



Universität  
Bremen

Fachbereich 3: Mathematik und Informatik

# Bachelorarbeit

## Entwicklung einer Anwendungsoberfläche für die Datenbankmigration mit GuttenBase

Sirajeddine Ben Zinab

Matrikel-Nr. 3094966

28. Februar 2021

**Betreuender Prüfer:** Prof. Dr. Sebastian Maneth

**Zweitgutachter:** Prof. Dr. Martin Gogolla



## **Selbstständigkeitserklärung**

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sämtliche wissentlich verwendete Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.

Bremen, den 28. Februar 2021

---

Sirajeddine Ben Zinab

## **Zusammenfassung**

Migration ist in der Wissenschaft kein neues Thema. Es existieren zahlreiche Methoden und Frameworks zur Beschreibung, Analyse und Implementierung der Migration. Dies gilt auch für Datenbankverwaltungssysteme (DBMS).

In dieser Arbeit werden aktuelle Tools für die Datenbankmigration vorgestellt. Dabei werden bedeutsamen Eigenschaften der Open-Source-Bibliothek GuttenBase erläutert.

Außerdem hat diese Arbeit hauptsächlich den Entwurf, die Implementierung und die Evaluation eines Tools für Datenbank Migration zwischen verschiedenen Datenbanksystemen (DBMS) basierend auf GuttenBase zum Thema. Um die Nutzung der GuttenBase-Bibliothek für möglichst viele Nutzer zu ermöglichen, erfolgt die Umsetzung als ein IntelliJ Plugin (IDEA).

Diese Bachelorarbeit wurde bei der Firma Akquinet AG in Bremen im Zeitraum von September 2020 bis zum März 2021 erstellt und stellt den Abschluss meines Bachelorstudiums an der Universität Bremen dar.

Der Text liegt in deutscher Sprache vor.

# Inhaltsverzeichnis

<b>Inhaltsverzeichnis</b>	<b>i</b>
<b>Abbildungsverzeichnis</b>	<b>iii</b>
<b>Tabellenverzeichnis</b>	<b>v</b>
<b>1 Einleitung</b>	<b>1</b>
1.1 Problemstellung und Motivation . . . . .	1
1.2 Zielsetzung . . . . .	1
1.3 Aufbau der Arbeit . . . . .	3
<b>2 Grundlagen</b>	<b>5</b>
2.1 Datenbankmanagementsysteme . . . . .	5
2.2 Datenbankmigration . . . . .	6
2.3 Aktueller Stand . . . . .	7
2.3.1 GuttenBase . . . . .	8
2.3.2 IntelliJ . . . . .	9
<b>3 Anforderungsmanagement</b>	<b>13</b>
3.1 Festlegung der Anforderungen . . . . .	13
3.2 Vereinfachtes Datenmodell . . . . .	14
3.3 Detaillierte Beschreibung der Anforderungen . . . . .	15
3.3.0.1 Migrationsoperation: Umbenennen erstellen . . . . .	16
3.3.0.2 Migrationsoperation erstellen: Datentyp ändern . . . . .	19
3.3.0.3 Migrationsoperationen verwalten . . . . .	21
3.3.0.4 Datenbankmigration durchführen . . . . .	22
<b>4 Konzeption und Implementierung</b>	<b>31</b>
4.1 Konzeptentwicklung . . . . .	31
4.1.1 Konzeptionelle Sicht . . . . .	31
4.1.1.1 GuttenBase Plugin . . . . .	32
4.1.1.2 IntelliJ Plattform . . . . .	33
4.1.1.3 GuttenBase . . . . .	34

4.1.2	Modulsicht . . . . .	34
4.1.2.1	Modulsicht der Übersicht der Konfigurationsschritte . . . . .	35
4.1.2.2	Modulsicht der allgemeinen Übersicht . . . . .	37
4.1.2.3	Modulsicht der Konfigurationsübersicht . . . . .	38
4.1.2.4	Modulsicht der Ergebnisübersicht . . . . .	39
4.2	Technologieauswahl . . . . .	41
4.2.1	Umsetzungsform . . . . .	41
4.2.2	IntelliJ Platform . . . . .	43
4.2.3	GuttenBase . . . . .	44
4.3	PlugIn Implementierung . . . . .	46
4.3.1	Funktion: Liste der vorhandenen Migrationsoperationen . . . . .	47
4.3.2	Funktion: Hinzufügen einer neuen Migrationsoperation . . . . .	48
4.3.3	Funktion: Verbindungerstellung . . . . .	48
4.3.4	Funktion: Konfiguration . . . . .	49
4.3.5	Funktion: Migrationsdurchführung . . . . .	52
<b>5</b>	<b>Evaluation</b>	<b>55</b>
5.1	Evaluationsmethodik . . . . .	55
5.2	Durchführung . . . . .	55
5.3	Ergebnisse . . . . .	56
5.4	Anpassungen . . . . .	58
<b>6</b>	<b>Fazit und Ausblick</b>	<b>59</b>
6.1	Fazit . . . . .	59
6.2	Ausblick . . . . .	59
	<b>Literatur</b>	<b>61</b>

# Abbildungsverzeichnis

3.1	Vereinfachtes Datenmodell . . . . .	15
3.2	Migrationsoperation „Umbenennen“ erstellen . . . . .	16
3.3	Übersicht Migrationsoperationen . . . . .	17
3.4	Migrationsoperation: Umbenennen . . . . .	17
3.5	Migrationsoperation „Datentyp ändern“ erstellen . . . . .	19
3.6	Migrationsoperationen verwalten . . . . .	21
3.7	Datenbanken verbinden . . . . .	22
3.8	Datenbankmigration View . . . . .	23
3.9	Migrationsprozess konfigurieren . . . . .	25
3.10	Übersicht Quelldatenbank . . . . .	26
3.11	Hinzugefügte Migrationsoperationen löschen . . . . .	27
3.12	Übersicht über die hinzugefügten Migrationsoperationen . . . . .	28
3.13	Migration starten . . . . .	29
3.14	Fortschritt des Migrationsprozesses . . . . .	29
4.1	Komponentendiagramm für die konzeptionelle Sicht . . . . .	32
4.2	Modulsicht der Übersicht der Konfigurationsschritte . . . . .	36
4.3	Modulsicht der allgemeinen Übersicht . . . . .	37
4.4	Modulsicht der Konfigurationsübersicht . . . . .	38
4.5	Modulsicht der Ergebnisübersicht . . . . .	40
4.6	Top IDE Index . . . . .	43
4.7	ConnectionInfo konfigurieren . . . . .	45
4.8	Connector Repository konfigurieren . . . . .	45
4.9	ColumnMapperHint hinzufügen . . . . .	46
4.10	Datenbank Migration durchführen . . . . .	46
4.11	Übersicht der Migrationsoperationen öffnen . . . . .	47
4.12	Übersicht der Migrationsoperationen . . . . .	48
4.13	Migrationsoperation Umbenennen erstellen . . . . .	49

4.14	Migrationsoperationen exportieren und importieren . . . . .	49
4.15	Allgemeine Übersicht der Datenbank Migration (generalView) . . . . .	50
4.16	Konfigurationsübersicht (overview) . . . . .	51
4.17	matches Methode der Migrationsoperation Umbenennen . . . . .	51
4.18	das Hinzufügen von der Migrationsoperation Umbenennen . . . . .	51
4.19	Ergebnisübersicht . . . . .	52
4.20	ConnectorHints hinzufügen . . . . .	53
4.21	Migration durchführen (MapperHelper Klasse) . . . . .	53
4.22	Fortschrittsübersicht . . . . .	54



# Tabellenverzeichnis

2.1	Tools für die Datenbankmigration . . . . .	10
2.2	Vergleich freier Migrationstools . . . . .	11
3.1	Anwendungsfall: Migrationsoperation Umbenennen erstellen . . . . .	18
3.2	Anwendungsfall: Migrationsoperation ‚Datentyp ändern‘ erstellen . . . . .	20
3.3	Anwendungsfall Migrationsoperationen verwalten . . . . .	22
3.4	Anwendungsfall Datenbanken verbinden . . . . .	24
3.5	Anwendungsfall Migrationsprozess konfigurieren . . . . .	27
3.6	Anwendungsfall Anwendungsfall ‚hinzugefügte Migrationsoperationen löschen‘ . .	28
3.7	Anwendungsfall Migration starten . . . . .	29
4.1	Umsetzungsmöglichkeiten . . . . .	42



# Einleitung

## 1.1 Problemstellung und Motivation

Die IT-Migration ist seit Anbeginn des Informationszeitalter ein zentraler Bestandteil der Informationsverarbeitung [WZ15]. Ebenso wie Hardware, Betriebssysteme sowie Programme, werden auch Datenbanken häufig migriert. Bei einer Datenbankmigration werden Daten von einer Quelldatenbank in eine Zieldatenbank verschoben. Die Notwendigkeit einer Datenbankmigration besteht darin, ein vorhandenes System auf ein modernes auszutauschen, das den aktuellen Anforderungen entspricht [HLS74].

Es gibt zahlreiche Tools zum Visualisieren oder Analysieren von Datenbanken (Tableau [DW18], SAP Crystal Reports [Dut16] etc.). Zudem gibt es einige Programme für die Datenbankmigration [Hor05]. Dazu gehört die GuttenBase-Bibliothek. Diese bietet durch die Möglichkeit der Hinzufügens von Migrationsoperationen eine gewisse Flexibilität während des Migrationsprozesses. Um diese Migrationsoperationen effizient auszunutzen und eine raschere und anpassbare Migration durchzuführen, lässt sich GuttenBase optimieren.

In diese Arbeit wird der Frage nachgegangen, wie sich die von der Firma Akquinet AG entwickelte Bibliothek optimieren lässt.

## 1.2 Zielsetzung

Die GuttenBase-Bibliothek lässt sich mithilfe unterschiedlicher Weiterentwicklungen optimieren. Im Rahmen dieser Arbeit soll eine Anwendungsoberfläche (GuttenBase-Plugin) für die Datenbankmigration basierend auf GuttenBase konzipiert, implementiert und anschließend evaluiert werden.

Das GuttenBase-Plugin soll die bedeutendsten Funktionalitäten von GuttenBase unterstützen.

Diese werden in der Anforderungsanalyse erläutert (siehe Kapitel 3).

Um ein benutzerfreundliches System zu erzielen, ist es ausschlaggebend, dass die zu entwickelnde Anwendungsoberfläche den Grundsätzen der Informationsdarstellung entspricht. Diese sind in der Norm DIN EN ISO 9241-112 zu finden und beinhalten die folgenden Grundsätze:

- Entdeckbarkeit: Die Informationen sollen in der Darstellung erkennbar sein und als vorhanden wahrgenommen werden können.
- Ablenkungsfreiheit: Die erforderlichen Informationen sollen ohne eine durch weitere Informationen verursachte Störung wahrgenommen werden.
- Unterscheidbarkeit: Elemente oder Gruppen von Elementen sollen voneinander unterschieden werden können. Die Darstellung sollte die Unterscheidung bzw. Zuordnung von Elementen und Gruppen unterstützen.
- Eindeutige Interpretierbarkeit: Die Informationen sollen auf eine Weise verstanden werden, wie es vorgesehen ist.
- Kompaktheit: Nur notwendige Informationen sollen dargestellt werden.
- Konsistenz: Ähnliche Absichten sollen ähnlich dargestellt werden und unterschiedliche Absichten sollen in unterschiedlicher Form dargestellt werden.

Die genannten Grundsätze sollen im Zusammenhang mit den Grundsätzen der Benutzer-System-Interaktion („Dialogprinzipien“) angewendet werden. Diese beinhalten nach der Norm DIN EN ISO 9241-11 die folgenden Grundsätze:

- Aufgabenangemessenheit: Die Schritte sollen nicht überflüssig sein und keine irreführende Informationen beinhalten.
- Selbstbeschreibungsfähigkeit: Es sollen lediglich jene Informationen dargestellt werden, die für einen bestimmten Schritt erforderlich sind.
- Erwartungskonformität: Das System verhält sich nach der Durchführung einer bestimmten Aufgabe wie erwartet.
- Lernförderlichkeit: Der Benutzer kann den entsprechenden Schritt durchführen, ohne Vorwissen zu haben bzw. eine Schulung zu besuchen.
- Steuerbarkeit: Der Benutzer kann konsequent und ohne Umwege in Richtungen gehen, die für die zu erledigende Aufgabe erforderlich sind.
- Fehlertoleranz: Das System soll den Benutzer vor Fehlern schützen. Wenn Fehler gemacht werden, sollen diese mit minimalem Aufwand behoben werden können.
- Individualisierbarkeit: Der Benutzer kann die Anwendungsoberfläche mittels individueller Voreinstellungen anpassen.

Mit den oben genannten Grundsätzen wird sichergestellt, dass das GuttenBase-Plugin effektiv, effizient und zufriedenstellend ist. Diese drei Merkmale entsprechen den drei Zielen der Gebrauchstauglichkeit (Usability).

## 1.3 Aufbau der Arbeit

Zu Beginn der Arbeit werden einige Grundbegriffe der Datenbankmigration erläutert. Außerdem werden die Eigenschaften der GuttenBase-Bibliothek aktuelle Tools für die Datenbankmigration erläutert (Kapitel 2).

Der Hauptteil dieser Arbeit hat die Umsetzung des Guttenbase-Plugins zum Thema. Dabei wird zuerst eine Anforderungsanalyse durchgeführt, um den Sollzustand zu definieren (Kapitel 3). Um die technische Machbarkeit zu prüfen und Zeit bei der Entwicklung zu sparen, werden bei der Analyse einige GUI-Prototypen erstellt. Somit wird bereits zu Beginn der Umsetzung klar sein, wie die zu entwickelnde Anwendungsoberfläche die Funktionalitäten von GuttenBase unterstützen würde.

Die Softwarearchitektur erfolgt im darauffolgenden Kapitel (Kapitel 4). Diese wird basierend auf dem Siemens-Blickwinkel erstellt. Zunächst werden die verwendeten Technologien sowie das Ergebnis vorgestellt.

Im darauffolgenden Kapitel (5) wird das Ergebnis evaluiert. Dabei wird ein Experteninterview durchgeführt.

Anschließend folgen eine Zusammenfassung sowie Ideen für Optimierungsmöglichkeiten (Kapitel 6).



# Grundlagen

In diesem Kapitel wird einen allgemeinen Einblick in einige Grundaspekte der Datenbankmigration gegeben. Außerdem werden vorhandene Arbeiten vorgestellt und einige Tools für die Datenbankmigration untersucht. Anschließend folgt eine Einführung in GuttenBase sowie die IntelliJ-Plugin-Entwicklung.

## 2.1 Datenbankmanagementsysteme

Damit Daten auf einem Computer verwaltet werden können, werden Datenbankmanagementsysteme (DBMS) benötigt. Diese sind leistungsfähige Programme für die flexible Speicherung und Abfrage strukturierter Daten.

Außerdem hilft ein DBMS bei der Organisation und Integrität von Daten und regelt den Zugriff auf Datengruppen [Gei14].

Ein DBMS kann aus einem einzelnen Programm bestehen. Dies ist z. B. bei einem Desktop-DBMS der Fall. Es kann jedoch auch aus verschiedenen Programmen bestehen, die interagieren und die Funktion des DBMS bereitstellen. Dies ist z. B. bei servergestützten Datenbanksystemen der Fall.

Um eine Datenbankanwendung zu implementieren, muss auf das Datenbankmodell geachtet werden. Dieses stellt die Daten einer Datenbank und deren Beziehungen abstrakt dar. Meist wird ein relationales Datenbankmodell eingesetzt [NK92]. Dieses weist im Gegensatz zu anderen Datenbankmodellen keine strukturelle Abhängigkeit auf und versteckt die physikalische Komplexität der Datenbank vollständig vor den Anwendern.

Es stehen zahlreiche Datenbankmanagementsysteme zur Verfügung. Die folgenden sind einige der gängigsten DBMS:

- Microsoft SQL Server
- MS-Access
- MySQL

- PostgreSQL
- HSQLDB
- H2 Derby
- Oracle
- DB2
- Sybase

Um ein geeignetes DBMS auszuwählen, sind zahlreiche Kriterien wie die Ausführungszeit sowie die CPU- und Speicher Nutzung zu beachten. Im Artikel von Bassil, A Comparative Study on the Performance of the Top DBMS Systems, aus dem Jahr 2011 werden einige Datenbankmanagementsysteme anhand der genannten Kriterien überprüft [Bas12].

## 2.2 Datenbankmigration

Die Migration von Datenbanken dient zum Verschieben von Daten von der Quelldatenbank zur Zieldatenbank einschließlich der Schemaübersetzung und der Datentransformation. Das Quellschema wird dabei semantisch in ein Zielschema übersetzt. Darauf basierend werden die enthaltenen Daten entsprechend den Datenbanktypen konvertiert [EA15].

Gründe für eine Datenbankmigration sind:

- Upgrade auf eine neue Software oder Hardware,
- Änderung der Unternehmensrichtlinien,
- Investition in IT-Dienstleistungen,
- Zusammenführen mehrerer Datenbanken in eine Datenbank für eine einheitliche Datenansicht;
- die Wartung des existierenden Systems ist schwierig oder nicht möglich.

Wie jede Migration, erfolgt eine Datenbankmigration i. d. R. in mehreren Schritten. Diese beinhalten die Planung, den Entwurf, das Laden sowie die Überprüfung der Daten [Kow18].

Bei einer Datenbankmigration ergeben sich stets Herausforderungen. Dies liegt meist daran, dass Datenbankmanagementsysteme unterschiedliche Formate zum Speichern der Daten aufweisen. Zu den Herausforderungen einer Datenbankmigration gehören ein Datenverlust, ungeeignete Migrationstools, Fehler beim Validieren bzw. Testen des Migrationsprozesses und unzureichende Kenntnisse bezüglich der Verwendung von Migrationstools [KT18].

Während des Migrationsprozesses ist häufig eine benutzerdefinierte Lösung erforderlich, um die Daten entsprechend dem Zielsystem konvertieren zu können. Dies kann durch das Definieren von Migrationsoperationen erreicht werden. Im Folgenden werden einige bedeutende Migrationsoperationen erläutert, die während einer Datenbankmigration benötigt werden:

- **M1: Schemaauswahl**



Der Benutzer kann ein bestimmtes Schema für die Migration auswählen.

- **M2: Tabellenauswahl**

Der Benutzer kann mehrere Tabellen für die Migration auswählen. Alternativ dazu können Tabellen ausgeschlossen werden.

- **M3: Spaltenauswahl**

Der Benutzer kann mehrere Spalten für die Migration auswählen. Alternativ dazu können Spalten ausgeschlossen werden.

- **M4: Tabellenumbenennung**

Die Tabellen lassen sich in der Zieldatenbank umbenennen.

- **M5: Spaltenumbenennung**

Die Spalten lassen sich in der Zieldatenbank umbenennen.

- **M6: Benutzerdefinierte SQL-Klausel**

Damit lassen sich die SELECT Anweisungen konfigurieren, die für das Lesen der Daten aus der Quelldatenbank verwendet werden.

- **M7: Migrationsfortschritt**

Der Fortschritt der Migration wird angezeigt. Eventuell wird ein Protokoll des Migrationsprozesses ausgegeben.

- **M8: Zielschemaerstellung**

Meist verfügt die Zieldatenbank nicht über ein geeignetes Schema für die Migration. Mit dieser Migrationsoperation kann ein Zielschema vor der Migration erstellt werden.

## 2.3 Aktueller Stand

Die mit dem Migrationsworkflow verbundenen Aufgaben, etwa das Schreiben von Skripten und das Mapping der Datenbankelementen, sind vielfältig und z. T. komplex. Das manuelle Ausführen dieser Aufgaben erfordert häufig viel Zeit und ein erfahrenes Team. Um Zeit und Kosten bei der Migration zu sparen und um wiederholende Aufgaben zu automatisieren, bieten sich zahlreiche Tools bzw. Prototypen für die Datenbankmigration (DBMT für das Database-Migration-Tool). Einige dieser Tools werden in Tabelle 2.1 vorgestellt [Hor05]. Dabei werden sowohl Open-Source als auch kommerzielle Tools betrachtet. Die Tabelle liefert außerdem einen Einblick in die unterstützte Quell- und Zieldatenbank sowie die Plattformabhängigkeit. Die Einträge wurden nach dem Datum des letzten Release sortiert.

Um einen Überblick zu erlangen, wurden die jene Tools untersucht, für die eine freie Lizenz angeboten wird (MySQL Workbench Migration Wizard, Open-DBcopy und Squirrel-DBCOPY-Plugin). Tabelle 2.2 zeigt die unterstützten Migrationsoperationen (siehe Abschnitt 2.2) sowie einige Vor- und Nachteile für jedes der genannten Tools. Das MySQL-Workbench-Migrationstool ist ein Teil der MySQL Workbench, die von Oracle<sup>1</sup> entwickelt wurde. Es verfügt über eine benutzerfreundliche Anwendungsoberfläche, wobei die Migration in mehreren Schritten erfolgt. Während der Migration hat der Benutzer eine Übersicht über die Quell- und Zieldatenbanken. Außerdem gibt es die Möglichkeit, die SQL-Anfragen für jedes Datenbankelement zu editieren. Die ermöglicht eine angemessene Konfiguration des Migrationsprozesses.

Das Open-DBCopy-Tool, das von Puzzle ITC entwickelt wurde<sup>5</sup>, wird als eigenständiges Programm bereitgestellt. Bemerkenswert ist dabei die Unterstützung von zahlreichen Migrationsoperationen inklusive der Tabellen- und Spaltenumbenennung. Diese sind allerdings nicht vollständig vom Tool automatisiert, sondern müssen vom Nutzer übernommen werden, indem die Hibernate<sup>2</sup> Mapping Dateien überschrieben werden. Das Open-DBCopy-Tool verfügt zwar über eine umfangreiche Benutzeroberfläche, diese ist allerdings nicht benutzerfreundlich und modern genug<sup>3</sup>, um eine angemessene Konfiguration des Migrationsprozesses durchzuführen.

Das DBCopy-Plugin<sup>6</sup> ist ein Teil des Squirrel-SQL-Client<sup>4</sup>, der ein grafisches Werkzeug für die Verbindung zu Datenbanken darstellt. Dieses Migrationstool bietet eine minimale Datenbankmigration, welche nach dem Klick auf die zu migrierende Tabelle erfolgt. Dabei hat der Nutzer ein einfaches und intuitives Benutzererlebnis, allerdings ist der Funktionsumfang eingeschränkt. Zusammenfassend lässt sich sagen, dass keines der erwähnten Tools eine vollständige Liste von Migrationsoperationen beinhaltet. Außerdem ist auffallend, dass die Tabellen- bzw. Spaltenumbenennung lediglich von einem Tool teilweise unterstützt wird.

### 2.3.1 GuttenBase

Zahlreiche Softwareunternehmen haben sich dafür entschieden, ein eigenes Tool für Datenbankmigration zu entwickeln. Dies ist der Fall in der Firma Akquinet AG, in der die Open-Source-Bibliothek GuttenBase im Jahr 2012 entwickelt wurde.

---

<sup>i</sup><https://www.altova.com/> [25.03.2021]

<sup>ii</sup><https://www.ispirer.com/products/why-sqlways> [25.03.2021]

<sup>iii</sup><https://dbconvert.com/> [25.03.2021]

<sup>iv</sup><https://www.mysql.com/products/workbench/migrate> [25.03.2021]

<sup>v</sup><http://opendbcopy.sourceforge.net/> [25.03.2021]

<sup>vi</sup><http://dbcopypugin.sourceforge.net/> [25.03.2021]

<sup>vii</sup><https://www.astera.com/de/products/centerprise-data/> [25.03.2021]

<sup>viii</sup><https://sourceforge.net/projects/progressiondb/> [25.03.2021]

<sup>ix</sup><https://sourceforge.net/projects/mssql2pgsql/> [25.03.2021]

<sup>x</sup><http://shift2ingres.sourceforge.net/> [25.03.2021]

<sup>1</sup><https://www.oracle.com/> [26.03.2021]

<sup>2</sup><http://hibernate.org/> [26.03.2021]

<sup>3</sup><http://opendbcopy.sourceforge.net/user-manual.pdf>, S30 [26.03.2021]

<sup>4</sup><http://www.squirrelsql.org/> [26.03.2021]

GuttenBase verfügt zwar über keine Benutzeroberfläche, beinhaltet aber zahlreiche Funktionalitäten für die Datenbankmigration. Es werden mehrere Migrationsoperationen angeboten. Bis auf die Schemaerstellung (M8), werden alle Migrationsoperationen unterstützt, die in Abschnitt 2.2 definiert wurden. Zusätzlich beinhaltet GuttenBase weitere Migrationsoperationen, mit denen die Datenbankmigration zusätzlich individualisiert werden kann.

GuttenBase kann aktuell in jedes Java-Projekt (z. B. Gradle oder Maven) importiert und darin genutzt werden. Die Nutzung von GuttenBase wird in Abschnitt 4.2.3 detailliert erläutert.

### 2.3.2 IntelliJ

Die von der Firma JetBrains und in Java entwickelte Entwicklungsumgebung (IDE), IntelliJ IDEA, ist ein Teil einer Reihe von ähnlich JetBrains Entwicklungsumgebungen wie Clion, PyCharm, PhpStorm, DataGrip usw. Diese basieren auf dem gleichen Kern, nämlich dem IntelliJ-Plattform-SDK, der eine freie Lizenz aufweist und von Dritten zum Erstellen von IDEs verwendet werden kann.

IntelliJ IDEA eignet sich für die Implementierung in Java, Kotlin, Groovy und Scala. Die Umgebung hat außerdem unterschiedliche Funktionalitäten. Dazu gehören ein GUI-Editor, ein Build-Management-Tool (z. B. Gradle) und andere Frameworks etwa für die Versionskontrolle, das Refactoring, und einen Test.

Der Funktionsumfang dieser IDE kann mittels Plugins erweitert werden. Ein Beispiel davon ist das Database-Tool and SQL-Plugin, die zur Datenbankverwaltung dienen. Damit können Datenbanken erstellt, verwaltet und gelöscht werden. Außerdem werden die meisten Datenbankmanagementsysteme unterstützt.

Name	Quell-DBMS	Ziel-DBMS	Lizenz	Betriebs-systeme	Datum des neusten Release
MapForce (Altova) <sup>i</sup>	SQL Server, DB2, MS Access, MySQL und PostgreSQL	SQL Server, DB2, MS Access und Oracle	Kommerziell	Windows, Linux und Mac OS	2021
SQLWays (Inspirer) <sup>ii</sup>	Alle RDMBS	PostgreSQL und MySQL	Kommerziell	Windows	2020
DBConvert (DB Convert) <sup>iii</sup>	Oracle, DB2, SQLite, MySQL, PostgreSQL, MS Access und Foxpro	Oracle, DB2, SQLite, MySQL, PostgreSQL, MS Access und Foxpro	Kommerziell	Windows	15.12.2020
MySQL Workbench Migration Wizard (MySQL AB) <sup>iv</sup>	MS Access und Oracle	MySQL	Frei	Windows	07.12.2020
Open DBCopy (Puzzle ITC) <sup>v</sup>	Alle RDBMS	Alle RDBMS	Frei	Alle Betriebssysteme	27.07.2020
Squirrel DBCopy-Plugin (Sourceforge) <sup>vi</sup>	Alle RDBMS	Alle RDBMS	Frei	Alle Betriebssysteme	30.04.2020
Centerprise Data Integrator (Astera) <sup>vii</sup>	SQL Server, DB2, MS Access, MySQL und PostgreSQL	SQL Server, DB2, MS Access, MySQL und PostgreSQL	Kommerziell	Windows	02.2020
Progression DB (Versora) <sup>viii</sup>	MS SQL	PostgreSQL, MySQL und Ingres	Frei	Linux und Windows	01.05.2015
Mssql2Pgsql (OS Project) <sup>ix</sup>	MS SQL	PostgreSQL	Frei	Windows	17.06.2005
Shift2Ingres (OS Project) <sup>x</sup>	Oracle und DB2	Ingres	Frei	Alle Betriebssysteme	20.05.2005

**Tabelle 2.1** Tools für die Datenbankmigration

Migrationsooperationen	MySQL Workbench Migration Wizard	Open DBcopy	Squirrel DBCOPY Plugin
Schemaauswahl (M1)		✓	
Tabellenauswahl (M2)	✓	✓	✓
Spaltenauswahl (M3)	✓	✓	
Tabellenumbenennung (M4)		✓	
Spaltenumbenennung (M5)		✓	
Benutzerdefinierte SQL-Klausel (M6)	✓		
Migrationsfortschritt (M7)	✓	✓	✓
Zielschemaerstellung (M8)	✓		

**Tabelle 2.2** Vergleich freier Migrationstools



## Anforderungsmanagement

In diesem Kapitel werden die funktionalen Anforderungen für das GuttenBase-Plugin festgelegt. Zunächst werden die Anforderungen grob definiert. Danach werden sie detailliert mithilfe eines vereinfachten Datenmodells beschrieben.

### 3.1 Festlegung der Anforderungen

Für die Anforderungsanalyse fanden mehrere Kundengespräche statt. Diese ergaben die folgenden Punkte, die vom GuttenBase-Plugin erfüllt werden sollen:

- **A1: Migrationsoperationen verwalten:**  
Um den Migrationsprozess zu individualisieren, soll der Benutzer die Möglichkeit haben, neue Migrationsoperationen zu erstellen, zu editieren und zu löschen.
- **A2: Migrationsoperationen speichern:**  
Die hinzugefügten Migrationsoperationen sollen nach der Bestätigung vom Benutzer gespeichert werden können. Sie sollen auch nach einem Neustart der Anwendung zur Verfügung stehen.
- **A3: Überblick über alle Migrationsoperationen:**  
Der Benutzer soll über eine tabellarische Auflistung aller erstellten Migrationsoperationen verfügen.
- **A4: Datenbanken verbinden:**  
Um eine erfolgreiche Migration durchzuführen, soll der Benutzer in der Lage sein, eine Verbindung zwischen der Quell- und Zieldatenbank herzustellen. Die zu migrierende Datenbank sowie die Zieldatenbank sollen aus den existierenden Datenbanken ausgewählt werden können.
- **A5: Überblick über enthaltene Datenbankelemente:**  
Während des Migrationsprozesses soll der Benutzer einen Überblick über alle in der Quelldatenbank enthaltenen Tabellen bzw. Spalten haben.

- **A6: Existierende Migrationsoperationen zur Migration hinzufügen:**

Die gespeicherten Migrationsoperationen sollen in der Übersicht der Datenbankelemente zur Verfügung stehen. Diese können auf die entsprechenden Datenbankelemente angewendet werden.

- **A7: Hinzugefügte Migrationsoperationen löschen**

Der Benutzer soll die Möglichkeit haben, hinzugefügte Migrationsoperationen zu löschen, nachdem sie zur Migration hinzugefügt worden sind.

- **A8: Migrationsprozess starten**

Im letzten Schritt der Migration kann der Benutzer den Migrationsprozess mit den hinzugefügten Migrationsoperationen starten.

- **A9: Überblick über den Fortschritt der Migrationsprozesse**

Der Benutzer soll die Möglichkeit haben, hinzugefügte Migrationsoperationen zu löschen, nachdem sie zur Migration hinzugefügt worden sind.

Um den Umfang dieser Arbeit in Grenzen zu halten, wurden folgende Migrationsoperationen für die Umsetzung ausgewählt:

- Tabellenauswahl,
- Spaltenauswahl,
- Tabellenumbenennung,
- Spaltenumbenennung,
- Migrationsfortschritt,
- Datentypen der Spalten ändern.

## 3.2 Vereinfachtes Datenmodell

In diesem Abschnitt wird ein vereinfachtes Datenmodell erstellt. Mit diesem wird gezeigt, welche Einheiten des Systems relevant sind und welche Beziehungen zwischen diesen Einheiten gelten. Es handelt sich dabei noch nicht um eine Spezifikation von Klassen für die Implementierung, sondern um eine Modellierung der realen Welt. Das Datenmodell ist maßgebend für die Architektur des Plugins. Aus diesem Grund wird auf unnötige Details bzw. Attribute verzichtet. Das Datenmodell wird als UML-Klassendiagramm in Abbildung 3.1 angegeben. Dabei werden hauptsächlich die Migrationsoperationen und deren Beziehungen dargestellt. Die abstrakte Klasse ‚GBAction‘ definiert alle möglichen Migrationsoperationen. Sie wird durch die folgenden Unterklassen erweitert:

- **Rename:**

Damit wird das Umbenennen eines Datenbankelementes modelliert. Jede Rename-Klasse hat einen Typ (Rename-Type). Dieser definiert, wie das Umbenennen des Datenbankelementes erfolgen soll und entspricht den in Abschnitt 3.3.0.1 vorgestellten Optionen für das



Umbenennen. Außerdem wird die Umbenennung von Spalten bzw. Tabellen in zwei weiteren Unterklassen spezifiziert.

- ChangeColumnType:

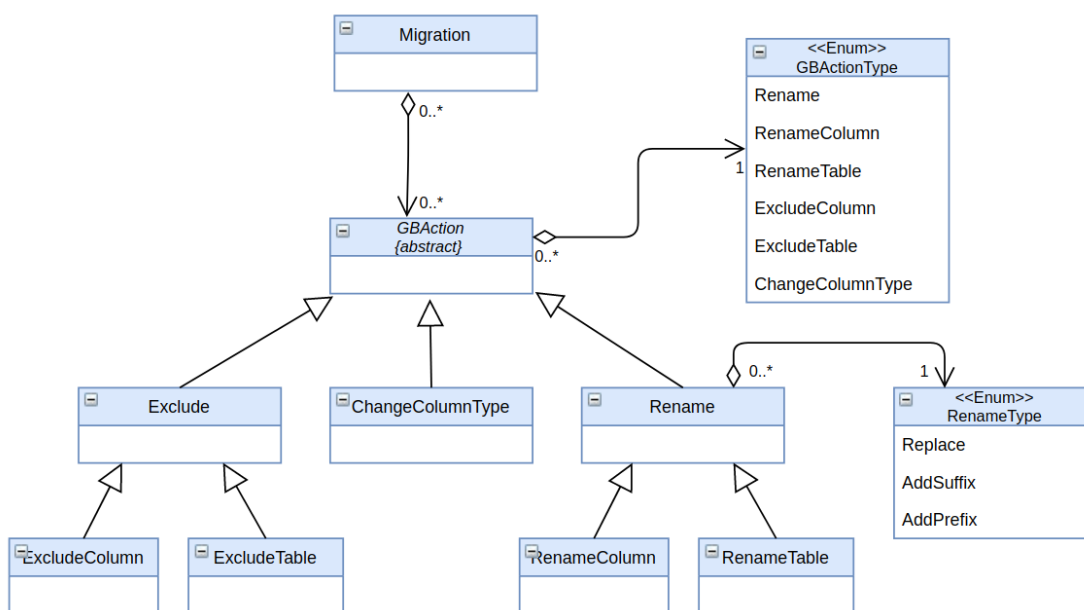
Diese Klasse entspricht der Migrationsoperation „Datentypen von Spalten ändern“.

- Exclude:

Mit der Exclude-Klasse wird die Migrationsoperation für das Ausschließen einer Spalte bzw. einer Tabelle modelliert.

Außerdem enthält die Klasse Migration alle Migrationsoperationen die beim Migrationsprozess angewendet werden.

**Abbildung 3.1** Vereinfachtes Datenmodell

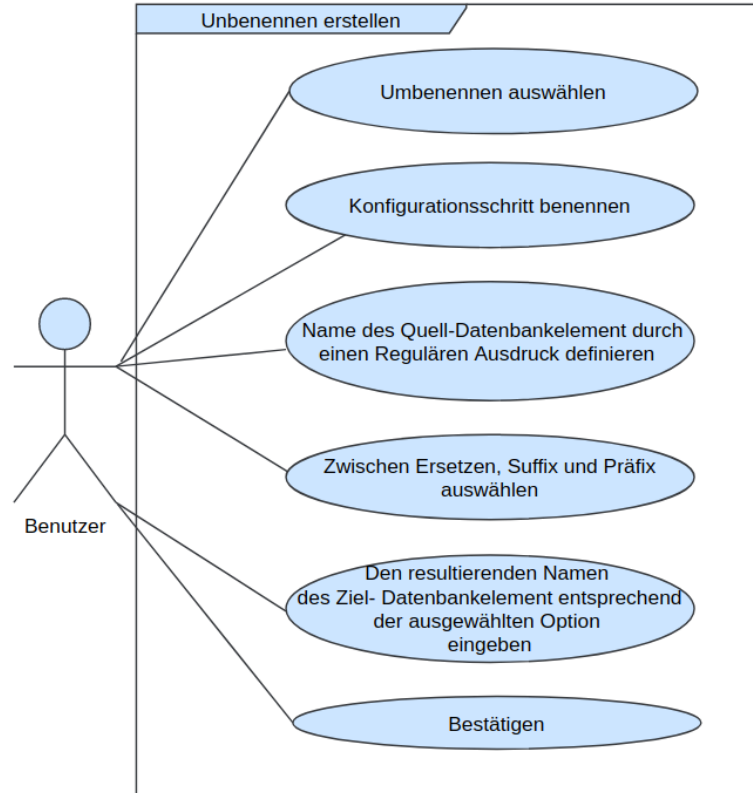


### 3.3 Detaillierte Beschreibung der Anforderungen

In diesem Abschnitt werden die zu implementierenden Anforderungen behandelt. Diese decken den zentralen Funktionsumfang des Systems ab. Neben der textuellen Beschreibung werden auch Anwendungsfalldiagramme erstellt. Dabei wird lediglich ein Akteur identifiziert. Dieser ist der Benutzer, der die Datenbankmigration durchführt. Um die Benutzungsführung in den Anwendungsfällen zu illustrieren und die konkrete Benutzeroberfläche, die es zu implementieren gilt, zu spezifizieren, wurden Papierprototypen für die ausschlaggebenden Teile des Systems erzeugt. Diese basieren auf der Norm DIN EN ISO 9241-210.

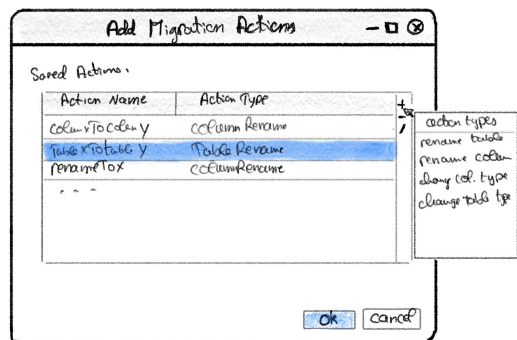
### 3.3.0.1 Migrationsoperation: Umbenennen erstellen

Abbildung 3.2 Migrationsoperation „Umbenennen“ erstellen



Dieser Anwendungsfall bildet den Vorgang ab, wenn ein Benutzer eine neue Migrationsoperation für das Umbenennen von Spalten bzw. Tabellen in der Zieldatenbank erstellt. Es wird vorausgesetzt, dass der Benutzer die Übersicht aller Migrationsoperationen bereits geöffnet hat (siehe Abbildung 3.3).

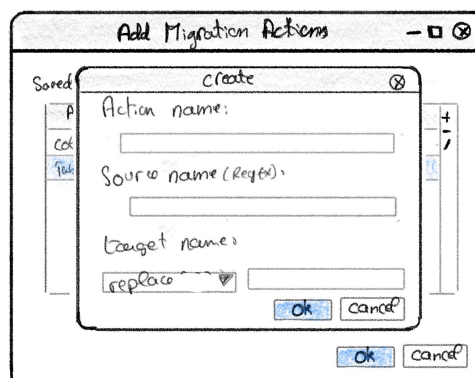
Abbildung 3.3 Übersicht Migrationsoperationen



Zu Beginn soll der Benutzer die zu erstellende Migrationsoperation benennen (z. B. ‚Rename id to identifier‘). Das Quelldatenbankelement (Spalte oder Tabelle) soll durch einen regulären Ausdruck definiert werden (siehe Abbildung 3.4). Dieser wird in der Migrationsoperation gespeichert, damit er später auf das Datenbankelement angewendet werden kann, das diesen Ausdruck erfüllt. Anschließend wird der Zielname des Datenbankelementes festgelegt. Dieser wird vom Benutzer als eine Zeichenkette angegeben. Dabei stehen drei Optionen zur Verfügung:

- **Ersetzen:** Der vollständige Name des entsprechenden Datenbankelements wird durch die übergebene Zeichenkette ersetzt.
- **Suffix hinzufügen:** Die übergebene Zeichenkette wird als Suffix zum ursprünglichen Namen hinzugefügt.
- **Präfix hinzufügen:** Die übergebene Zeichenkette wird als Präfix zum ursprünglichen Namen hinzugefügt.

Abbildung 3.4 Migrationsoperation: Umbenennen



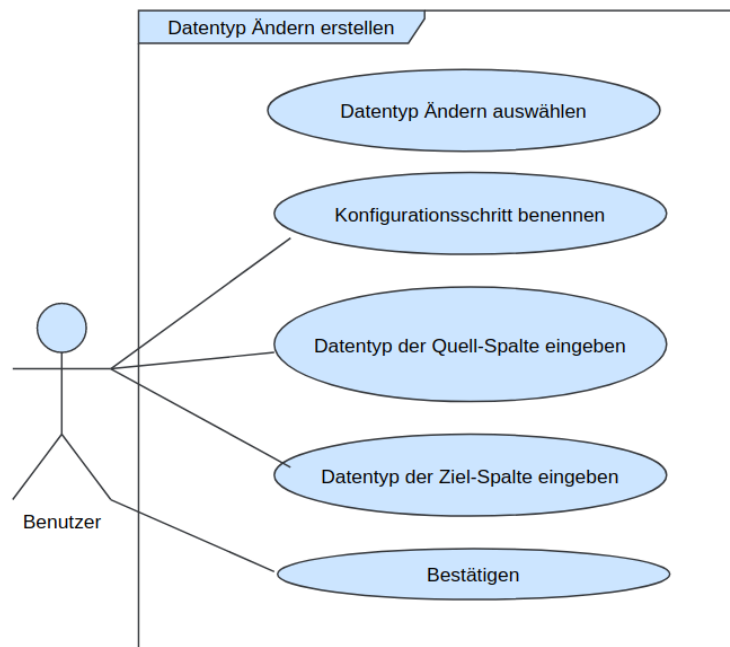
Nach dem Bestätigen der Eingaben wird die neu erstellte Migrationsoperation der Liste aller Migrationsoperationen hinzugefügt.

<b>Name</b>	Migrationsoperation Umbenennen erstellen
<b>Akteure</b>	Benutzer
<b>Auslöser</b>	Der Nutzer ist in der Übersicht der Migrationsoperationen und hat auf den „+“Button geklickt.
<b>Vorbedingung</b>	Der Nutzer besitzt eine List von Migrationsoperationen.
<b>Nachbedingung</b>	Eine Migrationsoperation vom Typ Umbenennen wird zur Liste aller Migrationsoperationen hinzugefügt.
<b>Ablauf</b>	<ol style="list-style-type: none"> <li>1. Die Option „Umbenennen“auswählen.</li> <li>2. Einen Namen für die zu erstellende Migrationsoperation eingeben.</li> <li>3. Den Namen des Quelldatenbankelement durch einen regulären Ausdruck definieren.</li> <li>4. Zwischen Ersetzen, Suffix und Präfix auswählen.</li> <li>5. Den resultierenden Namen des Ziedatenbankelements entsprechend der ausgewählten Option eingeben.</li> <li>6. Bestätigen.</li> </ol>

**Tabelle 3.1** Anwendungsfall: Migrationsoperation Umbenennen erstellen

### 3.3.0.2 Migrationsoperation erstellen: Datentyp ändern

Abbildung 3.5 Migrationsoperation „Datentyp ändern“ erstellen



Dieser Anwendungsfall zeigt, wie der Benutzer die Migrationsoperation ‚Datentyp ändern‘ erstellt. Wie beim vorangegangenen Anwendungsfall soll sich der Benutzer in der Übersicht über alle Migrationsoperationen befinden, um in den Anwendungsfall einzutreten (siehe Abbildung 3.3).

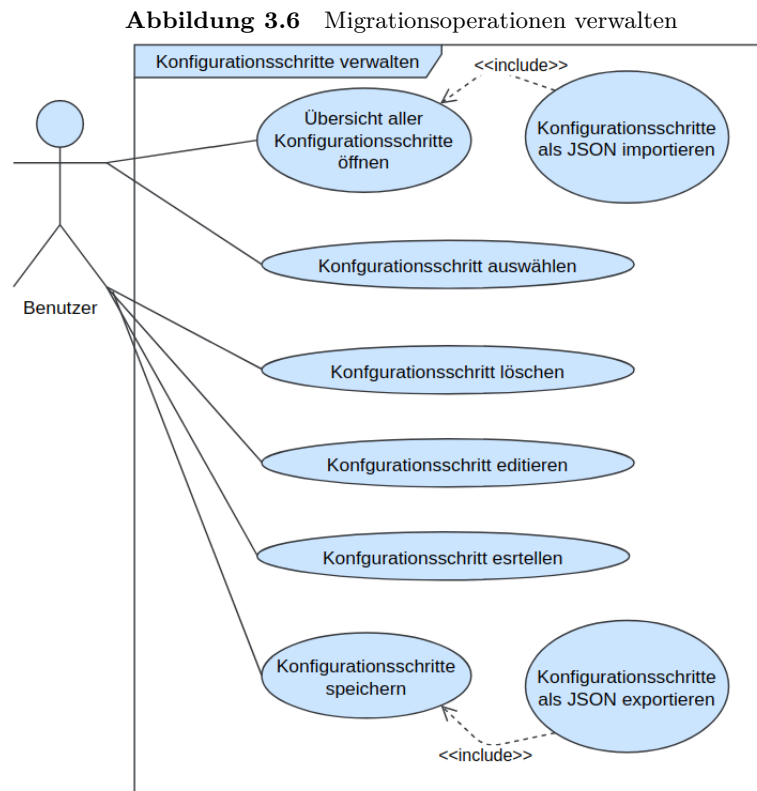
Nach dem Auslösen des Anwendungsfalles soll der Benutzer die Option ‚Datentyp ändern‘ auswählen. Danach hat der Benutzer die Möglichkeit, die Migrationsoperation zu benennen, den Datentyp der Quelldatenbank bzw. der Zieldatenbank als Zeichenkette einzugeben und anschließend die Eingaben zu bestätigen. Nachdem dieser Anwendungsfall beendet worden ist, wird eine neue Migrationsoperation hinzugefügt.

Das Erstellen der Migrationsoperation ‚Exkludieren‘ funktioniert ähnlich wie die zwei vorherigen Anwendungsfälle und wird daher nicht behandelt.

<b>Name</b>	Migrationsoperation Datentyp ändern erstellen
<b>Akteure</b>	Benutzer
<b>Auslöser</b>	Der Nutzer ist in der Übersicht der Migrationsoperationen und hat auf den ‚+‘-Button geklickt.
<b>Vorbedingung</b>	Der Nutzer besitzt eine List von Migrationsoperationen.
<b>Nachbedingung</b>	Eine Migrationsoperation vom Typ ‚Datentyp ändern‘ wird zur Liste aller Migrationsoperationen hinzugefügt.
<b>Ablauf</b>	<ol style="list-style-type: none"> <li>1. Die Option ‚Datentyp ändern‘ auswählen</li> <li>2. Einen Namen für die zu erstellende Migrationsoperation eingeben</li> <li>3. Den Datentyp der Quellspalte festlegen</li> <li>4. Den Datentyp der Zielspalte eingeben</li> <li>5. Bestätigen</li> </ol>

**Tabelle 3.2** Anwendungsfall: Migrationsoperation ‚Datentyp ändern‘ erstellen

### 3.3.0.3 Migrationsoperationen verwalten



Dieser Anwendungsfall beinhaltet die Verwaltung der Migrationsoperationen. Zunächst soll der Benutzer die Übersicht der Migrationsoperationen öffnen (siehe Abbildung 3.3). Dabei werden alle gespeicherten Migrationsoperationen geladen. Diese werden aus einer JSON-Datei erzeugt. In der Übersicht kann der Benutzer einzelne oder mehrere Migrationsoperationen zugleich löschen. Außerdem kann er Migrationsoperationen erstellen (Dieser Vorgang wurde in den vorangegangenen Anwendungsfällen beschrieben). Das Editieren der Migrationsoperationen erfolgt ebenso wie das Erstellen.

Anschließend können die Migrationsoperationen nach der Bestätigung durch den Benutzer als JSON gespeichert werden.

<b>Name</b>	Migrationsoperationen verwalten
<b>Akteure</b>	Benutzer
<b>Auslöser</b>	Der Benutzer klickt auf einen Button, um die Übersicht aller Migrationsoperationen zu sehen.
<b>Vorbedingung</b>	Der Benutzer verfügt über eine initiale Liste der Migrationsoperationen.
<b>Nachbedingung</b>	Die Änderungen sind vorgenommen und gespeichert.
<b>Ablauf</b>	<ol style="list-style-type: none"> <li>1. Übersicht aller Migrationsoperationen öffnen.</li> <li>2. eventuell Migrationsoperationen auswählen.</li> <li>3. eventuell Migrationsoperationen löschen.</li> <li>4. eventuell eine Migrationsoperation editieren.</li> <li>5. eventuell eine Migrationsoperation erstellen.</li> <li>6. Migrationsoperationen speichern.</li> </ol>

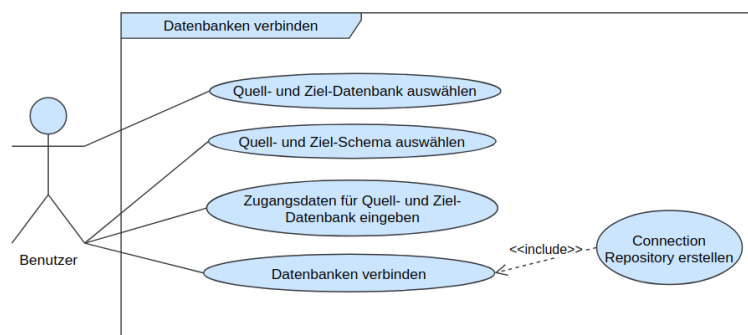
**Tabelle 3.3** Anwendungsfall Migrationsoperationen verwalten

#### 3.3.0.4 Datenbankmigration durchführen

Die Durchführung der Datenbankmigration deckt die Hauptfunktionalität des GuttenBase-Plugins ab. Die Migration wird in die folgenden Anwendungsfälle unterteilt:

##### Datenbanken verbinden

**Abbildung 3.7** Datenbanken verbinden





Für das Eintreten dieses Anwendungsfalls wird vorausgesetzt, dass der Benutzer über mindestens eine Datenbank verfügt. Zu Beginn muss er das Migrationsfenster öffnen, um die Eingabefelder zu sehen. Anschließend soll er die Datenbank, das Schema sowie die Zugangsdaten für das Quell- und Ziel-DBMS angeben (siehe Abbildung 3.8). Wenn die Eingaben stimmen, kann der Benutzer eine Verbindung zwischen den beiden Datenbanken herstellen. Ansonsten wird eine entsprechende Meldung angezeigt. Bei diesem Schritt wird das Connector-Repository der GuttenBase-Bibliothek erstellt und konfiguriert. Somit ist die Datenbankmigration bereit für die Konfiguration.

**Abbildung 3.8** Datenbankmigration View

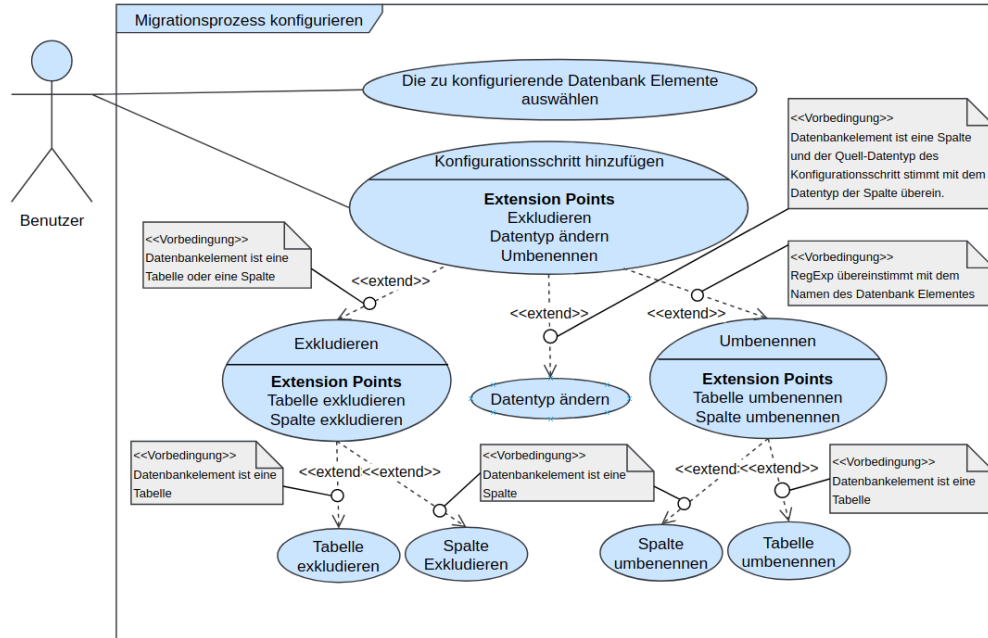
The image is a hand-drawn sketch of a software window titled "Migrate Database". The window has a standard title bar with a close button (an 'X' in a circle) on the right. The main content area is divided into two sections by a horizontal line. The top section is labeled "Source:" and contains four input fields with labels "database:", "schema:", "username:", and "password:". The bottom section is labeled "Target:" and also contains four input fields with the same labels: "database:", "schema:", "username:", and "password:". At the bottom right of the window, there is a small button labeled "next".

<b>Name</b>	Datenbanken verbinden
<b>Akteure</b>	Benutzer
<b>Auslöser</b>	Der Benutzer klickt auf einen Button um die Übersicht der Datenbankverbindung zu öffnen.
<b>Vorbedingung</b>	Die Quell- und Zieldatenbanken sind nicht mit dem GuttenBase-Tool verbunden.
<b>Nachbedingung</b>	Die Quell- und Zieldatenbanken sind verbunden.
<b>Ablauf</b>	<ol style="list-style-type: none"><li>1. Quelldatenbank auswählen</li><li>2. Quellschema auswählen</li><li>3. Benutzername der Quelldatenbank eingeben</li><li>4. Passwort der Quelldatenbank eingeben</li><li>5. Zieldatenbank auswählen</li><li>6. Zielschema auswählen</li><li>7. Benutzername der Zieldatenbank eingeben</li><li>8. Passwort der Zieldatenbank eingeben</li><li>9. Datenbanken verbinden</li></ol>

**Tabelle 3.4** Anwendungsfall Datenbanken verbinden

## Migrationsprozess konfigurieren

Abbildung 3.9 Migrationsprozess konfigurieren

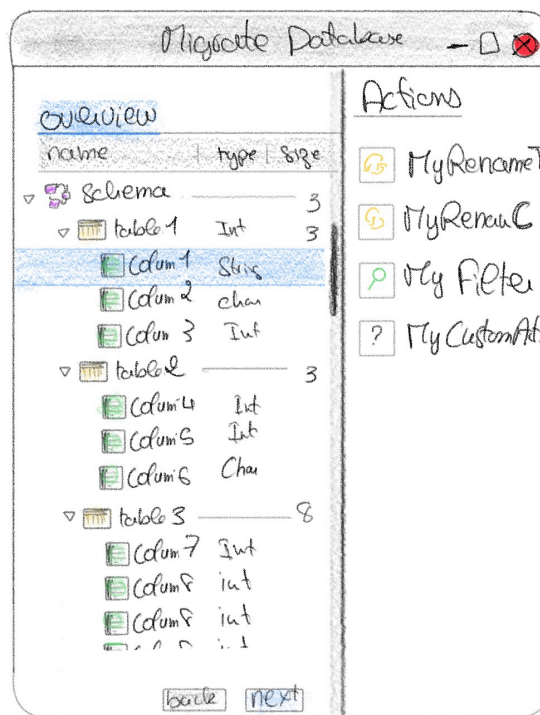


Dieser Anwendungsfall zeigt den Vorgang, wie der Benutzer Migrationsoperationen zum Migrationsprozess hinzufügen kann.

Es wird vorausgesetzt, dass die Quell- und Zieldatenbanken verbunden sind (siehe Anwendungsfall 3.4)

Wenn der Nutzer auf ‚Next‘ geklickt hat, soll eine Übersicht für alle in der Quelldatenbank enthaltenen Elemente angezeigt werden (Siehe Abbildung 3.10).

Abbildung 3.10 Übersicht Quelldatenbank



Der Benutzer kann zunächst Spalten bzw. Tabellen auswählen, um diesen Migrationsoperationen zuzuweisen.

Je nachdem wie die Auswahl der Datenbankelemente aussieht, stehen lediglich passende Migrationsoperationen zur Verfügung. Um eine Tabelle bzw. eine Spalte zu exkludieren, ist es ausreichend, wenn das selektierte Datenbankelement eine Tabelle bzw. eine Spalte ist.

Weiter wird beim Hinzufügen der Migrationsoperation ‚Datentyp ändern‘ geprüft, ob die ausgewählten Datenbankelemente Spalten sind und ob deren Datentypen dem Datentyp der Migrationsoperation entsprechen.

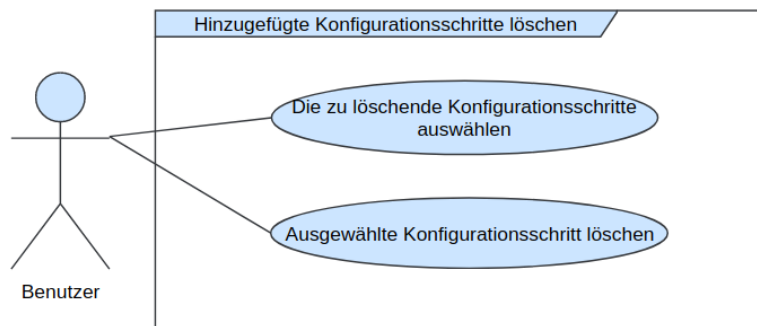
Außerdem wird beim Umbenennen der ausgewählten Tabellen bzw. Spalten kontrolliert, ob die Namen mit dem in der Migrationsoperation gespeicherten regulären Ausdruck übereinstimmen. Nachdem der Benutzer alle gewünschten Migrationsoperationen ausgewählt hat, werden diese gespeichert und zum Migrationsprozess hinzugefügt.

<b>Name</b>	Migrationsprozess konfigurieren.
<b>Akteure</b>	Benutzer
<b>Auslöser</b>	Der Benutzer klickt auf den „Next“-Button.
<b>Vorbedingung</b>	Die Migration ist nicht konfiguriert.
<b>Nachbedingung</b>	Die Migration ist nach den Wünschen des Benutzers konfiguriert.
<b>Ablauf</b>	<ol style="list-style-type: none"> <li>1. Die zu konfigurierenden Datenbankelemente auswählen</li> <li>2. Migrationsoperation hinzufügen (dieser Schritt kann mehrmals durchgeführt werden)</li> </ol>

**Tabelle 3.5** Anwendungsfall Migrationsprozess konfigurieren

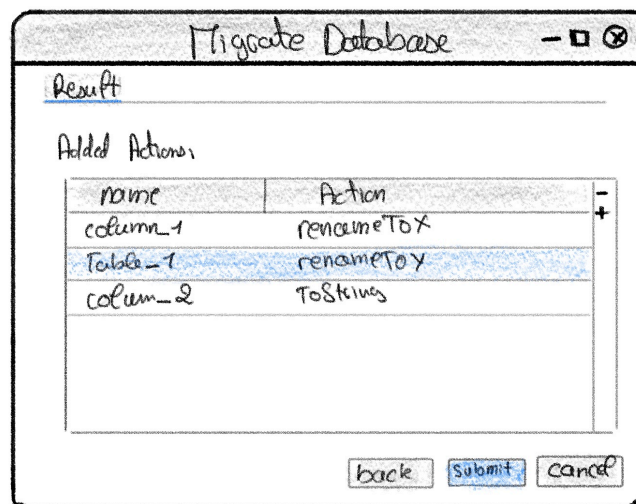
## Hinzugefügte Migrationsoperationen löschen

**Abbildung 3.11** Hinzugefügte Migrationsoperationen löschen



Das Löschen einer hinzugefügten Migrationsoperation ist erst möglich, wenn sich der Benutzer in der entsprechenden Übersicht befindet (siehe Abbildung 3.12). Dabei werden alle hinzugefügten Migrationsoperationen aufgelistet. Diese können ausgewählt und anschließend gelöscht werden.

Abbildung 3.12 Übersicht über die hinzugefügten Migrationsoperationen

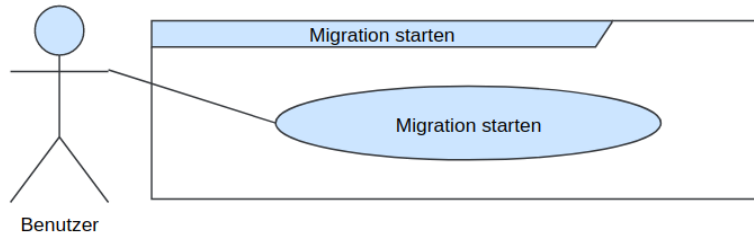


<b>Name</b>	Hinzugefügte Migrationsoperationen löschen.
<b>Akteure</b>	Benutzer.
<b>Auslöser</b>	Der Benutzer klickt auf den ‚Next‘-Button.
<b>Vorbedingung</b>	Die hinzugefügten Migrationsoperationen sind nicht gelöscht.
<b>Nachbedingung</b>	Die hinzugefügten Migrationsoperationen sind gelöscht.
<b>Ablauf</b>	<ol style="list-style-type: none"> <li>1. Die zu löschenden Migrationsoperationen auswählen</li> <li>2. Die ausgewählten Migrationsoperationen löschen</li> </ol>

Tabelle 3.6 Anwendungsfall Anwendungsfall ‚hinzugefügte Migrationsoperationen löschen‘

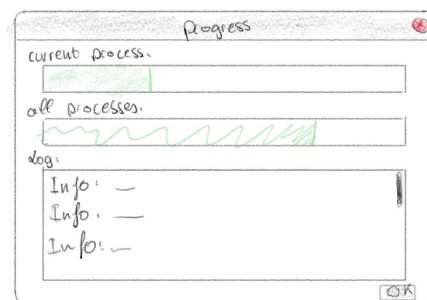
## Migration starten

**Abbildung 3.13** Migration starten



Nachdem der Benutzer alle Datenbanken verbunden und den Migrationsprozess konfiguriert hat, kann er die Migration der Datenbank durch eine einfache Bestätigung starten. Danach sollen die Daten entsprechend der Konfiguration migriert werden. Währenddessen soll der Benutzer über den Migrationsstand informiert werden (siehe Abbildung 3.14).

**Abbildung 3.14** Fortschritt des Migrationsprozesses



<b>Name</b>	Migration starten
<b>Akteure</b>	Benutzer
<b>Auslöser</b>	Der Benutzer klickt auf den ‚Migrieren‘-Button.
<b>Vorbedingung</b>	Die Quelldatenbank ist noch nicht migriert.
<b>Nachbedingung</b>	Die Quelldatenbank ist migriert.
<b>Ablauf</b>	1. Migration starten.

**Tabelle 3.7** Anwendungsfall Migration starten





# Konzeption und Implementierung

In diesem Kapitel wird die Konzeptentwicklung sowie die Technologieauswahl und die Implementierung des GuttenBase Plugins erläutert.

## 4.1 Konzeptentwicklung

In diesem Abschnitt werden grundlegende Aspekte der Softwarearchitektur vom GuttenBase Plugin vorgestellt. Diese werden basierend auf der Empfehlung im IEEE-Standard IEEE STD 1471-2000 erstellt.

Die Architektur wird aus zwei Sichten (Views) betrachtet. Jede Sicht stellt dabei spezifische Informationen bereit.

Es ist außerdem zu beachten, dass sich die Architektur auf die Ergebnisse der funktionalen Anforderungsanalyse sowie auf die Architektur der Zielplattform (IntelliJ Plattform) bezieht.

Im Folgenden werden die einzelnen Sichten genauer vorgestellt.

### 4.1.1 Konzeptionelle Sicht

In diesem Abschnitt wird die Konzeptionelle Sicht des Systems vorgestellt. Hier wird das System noch unabhängig von den Implementierungsentscheidungen betrachtet.

Die konzeptionelle Sicht wird als Komponentendiagramm in der Abbildung [4.1](#) dargestellt.

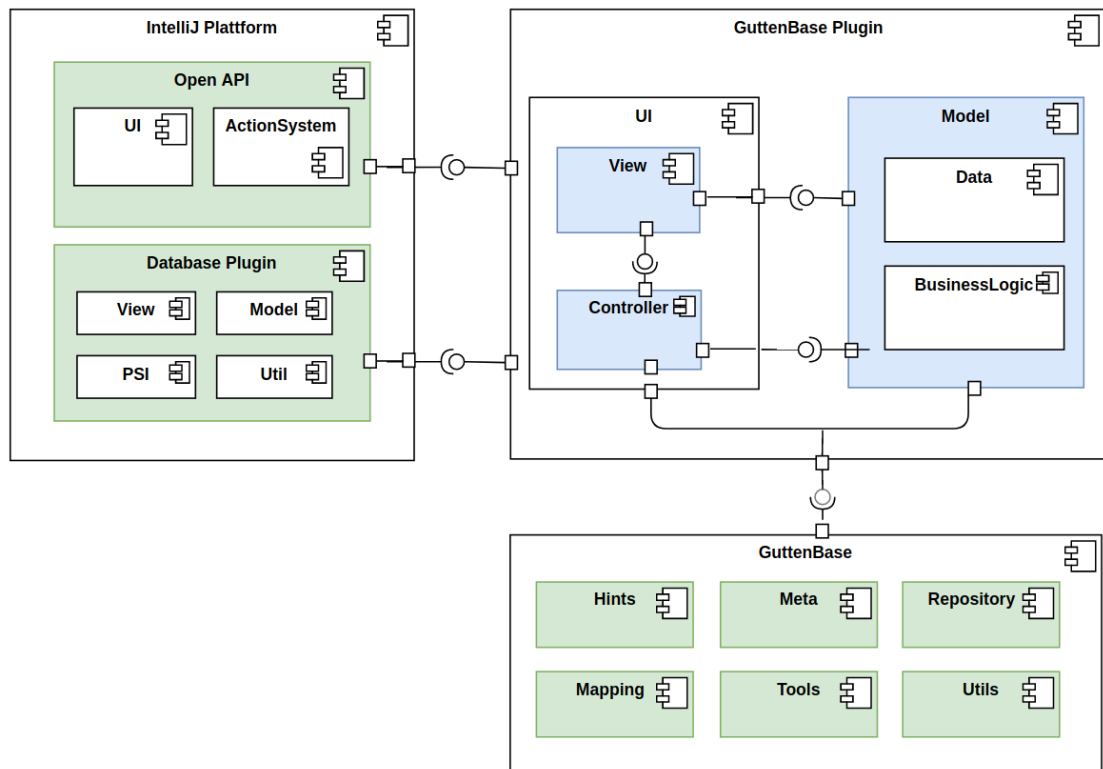


Abbildung 4.1 Komponentendiagramm für die konzeptionelle Sicht

Das Komponentendiagramm besteht aus der GuttenBase Plugin Komponente, welche das zu entwickelnde System darstellt, der IntelliJ Plattform Komponente und der GuttenBase Komponente. Es werden nur Komponenten veranschaulicht, die von unserem System benötigt sind. Diese werden im Folgenden genauer beschrieben.

#### 4.1.1.1 GuttenBase Plugin

Das GuttenBase Plugin wird nach dem MVC-Architekturstil konzipiert und besteht aus folgenden Komponenten:

##### Model

Das Modell enthält Daten, die von der View Komponente dargestellt werden. Hier liegt außerdem die Geschäftslogik (**BusinessLogik**), welche für die Änderung der Daten zuständig ist.

## View

Die View Komponente ist für die Darstellung der Daten für den Benutzer verantwortlich. Hier werden alle UI-abhängigen Aspekte wie Layout, Schriftart usw. behandelt.

## Controller

Die Controller Komponente ist für die Interaktion mit dem Benutzer verantwortlich. Sie wird von der View Komponente über Benutzerinteraktionen informiert und wertet sie aus. Anschließend können Änderungen an den Daten der Model Komponente sowie Anpassungen an der View Komponente angenommen werden.

### 4.1.1.2 IntelliJ Plattform

Wie in der Abbildung 4.1 zu sehen ist, interagiert das GuttenBase Plugin mit mehreren Komponenten der IntelliJ Plattform. Es wurden dabei nur die verwendeten Komponenten dargestellt. Diese sind in zwei Komponenten beinhaltet:

## Open API

Die IntelliJ Open API beinhaltet viele Komponenten, die auch von der IntelliJ IDEA Community Edition verwendet werden und für Plugin-Entwickler zur Verfügung stehen. Es können z. B. UI-Komponenten aus der Komponente **UI** für die Implementierung des Plugins benutzt werden. Außerdem wird **ActionSystem** Komponente benötigt, um bestimmte Aktionen, wie das Starten des Plugins, auszulösen.

## Database Plugin

Da das GuttenBase Plugin viel mit den Datenbanken umgeht, die in IntelliJ konfiguriert wurden, ist eine Interaktion mit dem Database Plugin sehr sinnvoll. Dazu bieten sich viele Komponenten für unterschiedliche Zwecke. Die für das GuttenBase Plugin benötigten Komponenten sind die **Model** Komponente (um evt. eine Datenbank Konfiguration zu bekommen), die **PSI** Komponente (um die Elemente der Datenbank zu bekommen), die **Util** Komponente (um ein Datenbank-Schema zu bekommen) und die **View** Komponente (um aus der Übersicht des Database Plugins Datenbank-Inhalte zu bekommen).

#### 4.1.1.3 GuttenBase

##### Repository

Unter der Repository Komponente befindet sich das ConnectorRepository, welches Informationen über die Quell- und Ziel-Datenbank enthält sowie die konfigurierten Migrationsoperationen enthält. Das ConnectorRepository wird während der ganzen Migration vom GuttenBase Plugin verwendet.

##### Meta

Hier sind Klassen enthalten, die die Datenbankelemente (Schema, Tabellen und Spalten) darstellen.

##### Hints

Die Hints Komponente enthält die Standardimplementierung von den Migrationsoperationen (z. B. Tabellenumbenennung). Diese können während der Migration überschrieben werden.

##### Mapping

Hier befinden sich alle Mapper Klassen. Diese können beim Konfigurieren der Migrationsoperationen benutzt werden.

##### Tools

Die Tools Komponente übernimmt das ausführen von bestimmten Aufgaben, wie das Kopieren von den Tabellen während der Migration.

##### Utils

Unter Utils liegen Klassen, die für das Logging des Migrationsprozesses benötigt sind. Z. B. kann die LoggingScriptExecutorProgressIndicator Klasse benutzt werden, um wichtige Informationen über den Migrationsprozess (z. B. die verstrichene Zeit oder die aktuelle SQL-Anfrage) zu haben. Diese können für das Darstellen des Migrationsfortschrittes benutzt werden.

#### 4.1.2 Modulsicht

Die Modulsicht zeigt die Struktur des GutenBase Plugins in Form von Modulen und deren Beziehungen zueinander. Hierbei werden die Komponenten und Konnektoren der konzeptionellen Sicht

auf Module, Schichten und Subsysteme abgebildet. Diese werden in Paket- und Klassendiagramme verfeinert. Zur Wahrung der Übersichtlichkeit wird die Modulsicht nach den unterschiedlichen Übersichten unterteilt. Es gibt insgesamt vier Übersichten, die das gesamte System abdecken. Pro Übersicht werden nur Klassen bzw. Methoden beschrieben, die entsprechenden Anwendungsfälle realisieren.

#### 4.1.2.1 Modulsicht der Übersicht der Konfigurationsschritte

Die Modulsicht der Abbildung 4.2 beschreibt alle beteiligten Klassen, die beim Erstellen und Verwalten der Migrationsoperationen eingesetzt sind. Diese werden entsprechend der Konzeptuellen Sicht aufgeteilt, nämlich in den View, Model und Controller Paketen.

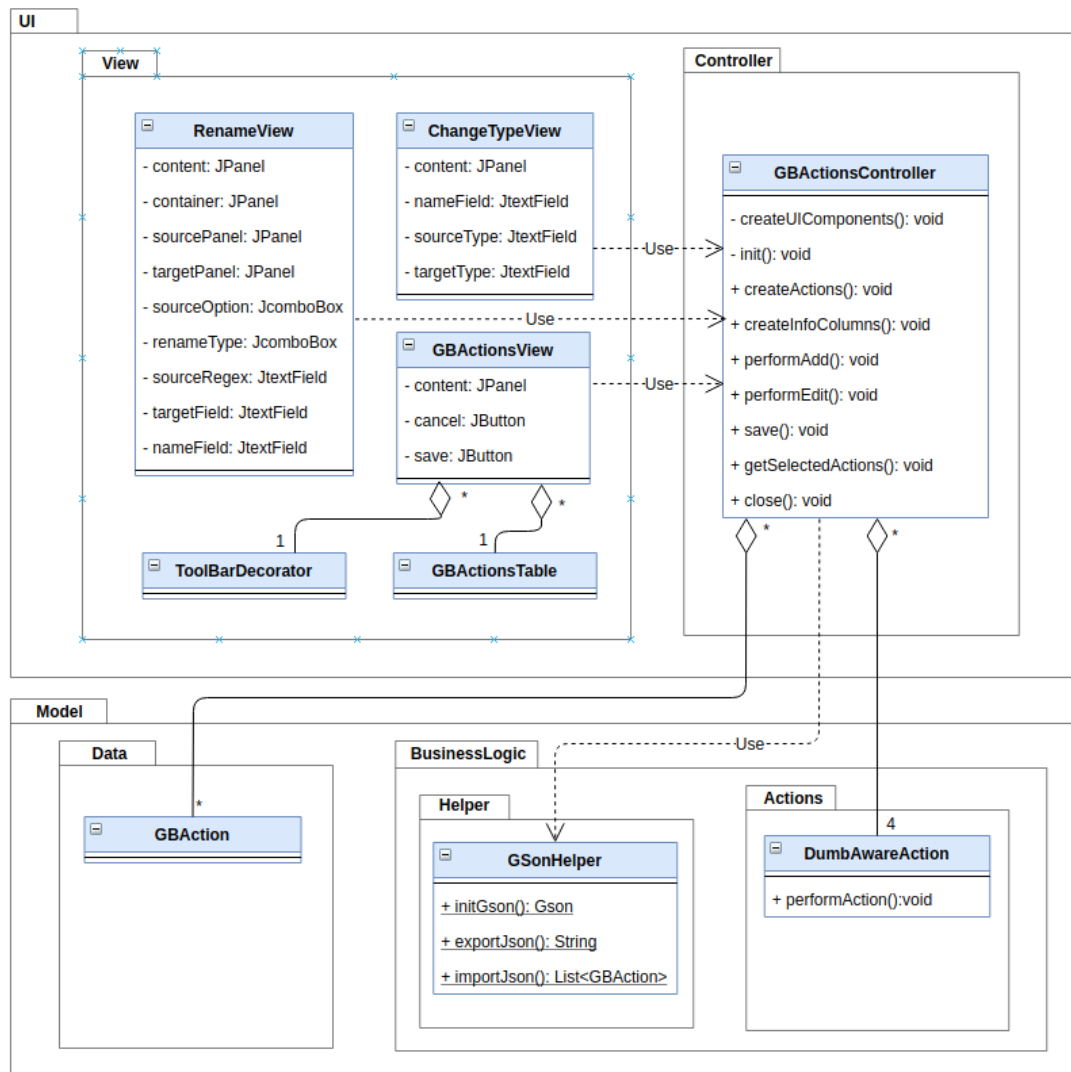


Abbildung 4.2 Modulsicht der Übersicht der Konfigurationsschritte

In dem View Paket werden alle Klassen dargestellt, die für die Darstellung der Migrationsoperationen zuständig sind. Diese basieren auf unterschiedliche Swing Komponenten. Die Klasse GBActionsView repräsentiert dabei die Hauptübersicht der Migrationsoperationen und beinhaltet die GBActionsTable Klasse, welche die Auflistung der Migrationsoperationen übernimmt. Außerdem wird die ToolbarDecorator Klasse für die Darstellung der möglichen Aktionen benötigt.

Das Controller Paket enthält hierbei die GBActionsController Klasse. Diese hat folgende Aufgaben:

- UI-Komponenten vor dem Anzeigen vorbereiten, indem Daten aus dem Paket Model erzeugt bzw. geladen werden.
- Migrationsoperationen erstellen. Dies passiert, wenn der Benutzer eine bestimmte Migrationsoperation hinzufügen möchte und diese in der GBActionsView übersicht selektiert. Zunächst wird die entsprechende DumbAwareAction Klasse aufgerufen, um die entsprechende Aktion durchzuführen. Die DumbAwareAction Klasse ist im Paket BusinessLogic zu finden und könnte für das Erstellen einer Migrationsoperation verwendet werden. Diese werden durch die RenameView und ChangeTypeView Klassen realisiert. Analog dazu erfolgt das Editieren der Migrationsoperationen.
- Migrationsoperationen speichern, indem die hinzugefügte Migrationsoperationen in JSON konvertiert und dann exportiert werden. Dafür ist die GsonHelper Klasse des BusinessLogic Pakets zuständig.

#### 4.1.2.2 Modulsicht der allgemeinen Übersicht

Diese Modulsicht zeigt die beteiligten Klassen, die beim Verbinden der zu migrierenden Datenbanken genutzt werden. Diese wird in der Abbildung 4.3 dargestellt.

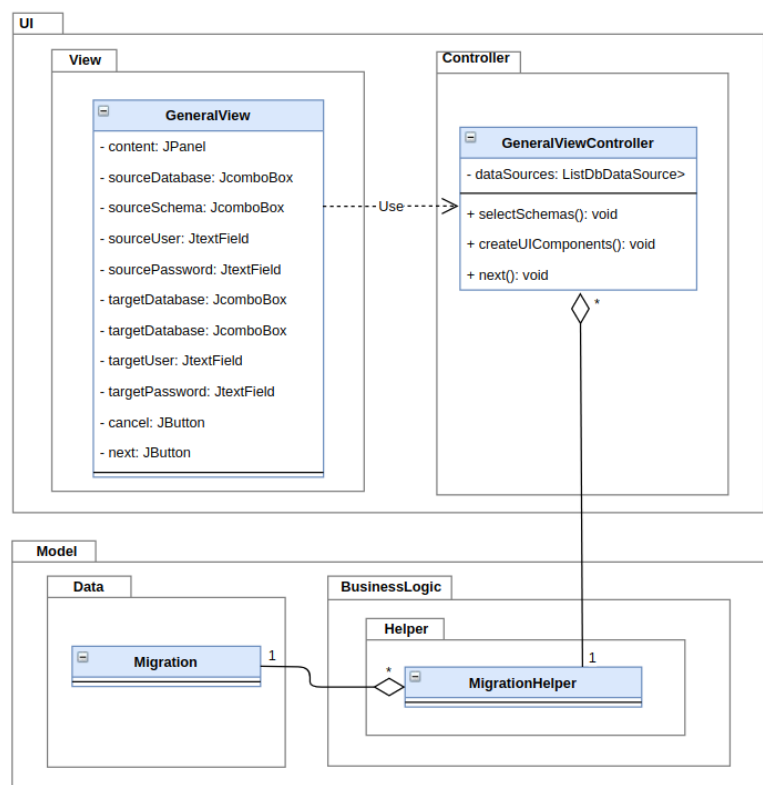
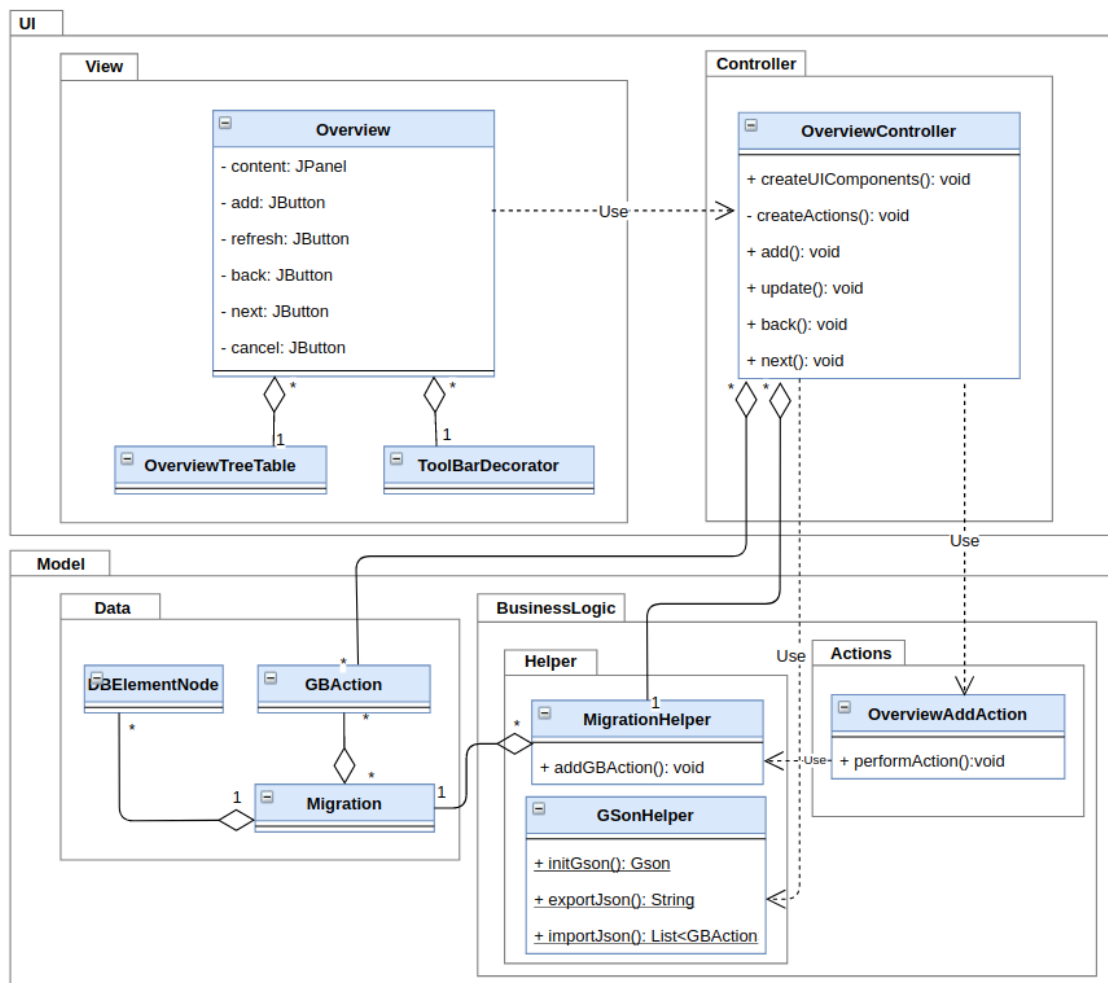


Abbildung 4.3 Modulsicht der allgemeinen Übersicht

Die GeneralView Klasse ist für die Interaktion mit dem Benutzer verantwortlich und enthält alle Eingabefelder und Texte. Die GeneralViewController Klasse ist für das Laden der Datenbank-elemente zuständig. Klickt der Benutzer auf das „Next“-Button, werden alle Eingaben über die MigrationHelper Klasse des Pakets BusinessLogic übergeben. Diese Informationen werden dann in der Migration Klasse des Model Pakets gespeichert.

#### 4.1.2.3 Modulsicht der Konfigurationsübersicht

Die Modulsicht in der Abbildung 4.4 zeigt die für die Konfigurationsübersicht relevanten Klassen.



**Abbildung 4.4** Modulsicht der Konfigurationsübersicht

In der Konfigurationsübersicht (Overview) enthält die OverviewTreeTable Klasse, welche alle Quell-Datenbankelemente auflistet und die ToolbarDecorator Klasse, die gespeicherten Migrati-



onsoperationen anzeigt. Außerdem stehen einige Buttons für die Benutzerinteraktion zur Verfügung.

Wie bei den vorherigen Modulsichten, stellt die OverviewController Klasse Methoden für folgende Zwecke bereit:

- Das Laden der Datenbankelemente: Hierbei werden die gespeicherten Migrationsoperationen mithilfe der GsonHelper Klasse importiert und die Datenbankelement (DBElementNode) aus der Quell-Datenbank erstellt.
- das Anzeigen der Migrationsoperationen: Migrationsoperationen werden nach Kompatibilität mit den selektierten Datenbankelementen behandelt. Ist eine Migrationsoperation mit einem Datenbankelement geeignet, wird diese klickbar angezeigt, außerdem wird stattdessen ein ausgegrautes Button dargestellt.
- Hinzufügen von Migrationsoperationen: Wenn der Benutzer ein Datenbankelement selektiert und dann eine entsprechende Migrationsoperation hinzufügt, wird die OverviewAddAction Aktion von dem BusinessLogic Paket ausgelöst. Dabei wird die entsprechende Migrationsoperation durch die MigrationHelper Klasse zur Migration hinzugefügt.

#### 4.1.2.4 Modulsicht der Ergebnisübersicht

Die Modulsicht der Abbildung 4.5 stellt die für die Ergebnisübersicht (ResultView) zuständigen Klassen. Außerdem wird die Fortschritt Übersicht (ProgressView) miteingebunden, da diese vom selben Controller verwaltet wird.

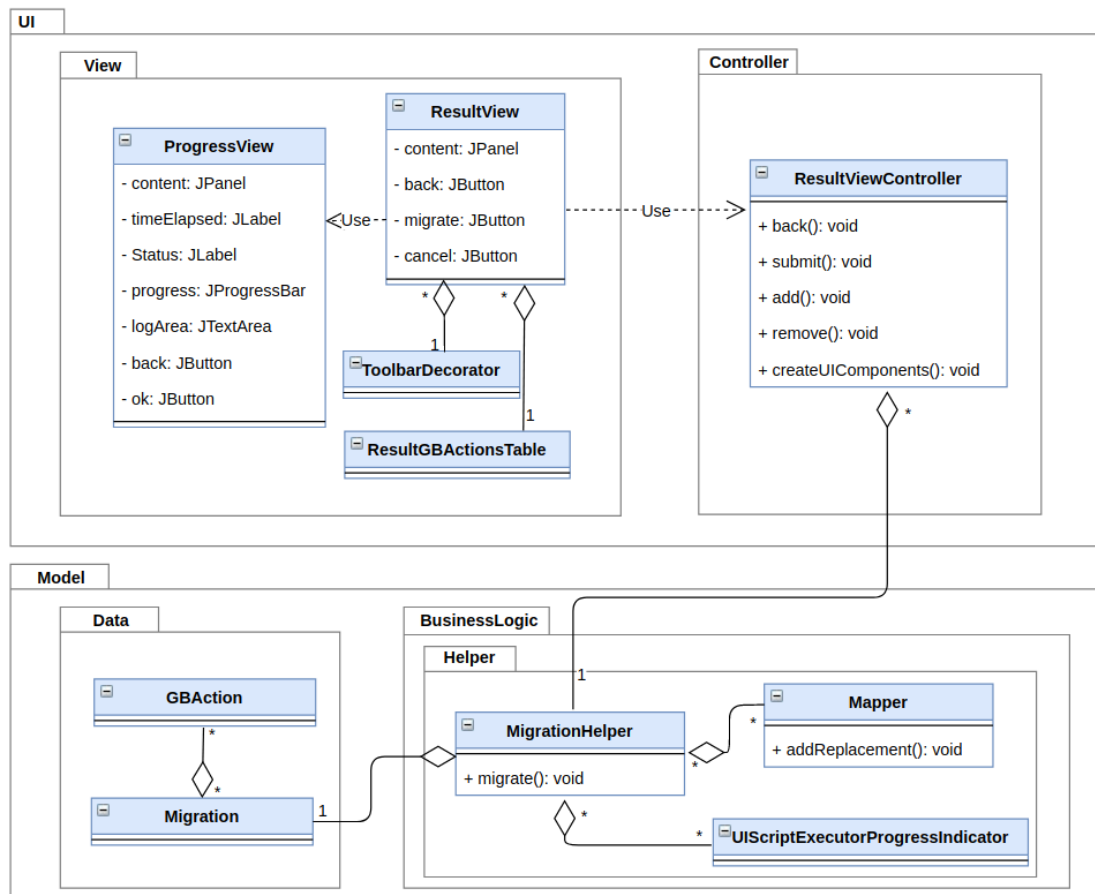


Abbildung 4.5 Modulsicht der Ergebnisübersicht

Ähnlich wie bei den anderen Übersichten, enthält die ResultView Klasse eine Tabelle (ResultGBActionsTable), die alle hinzugefügten Migrationsoperationen enthält und ein ToolbarDecorator, wo die Aktionen zum Löschen und Hinzufügen angezeigt werden.

Auf der anderen Seite stellt die ResultViewController Klasse Methoden für das Löschen und das Hinzufügen von Migrationsoperationen sowie für das Starten des Migrationsprozesses bereit. Diese werden im Folgenden genauer erklärt:

- Hinzufügen: Wenn der Benutzer noch mehr Migrationsoperationen zur Migration hinzufügen möchte, wird die aktuelle Übersicht zur Übersicht der Migrationsoperationen (Overview) umgeleitet.
- Löschen: Wie das Hinzufügen, erfolgt das Löschen von Migrationsoperationen (die schon zur Migration hinzugefügt wurden) durch die Entfernung von der ausgewählten Migrationsoperation (GBAction) aus der Migration Klasse.
- Migration starten: Nach dem Klick auf das „Migrate“Button, wird der Migrationsprozess

gestartet. Dieser erfolgt durch die MigrationHelper Klasse. Dabei wird das Connector Repository der GuttenBase Bibliothek entsprechend der Migrationsoperationen konfiguriert (siehe 2.3.1) und anschließend das Kopieren durch das DefaultTableCopyTool gestartet. Parallel dazu wird die Fortschritt Übersicht (ProgressView) angezeigt, um Informationen über den laufenden Prozess zu sehen. Dies ist durch die UIScriptExecutorProgressIndicator Klasse ermöglicht.

## 4.2 Technologieauswahl

Im Folgenden wird die Umsetzungsform des GuttenBase Plugins begründet. Außerdem wird die Nutzung der GuttenBase Bibliothek und die IntelliJ Plugin Entwicklung genauer beschrieben.

### 4.2.1 Umsetzungsform

Um eine optimale Nutzung des GuttenBase Plugins zu erzielen, soll auf die Umsetzungsform geachtet werden.

Das zu entwickelnde Tool kann z. B. als eine Desktop Applikation, Web Applikation oder als Plugin einer anderen Anwendung realisiert werden.

In der Tabelle 4.1 werden einige Vor- und Nachteile jeder Alternative erläutert.

Alle drei Alternativen haben Pros und Contras allerdings ist die schnellere Erreichung von vielen Nutzern sowie die Einfache Installation bei der IDE Plugin Entwicklung entscheidend.

Zunächst soll für eine konkrete IDE entschieden werden. Um diese auszuwählen, muss auf die Anzahl der Nutzer, die Verfügbarkeit der Dokumentation für Plugin Entwicklung sowie die Unterstützung von Datenbanken geachtet werden.

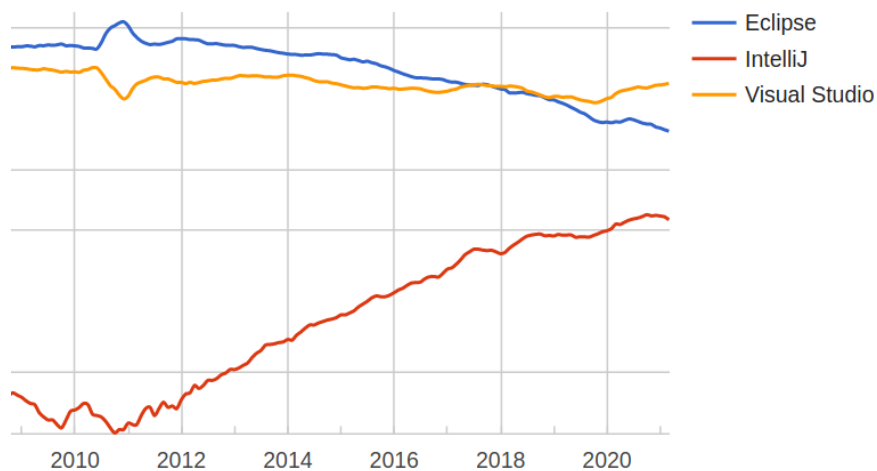
Einer der bekanntesten Methoden, um die genaue Beliebtheit einer Programmiersprache bzw. eine IDE herauszufinden, ist der PYPL-Index. Er basiert auf Rohdaten aus Google Trends. PYPL enthält den TOP-IDE-Index, welches analysiert, wie oft IDEs bei Google durchgesucht werden. Die Suchanfragen spiegeln zwar nicht unbedingt die Beliebtheit der IDEs. Allerdings hilft einen solchen Index enorm bei der Wahl einer Entwicklungsumgebung. Bei dieser Analyse sind die drei bekanntesten und für unseren Fall relevanten Entwicklungsumgebungen Visual Studio (erster Platz), Eclipse (zweiter Platz) und IntelliJ (sechster Platz). Außerdem hat sich der Index von IntelliJ IDE am stärksten erhöht (siehe Abbildung 4.6)

Bei einer anderen Umfrage (Jaxenter), mit welcher Entwicklungsumgebung am liebsten in Java programmiert wird, war IntelliJ sogar im ersten Platz mit 1660 Stimmen von 2934.

Alternative	Vorteile	Nachteile
Desktop App	<ul style="list-style-type: none"> <li>• Offline immer verfügbar</li> <li>• Volle Kontrolle über die Anwendung und die enthaltenen Daten.</li> <li>• Bessere Leistung, da kein Browser als Zwischenschicht existiert.</li> </ul>	<ul style="list-style-type: none"> <li>• Plattformabhängig</li> <li>• Hohe Entwicklungskosten</li> <li>• Installation ist notwendig</li> </ul>
Web App	<ul style="list-style-type: none"> <li>• Installation oder manuelle Updates sind nicht notwendig.</li> <li>• geringere Entwicklungs- und Wartungskosten, da die Anwendung unabhängig von lokalen Endgeräten ist.</li> </ul>	<ul style="list-style-type: none"> <li>• Offline meistens nicht verfügbar.</li> <li>• Geringere Leistung.</li> </ul>
IDE Plugin Entwicklung	<ul style="list-style-type: none"> <li>• Für IntelliJ Benutzer ist das einfach und intuitiv zu benutzen.</li> <li>• Manche Komponenten bzw. Funktionalitäten der zu erweiternden IDE können wiederverwendet werden, was die Entwicklungsdauer verkürzt.</li> <li>• Intuitive Nutzung sowie eine einheitliche Benutzeroberfläche wie die benutzte IDE.</li> </ul>	<ul style="list-style-type: none"> <li>• Die Flexibilität beim Entwickeln ist durch die limitierte Erweiterbarkeit der IDE eingeschränkt.</li> </ul>

**Tabelle 4.1** Umsetzungsmöglichkeiten

Abbildung 4.6 Top IDE Index



Aus den oben erläuterten Daten und aufgrund der guten Dokumentation für Plugin Entwicklung wird das GuttenBase als ein IntelliJ Plugin umgesetzt.

#### 4.2.2 IntelliJ Platform

Für die erfolgreiche IntelliJ Plugin Entwicklung muss auf mehrere Aspekte geachtet werden wie die Projektstruktur und die häufig verwendeten Komponenten.

Die Plugin Entwicklung erfolgt in der IntelliJ IDE selbst. Deswegen kann das Plugin entweder in Java, Kotlin, Groovy oder Scala geschrieben werden.

Der von JetBrains empfohlene Weg für das Erstellen eines neuen Plugins ist das Gradle Projekt. Dabei muss die Option IntelliJ Platform Plugin ausgewählt werden, damit die Plugin Abhängigkeiten sowie die Basis-IDE automatisch konfiguriert werden. Zusätzlich muss die Datei plugin.xml entsprechend des zu entwickelnden Plugins angepasst werden. Diese enthält wichtige Informationen, die in den folgenden Tags (Auszeichnungen) erklärt werden:

- **<name>**  
Der Name des Plugins. Er soll kurz und beschreibend sein.
- **<id>**  
Eine eindeutige Bezeichnung des Plugins. Diese kann nicht während der Entwicklung geändert werden.
- **<description>**  
Eine Kurze Beschreibung des Plugins.
- **<change-notes>**  
Eine Beschrei Beschreibung der Änderungen in der neusten Version des Plugins.

- **<version>**  
Die aktuelle Plugin Version.
- **<vendor>**  
Der Anbieter des Plugins. Hier kann zusätzlich eine Email Adresse angegeben werden.
- **<depends>**  
Abhängigkeiten zu Plugins oder Modulen.
- **<idea-version>**  
Die minimale und maximale Version der IDE, mit der das Plugin kompatibel ist.
- **<actions>**  
Definiert wie die Funktionalität des Plugins aufgerufen wird. Dies wird im folgenden Abschnitt behandelt.
- **<extensionPoints>**  
Die vom Plugin definierte Erweiterungspunkte. Diese können erlauben anderen Plugin-Entwicklern, auf bestimmte Daten zuzugreifen.
- **<extensions>**  
Erweiterungspunkte, die von IntelliJ-Plattform bzw. von anderen Plugins definiert sind und von dem zu entwickelnden Plugin verwendet werden.

## Action-System

Die am häufigsten verwendete Methode, um die Plugin Funktionalität aufzurufen, ist die Nutzung der sogenannten Actions vom Action-System der IntelliJ-Plattform.

Eine Aktion kann über ein Menüpunkt (menu item) oder einen Eintrag in der Symbolleiste ausgelöst werden. Dazu muss ein Eintrag in dem Actions Tag der plugin.xml Datei erfolgen. Dabei muss jede Action mindestens eine Id, eine Klasse und einen beschreibenden Text haben. I.d.R werden Menüpunkte nach Funktionalität gruppiert. Um die Implementierte Action zu einer bestimmten Gruppe hinzufügen zu können, muss der Tag **<add-to-group>** verwendet werden.

Die Action Klasse muss von der AnAction Klasse abgeleitet werden und die actionPerformed() Methode überschreiben. Diese wird nach dem Klick auf das entsprechende Menüpunkt bzw. Symbolleiste aufgerufen.

### 4.2.3 GuttenBase

Für die Nutzung der GuttenBase Bibliothek sollen als erstes Abhängigkeiten zu dem laufenden Projekt (z. B Gradle) hinzugefügt werden. Die für GuttenBase benötigten Informationen sind:

- groupId: de.akquinet.jbosscg.guttenbase

- artifactId: GuttenBase
- version: 2.0.0

Es ist außerdem zu beachten, dass die Abhängigkeiten für die Treiber-Klassen der Quell- und Ziel-DBMS auch hinzugefügt werden sollen.

Zunächst sollen die ConnectionInfo Klassen erstellt werden. Diese beschreiben die Quell- und Ziel-Datenbanken und enthalten die für eine JDBC (Java Database Connectivity) Verbindung erforderlichen Attribute.

```
public class MySQLConnectionsInfo extends URLConnectorInfoImpl {
    public MySQLConnectionsInfo() {
        super("jdbc:mysql://localhost:3306/testdb", "user", "password",
            "com.mysql.jdbc.Driver", "aev", DatabaseType.MYSQL);
    }
}
```

**Abbildung 4.7** ConnectionInfo konfigurieren

Im nachhinein wird das Connector Repository konfiguriert werden. Dies enthält alle Konnektoren, die in der Datenbank Migration beteiligt sind.

```
public static final String SOURCE = "source";
public static final String TARGET = "target";
...
final ConnectorRepository connectorRepository = new
ConnectorRepositoryImpl();
connectorRepository.addConnectionInfo(SOURCE, new
PostgreSQLConnectionInfo());
connectorRepository.addConnectionInfo(TARGET, new
MySQLConnectionsInfo());
```

**Abbildung 4.8** Connector Repository konfigurieren

Meistens werden Konfigurationshinweise (hints) benötigt, um die Migration zu individualisieren. In der Dokumentation von GuttenBase<sup>1</sup> befindet sich eine Liste aller unterstützten Konfigurationshinweise.

Als Beispiel wird der Konfigurationshinweis ColumnMapperHint in der Abbildung dargestellt.

---

<sup>1</sup>Getting Started with GuttenBase (2018, 04.01)

<https://github.com/akquinet/GuttenBase/blob/master/Getting%20Started%20with%20GuttenBase.pdf>

```
connectorRepository.addConnectorHint(TARGET, new ColumnMapperHint() {  
    @Override  
    public ColumnMapper getValue() {  
        return new CustomColumnRenameName()  
            .addReplacement("name", "id_name")  
            .addReplacement("username", "id_username");  
    }  
})
```

Abbildung 4.9 ColumnMapperHint hinzufügen

Anschließend kann die Migration mit den eventuell hinzugefügten Hinweisen durchgeführt werden. Dies passiert wie folgt:

- Das Datenbank Schema wird von der Quell-Datenbank in die Ziel-Datenbank kopiert. Dabei werden möglichst viele Unterschiede in Datentypdarstellung standardmäßig berücksichtigt.
- Die Kompatibilität von den Schemata wird geprüft. Hierbei wird nach gleichen Tabellen bzw Spalten gesucht.
- Falls es keine Fehler beim Prüfen gibt, werden die Daten dann kopiert.

```
new CopySchemaTool(connectorRepository).copySchema(SOURCE, TARGET);  
final SchemaCompatibilityIssues schemaCompatibilityIssues = new  
SchemaComparatorTool(connectorRepository).check(SOURCE, TARGET);  
if (schemaCompatibilityIssues.isSevere()) {  
    throw new SQLException(schemaCompatibilityIssues.toString());  
}  
new DefaultTableCopyTool(connectorRepository).copyTables(SOURCE,  
TARGET);  
new CheckEqualTableDataTool(connectorRepository).checkTableData(SOURCE,  
TARGET);
```

Abbildung 4.10 Datenbank Migration durchführen

## 4.3 PlugIn Implementierung

Dieser Abschnitt beschäftigt sich mit der Implementierung des GuttenBase Plugins.

Die resultierende Anwendungsoberfläche wird anhand den folgenden Anwendungsfunktionalitäten veranschaulicht:

- Liste der vorhandenen Migrationsoperationen.
- Hinzufügen einer neuen Migrationsoperation.
- Verbindungerstellung.
- Konfiguration.
- Migrationsdurchführung.



### 4.3.1 Funktion: Liste der vorhandenen Migrationsoperationen

Um die Funktionalität des GuttenBase Plugins einfach und intuitiv für IntelliJ Nutzer zur Verfügung zu stellen, wurden Menüpunkte zum Database Plugin hinzugefügt. Diese wurden gemäß dem Grundsatz der Unterscheidbarkeit platziert. Somit kann die Übersicht der Migrationsoperationen nach dem Klick auf „Show Migration Actions“ geöffnet werden. Dies wird in der Abbildung 4.12 dargestellt. Diese enthält standardmäßig nur zwei Migrationsoperationen (Tabellen bzw. Spalten Ausschließen).

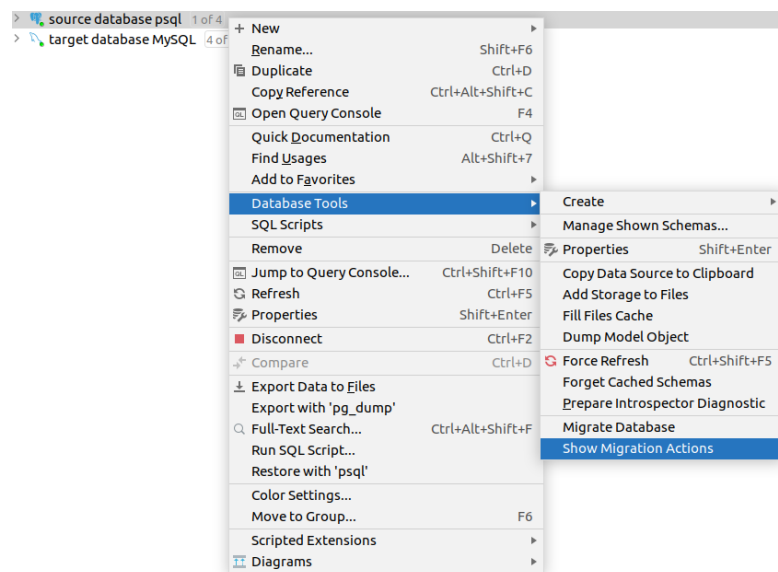


Abbildung 4.11 Übersicht der Migrationsoperationen öffnen

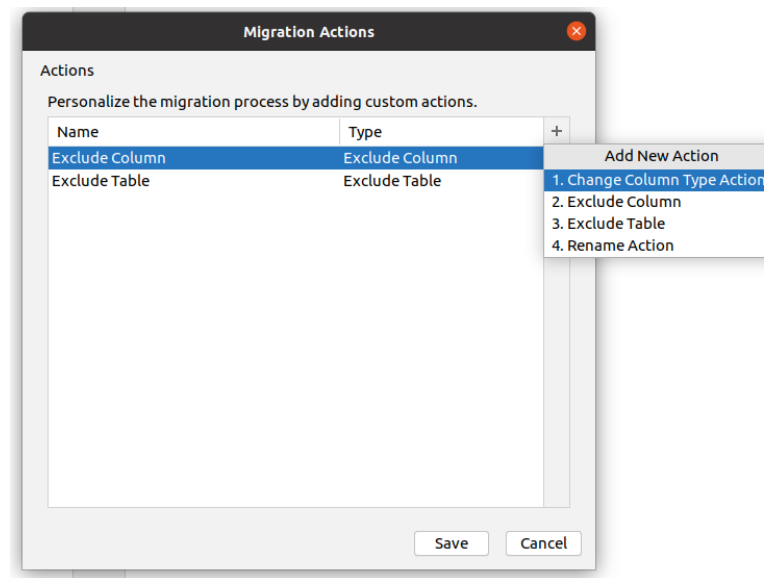


Abbildung 4.12 Übersicht der Migrationsoperationen

#### 4.3.2 Funktion: Hinzufügen einer neuen Migrationsoperation

Bei der Übersicht in der Abbildung 4.12 kann der Benutzer verschiedene Migrationsoperationen erstellen, wenn diese nicht bereits existieren. In diesem Szenario wird nur das Hinzufügen von der Migrationsoperation Umbenennen (Rename Action) dargestellt.

Nach dem Klick auf das entsprechende Button wird ein Dialog angezeigt, um die erforderliche Informationen einzugeben. Dabei wird der Name der Migrationsoperation festgelegt. Außerdem der Quell-Name durch einen regulären Ausdruck definiert. Anschließend wird die der Zeil-Name festgelegt.

Die in der Abbildung 4.13 angezeigte Migrationsoperation gilt für alle Datenbankelemente, deren Name die Zeichenkette „table“ enthält. Wenn diese an einem entsprechenden Datenbankelement angewendet wird, wird das Suffix „\_Test“ hinten hinzugefügt. Anschließend lässt sich die neu hinzugefügte Migrationsoperation durch das Klick auf das „Save“ Button speichern (siehe Abbildung 4.12). Dabei werden alle Migrationsoperationen nach JSON konvertiert und in einer externen Datei gespeichert. Dafür ist die Klasse GsonHelper verantwortlich (siehe Abbildung 4.14).

#### 4.3.3 Funktion: Verbindungserstellung

Wenn die Quell- und Ziel-Datenbank im Database Plugin eingerichtet sind, kann das Aktionsmenü nach Rechtsklick auf die Quell-Datenbank aktiviert werden. Wie in der Abbildung 4.11

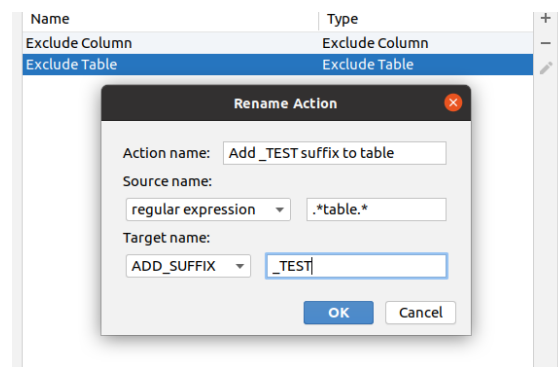


Abbildung 4.13 Migrationsoperation Umbenennen erstellen

```
public class GsonHelper {
    public static String exportJSON(List<GBAction> gbActions, String fileName) throws IOException {
        gbActions.forEach(gbAction -> System.out.println(gbAction.getName() + gbAction.getGBActionType()));
        Gson gson = GsonHelper.initGson();
        GBActionsJSON gbActionsJSON = new GBActionsJSON(gbActions);
        String actionsString = gson.toJson(gbActionsJSON, GBActionsJSON.class);
        FileWriter file = new FileWriter(fileName, append: false); //replace file
        file.write(actionsString);
        file.flush();
        return new File(fileName).getAbsolutePath();
    }

    public static List<GBAction> importJSON(String fileName) throws FileNotFoundException {
        Gson gson = GsonHelper.initGson();
        JsonReader reader = new JsonReader(new FileReader(fileName));
        GBActionsJSON gbActionsJSON = gson.fromJson(reader, GBActionsJSON.class);
        gbActionsJSON.getGBActions().forEach(gbAction -> System.out.println(gbAction.getName() + gbAction.getGBActionType()));
        return gbActionsJSON.getGBActions();
    }
}
```

Abbildung 4.14 Migrationsoperationen exportieren und importieren

angezeigt, kann der Benutzer auf das Button „Migrate Database“ klicken um die Übersicht der Datenbank Migration zu öffnen (siehe Abbildung 4.15). Dabei wird die Quell-Datenbank anhand der selektierten Datenbank automatisch selektiert. Außerdem muss das zu migrierende Schema der Quell-Datenbank sowie das Ziel-Schema ausgewählt werden. Zusätzlich müssen die Zugangsdaten jeder Datenbank angegeben werden, um die Datenbanken zu verbinden. Nach dem Klick auf das „Next“ Button, werden die Angaben erst geprüft. Wenn diese fehlerhaft sind, dann wird eine entsprechende Fehlermeldung ausgegeben. Ansonsten wird das ConnectorRepository (siehe 4.2.3) anhand der angegebenen Informationen erstellt und anschließend die nächste Übersicht (overview) angezeigt.

#### 4.3.4 Funktion: Konfiguration

Bei der Konfigurationsübersicht (siehe Abbildung 4.16) werden alle Elemente der Quell-Datenbank angezeigt. Hierbei wird Einzel- sowie Mehrfachauswahl ermöglicht.

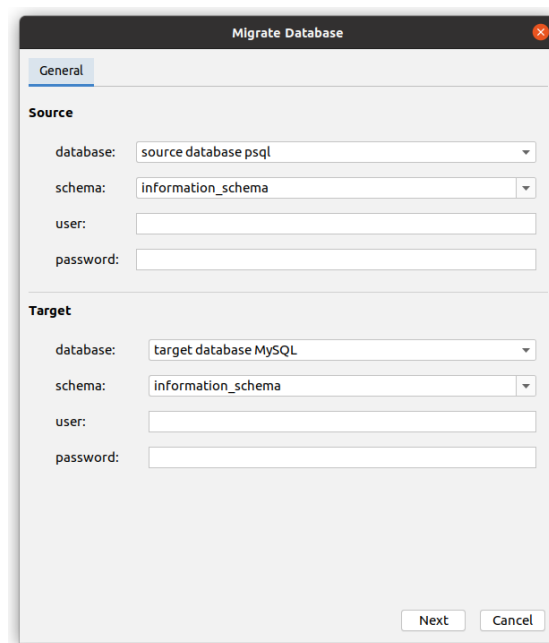


Abbildung 4.15 Allgemeine Übersicht der Datenbank Migration (generalView)

Außerdem werden alle Migrationsoperationen mithilfe der **GsonHelper** Klasse geladen werden. Diese werden abhängig von den selektierten Elementen unterschiedlich dargestellt. Wenn eine Migrationsoperation (GBAction) zu den ausgewählten Elementen passt, wird diese klickbar angezeigt, ansonsten wird diese deaktiviert und ausgegraut dargestellt. Dabei spielt die **matches** Methode der **GBAction** Klasse eine entscheidende Rolle. Diese wird entsprechend des Typen der Migrationsoperation. Für das Umbenennen wird z. B. geprüft, ob der gespeicherte reguläre Ausdruck zum Namen der selektierten Elemente passt (siehe Abbildung 4.17).

Bei jedem Hinzufügen wird die entsprechende Migrationsoperation zu der Liste aller Operationen hinzugefügt. Dabei wird eine neue Instanz erzeugt, die die benötigten Informationen des entsprechenden Datenbankelementes enthält (siehe Abbildung 4.18).

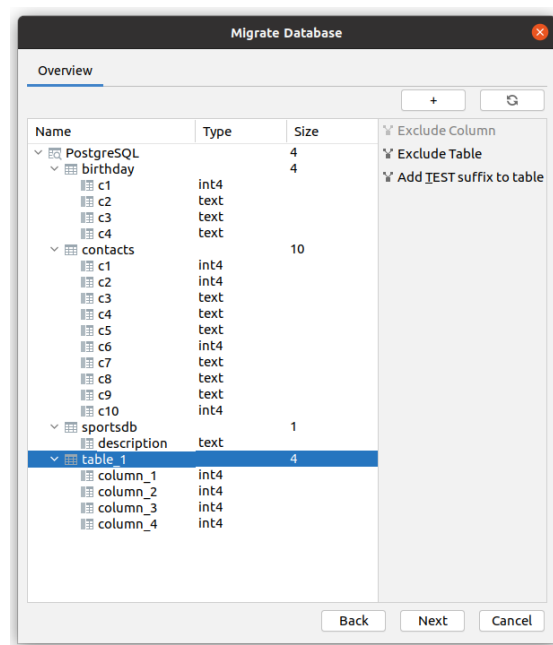


Abbildung 4.16 Konfigurationsübersicht (overview)

```
@Override
public boolean matches(MyDataNode node){
    return node.getName().matches(regExp);
}
```

Abbildung 4.17 matches Methode der Migrationsoperation Umbenennen

```
for (int row : rows) {
    MyDataNode node = (MyDataNode) overviewTreeTable.getValueAt(row, column: 0);
    GBAAction newGBAction;
    try {
        // create a new instance of the action (otherwise the same object will be overwritten).
        newGBAction = (GBAction) gbAction.clone();
    } catch (CloneNotSupportedException cloneNotSupportedException) {
        cloneNotSupportedException.printStackTrace();
        Messages.showErrorDialog(cloneNotSupportedException.getMessage(), title: "Error!");
        return;
    }
    newGBAction.setSource(node);
    if (node instanceof ColumnNode && newGBAction.getGBActionType().equals(GBActionType.RENAME)) {
        newGBAction.setGBActionType(GBActionType.RENAME_COLUMN);
    }
    else if (node instanceof TableNode && newGBAction.getGBActionType().equals(GBActionType.RENAME)) {
        newGBAction.setGBActionType(GBActionType.RENAME_TABLE);
    }
    migration.addGBAction(newGBAction);
}
```

Abbildung 4.18 das Hinzufügen von der Migrationsoperation Umbenennen

### 4.3.5 Funktion: Migrationsdurchführung

Nach dem Klick auf das „Next“ Button, erhält der Benutzer eine Übersicht von allen hinzugefügten Migrationsoperationen (siehe Abbildung 4.19). Diese können nach Bedarf gelöscht werden.

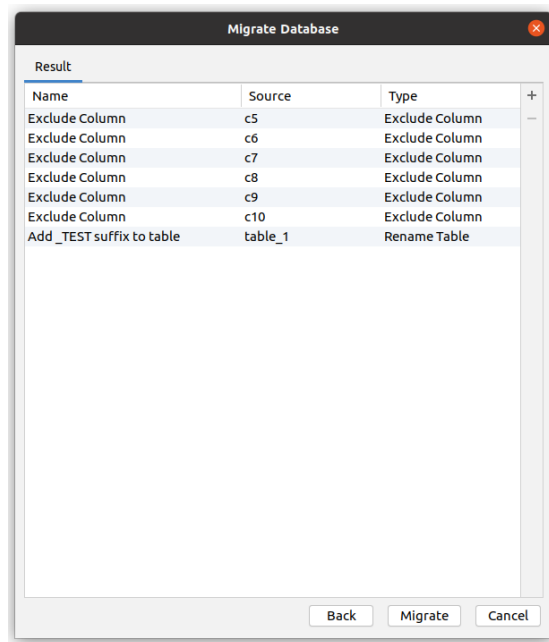


Abbildung 4.19 Ergebnisübersicht

Nach dem Klick auf das „Migrate“ Button, wird die Fortschrittsübersicht angezeigt (siehe Abbildung 4.22). Diese veranschaulicht den Migrationsprozess, welcher in einem neuen Thread ausgeführt wird. Bei der Migration werden die Mapper Klassen sowie die GuttenBase Connectors (siehe Abschnitt 4.2.3) entsprechend der hinzugefügten Migrationsoperationen zum Connector-Repository hinzugefügt. Danach werden die Daten von der Quell-Datenbank zur Zie-Datenbank kopiert (siehe Abbildung 4.20 bzw. Abbildung 4.21).

```
private void addConnectors() {
    connectorRepository.addConnectorHint(TARGET, (ColumnMapperHint) () -> {
        return columnRenameMapper;
    });
    connectorRepository.addConnectorHint(TARGET, (TableMapperHint) () -> {
        return tableRenameMapper;
    });
    connectorRepository.addConnectorHint(TARGET, (ColumnTypeMapperHint) () -> {
        return columnTypeMapper;
    });
    connectorRepository.addConnectorHint(SOURCE, (RepositoryColumnFilterHint) () -> {
        return column -> !excludedColumns.contains(column);
    });
    connectorRepository.addConnectorHint(SOURCE, (DefaultRepositoryTableFilterHint) getValue() -> {
        return table -> !excludedTables.contains(table);
    });
    connectorRepository.addConnectorHint(SOURCE, (ScriptExecutorProgressIndicatorHint) () -> {
        return new UIScriptExecutorProgressIndicator(progressView);
    });
    connectorRepository.addConnectorHint(TARGET, (ScriptExecutorProgressIndicatorHint) () -> {
        return new UIScriptExecutorProgressIndicator(progressView);
    });
}
```

Abbildung 4.20 ConnectorHints hinzufügen

```
@Override
public void run() {
    updateMappers();
    addConnectors();
    try {
        new CopySchemaTool(connectorRepository).copySchema(SOURCE, TARGET);
        checkCompatibilityIssues();
        new DefaultTableCopyTool(connectorRepository).copyTables(SOURCE, TARGET);
        new CheckEqualTableDataTool(connectorRepository).checkTableData(SOURCE, TARGET);
    } catch (SQLException e) {
        e.printStackTrace();
        ApplicationManager.getApplication().invokeLater(() -> Messages.showErrorDialog(e.getMessage(), ERROR_TITLE));
        progressView.enableBack();
    }
}
```

Abbildung 4.21 Migration durchführen (MapperHelper Klasse)

Falls ein Fehler bei der Migration auftritt, wird eine entsprechende Fehlermeldung angezeigt und das „Back“ aktiviert, um Änderungen durchzuführen und die Migration nochmal zu starten. Der Benutzer bekommt außerdem einen Hinweis, Falls die Migration erfolgreich abgeschlossen ist.

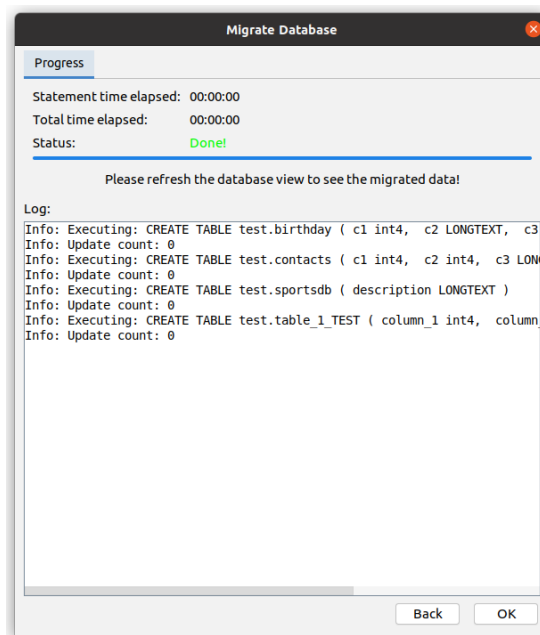


Abbildung 4.22 Fortschrittsübersicht



# Evaluation

Die folgenden Abschnitte beschäftigen sich mit der Bewertung des erstellten Plugins. Zuerst wird die Evaluationsmethodik sowie der Ablauf der Evaluation beschrieben und anschließend werden die Ergebnisse vorgestellt.

## 5.1 Evaluationsmethodik

Um einen Gesamteindruck über die Gebrauchstauglichkeit (Usability) zu haben und Anpassungsmöglichkeiten zu identifizieren, wurde das entwickelte Tool in Form eines qualitativen Experteninterviews evaluiert.

Es werden in der Regel mindestens fünf Testpersonen benötigt um die meisten auftretenden Probleme (85%) in einem System zu identifizieren[NL93]. Allerdings wurde im Rahmen dieser Bachelorarbeit aufgrund beschränkter zeitlicher Ressourcen nur ein Experteninterview durchgeführt.

die wichtigste Voraussetzung bei der Auswahl der Expertin war lediglich die Erfahrung mit Benutzerschnittstellen und User Experience. Außerdem waren Kenntnisse in der Datenbankverwaltung bzw. Datenbankmigration von Vorteil.

## 5.2 Durchführung

Üblicherweise wird das Experteninterviews Leitfadeninterview durchgeführt [May94]. Dies ermöglicht die Erhebung von einzelnen, bestimmbar Informationen sowie die Verfolgung von bestimmten Informationszielen. Deswegen wurde diese Befragungsart für die Evaluation gewählt. Bei dieser Befragungsmethode werden oft offene Fragen gestellt, um mehr Freiraum für Antworten und möglichst viele Informationen aus dem Interview zu gewinnen.

Das Experteninterview hat online über Teams<sup>1</sup> stattgefunden. Der Hauptgrund dafür war die aktuelle Corona Pandemie. Außerdem bieten Online Interviews eine flexible Vereinbarung gegenüber den physischen Interviews.

Während des Experteninterviews wurde das GuttenBase Plugin über ein Share-Screen gezeigt. Dabei wurden die wichtigsten Funktionalitäten vorgestellt. Zunächst wurden Fragen aus dem Leitfaden gestellt, welches vorab erstellt wurde. Das Leitfaden beinhaltet hauptsächlich Fragen über die Grundsätze der Informationsdarstellung und Benutzer-System-Interaktion, welche im Abschnitt 1.2 erläutert wurden. Außerdem wurden Fragen hinsichtlich der Verbesserungsmöglichkeiten gestellt.

## 5.3 Ergebnisse

Um das Interview auswerten zu können, wurden die Gespräche nach der Durchführung transkribiert. Zunächst wurde das transkribierte Interview gemäß der qualitativen Inhaltsanalyse kodiert [May94]. Dabei wurde das Transkript genau durchgelesen und den Textfragmenten wurden bestimmte Codes zugeordnet. Nach der Transkription erfolgte die Kategorienbildung. Dabei wurden folgende Kategorien identifiziert, die relevante Codes zusammenfassen:

- Eindruck
- Usability
- Verbesserungsvorschläge

Im Folgenden werden die Ergebnisse des Interviews entsprechend der genannten Kategorien zusammen gefasst.

### Eindruck

Der Gesamteindruck der Anwendungsoberfläche bezeichnet die Expertin als zufriedenstellend und positiv. Die Benutzeroberfläche wirkte insgesamt intuitiv und einheitlich mit der IntelliJ Entwicklungsumgebung. Außerdem findet die Expertin die Datenbankmigration in mehreren Schritten relativ einfach.

### Usability

Es wurden mehrere Aspekte der Gebrauchstauglichkeit von den Teilnehmerin bewertet. Beispielsweise bewertet sie die **Fehlertoleranz** positiv. Sie gibt an, dass an fast allen Stellen wird eine

---

<sup>1</sup><https://teams.microsoft.com/>

Fehlermeldung angezeigt, die die Ursache des Fehlers erklärt. Dies ist der Fall bei der Datenbankverbindung zu sehen, wenn die Zugangsdaten nicht stimmen. Dabei wird die Fehlermeldung allerdings relativ spät angezeigt (erst nach dem Klick auf das Next Button), was die Expertin als verbesserungswürdig bezeichnet.

Die **Selbstbeschreibfähigkeit** bezeichnet die Teilnehmerin als verbesserungsfähig. Dies liegt hauptsächlich an der Platzierung des Plus Button bei der Konfigurationsübersicht des Plugins. Die Teilnehmerin findet konnte nicht herausfinden, was die Funktion des Buttons ist. Außerdem gibt sie an, dass die Bezeichnungen von den Menüpunkten für das Öffnen der Benutzeroberfläche nicht selbstbeschreibend sind.

Die Teilnehmerin bewertet die **Aufgabenangemessenheit** generell positiv. Allerdings findet sie das Hinzufügen von Migrationsoperationen während der Migration wichtig. Dies wird aktuell allerdings nicht unterstützt.

Die **Konsistenz** äußert sich die Expertin insgesamt positiv. Sie findet das Trennen von den Quell- und Zieldatenbanken bei der Datenbankverbindung gut. Allerdings ist ihr aufgefallen, dass die Schriftgröße von den „Source“ und „Target“ Überschriften zu klein. Sie findet außerdem Die Gruppierung von den Menüpunkten beim Öffnen der Benutzeroberfläche verbesserungswürdig.

Die Teilnehmerin bezeichnet die **Erwartungskonformität** als relativ gut. Sie gibt an, dass das System sich erwartungsgemäß. Auf der anderen Seite findet sie es schlecht, dass die Datenbanken nach dem Abschluß der Migration nicht automatisch aktualisiert werden.

## Verbesserungsvorschläge

Während des Interviews hat die Expertin vier Ideen vorgeschlagen, um das GuttenBase zu verbessern. Diese werden im Folgenden zusammengefasst:

- **V1:** Menüpunkt „Show Actions“ zu „Show Migration Action“ umbenennen.
- **V2:** Menüpunkte „Migrate Database“ und „Show Migration Action“ gruppieren und unter dem Menüpunkt „Database Tools“ bewegen.
- **V3:** Schriftgröße von „Source“ und „Target“ bei der Datenbankverbindung vergrößern.
- **V4:** Echtzeit Überprüfung von den Benutzereingaben während der Datenbankverbindung.
- **V5:** „+“ Button bei der Konfigurationsübersicht umbenennen oder neben den Migrationsoperationen platzieren.
- **V6:** Ein automatisches Aktualisieren der Zieldatenbank ermöglichen oder einen entsprechenden Hinweis am Ende der Migration in der Fortschrittsübersicht anzeigen, damit der Benutzer informiert wird.

## 5.4 Anpassungen

Nach der Evaluation wurden Anpassungen vorgenommen, die schnell einsetzbar sind. Diese werden im Folgenden aufgelistet:

- **V1:** Menüpunkt „Show Actions“ zu „Show Migration Action“ umbenennen.
- **V2:** Menüpunkte „Migrate Database“ und „Show Migration Action“ gruppieren und unter dem Menüpunkt „Database Tools“ bewegen.
- **V3:** Schriftgröße von „Source“ und „Target“ bei der Datenbankverbindung vergrößern.
- **V5:** „+“ Button bei der Konfigurationsübersicht neben den Migrationsoperationen platzieren.
- **V6:** Hinweis am Ende der Migration in der Fortschrittsübersicht anzeigen, damit der Benutzer informiert wird.

Es ist zu beachten, dass der Abschnitt 4.3 entsprechend der neuen Anpassungen aktualisiert wurde.

## Fazit und Ausblick

### 6.1 Fazit

Nach einer Einführung in die Grundlagen der Datenbank Migration, Guttenbase und die IntelliJ Plugin Entwicklung, wurde eine Anforderungsanalyse durchgeführt. basierend darauf wurde das GuttenBase Plugin entworfen, implementiert und evaluiert.

Mit dem GuttenBase Plugin lassen sich Datenbanken durch wenige Klicks migrieren. Während des Migrationsprozesses können Migrationsoperationen hinzugefügt werden wie das Umbenennen der Quell-Tabellen bzw. Spalten sowie das Ausschließen bestimmter Tabellen bzw. Spalten und das Ändern von Spalten-Datentypen. Diese können vor- oder in dem Migrationsprozess erstellt werden und werden gespeichert, um sie bei der nächsten Migration wiederverwenden zu können. Das GuttenBase Plugin läuft basierend auf der GuttenBase Bibliothek und interagiert mit dem Database Plugin von IntelliJ.

Mit den oben genannten Funktionalitäten erreicht das GuttenBase Plugin das zu Beginn der Bachelorarbeit gesetzte Ziel.

### 6.2 Ausblick

Die in dieser Bachelorarbeit implementierten Funktionalitäten stellen eine Grundlage dar, die sich beliebig ausbauen lässt. Zum Einen können die fehlenden Verbesserungsvorschläge aus dem Abschnitt 5.3 umgesetzt werden. Zum Anderen kann das GuttenBase Plugin um weitere Migrationsoperationen erweitert werden.

Es spricht im Grunde nichts dagegen, das GuttenBase Plugin auf andere Entwicklungsumgebungen wie Eclipse oder NetBeans zu übertragen. Da einige Implementierungen von der IntelliJ API abhängig sind, müsste jede Funktionalität entsprechend der Schnittstelle der jeweiligen Entwicklungsumgebung implementiert werden. Eine Voraussetzung für eine solche Übertragung ist die

Unterstützung von Datenbanken. Ansonsten müssen zusätzliche Informationen über die Quell- und Ziel-Datenbank erforderlich.

Es könnte außerdem vorkommen, dass einige Bugs während der Nutzung des Plugins auftreten. Diese können natürlich durch regelmäßige Wartung gelöst werden.

# Literatur

- [Bas12] Youssef Bassil. »A comparative study on the performance of the Top DBMS systems«. In: *arXiv preprint arXiv:1205.2889* (2012).
- [Dut16] *SAP analytics products*. Springer, 2016, S. 71–85.
- [DW18] Ilka Datig und Paul Whiting. »Telling your library story: tableau public for data visualization«. In: *Library Hi Tech News* (2018).
- [EA15] M Elamparithi und V Anuratha. »A Review on Database Migration Strategies, Techniques and Tools«. In: *World Journal of Computer Application and Technology* 3.3 (2015), S. 41–48.
- [Gei14] Frank Geisler. *Datenbanken: Grundlagen und Design*. mitp Verlags GmbH & Co. KG, 2014.
- [HLS74] *Architecture to an interactive migration system (AIMS)*. 1974, S. 157–169.
- [Hor05] Jutta Horstmann. »Migration to open source databases«. In: *Computation and Information Structures (CIS)* (2005), S. 12–13.
- [Kow18] Stacy T Kowalczyk. *Digital Curation for Libraries and Archives*. ABC-CLIO, 2018.
- [KT18] Mutale Kasonde und Simon Tembo. »A Seamless Network Database Migration Tool for Institutions in Zambia«. In: *International Journal of Advanced Computer Science and Applications* 9.1 (2018), S. 191–199.
- [May94] Philipp Mayring. *Qualitative Inhaltsanalyse*. Bd. 14. UVK Univ.-Verl. Konstanz, 1994.
- [NK92] *Relationales Datenbankmodell (REL)*. Springer, 1992, S. 15–49.
- [NL93] *A mathematical model of the finding of usability problems*. 1993, S. 206–213.
- [Ram13] *Was ist „Qualitative Inhaltsanalyse?“*. Springer, 2013, S. 23–42.
- [WZ15] Sabine Wachter und Thomas Zaelke. *Systemkonsolidierung und Datenmigration als Erfolgsfaktoren: HMD Best Paper Award 2014*. Springer-Verlag, 2015.