

Dein eigener KI-Agent mit Java!

Michael Niedermair und andere, realisiert mit KI-Unterstützung



generiert mit Google AI Studio



1	Einführung	3
1.1	Das Ziel: Der „Datenbank-Abfrager“	3
1.2	Einleitung	4
1.3	Voraussetzungen	6
1.4	Die Datenbank aufsetzen (MariaDB)	7
2	Agenten in Java erstellen	8
2.1	Das Maven-Projekt anlegen & konfigurieren	8
2.2	Übersicht: So greifen die Komponenten ineinander	10
2.3	Das Werkzeug (Tool) für den Datenbankzugriff	11
2.4	Datenbank-Helfer	12
2.5	KI-Helfer	15
2.6	Die Persönlichkeit des Agenten definieren	20
2.7	Sequenz-Diagramm	21
2.8	Klassen-Diagramm	22
2.9	Alles zusammenfügen und den Agenten starten	23
3	Ausprobieren und Testen	26
4	Aufgaben	27
5	Voraussetzungs-Review	28
5.1	Grundlagen Java (OpenJDK 21)	29
5.1.1	Welche Themen solltet ihr auffrischen?	29
5.1.2	Hilfreiche Links zur Wiederholung	29
5.2	Grundlagen SQL (mit MariaDB 11.3)	29
5.2.1	Welche Themen solltet ihr auffrischen?	30
5.2.2	Hilfreiche Links zur Wiederholung	30
5.3	Grundlagen JDBC	30
5.3.1	Welche Themen solltet ihr auffrischen?	30
5.3.2	Hilfreiche Links zur Wiederholung	31
5.4	Zugriff auf das GitHub-Repository	31
5.4.1	Was müsst ihr können?	31
5.4.2	Hilfreiche Links zur Wiederholung	31
5.5	Debugging-Tipps: Euer mächtigstes Werkzeug	31
5.5.1	Warum ist der Debugger hier so nützlich?	31
5.5.2	Ein praktisches Beispiel: Den Agenten beim Denken beobachten	32
6	Mit der KI lernen	33



1 Einführung

1.1 Das Ziel: Der „Datenbank-Abfrager“

DB-Genius

Wir entwickeln einen Java-Agenten namens „DB-Genius“. Dieser Agent wird über die Konsole gestartet. Ihr könnt ihm dann Fragen auf Deutsch stellen, wie „Wie viele Schüler haben wir?“ oder „Liste alle aus der Klasse FIAE-23a auf“. Der Agent wird diese Fragen verstehen, sie in SQL-Code für unsere MariaDB übersetzen, die Datenbank abfragen und eine freundliche Antwort auf Deutsch geben.

Die Dateien etc. findet Ihr unter:

https://github.com/Michael2024abc/EduKI/tree/main/einheiten/ki_agent_einfach

ARD-Audiothek – Podcast

🎵 Wie baue ich meinen eigenen KI-Agenten?

Einfach mal anhören 😊

„KI-Agenten gelten als der nächste große Schritt nach Chatbots – doch was machen sie eigentlich anders?“

In dieser Folge testen Marie und Gregor, wie sich Agenten in den Alltag holen lassen, was beim Bauen schiefgehen kann und warum manchmal schon ein kleines Kalender-Widget die größte Hürde ist.“

<https://www.ardaudiothek.de/episode/urn:ard:section:87c47940afea7273/>



1.2 Einleitung

Eure Reise in die Welt der KI-Agenten

Hello zusammen und herzlich willkommen zu einem der spannenden Projekt, welches euch in die KI-Agenten einführt! Ihr steht kurz davor, nicht nur Code zu schreiben, sondern einer Maschine beizubringen, menschliche Sprache zu verstehen und darauf zu reagieren.

Ihr werdet einen eigenen KI-Agenten bauen – den „DB-Genius“.

Stellt euch vor, ihr könntet eurer Datenbank einfach in normalen Sätzen Fragen stellen, und sie antwortet euch. Genau das werdet ihr möglich machen!

Das „Gehirn“ eures Agenten wird ein sogenanntes großes Sprachmodell (**Large Language Model, LLM**) sein. Das sind extrem leistungsfähige KIs, die darauf trainiert wurden, Texte zu verstehen, zu verarbeiten und zu erzeugen. Für unser Projekt habt ihr die Wahl zwischen drei fantastischen, aber sehr unterschiedlichen „Gehirnen“.

Die Wahl des richtigen Werkzeugs für die richtige Aufgabe ist eine der wichtigsten Fähigkeiten eines modernen Entwicklers. Lasst uns eure Optionen kennenlernen:

Google Gemini: Der leistungsstarke Allrounder

Stellt euch Gemini als ein riesiges, kreatives Gehirn in der Cloud vor, das von Google entwickelt wurde. Es ist unglaublich gut darin, komplexe Anweisungen zu verstehen und präzise Antworten zu geben.

- **Zugang:**

Das Beste für den Einstieg: Der Zugang zu vielen Gemini-Modellen ist für Entwickler und zum Ausprobieren **kostenlos**. Ihr könnt also sofort und ohne Hürden loslegen.

- **Performance:**

Da Gemini auf Googles gewaltiger Infrastruktur läuft, erhaltet ihr Antworten extrem schnell und in sehr hoher Qualität.

- **DSGVO und Datenschutz:**

Hier müsst ihr als verantwortungsvolle Entwickler aufmerksam sein. Gemini ist ein Cloud-Dienst. Das bedeutet, jede Frage, die ihr dem Agenten stellt, wird zur Verarbeitung an die Server von Google in den USA gesendet. Für unser Projekt mit einer fiktiven Schüler-Datenbank ist das unbedenklich. Aber ihr lernt hier eine entscheidende Lektion:

Niemals echte, sensible oder personenbezogene Daten an solche öffentlichen Cloud-Dienste senden!

Die Nutzung ist toll zum Lernen, aber für echte Firmendaten gelten strengere Regeln.

GitHub Copilot: Euer persönlicher Coding-Partner

Copilot, entwickelt von GitHub und OpenAI, ist mehr als nur ein allgemeines Sprachmodell – es ist speziell darauf trainiert, Programmierern zu helfen. Es ist, als hättet ihr einen extrem erfahrenen Entwickler als „Beifahrer“, der euch ständig Tipps gibt.

- **Zugang:**

Hier habt ihr einen riesigen Vorteil! Über das **GitHub Education Pack** erhalten Schüler und Lehrer kostenlosen Zugang zu GitHub Copilot. Nutzt diese Chance unbedingt!

- **Performance:**

Copilot ist für Code-Aufgaben optimiert und kann oft noch präzisere SQL-Abfragen oder Code-Strukturen vorschlagen als allgemeinere Modelle.



- **DSGVO und Datenschutz:**

Genau wie bei Gemini handelt es sich um einen Cloud-Dienst von Microsoft/GitHub. Eure Anfragen werden zur Verarbeitung an deren Server gesendet. Es gelten also die gleichen Datenschutz-Überlegungen: Perfekt zum Lernen und für unkritische Daten, aber Vorsicht bei echten, schützenswerten Informationen.

Ollama: Eure private und lokale KI-Werkstatt

Ollama ist der revolutionäre Ansatz in unserer Liste. Anstatt eure Daten zu einem großen Unternehmen in die Cloud zu schicken, holt ihr euch mit Ollama das Sprachmodell direkt auf euren eigenen Computer!

- **Zugang:**

Ollama ist eine **kostenlose Open-Source-Software**, die ihr einfach installieren könnt. Damit könnt ihr eine Vielzahl an leistungsstarken Open-Source-Sprachmodellen (wie Mistral, Llama, etc.) lokal ausführen.

- **Performance:**

Die Geschwindigkeit eures lokalen Modells hängt vollständig von eurer eigenen Hardware ab. Ein PC mit einer guten Grafikkarte (GPU) und viel Arbeitsspeicher (RAM) wird deutlich schneller sein als ein älterer Laptop. Die Antworten kommen vielleicht nicht ganz so schnell wie von Gemini, aber die Qualität ist oft erstaunlich gut.

- **DSGVO und Datenschutz:**

Das ist die Superkraft von Ollama. Da das Modell zu 100% auf eurem Rechner läuft, **verlässt kein einziges Zeichen eurer Anfragen jemals euren Computer**. Damit ist Ollama die perfekte Lösung für datenschutzkritische Anwendungen und zu **100% DSGVO-konform**. Ihr habt die volle Kontrolle.

Ihr habt nun die Wahl: die brachiale Power der Cloud-Giganten für maximale Leistung oder die absolute Kontrolle und den Datenschutz eurer eigenen, lokalen KI. Experimentiert mit allen drei Systemen! Findet heraus, welche Vor- und Nachteile sie in der Praxis haben. Diese Erfahrung ist unbezahlbar.



1.3 Voraussetzungen

Das solltet Ihr können!

- Grundlagen Java (OpenJDK 21)
- Grundlagen SQL (mit MariaDB 11.3)
- ein wenig JDBC
- Zugriff auf das github-Repository, damit Ihr die Dateien nicht abtippen müsst.

Siehe hierzu auch das Voraussetzungs-Review auf Seite [28](#).



generiert mit Perplexity



1.4 Die Datenbank aufsetzen (MariaDB)

Jeder gute Agent braucht Daten. Wir erstellen eine einfache Datenbank und eine Tabelle für unsere Schüler.



Was ist zu tun?

1. Öffnet dazu eine SQL-Konsole (z. B. mariadb, HeidiSQL, DBeaver, ...).
2. Führt die folgenden SQL-Befehle aus, um die Datenbank, die Tabelle und die Beispieldaten zu erstellen.
3. Die zusätzlichen Kommentare helfen euch, die Schritte besser zu verstehen.

create_db.sql

```
1 -- Ausführen als DB-Administrator, normalerweise 'root'.
2
3 -- Wechselt zur Hauptdatenbank für die Verwaltung.
4 USE mysql;
5
6 -- Erstellt eine neue Datenbank, falls sie noch existiert, vorher löschen.
7 DROP DATABASE IF EXISTS dbagent;
8 CREATE DATABASE dbagent CHARACTER SET utf8;
9
10 -- Erstellt einen neuen User, falls dieser noch existiert, vorher löschen.
11 -- User 'dbagent', PW ist leer
12 DROP USER IF EXISTS 'dbagent'@'localhost';
13 CREATE USER 'dbagent'@'localhost' IDENTIFIED BY '';
14
15 -- Rechte vergeben.
16 GRANT ALL PRIVILEGES ON dbagent.* TO 'dbagent'@'localhost';
17
18 -- Rechte aktualisieren.
19 FLUSH PRIVILEGES;
20
21 -- Wählt die neue Datenbank für die folgenden Befehle aus.
22 USE dbagent;
23
24 -- Erstellt die Tabelle 'schueler' mit den notwendigen Spalten.
25 CREATE TABLE schueler (
26     id INT AUTO_INCREMENT PRIMARY KEY,
27     vorname VARCHAR(50),
28     nachname VARCHAR(50),
29     klasse VARCHAR(20),
30     eintrittsdatum DATE
31 ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
32
33 -- Fügt Beispieldaten in unsere Tabelle ein.
34 -- Diese Daten wird unser Agent später abfragen.
35 INSERT INTO schueler (vorname, nachname, klasse, eintrittsdatum) VALUES
36 ('Anna', 'Schmidt', 'FIAE-23a', '2023-08-01'),
37 ('Ben', 'Müller', 'FIAE-23b', '2023-08-01'),
38 ('Carla', 'Weber', 'FIAE-23a', '2023-09-01'),
39 ('David', 'Wagner', 'FISI-24', '2024-08-01'),
40 ('Elена', 'Becker', 'FIAE-23b', '2023-08-01'),
41 ('Felix', 'Hoffmann', 'FISI-24', '2024-09-01');
42
43 -- Überprüfen, ob die Daten korrekt eingefügt wurden.
44 SELECT * FROM schueler;
```



2 Agenten in Java erstellen

2.1 Das Maven-Projekt anlegen & konfigurieren



☰ Was ist zu tun?

Jetzt erstellen wir das Skelett unseres Java-Projekts mit Maven.

1. Erstellt ein neues, leeres Maven-Projekt in eurer IDE
(z. B. IntelliJ IDEA: File > New > Project... und wählt Maven Archetype).
2. Öffnet die generierte pom.xml-Datei und ersetzt den Inhalt durch den folgenden Code. Jeder Abschnitt ist kommentiert, um zu erklären, was er tut.
3. **Wichtig:** Nachdem die pom.xml gespeichert wurde, lässt Maven die Abhängigkeiten aktualisieren (in IntelliJ passiert dies oft automatisch oder über ein kleines Icon).

pom.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project
3   xmlns="http://maven.apache.org/POM/4.0.0"
4   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
6     http://maven.apache.org/xsd/maven-4.0.0.xsd">
7
8   <modelVersion>4.0.0</modelVersion>
9   <groupId>de.gc</groupId>
10  <artifactId>dbagent</artifactId>
11  <version>0.1.1</version>
12
13  <properties>
14    <!-- Wir legen hier zentrale Versionen fest, um sie leicht zu aktualisieren. --&gt;
15    &lt;maven.compiler.source&gt;21&lt;/maven.compiler.source&gt;
16    &lt;maven.compiler.target&gt;21&lt;/maven.compiler.target&gt;
17    &lt;project.build.sourceEncoding&gt;UTF-8&lt;/project.build.sourceEncoding&gt;
18
19    &lt;langchain4j.version&gt;1.2.0&lt;/langchain4j.version&gt;
20    &lt;langchain4j-google-ai-gemini.version&gt;1.2.0&lt;/langchain4j-google-ai-gemini.version&gt;
21    &lt;langchain4j-github-models.version&gt;1.2.0-beta8&lt;/langchain4j-github-models.version&gt;
22    &lt;langchain4j-ollama.version&gt;1.2.0&lt;/langchain4j-ollama.version&gt;
23
24    &lt;mariadb-java-client.version&gt;3.5.4&lt;/mariadb-java-client.version&gt;
25    &lt;slf4j-simple.version&gt;2.0.17&lt;/slf4j-simple.version&gt;
26    &lt;text-table-formatter.version&gt;1.2.4&lt;/text-table-formatter.version&gt;
27
28  &lt;/properties&gt;
29
30  &lt;dependencies&gt;
31
32    &lt;!-- ===== --&gt;
33    &lt;!-- ABHÄNGIGKEITEN FÜR LANGCHAIN4J &amp; GOOGLE GEMINI --&gt;
34    &lt;!-- ===== --&gt;
35
36    &lt;!-- Das Herzstück von LangChain4j. Bietet die Kern-APIs. --&gt;
37    &lt;dependency&gt;
38      &lt;groupId&gt;dev.langchain4j&lt;/groupId&gt;
39      &lt;artifactId&gt;langchain4j&lt;/artifactId&gt;
40      &lt;version&gt;${langchain4j.version}&lt;/version&gt;
41    &lt;/dependency&gt;
42
43    &lt;!-- Der Konnektor speziell für Google Gemini AI). --&gt;
44    &lt;!-- Damit kann unsere Java-Anwendung mit der Gemini-API sprechen. --&gt;
45    &lt;dependency&gt;
46      &lt;groupId&gt;dev.langchain4j&lt;/groupId&gt;
47      &lt;artifactId&gt;langchain4j-google-ai-gemini&lt;/artifactId&gt;
48      &lt;version&gt;${langchain4j-google-ai-gemini.version}&lt;/version&gt;
49    &lt;/dependency&gt;
50
51    &lt;!-- Der Konnektor speziell für GitHub Copilot. --&gt;</pre>
```



```

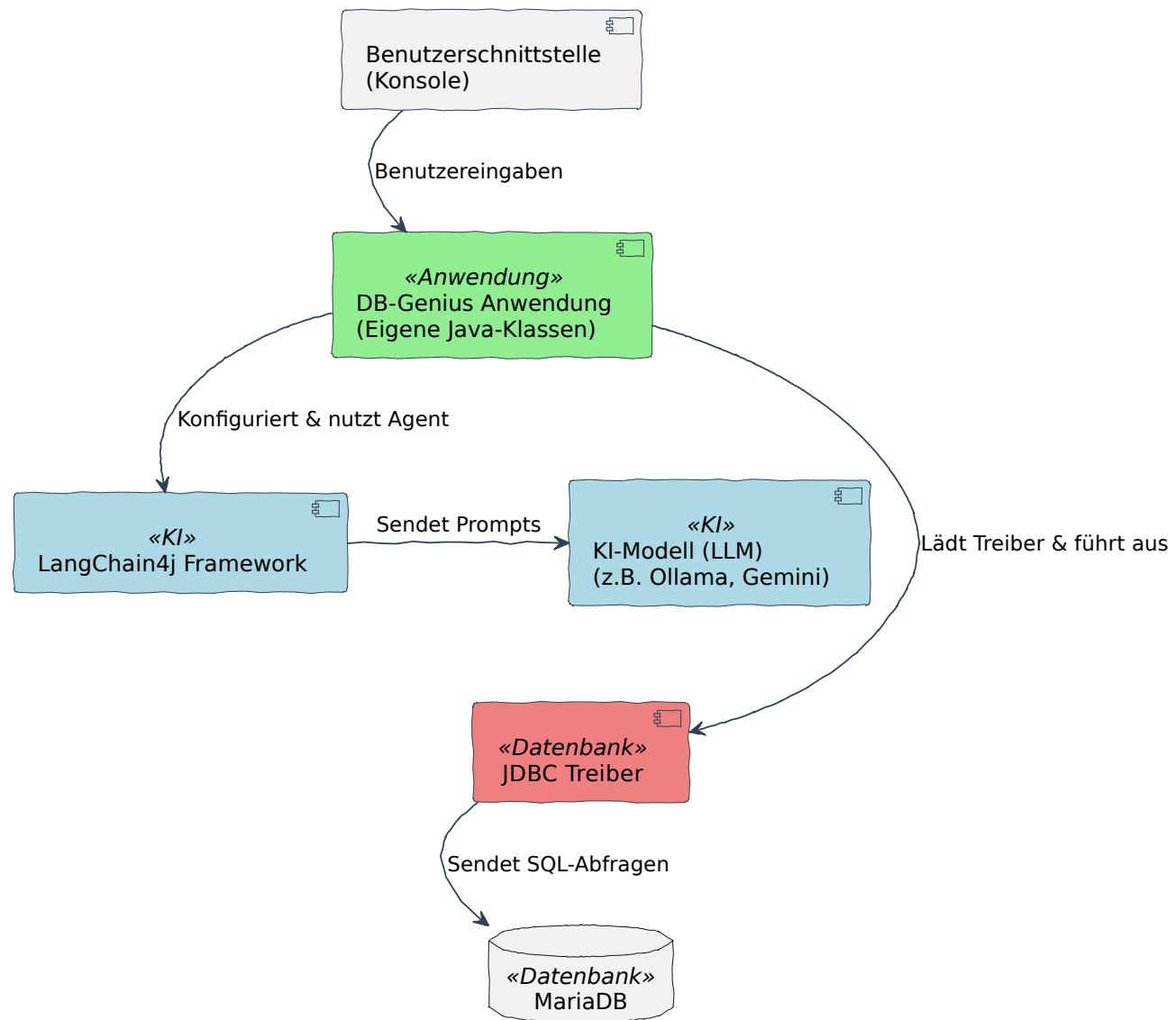
52 <dependency>
53   <groupId>dev.langchain4j</groupId>
54   <artifactId>langchain4j-github-models</artifactId>
55   <version>${langchain4j-github-models.version}</version>
56 </dependency>
57
58 <!-- Der Konnektor speziell für Ollama. -->
59 <dependency>
60   <groupId>dev.langchain4j</groupId>
61   <artifactId>langchain4j-ollama</artifactId>
62   <version>${langchain4j-ollama.version}</version>
63 </dependency>
64
65 <!-- ===== -->
66 <!-- ABHÄNGIGKEITEN FÜR DIE DATENBANK & LOGGING, ... -->
67 <!-- ===== -->
68
69 <!-- Der offizielle JDBC-Treiber für MariaDB. -->
70 <dependency>
71   <groupId>org.mariadb.jdbc</groupId>
72   <artifactId>mariadb-java-client</artifactId>
73   <version>${mariadb-java-client.version}</version>
74 </dependency>
75
76 <!-- Erzeugt schöne Text-Tabellen -->
77 <dependency>
78   <groupId>org.nocrala.tools.texttablefmt</groupId>
79   <artifactId>text-table-formatter</artifactId>
80   <version>${text-table-formatter.version}</version>
81 </dependency>
82
83 <!-- Eine einfache Logging-Implementierung. Sehr nützlich, um zu sehen, -->
84 <!-- was LangChain4j im Hintergrund macht (z.B. welche Prompts es sendet). -->
85 <dependency>
86   <groupId>org.slf4j</groupId>
87   <artifactId>slf4j-simple</artifactId>
88   <version>${slf4j-simple.version}</version>
89 </dependency>
90 </dependencies>
91
92 </project>

```



2.2 Übersicht: So greifen die Komponenten ineinander

Komponentendiagramm: DB-Genius Architektur



2.3 Das Werkzeug (Tool) für den Datenbankzugriff

Der Agent selbst kann nicht direkt mit der Datenbank sprechen. Wir geben ihm ein „Werkzeug“ an die Hand. Das ist eine einfache Java-Methode, die der Agent bei Bedarf aufrufen kann.



DatabaseTools.java

```
1 package de.gc.agent.db;
2
3 import java.sql.Connection;
4 import java.sql.ResultSet;
5 import java.sql.SQLException;
6 import java.sql.Statement;
7
8 import dev.langchain4j.agent.tool.Tool;
9
10 /**
11  * Diese Klasse enthält die Werkzeuge, die unser KI-Agent verwenden
12  * kann. Jede Methode, die mit @Tool annotiert ist, steht dem Agenten
13  * zur Verfügung.
14 */
15 public class DatabaseTools {
16
17     /**
18      * Dies ist das einzige Werkzeug, das wir dem Agenten geben.
19      *
20      * Die Beschreibung in der @Tool-Annotation ist EXTREM WICHTIG.
21      *
22      * Der Agent liest diesen Text, um zu verstehen, was das Werkzeug tut
23      * und wann er es einsetzen soll.
24      *
25      * @param sqlQuery Die SQL-Anfrage, die vom Agenten generiert wurde.
26      *
27      * @return Ein String mit dem Ergebnis der Datenbankabfrage, formatiert
28      *         als Tabelle.
29     */
30     @Tool("Führt eine SQL-Abfrage in der Schüler-Datenbank aus und gibt das Ergebnis zurück.")
31     public String executeQuery(final String sqlQuery) {
32
33         System.out.println(" SQL query: " + sqlQuery);
34
35         // Wir stellen eine Verbindung zur Datenbank her - singelton.
36         final Connection con = DbUtil.getConnection();
37
38         // Wir verwenden try-with-resources, um sicherzustellen, dass die
39         // Ressourcen am Ende automatisch geschlossen werden.
40         try (Statement stmt = con.createStatement();
41              ResultSet rs = stmt.executeQuery(sqlQuery)) {
42
43             // Wir verwenden die Hilfsmethode DbUtil.getOutputRs(), um das
44             // ResultSet in eine String (Tabelle) umzuwandeln.
45             final String output = DbUtil.getOutputRs(rs);
46
47             // Wenn keine Zeilen gefunden wurden, geben wir eine freundliche
48             // Nachricht zurück.
49             if (output.isBlank()) {
50                 return "Die Abfrage hat keine Ergebnisse geliefert.";
51             }
52
53             return output;
54         }
55     }
56 }
```



```

55     } catch (final SQLException e) {
56         // Wenn etwas schiefgeht (z.B. ungültiges SQL), geben wir die
57         // Fehlermeldung an den Agenten zurück.
58         // Er kann dann versuchen, seinen Fehler zu korrigieren.
59         return "Fehler bei der Ausführung der SQL-Abfrage: " + e
60             .getMessage();
61     }
62 }

```

2.4 Datenbank-Helfer

Damit die Logik von „Tool“ und „Datenbank“ getrennt ist, wird für die Datenbank eine eigene Klasse erzeugt.



DbUtil.java

```

1 package de.gc.agent.db;
2
3 import java.io.IOException;
4 import java.sql.Connection;
5 import java.sql.DriverManager;
6 import java.sql.ResultSet;
7 import java.sql.ResultSetMetaData;
8 import java.sql.SQLException;
9 import java.util.Properties;
10
11 import org.nocrala.tools.texttablefmt.Table;
12 import org.slf4j.Logger;
13 import org.slf4j.LoggerFactory;
14
15 /**
16 * Diese Klasse bietet Hilfsfunktionen für die Datenbankverbindung und
17 * die Verarbeitung von SQL-Abfragen. Sie ermöglicht das Laden der
18 * Datenbankkonfiguration aus einer Datei und stellt eine
19 * Singleton-Verbindung zur Datenbank bereit.
20 */
21 public class DbUtil {
22
23     /** Logger für die Protokollierung von Ereignissen in dieser Klasse. */
24     private static final Logger logger = LoggerFactory.getLogger(DbUtil.class);
25
26     /** Die Datenbankverbindung, die von dieser Klasse verwaltet wird. */
27     private static Connection connection;
28
29     /**
30      * Schließt die Datenbankverbindung, wenn sie geöffnet ist. Setzt die
31      * Verbindung auf null, um anzugeben, dass sie geschlossen wurde.
32      */
33     public static void close() {
34         if (connection != null) {
35             try {
36                 if (!connection.isClosed()) {
37                     connection.close();
38                     logger.info("Database connection closed.");
39                     // Setze die Verbindung auf null, um sie als geschlossen zu
40                     // markieren.
41                     connection = null;
42                 }
43             }
44         }
45     }

```



```

43     } catch (final SQLException e) {
44         logger.error("Error closing database connection.", e);
45         throw new RuntimeException("Error closing database connection.", e);
46     }
47 }
48 }
49
50 /**
51 * Stellt eine Verbindung zur Datenbank her, wenn sie noch nicht
52 * besteht. Lädt die Datenbankkonfiguration aus der Datei db.properties.
53 *
54 * @return Eine aktive {@link Connection} zur Datenbank.
55 */
56 public static Connection getConnection() {
57
58     // Überprüfen, ob die Verbindung bereits besteht
59     if (connection == null) {
60
61         // Wenn die Verbindung nicht besteht, versuchen wir, sie zu laden
62         logger.debug("Loading database connection from db.properties...");
63         final Properties dbProperties = new Properties();
64         try {
65
66             // Laden der Datenbankkonfiguration aus der Datei db.properties
67             dbProperties
68                 .load(DbUtil.class.getResourceAsStream("/db.properties"));
69             final String dbUrl = dbProperties.getProperty("db.url");
70             final String dbUser = dbProperties.getProperty("db.user");
71             final String dbPassword = dbProperties.getProperty("db.password");
72
73             // Überprüfen, ob die erforderlichen Eigenschaften gesetzt sind
74             if (dbUrl == null || dbUser == null || dbPassword == null) {
75                 logger.error(
76                     "Die Datei db.properties muss die Eigenschaften db.url, db.user und db.password enthalten.");
77                 throw new RuntimeException(
78                     "Die Datei db.properties muss die Eigenschaften db.url, db.user und db.password enthalten.");
79             }
80
81             // Herstellen der Verbindung zur Datenbank
82             connection = DriverManager.getConnection(dbUrl, dbUser, dbPassword);
83             logger.debug("Datenbankverbindung war erfolgreich.");
84
85             // Registrieren eines Shutdown-Hooks, um die Verbindung zu
86             // schließen, wenn die VM beendet wird.
87             Runtime.getRuntime()
88                 .addShutdownHook(new Thread(() -> { DbUtil.close(); }));
89
90         } catch (final IOException e) {
91             logger.error("Fehler beim Laden der Datenbankkonfiguration", e);
92             throw new RuntimeException(
93                 "Fehler beim Laden der Datenbankkonfiguration.", e);
94         } catch (final SQLException e) {
95             logger.error("Fehler beim Herstellen der Datenbankverbindung.", e);
96             throw new RuntimeException(
97                 "Fehler beim Herstellen der Datenbankverbindung.", e);
98     }

```



```

99     }
100
101    return connection;
102 }
103
104 /**
105 * Gibt das Ergebnis eines ResultSets als formatierte Tabelle zurück.
106 *
107 * @param rs Das ResultSet, das die Abfrageergebnisse enthält.
108 *
109 * @return Ein String, der das Ergebnis der Abfrage in Tabellenform
110 *         darstellt.
111 */
112 public static String getOutputRs(final ResultSet rs) {
113
114     try {
115
116         // Ermittelt die Metadaten des ResultSets,
117         // um die Spalteninformationen zu erhalten.
118         final ResultSetMetaData rsmeta = rs.getMetaData();
119         final int cols = rsmeta.getColumnCount();
120         final Table t = new Table(cols);
121
122         // Fügt die Spaltenüberschriften zur Tabelle hinzu.
123         for (int i = 1; i <= cols; i++) {
124             t.addCell(rsmeta.getColumnLabel(i));
125         }
126
127         // Fügt die Datenzeilen zur Tabelle hinzu.
128         while (rs.next()) {
129             for (int i = 1; i <= cols; i++) {
130                 final Object obj = rs.getObject(i);
131                 t.addCell(obj == null ? "" : obj.toString());
132             }
133         }
134         return t.render();
135
136     } catch (final SQLException e) {
137         throw new RuntimeException(e);
138     }
139 }
140
141 private DbUtil() {
142     // keine Instanzierung erlaubt
143 }
144
145 }

```



2.5 KI-Helper

Damit das Erzeugen dieses KI-Models mit dem Login getrennt ist, wird hierfür eine eigene Klasse erzeugt.



XKILogin.java

```
1 /*
2  * Copyright (C) 2025 GNU GENERAL PUBLIC LICENSE Version 3, 29 June 2007
3  *
4  * Lizenzhinweis / License Notice
5  *
6  * Deutsch: Dieses Programm ist freie Software. Sie dürfen es unter den
7  * Bedingungen der GNU General Public License, Version 3, wie von der
8  * Free Software Foundation veröffentlicht, weitergeben und/oder
9  * modifizieren. Weitere Informationen finden Sie unter:
10 * https://www.gnu.org/licenses/gpl-3.0.de.html
11 *
12 * English: This program is free software. You can redistribute it
13 * and/or modify it under the terms of the GNU General Public License
14 * version 3 as published by the Free Software Foundation. For more
15 * information, see: https://www.gnu.org/licenses/gpl-3.0.html
16 */
17 package de.gc.agent.ki.model;
18
19 import java.io.File;
20 import java.io.FileInputStream;
21 import java.io.IOException;
22 import java.io.InputStream;
23 import java.time.Duration;
24 import java.util.Optional;
25 import java.util.Properties;
26
27 import dev.langchain4j.model.chat.ChatModel;
28 import dev.langchain4j.model.github.GitHubModelsChatModel;
29 import dev.langchain4j.model.googleai.GoogleAiGeminiChatModel;
30 import dev.langchain4j.model.googleai.GoogleAiGeminiChatModel.GoogleAiGeminiChatModelBuilder;
31 import dev.langchain4j.model.llama.OllamaChatModel;
32 import dev.langchain4j.model.llama.OllamaChatModel.OllamaChatModelBuilder;
33
34 /**
35  * XKILogin is a utility class for creating chat models for various KI
36  * systems.
37  *
38  * @author Michael Niedermair
39  */
40 public class XKILogin {
41
42     /** Enum for the supported KI systems. */
43     public enum KiSystem {
44         GEMINI, GITHUB, OLLAMA, NONE
45     }
46
47     /**
48      * Creates a {@link ChatModel} based on the provided KI system and model
49      * name.
50      *
51      * @param kisystem The KI system to use.
52      * @param modelName The name of the model.
53      * @param token The authentication token.
54      * @param url The base URL for the KI service.
55      *
```



```

56     * @return A configured {@link ChatModel}.
57     */
58     public static ChatModel createChatModel(final KiSystem kisystem,
59         final String modelName,
60         final String token, final String url) {
61
62     return switch (kisystem) {
63
64     case GITHUB ->
65         createGitHubBuilder(modelName, token, 0.7, 0.95, false,
66             Duration.ofSeconds(60));
67
68     case OLLAMA -> createOllamaBuilder(modelName, url, 0.7, 0.95, false,
69             Duration.ofSeconds(180));
70
71     case GEMINI ->
72         createGeminiBuilder(modelName, token, 0.7, 0.95, false,
73             Duration.ofSeconds(60));
74
75     default -> throw new IllegalArgumentException(
76         "Chat model for " + kisystem.name() + " not implemented yet.");
77     };
78 }
79 /**
80 * Creates a {@link ChatModel} based on the provided configuration.
81 *
82 * @param kisystem The KI system to use.
83 * @param modelName The name of the model.
84 * @param token The authentication token.
85 * @param url The base URL for the KI service.
86 * @param temperature The temperature for the model.
87 * @param topP The top P value for the model.
88 * @param logRequests Whether to log requests.
89 * @param timeout The timeout for requests in seconds.
90 */
91 public static ChatModel createChatModel(final KiSystem kisystem,
92     final String modelName,
93     final String token, final String url, final double temperature,
94     final double topP,
95     final boolean logRequests, final int timeout) {
96
97     // timeout
98     final Duration tout = timeout >= 60 ? Duration.ofSeconds(timeout)
99         : Duration.ofSeconds(60);
100
101     return switch (kisystem) {
102
103     case GITHUB -> createGitHubBuilder(modelName, token, temperature,
104             topP, logRequests, tout);
105
106     case OLLAMA -> createOllamaBuilder(modelName, url, temperature,
107             topP, logRequests, tout);
108
109     case GEMINI -> createGeminiBuilder(modelName, token, temperature,
110             topP, logRequests, tout);
111
112     default -> throw new IllegalArgumentException(
113         "Chat model for " + kisystem.name() + " not implemented yet.");
114     };
115 }
116 }
```



```

119
120 /**
121 * Creates a {@link ChatModel} for the Gemini KI system.
122 *
123 * @param modelName The name of the model.
124 * @param token The authentication token.
125 * @param temperature The temperature for the model.
126 * @param topP The top P value for the model.
127 * @param logRequests Whether to log requests.
128 * @param tout The timeout for requests.
129 */
130 private static ChatModel createGeminiBuilder(final String modelName,
131     final String token, final double temperature,
132     final double topP, final boolean logRequests,
133     final Duration tout) {
134
135     final GoogleAiGeminiChatModelBuilder builder = GoogleAiGeminiChatModel
136         .builder();
137     builder.modelName(modelName);
138     builder.apiKey(token);
139     builder.timeout(tout);
140
141     // Optional
142     Optional.ofNullable(temperature)
143         .ifPresent(builder::temperature);
144     Optional.ofNullable(topP)
145         .ifPresent(builder::topP);
146     Optional.ofNullable(logRequests)
147         .ifPresent(builder::logRequestsAndResponses);
148
149     return builder.build();
150 }
151
152 /**
153 * Creates a {@link ChatModel} for the GitHub KI system.
154 *
155 * @param modelName The name of the model.
156 * @param token The authentication token.
157 * @param temperature The temperature for the model.
158 * @param topP The top P value for the model.
159 * @param logRequests Whether to log requests.
160 * @param tout The timeout for requests.
161 */
162 private static ChatModel createGitHubBuilder(final String modelName,
163     final String token, final double temperature, final double topP,
164     final boolean logRequests, final Duration tout) {
165
166     final GitHubModelsChatModel.Builder builder = GitHubModelsChatModel
167         .builder();
168     builder.modelName(modelName);
169     builder.gitHubToken(token);
170     builder.timeout(tout);
171
172     // Optional
173     Optional.ofNullable(temperature)
174         .ifPresent(builder::temperature);
175     Optional.ofNullable(topP)
176         .ifPresent(builder::topP);
177     Optional.ofNullable(logRequests)
178         .ifPresent(builder::logRequestsAndResponses);
179
180     return builder.build();

```



```

181     }
182
183     /**
184      * Creates a {@link ChatModel} for the Ollama KI system.
185      *
186      * @param modelName The name of the model.
187      * @param url The base URL for the Ollama service.
188      * @param temperature The temperature for the model.
189      * @param topP The top P value for the model.
190      * @param logRequests Whether to log requests.
191      * @param tout The timeout for requests.
192      */
193     private static ChatModel createOllamaBuilder(final String modelName,
194         final String url, final double temperature, final double topP,
195         final boolean logRequests, final Duration tout) {
196
197         final OllamaChatModelBuilder builder = OllamaChatModel.builder();
198         builder.modelName(modelName);
199         builder.baseUrl(url);
200         builder.timeout(tout);
201
202         // Optional
203         Optional.ofNullable(temperature)
204             .ifPresent(builder::temperature);
205         Optional.ofNullable(topP)
206             .ifPresent(builder::topP);
207         Optional.ofNullable(logRequests)
208             .ifPresent(builder::logRequests);
209
210         return builder.build();
211     }
212
213     /**
214      * Retrieves the token for a given KI system from a properties file.
215      * <p>
216      * Example:
217      * <br>
218      * <pre>
219      * String token = getToken(new
220      * File("path/to/propsfile.properties"), "ge");
221      * <br>
222      * ge_token = YOUR_GEMINI_API_KEY
223      * </pre>
224      * <br>
225      * @param propsfile The properties file containing the tokens.
226      * @param kisystem_short_name The short name of the KI system.
227      * <br>
228      * @return The token for the specified KI system.
229      */
230     public static String getToken(final File propsfile,
231         final String kisystem_short_name) {
232         try {
233             final Properties properties = new Properties();
234             try (InputStream input = new FileInputStream(propsfile)) {
235                 properties.load(input);
236                 return properties.getProperty(kisystem_short_name + "_token");
237             }
238         } catch (final IOException e) {
239             throw new RuntimeException(
240                 "Error reading properties file: " + propsfile.getAbsolutePath(),
241                 e);
242     }

```



```
242 }
243 /**
244 * Converts a string to a KiSystem enum.
245 */
246 public static KiSystem kiSystemFromString(final String name) {
247     for (final KiSystem type : KiSystem.values()) {
248         if (type.name()
249             .equalsIgnoreCase(name)) {
250             return type;
251         }
252     }
253     return KiSystem.NONE;
254 }
255 }
256 }
```



2.6 Die Persönlichkeit des Agenten definieren

Wir erstellen jetzt die „Seele“ unseres Agenten. Dies geschieht über ein Java-Interface. Das Besondere hier ist die `@SystemMessage` – sie ist die Anweisung an die KI, wer sie sein soll und was ihre Aufgabe ist.



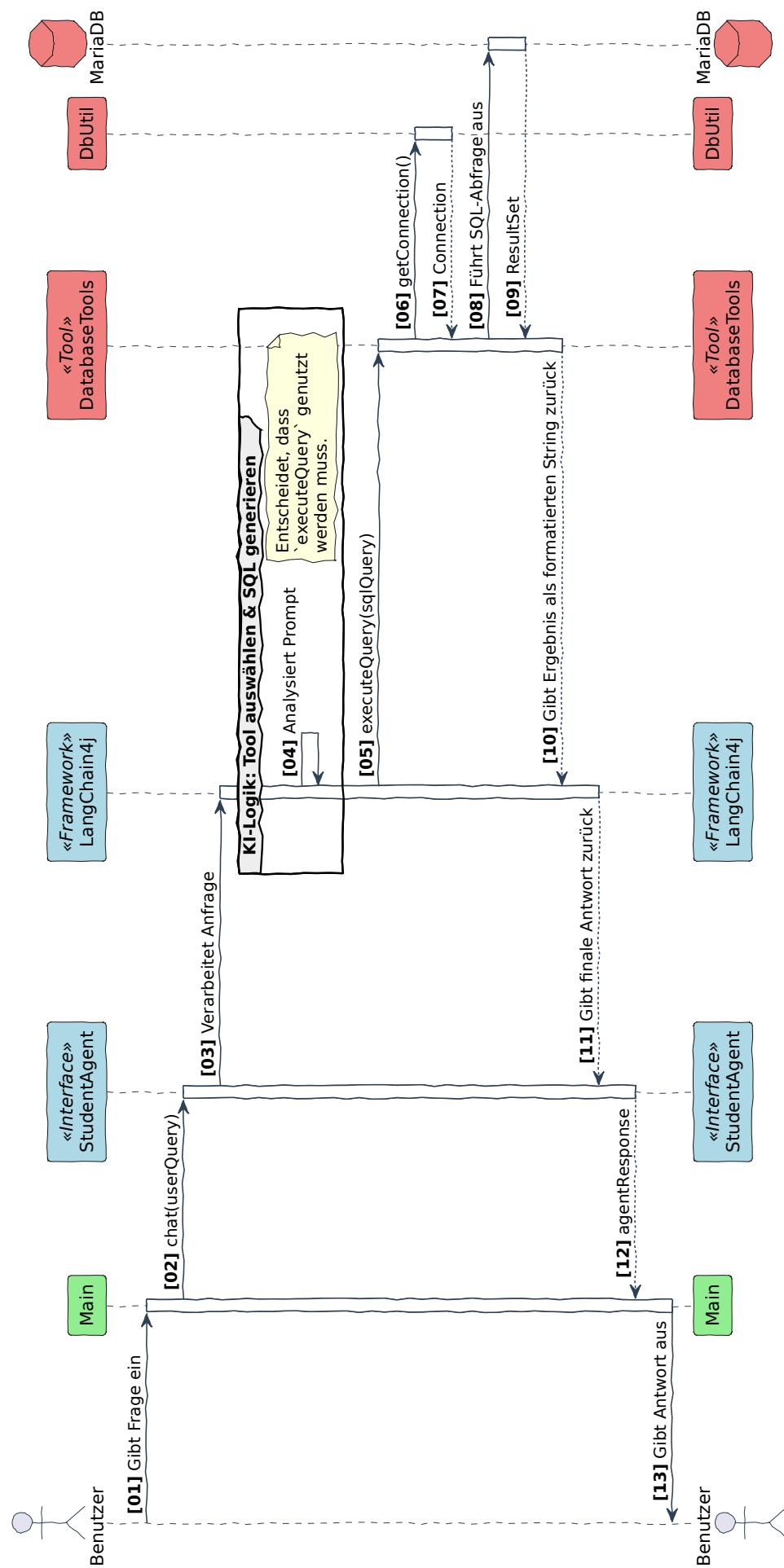
StudentAgent.java

```
1 package de.gc.agent.db;
2
3 import dev.langchain4j.service.SystemMessage;
4
5 /**
6  * Dies ist das Herzstück unseres Agenten. Es ist ein Interface, das
7  * definiert, wie wir mit dem Agenten interagieren. LangChain4j
8  * implementiert dieses Interface im Hintergrund für uns, wenn wir den
9  * AiServices.builder() aufrufen.
10 */
11 public interface StudentAgent {
12
13     /**
14      * Die @SystemMessage ist die wichtigste Anweisung an das Sprachmodell.
15      *
16      * Sie gibt dem Agenten seine Persönlichkeit, seine Anweisungen und den
17      * Kontext. Je detaillierter diese Anweisung, desto besser wird das
18      * Ergebnis.
19      *
20      */
21     @SystemMessage("""
22         Du bist 'DB-Genius', ein freundlicher und cleverer Assistent
23         für die Schüler-Datenbank der Fachinformatiker.
24         Deine Aufgabe ist es, Fragen von Schülern in natürlicher
25         Sprache in präzise MariaDB SQL-Abfragen zu übersetzen.
26
27         WICHTIGE REGELN:
28         1. Du darfst NUR SQL-Abfragen erstellen,
29             die Daten lesen (SELECT).
30             Du darfst niemals Daten ändern (UPDATE, DELETE, INSERT).
31         2. Das Schema der einzigen verfügbaren Tabelle ist:
32             schueler(ID, Vorname, Nachname, Klasse, Eintrittsdatum).
33         3. Nutze das dir zur Verfügung gestellte Werkzeug
34             'executeQuery', um die von dir erstellte SQL-Abfrage
35             auszuführen.
36         4. Antworte IMMER auf Deutsch.
37             Formuliere die Antwort freundlich und
38             gib das Ergebnis der Datenbankabfrage wieder.
39         5. Wenn du das Ergebnis hast, gib es direkt aus,
40             ohne die SQL-Abfrage selbst zu wiederholen.
41     """)
42     String chat(String userMessage);
43 }
```

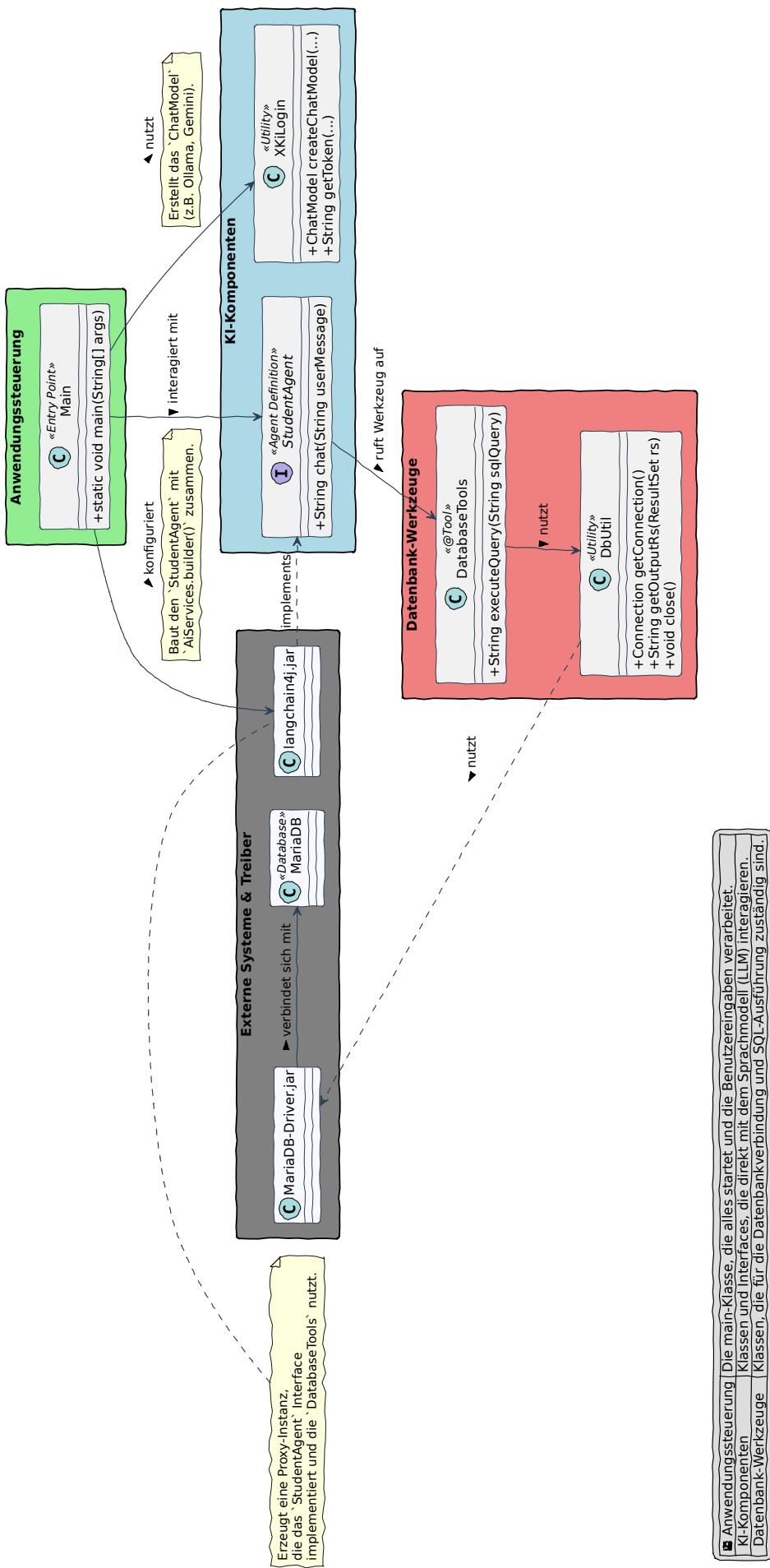


2.7 Sequenz-Diagramm

Sequenzdiagramm: Ablauf einer Benutzeranfrage



2.8 Klassen-Diagramm



■ Anwendungssteuerung Die main-Klasse, die alles startet und die Benutzererwartungen verarbeitet.
 ■ KI-Komponenten Klassen und Interfaces, die direkt mit dem Sprachmodell (LLM) interagieren.
 ■ Datenbank-Werkzeuge Klassen, die für die Datenbankverbindung und SQL-Ausführung zuständig sind.



2.9 Alles zusammenfügen und den Agenten starten

Jetzt kommt der finale Schritt: Wir schreiben die Main-Klasse, die den Agenten initialisiert und die interaktive Konsole startet.

KI auswählen

Jetzt müsst ihr euch entscheiden, welches KI-System ihr verwenden wollt.

Google Gemini

1. Gehe zu Google AI Studio. <https://aistudio.google.com/>
2. Klicke auf Get API key -> Create API key in new project.
3. Kopiere den Schlüssel. **Behandle diesen Schlüssel wie ein Passwort!**

GitHub Copilot

1. Seite zur Token-Erstellung öffnen.
Klicke auf den folgenden Link, um direkt zur Seite für die Erstellung eines neuen „klassischen“ Tokens auf GitHub zu gelangen:
<https://github.com/settings/tokens/new>
2. Token konfigurieren
Auf der Seite, die sich öffnet, musst du Folgendes tun:
 - Note (Name): Gib dem Token einen eindeutigen Namen, z. B. Copilot Token.
 - Expiration (Gültigkeit): Wähle aus, wie lange der Token gültig sein soll.
 - Select scopes (Berechtigungen auswählen): Scrolle durch die Liste und setze nur einen einzigen Haken bei der Option: copilot (Grant read-only access to GitHub Copilot)
 - Klicke ganz unten auf die grüne Schaltfläche "Generate token".
3. Token kopieren
Sofort kopieren! Direkt nach dem Klick wird dir dein neuer Token einmalig angezeigt.
Klicke auf das Kopieren-Symbol daneben, um ihn sicher in deine Zwischenablage zu legen.
Achtung: Bewahre diesen Token wie ein Passwort an einem sicheren Ort auf. Nachdem du die Seite verlassen hast, kannst du ihn nie wieder anzeigen lassen.

Ollama

- Ausführen und installieren großer Sprachmodelle lokal mit Ollama.

<https://www.centrон.de/tutorial/ollama-anleitung-llms-lokal-installieren-und-ausfuehren/>

Code anpassen

Je nach verwendetem KI-System, müssen die Codezeilen (für den Token und das Erstellen des Sprachmodells) angepasst werden.



Main.java



```
1 package de.gc.agent.db;
2
3 import java.util.Scanner;
4
5 import de.gc.agent.ki.model.XKiLogin;
6 import de.gc.agent.ki.model.XKiLogin.KiSystem;
7 import dev.langchain4j.memory.ChatMemory;
8 import dev.langchain4j.memory.chat.MessageWindowChatMemory;
9 import dev.langchain4j.model.chat.ChatModel;
10 import dev.langchain4j.service.AiServices;
11
12 /**
13 * Dies ist die Hauptklasse, die unsere Anwendung startet. Sie baut den
14 * "DB-Genius"-Agenten zusammen und startet die interaktive Konsole.
15 */
16 public class Main {
17
18     public static void main(final String[] args) {
19
20         try {
21
22             // =====
23             // 1. KONFIGURATION & SETUP
24             // =====
25
26             // Hier fügt jeder Schüler seinen eigenen, kostenlosen API-Schlüssel
27             // vom Google AI Studio ein oder einem anderem System.
28             //
29             // über die Umgebungsvariable:
30             // String token = System.getenv("GEMINI_API_KEY");
31             //
32             // Als String im Code - das ist nicht empfohlen
33             // String token = "DEIN_KOSTENLOSER_GEMINI_API_SCHLUESSEL_HIER";
34             //
35             // oder aus einer Datei, die wir hier angeben:
36             // z.B.
37             // ge_token=XXXGEHEIMXXX
38             // rest_token=YYYGEHEIMYYY
39             // siehe auch ki/model/XKiLogin.java
40
41             // -----
42             // GEMINI -> ANPASSEN
43             // final String token = XKiLogin.getToken(new
44             // File("/tmp/gi.properties"), "ge");
45             //
46             // // Wir erstellen das Sprachmodell - hier für Google Gemini.
47             // final ChatModel model = XKiLogin.createChatModel(KiSystem.GEMINI,
48             // "gemini-1.5-flash-latest", token, null);
49
50             // -----
51             // GITHUB -> ANPASSEN
52             // final String token = XKiLogin.getToken(new
53             // File("/tmp/gi.properties"), "rest");
54             //
55             // // Wir erstellen das Sprachmodell - hier für Github Copilot.
56             // final ChatModel model = XKiLogin.createChatModel(KiSystem.GITHUB,
57             // "GPT-4.1", token, null);
58
59             // -----
60             // OLLAMA -> ANPASSEN
61             // Wir erstellen das Sprachmodell - hier für Ollama - lokal.
62             //
63             final ChatModel model = XKiLogin.createChatModel(KiSystem.OLLAMA,
64                 "mistral:7b", null,
65                 "http://127.0.0.1:11434");
66
67             // Wir geben dem Agenten ein Kurzzeitgedächtnis für 10 Nachrichten.
68             final ChatMemory chatMemory = MessageWindowChatMemory
69                 .withMaxMessages(10);
70
71             // =====
72             // 2. DER ZUSAMMENBAU DES AGENTEN
73             // =====
74
75             // Mit 'AiServices.builder()' beginnt die "Montage" unseres Agenten.
```



```

76     final StudentAgent agent = AiServices.builder(StudentAgent.class)
77         .chatModel(model)
78         .tools(new DatabaseTools())
79         .chatMemory(chatMemory)
80         .build();
81
82     // =====
83     // 3. DIE INTERAKTIVE KONSOLE
84     // =====
85
86     System.out.println("""
87         Hallo! Ich bin DB-Genius.
88         Stell mir eine Frage über unsere Schüler-Datenbank.
89
90         Gib 'exit' ein, um das Programm zu beenden.
91         -----
92         """);
93
94     try (Scanner scanner = new Scanner(System.in)) {
95         while (true) {
96             System.out.print("Deine Frage: ");
97             final String userQuery = scanner.nextLine();
98
99             if ("exit".equalsIgnoreCase(userQuery)) {
100                 System.out.println("Auf Wiedersehen!");
101                 break;
102             }
103
104             // Wenn die Eingabe leer ist, überspringen wir die Verarbeitung.
105             // Das Framework wirft einen Fehler (IllegalArgumentException),
106             // wenn der Prompt leer ist.
107             if (userQuery.isBlank()) {
108                 continue;
109             }
110
111             final String agentResponse = agent.chat(userQuery);
112             System.out.println("\nDB-Genius: " + agentResponse);
113         }
114     }
115
116     } catch (final Exception e) {
117         System.err.println("Ein Fehler ist aufgetreten: " + e.getMessage());
118         e.printStackTrace();
119     }
120 }
121 }
```



3 Ausprobieren und Testen

Die Einstellungen sind so gewählt, dass hier zusätzliche Informationen ausgegeben werden, z. B. der erzeugte SQL-Befehl.

Test mit Gemini

```
Hallo! Ich bin DB-Genius.  
Stell mir eine Frage über unsere Schüler-Datenbank.  
  
Gib 'exit' ein, um das Programm zu beenden.  
  
Deine Frage: Wie viele Schüler hat die Schule?  
SQL query: SELECT COUNT(*) AS Anzahl FROM schueler  
  
DB-Genius: Die Schule hat 6 Schüler.  
  
Deine Frage: Gib mir die Vornamen der Schüler aus.  
SQL query: SELECT Vorname FROM schueler  
  
DB-Genius: Die Vornamen der Schüler lauten: Anna, Ben, Carla, David, Elena und Felix.  
  
Deine Frage: In welche Klasse ist Felix?  
SQL query: SELECT Klasse FROM schueler WHERE Vorname = 'Felix'  
  
DB-Genius: Felix besucht die Klasse FISI-24.  
  
Deine Frage: Wann ist er eingetreten?  
SQL query: SELECT Eintrittsdatum FROM schueler WHERE Vorname = 'Felix'  
  
DB-Genius: Felix ist am 01.09.2024 eingetreten.  
  
Deine Frage: exit  
Auf Wiedersehen!  
[Thread-0] INFO de.gc.agent.db.DbUtil - Database connection closed.
```

Beachte, bei der Frage „Wann ist er eingetreten?“ wird nicht angegeben, welcher Schüler gemeint ist. Die KI sucht dies aus dem Kontext heraus, da die letzten Antworten gespeichert worden sind.



4 Aufgaben

Bildet ein Team (2-3) und versucht eine oder mehrere Aufgaben zu lösen.

Schüler-Verwaltung mit natürlicher Sprache und Datenbankanbindung

Erweitert DB-Genius um die Möglichkeit, neue Schüler mit einem Befehl in natürlicher Sprache in die Datenbank einzufügen oder bestehende Datensätze zu ändern/löschen (z. B. „Füge Max Mustermann zur Klasse FIAE-23a hinzu!“ oder „Ändere die E-Mail von Anna Becker auf anna@beispiel.de“). Die KI muss die Absicht erkennen (CRUD) und in passende SQL-Statements übersetzen.

- **Ziel:**

Umgang mit Insert/Update/Delete, Fehlerbehandlung (z.B. „Name existiert nicht“), nochmal tieferes Verständnis für Datenintegrität.

- **Bonus:**

Bestätigung vor Datenänderung implementieren („Willst Du die Änderung wirklich durchführen?“).

Bücher-Ausleihe mit natürlicher Sprache und Datenbankanbindung

Programmiert einen smarten Agenten, der per natürlicher Sprache die Verwaltung einer kleinen Schulbibliothek übernimmt. Die Datenbasis ist eine Datenbank, in der Bücher, Ausleihende und Ausleihen gespeichert sind.

Beispiele für nutzernahe Sprachbefehle:

- „Leihe ‚Clean Code‘ an Max Müller aus!“
- „Welche Bücher sind aktuell ausgeliehen?“
- „Gib mir die Rückgabefrist für ‚Der Algorithmus der Menschlichkeit‘!“
- „Max Müller hat ‚Clean Code‘ zurückgegeben!“

Aufgabenstellung:

- Der Agent erkennt typischen Ausleih-/Rückgabe-Prozess und leitet daraus die passenden SQL-Operationen (INSERT, UPDATE, SELECT) ab.
- Die Dialogführung ist möglichst natürlich, und der Agent gibt verständliches Feedback (z.B. Hinweis, falls Buch schon ausgeliehen).
- Erweiterungsidee: Anzeige der Ausleihhistorie eines beliebigen Buchs.

Ziele:

- Arbeiten mit relationalen Daten (Bücher, Personen, Ausleihen)
- Umsetzung von Geschäftslogik (z. B. Disponibilität, Rückgabefristen)
- Starke Fokussierung auf Benutzerführung und verständliche Rückmeldungen



5 Voraussetzungs-Review

Hello zusammen!

Bevor ihr tief in die Entwicklung des KI-gestützten DB-Agenten eintaucht, wollen wir in diesem Kapitel sicherstellen, dass alle auf dem gleichen Stand sind. Wir wiederholen die wichtigsten Grundlagen, die ihr für dieses Projekt benötigt. Eure Aufgabe wird es sein, einen intelligenten Assistenten zu bauen, der normale Sprache in Datenbankabfragen übersetzt. Dafür sind Kenntnisse in Java, SQL, JDBC und im Umgang mit GitHub entscheidend.



generiert mit Perplexity



5.1 Grundlagen Java (OpenJDK 21)

Java ist die Programmiersprache, in der ihr den gesamten DB-Agenten schreiben werdet. Ein solides Grundwissen ist daher unerlässlich, um die bestehenden Klassen zu verstehen und neue Funktionalitäten zu implementieren.

5.1.1 Welche Themen solltet ihr auffrischen?

- **Klassen und Objekte:**

Das gesamte Projekt ist objektorientiert aufgebaut. Ihr solltet verstehen, wie Klassen (z. B. Main, DbUtil, StudentAgent) aufgebaut sind und wie Objekte dieser Klassen erzeugt und verwendet werden.

- **Methoden und Parameter:**

Ihr müsst verstehen, wie man Methoden aufruft und ihnen die richtigen Parameter übergibt, wie es z.B. in der Main-Klasse beim Aufruf von
`XKiLogin.createChatModel(...)`
geschieht.

- **Abhängigkeitsmanagement mit Maven:**

Das Projekt nutzt Maven, um externe Bibliotheken (Abhängigkeiten) zu verwalten. Werft einen Blick in die `pom.xml`, um zu sehen, welche Tools wie langchain4j oder der mariadb-java-client eingebunden werden. Ihr müsst verstehen, warum diese Datei die „einige Quelle der Wahrheit“ für die Bibliotheksversionen ist.

- **Exception Handling:**

Die `try-with-resources` und `try-catch`-Blöcke in den Klassen DatabaseTools und DbUtil sind entscheidend für eine robuste Fehlerbehandlung.

5.1.2 Hilfreiche Links zur Wiederholung

- **Java ist auch eine Insel (Online-Buch):**

Ein fantastisches, kostenloses Nachschlagewerk für alle Java-Themen.

<https://openbook.rheinwerk-verlag.de/javainsel/>

- **Maven in 5 Minuten (Anleitung):**

Eine offizielle, kurze Einführung in die Grundlagen von Maven.

<https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html>

- **Video-Tutorial zu Java-Grundlagen:**

Ein visueller Einstieg oder eine Auffrischung der Kernkonzepte.

https://www.youtube.com/watch?v=y_n1E1Q0r5I

5.2 Grundlagen SQL (mit MariaDB 11.3)

Euer DB-Agent wird Anfragen in natürlicher Sprache in SQL-Abfragen übersetzen. Daher müsst ihr die Sprache der Datenbank – SQL – verstehen, um zu wissen, was der Agent tun soll und um seine Ergebnisse überprüfen zu können.



5.2.1 Welche Themen solltet ihr auffrischen?

- **Die ‘SELECT’-Anweisung:**

Der Agent darf Daten nur lesen. Euer Fokus liegt also komplett auf SELECT. Ihr solltet wissen, wie man Spalten auswählt (SELECT Vorname, Nachname FROM ...) und alle Spalten abruft (SELECT * FROM ...).

- **Die ‘WHERE’-Klausel:**

Das Filtern von Daten ist eine der Hauptaufgaben. Wiederholt, wie man mit WHERE Bedingungen festlegt (z. B. WHERE Klasse = 'FIA12a').

- **Das Datenbankschema:**

Ihr müsst das Schema der Tabelle schueler (ID, Vorname, Nachname, Klasse, Eintrittsdatum) verinnerlichen. Nur mit diesem Wissen kann der Agent (und ihr) korrekte Abfragen formulieren.

5.2.2 Hilfreiche Links zur Wiederholung

- **Interaktives SQL-Tutorial:**

Hier könnt ihr SQL direkt im Browser ausprobieren und lernen.

<https://sqlbolt.com/>

- **MariaDB-Tutorial für Anfänger:**

Eine gute Einführung, die speziell auf MariaDB zugeschnitten ist.

[https://www.mariadbTutorial.com/](https://www.mariadbtutorial.com/)

- **Video-Tutorial zu SQL-Grundlagen:**

Eine visuelle Erklärung der wichtigsten SQL-Befehle.

https://www.youtube.com/watch?v=dp_d341tK78

5.3 Grundlagen JDBC

JDBC (Java Database Connectivity) ist die Brücke zwischen eurer Java-Anwendung und der MariaDB-Datenbank. Ohne JDBC könnte euer Agent nicht mit der Datenbank kommunizieren.

5.3.1 Welche Themen solltet ihr auffrischen?

- **Der Connection-Manager:**

Schaut euch die Klasse DbUtil an. Sie nutzt den java.sql.DriverManager, um die Verbindung herzustellen. Ihr solltet verstehen, warum die Verbindungsdaten (URL, User, Passwort) in die separate Datei db.properties ausgelagert sind.

- **Statement und ResultSet:**

In DatabaseTools seht ihr, wie ein Statement verwendet wird, um eine SQL-Abfrage auszuführen, und wie das Ergebnis als ResultSet zurückkommt. Versteht den grundlegenden Lebenszyklus: Verbindung aufbauen, Statement ausführen, Ergebnis verarbeiten, alles schließen.



5.3.2 Hilfreiche Links zur Wiederholung

- **Oracle JDBC-Grundlagen-Tutorial:**

Das offizielle Tutorial von Oracle, das die Basics sehr gut erklärt.

<https://docs.oracle.com/javase/tutorial/jdbc/basics/index.html>

- **JDBC-Tutorial von Baeldung:**

Eine sehr praxisnahe und gut erklärte Anleitung.

<https://www.baeldung.com/java-jdbc>

5.4 Zugriff auf das GitHub-Repository

Damit ihr nicht den gesamten Code abtippen müsst, liegt das Projekt in einem GitHub-Repository. Ihr müsst nur wissen, wie ihr an die Dateien kommt.

5.4.1 Was müsst ihr können?

Ihr müsst in der Lage sein, ein Repository von der GitHub-Weboberfläche herunterzuladen oder zu "klonen". Klonen ist der bevorzugte Weg, wenn ihr mit Git vertraut seid, aber für den Anfang reicht auch der Download als ZIP-Archiv.

5.4.2 Hilfreiche Links zur Wiederholung

- **GitHub-Dokumentation:**

Hilfe zu jeder Phase deiner GitHub-Reise.

<https://docs.github.com/de>

- **Video: Git & GitHub für Anfänger in 20 Minuten:**

Ein schneller und umfassender Überblick.

https://www.youtube.com/watch?v=P6jD53zt_yA

5.5 Debugging-Tipps: Euer mächtigstes Werkzeug

Der Code tut nicht, was er soll? Eine Fehlermeldung ist unverständlich? Willkommen im Entwickler-Alltag! Anstatt nur `System.out.println` an verschiedenen Stellen einzufügen, ist der Debugger euer bester Freund, um Fehler systematisch zu finden und die Logik des Programms zu verstehen.

5.5.1 Warum ist der Debugger hier so nützlich?

Mit einem Debugger könnt ihr die Ausführung eures Programms an einem bestimmten Punkt – einem sogenannten **Breakpoint** – anhalten. Dann könnt ihr euch den Zustand aller Variablen ansehen und den Code Zeile für Zeile weiter ausführen. Für euren DB-Agenten ist das Gold wert:

- **Die generierte SQL-Abfrage prüfen:**

Ihr könnt genau an der Stelle anhalten, an der der KI-Agent eine SQL-Abfrage an die Klasse `DatabaseTools` übergibt. So seht ihr sofort, ob der Agent eure Frage richtig verstanden und in korrektes SQL übersetzt hat.

- **Datenbank-Fehler verstehen:**

Wenn eine `SQLException` auftritt, könnt ihr euch die fehlerhafte SQL-Abfrage ansehen, die den Fehler verursacht hat. Oft sind es nur kleine Syntaxfehler.



- **Verbindungsprobleme analysieren:**

Lädt das Programm die `db.properties` korrekt? Sind Benutzername und Passwort richtig? Mit einem Breakpoint in der `DbUtil.getConnection()`-Methode könnt ihr das live überprüfen.

5.5.2 Ein praktisches Beispiel: Den Agenten beim Denken beobachten

Stellt euch vor, ihr fragt „Zeig mir alle Schüler aus der FIA12a“ und es kommt ein Fehler. So geht ihr vor:

1. **Breakpoint setzen:**

Öffnet die Datei `DatabaseTools.java` in eurer IDE (z. B. IntelliJ oder Eclipse). Klickt neben die Zeilennummer der ersten Zeile in der `executeQuery`-Methode. Dort sollte ein roter Punkt erscheinen – euer Breakpoint.

```
@Tool("Führt eine SQL-Abfrage in der Schüler-Datenbank aus...")  
public String executeQuery(final String sqlQuery) {  
    // <-- HIER DEN BREAKPOINT SETZEN  
    System.out.println(" SQL query: " + sqlQuery);  
    ...  
}
```

2. **Anwendung im Debug-Modus starten:**

Startet eure `Main.java` nicht mit „Run“, sondern mit „Debug“. Das Symbol dafür ist meist ein kleiner Käfer.

3. **Frage stellen:**

Gebt eure Frage in der Konsole ein. Sobald der Agent die SQL-Abfrage ausführen will, friert das Programm am Breakpoint ein und die IDE zeigt euch den Debugger-Tab.

4. **Variable inspizieren:**

Im Debugger-Fenster seht ihr nun alle Variablen, die an dieser Stelle bekannt sind. Sucht die Variable `sqlQuery` und schaut euch ihren Wert an. Vielleicht steht dort so etwas wie `SELECT * FROM schueler WHERE Klasse = FIA12a` und ihr erkennt sofort: Die Anführungszeichen um 'FIA12a' fehlen! Fehler gefunden.



6 Mit der KI lernen

Mit dem Informationsbeschaffungs-Prompt¹ könnt Ihr euch zusammen mit der KI in Themenbereiche einarbeiten.



☰ Was ist zu tun?

1. Wählt eine KI eures Vertrauens aus.^a
2. Kopiert den Informationsbeschaffungs-Prompt in die Prompt-Eingabe und führt diesen aus.

Vielen Dank für diese detaillierten Anweisungen. Ich habe die Rolle > und die Regeln für unsere Interaktion verstanden. Ich werde Ihnen > ausschließlich Informationen zu dem von Ihnen gewählten Thema liefern, > ohne bei der Erstellung von Artikeln zu helfen, und dabei die von > Ihnen festgelegte Struktur für "Ausgabe 1" und "Ausgabe 2" befolgen.

Ich bin bereit zu beginnen.

Bitte geben Sie das Thema (M) ein.

3. Aufgabenstellung definieren: Wählt das Thema aus. Je besser Ihr das Thema eingrenzt, um so besser weiß die KI, was Ihr lernen wollt.

Das Thema ist JDBC-Zugriff auf die SQL-Datenbank – mit MariaDB 11.3 > oder höher unter Verwendung von OpenJDK 21.

4. Prompt ausführen.

Danke. Das Thema ist "JDBC-Zugriff auf die SQL-Datenbank – mit MariaDB> 11.3 oder höher unter Verwendung von OpenJDK 21".

Welche Ausgabe möchten Sie erhalten? Bitte geben Sie die entsprechende> Nummer ein:

1: Grundinformationen
2: Spezialisierte Informationen

5. Jetzt seid Ihr dran...

1

6. Die Antwort...

Ausgabe 1: Grundinformationen

Einführung in den JDBC-Zugriff auf MariaDB mit OpenJDK 21

Der Zugriff auf eine MariaDB-Datenbank aus einer Java-Anwendung heraus ist ein fundamentaler Prozess für die moderne Softwareentwicklung. Dies wird durch eine standardisierte API namens JDBC (Java Database Connectivity) ermöglicht. In diesem speziellen Kontext betrachten wir die Synergie von drei modernen Technologien: **MariaDB 11.3+**, eine leistungsstarke, von der Community entwickelte relationale Datenbank; **OpenJDK 21**, die neueste Long-Term Support (LTS)-Version von Java, die innovative Funktionen wie virtuelle Threads einführt; und den **MariaDB Connector/J**, den spezialisierten JDBC-Treiber, der die Kommunikation zwischen Java-Anwendungen und der MariaDB-Datenbank

¹ Quelle: <https://www.ainauten.com/p/reddit-lernprompt-chatgpt-study-feature-ai-in-der-schule-ai-fun-beziehungen>



herstellt. Die Kombination dieser Komponenten ermöglicht es Entwicklern, robuste, performante und skalierbare datenbankgestützte Anwendungen zu erstellen, die von den neuesten Fortschritten in der Java- und Datenbanktechnologie profitieren.

Allgemeine Informationen

Um eine Verbindung herzustellen, benötigt eine Java-Anwendung, die auf OpenJDK 21 läuft, den passenden MariaDB JDBC-Treiber (Connector/J) in ihrem Klassenpfad. Dieser Treiber ist eine Typ-4-JDBC-Implementierung, was bedeutet, dass er eine reine Java-Bibliothek ist und direkt mit dem MariaDB-Netzwerkprotokoll kommuniziert, ohne auf native Bibliotheken oder einen zwischengeschalteten ODBC-Treiber angewiesen zu sein.

Der Prozess des Zugriffs lässt sich in mehrere grundlegende Schritte unterteilen:

- a) **Treiberbeschaffung:** Der MariaDB Connector/J muss als Abhängigkeit zum Projekt hinzugefügt werden. In modernen Java-Projekten geschieht dies typischerweise über ein Build-Management-Tool wie Maven oder Gradle, indem die entsprechende Koordinate (`org.mariadb.jdbc:mariadb-java-client`) in der Projektkonfigurationsdatei deklariert wird.
- b) **Treiber-Registrierung:** Früher war ein expliziter Aufruf von `Class.forName("org.mariadb.jdbc.Driver")` notwendig, um den Treiber im `DriverManager` zu registrieren. Dank des JDBC 4.0 Service Provider Mechanismus ist dies bei modernen Treibern und JDK-Versionen in der Regel nicht mehr erforderlich; der `DriverManager` findet und lädt den Treiber automatisch, solange er sich im Klassenpfad befindet.
- c) **Verbindungsaufbau:** Die Verbindung zur Datenbank wird über den `DriverManager` durch den Aufruf von `DriverManager.getConnection()` hergestellt. Diese Methode erfordert eine spezielle URL, den sogenannten „Connection String“, sowie Benutzername und Passwort.
- d) **Kommunikation:** Nach erfolgreichem Verbindungsaufbau gibt die Methode ein `Connection`-Objekt zurück. Über dieses Objekt können `Statement`-, `PreparedStatement`- oder `CallableStatement`-Objekte erzeugt werden, um SQL-Abfragen auszuführen, Daten zu bearbeiten (DML) und Datenstrukturen zu verwalten (DDL).
- e) **Ressourcen-Management:** Es ist entscheidend, alle Datenbankressourcen (`ResultSet`, `Statement`, `Connection`) nach ihrer Verwendung ordnungsgemäß zu schließen, um Ressourcenlecks zu vermeiden. Die `try-with-resources`-Anweisung von Java ist hierfür die empfohlene Vorgehensweise.

Wichtige Highlights und Punkte

- **Kompatibilität:** Der MariaDB Connector/J ist explizit mit Java 21 kompatibel. Die Release Notes, wie z.B. die für Version 3.3.1, bestätigen die Unterstützung für Java 8, 11, 17 und 21. Dies gewährleistet, dass Entwickler die neuesten Sprach- und Plattform-Features von OpenJDK 21 nutzen können.



- **Connection String:** Der Verbindungs-URL für MariaDB hat ein spezifisches Format: `jdbc:mariadb://<hostname>:<port>/<datenbankname>`. Wichtige Parameter wie `useSSL=true` können als Query-Parameter hinzugefügt werden, um eine sichere Verbindung zu erzwingen.
- **Treiberklassenname:** Der vollständig qualifizierte Name der Treiberklasse lautet `org.mariadb.jdbc.Driver`. Es ist wichtig, diesen nicht mit älteren oder von MySQL stammenden Treibern (z.B. `com.mysql.jdbc.Driver`) zu verwechseln.
- **OpenJDK 21 und Virtuelle Threads (Project Loom):** Eine der bedeutendsten Neuerungen in OpenJDK 21 sind virtuelle Threads. Diese ermöglichen eine massive Skalierung von nebenläufigen Anwendungen, da blockierende Operationen – wie das Warten auf eine Datenbankantwort – den zugrundeliegenden Betriebssystem-Thread nicht blockieren. JDBC-Operationen sind klassische blockierende Aufrufe. Die Verwendung des MariaDB Connector/J mit virtuellen Threads in OpenJDK 21 kann die Leistung und den Durchsatz von Anwendungen, die viele gleichzeitige Datenbankverbindungen verwalten, drastisch verbessern, ohne den Code grundlegend ändern zu müssen.
- **Sicherheit:** Nach der Installation von MariaDB wird dringend empfohlen, das Skript `mysql_secure_installation` auszuführen, um grundlegende Sicherheitsmaßnahmen wie das Festlegen eines Root-Passworts und das Entfernen von Testdatenbanken durchzuführen. Bei der Konfiguration der Verbindung sollten sichere Methoden zur Speicherung von Passwörtern verwendet werden, anstatt sie direkt im Code zu hinterlegen.
- **Abgrenzung zu MySQL:** Obwohl MariaDB als Fork von MySQL begann und eine hohe Kompatibilität aufweist, ist es wichtig, den dedizierten MariaDB-Treiber (`mariadb-java-client`) zu verwenden und nicht den MySQL-Treiber (Connector/J), um von MariaDB-spezifischen Funktionen und Optimierungen vollständig zu profitieren. Auch der URL-Präfix `jdbc:mariadb://` sollte anstelle von `jdbc:mysql://` verwendet werden.

Wenn Sie weitere Details zu diesen Grundinformationen wünschen, geben Sie bitte "more" ein.

Wenn Sie zu spezialisierten Informationen wechseln möchten, geben Sie bitte "2" ein.

7. und weiter...

a Mit ChatGPT, Gemini und Perplexity getestet.



Informationsbeschaffungs-Prompt

```
1 # *Informationsbeschaffungs-Prompt*
2
3 ---
4
5 ## *Prompt-Eingabe*
6
7 - Gib das Prompt-Thema ein = [.....]
8
9 - **Das eingegebene Thema ist eine Variable in geschweiften Klammern, die im gesamten Prompt als "M" bezeichnet wird.**
10
11 ---
12
13 ## *Prompt-Grundsätze*
14
15 - Ich bin Forscher und verfasse Artikel zu verschiedenen Themen.
16
17 - Du darfst mir **absolut nicht** dabei helfen, den Artikel zu entwerfen. (Wichtigster Punkt)
18
19 1. **Schlage mir niemals einen Artikel über "M" vor.**
20
21 2. **Gib mir keine Tipps, wie man einen Artikel über "M" entwirft.**
22
23 - Deine einzige Aufgabe ist es, mir Informationen über "M" zu liefern, damit **ich selbst** anhand dieser Informationen den Artikel erstellen kann.
24
25 - Im Abschnitt "Prompt-Ausgabe" werden verschiedene Ausgaben erstellt, die jeweils mit einer Nummer versehen sind, z. B. Ausgabe 1, Ausgabe 2 usw.
26
27 - **So funktionieren die Ausgaben:**
28
29 1. **Frage mich zu Beginn nach dem Absenden dieses Prompts, welche Ausgabe ich benötige.**
30
31 2. Ich tippe die Nummer der gewünschten Ausgabe, z. B. "1" oder "2" usw.
32
33 3. Du lieferst nur die Ausgabe mit genau dieser Nummer.
34
35 4. Nachdem die gewünschte Ausgabe geliefert wurde, erweitere dieselbe Art von nummerierter Ausgabe, wenn ich **"more"** tippe.
36
37 - Es spielt keine Rolle, welche Ausgabe du bereitstellst oder ob ich "more" tippe - in jedem Fall sollte deine Antwort **extrem detailliert** sein und **die maximal mögliche Zeichen- und Token-Anzahl** verwenden. (Äußerst wichtig)
38
39 - Vielen Dank für deine Kooperation, geschätzter Chatbot!
40
41 ---
42
43 ## *Prompt-Ausgabe*
44
45 ---
46
47 ### *Ausgabe 1*
48
49 - Diese Ausgabe heißt: **"Grundinformationen"**
50
51 - Enthält:
```



```

53 - Eine **Einführung** zu "M"
54
55 - **Allgemeine** Informationen über "M"
56
57 - **Wichtige** Highlights und Punkte zu "M"
58
59 - Wenn "2" getippt wird, gehe zur nächsten Ausgabe.
60
61 - Wenn "more" getippt wird, erweitere diese Art von Ausgabe.
62
63 ---
64
65 ### *Ausgabe 2*
66
67 - Diese Ausgabe heißt: **"Spezialisierte Informationen"**
68
69 - Enthält:
70
71 - Mehr akademische und spezialisierte Informationen
72
73 - Wenn das Prompt-Thema Charakterentwicklung ist:
74
75   - Für die Entwicklung von Fantasy-Charakteren: ausführlichere ↴
      Informationen wie Hardcore-Fanmeinungen, detaillierte ↴
      Hintergrundgeschichten und Spin-offs zur Figur.
76
77   - Für reale Personen: mehr persönliche Geschichten, Gewohnheiten, ↴
      Verhaltensweisen und detaillierte Informationen über die Person.
78
79 - So lieferst du die Ausgabe:
80
81 1. Zeige die verschiedenen Themen, die in den spezialisierten ↴
    Informationen über "M" behandelt werden, als Liste in Form eines ↴
    "Inhaltsverzeichnisses"; das sind die Anfangsthemen.
82
83 2. Schreibe darunter:
84
85   - "Welches Thema interessiert dich?"
86
87   - Wenn der Name des gewünschten Themas getippt wird, liefere ↴
      vollständige spezialisierte Informationen zu diesem Thema.
88
89   - "Wenn du mehr Themen zu 'M' brauchst, tippe bitte 'more'"
90
91   - Wenn "more" getippt wird, liefere zusätzliche Themen über die ↴
      ursprüngliche Liste hinaus. Wird nach der zweiten Runde erneut ↴
      "more" getippt, füge weitere Themen über die vorherigen beiden ↴
      Sets hinaus hinzu.
92
93   - Hinweis für dich: Versuche beim ersten Zusammenstellen der ↴
      Themen so viele relevante Themen wie möglich aufzunehmen, um den ↴
      Bedarf für diese Option zu minimieren.
94
95   - "Wenn du Unterthemen eines Themas brauchst, tippe bitte 'topics ↴
      ... (gewünschtes Thema)'."
96
97   - Wenn der genannte Text getippt wird, liefere die Unterthemen ↴
      (Sekundärthemen) der Anfangsthemen.
98
99   - Selbst wenn ich "topics ... (ein Sekundärthema)" tippe, liefere ↴
      trotzdem die Unterunterthemen dieser Sekundärthemen, die als ↴
      "Dritt niveau-Themen" bezeichnet werden können, und das kann ↴
      beliebig weitergehen.

```



100
101 - In jeder Ebenenphase (Anfangs-, Sekundär-, Dritt niveau- usw.) ↴
102 erweitert das Tippen von "more" immer die Themen auf derselben ↴
103 Ebene.
104
105 - **Zusammenfassung:**
106
107 - Wird nur der Themenname getippt, liefere spezialisierte ↴
108 Informationen im Format dieses Themas.
109
110 - Wird "topics ..." (ein anderes Thema) getippt, behandle die ↴
111 Unterthemen dieses Themas.
112
113 - Wird "more" nach einer Themenliste getippt, erweitere die Themen ↴
114 auf derselben Ebene.
115
116 - Wird "more" nach Informationen zu einem Thema getippt, gib weitere ↴
117 spezialisierte Informationen zu diesem Thema.
118
119 3. In jeder Phase: Wenn "1" getippt wird, verweise auf "Ausgabe 1".
120
121 - Beim Bereitstellen einer Themenliste auf irgendeiner Ebene ↴
122 erinnere mich daran, dass wir zu "Grundinformationen" zurückkehren, ↴
123 wenn ich einfach "1" tippe, und dass wir zum ersten Eintrag dieser ↴
124 Liste gehen, wenn ich "Option 1" tippe.
125
126 ---
127
128
129 --Ende==

