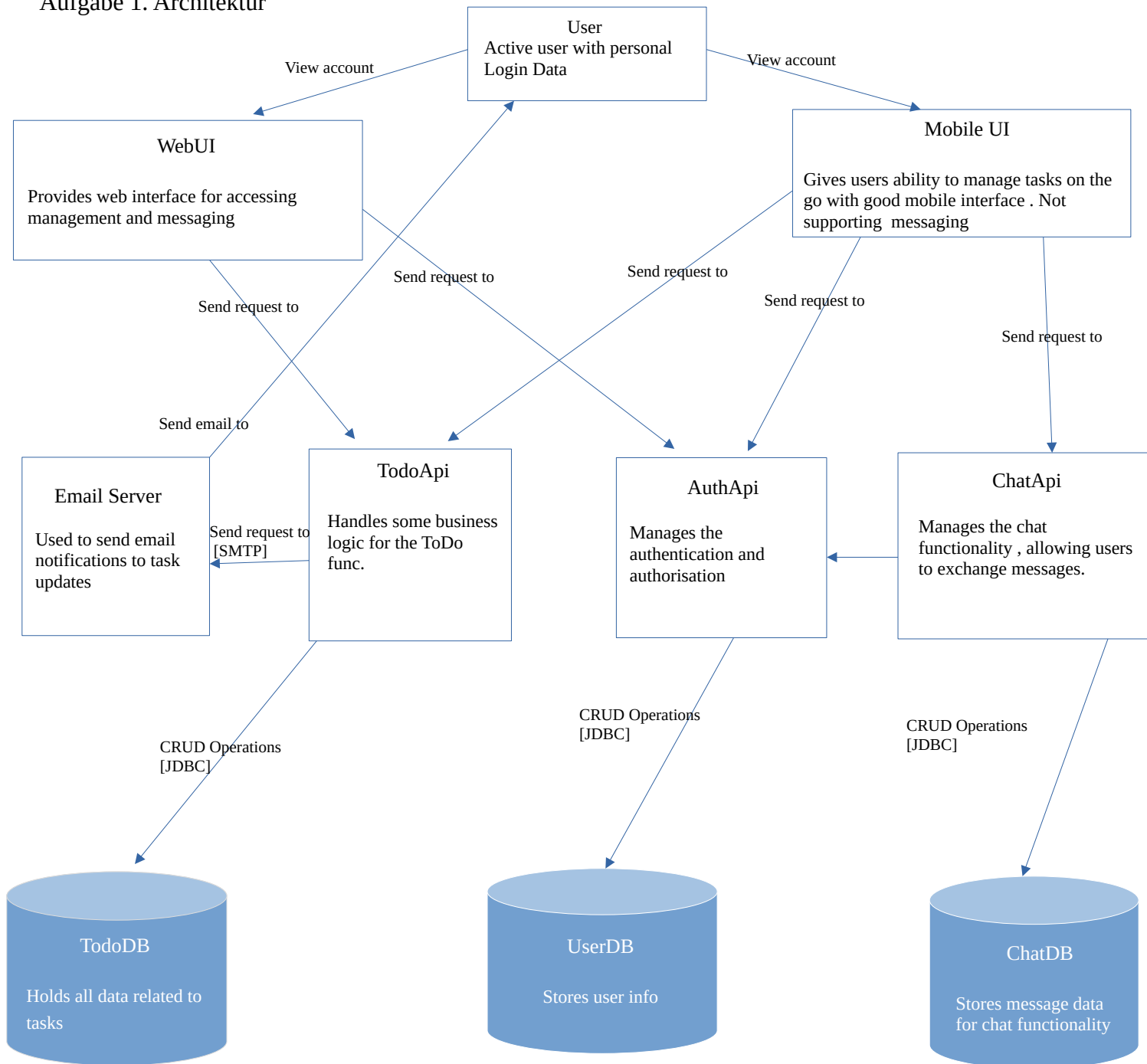


Aufgabe 1. Architektur



Aufgabe 2. SQL and JDBC

2.1

a) CREATE TABLE IF NOT EXISTS todos (
id INT PRIMARY KEY AUTO_INCREMENT NOT NULL,
title VARCHAR(100) NOT NULL,
description VARCHAR(500)
);

b) INSERT INTO todos (id, title, description)
VALUES (1, 'Some random title', 'Some random description');

c) *SQL- Befehl* : SELECT description
FROM todos
WHERE description LIKE '%Weihnacht%';

Ergebnisse :

Es ist nun endlich so weit! Mit dem 01. November wird es Zeit, zügig die Weihnachtsdekorationen auszupacken.

Bald sollte ich Weihnachtsplaetzchen backen.

2.2

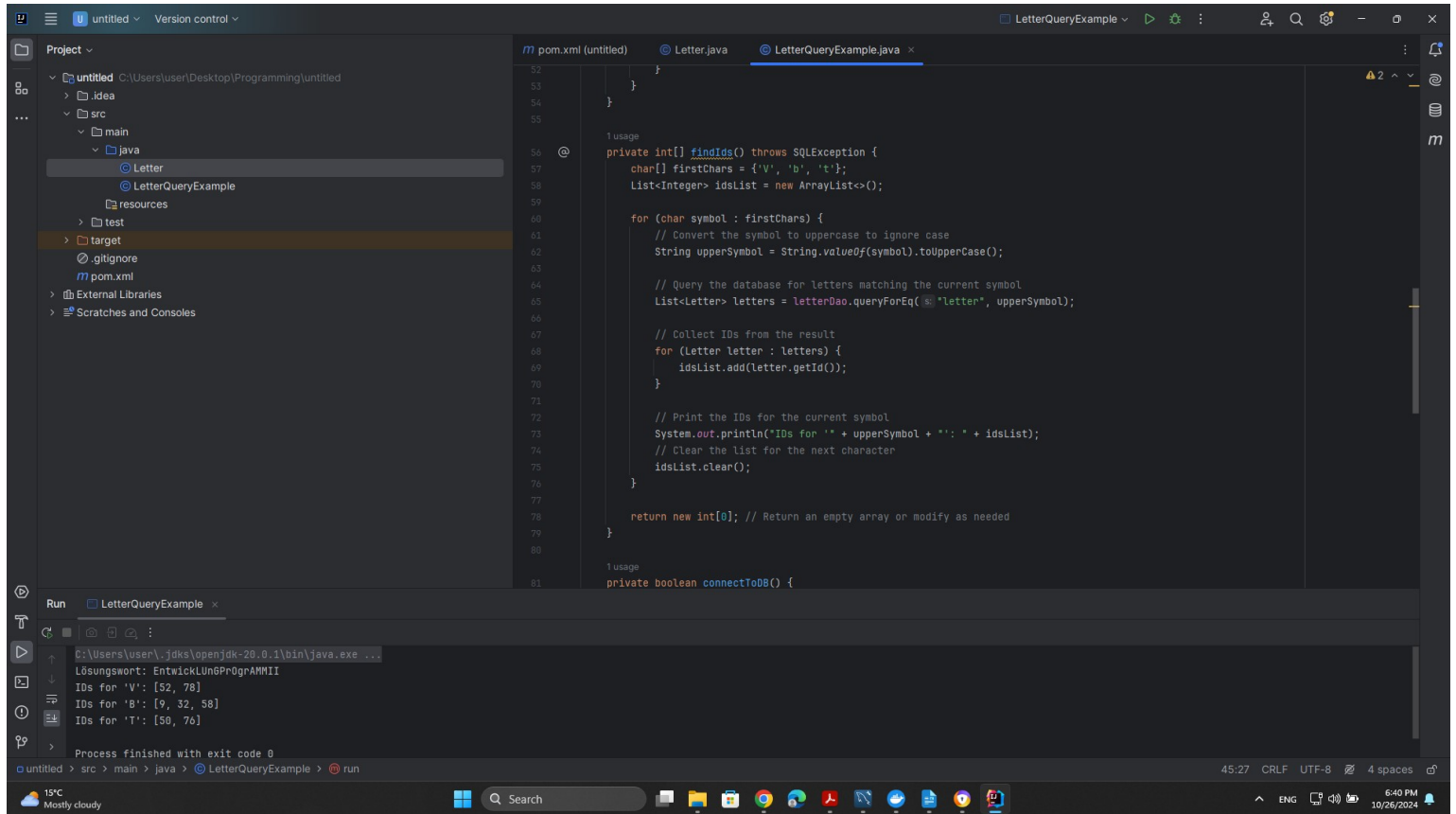
a) Lösungswort: EntwickLUnGPrOgrAMMII

```

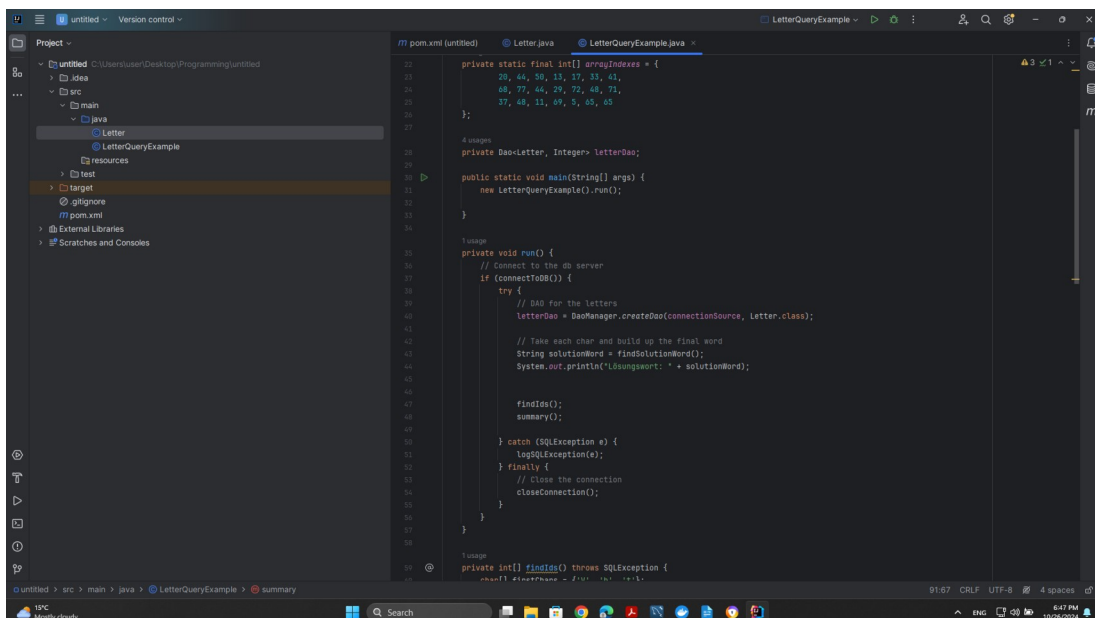
1 import java.sql.SQLException;
2 import java.util.logging.Level;
3 import java.util.logging.Logger;
4 import com.j256.ormlite.dao.Dao;
5 import com.j256.ormlite.dao.DaoManager;
6 import com.j256.ormlite.jdbc.JdbcConnectionSource;
7 import com.j256.ormlite.support.ConnectionSource;
8 import com.j256.ormlite.table.TableUtils;
9
10
11 public class LetterQueryExample {
12     private static final Logger LOGGER = Logger.getLogger(LetterQueryExample.class.getName());
13     private static final String DATABASE_URL = "jdbc:mariadb://bilbao.informatik.uni-stuttgart.de/pe2-db-a1";
14     private static final String DATABASE_USER = "pe2-nutzer";
15     private static final String DATABASE_PASSWORD = "esJLtFm6ksCT4mCy0S";
16
17     private ConnectionSource connectionSource;
18     private static final int[] arrayIndexes = {
19         20, 44, 58, 13, 17, 33, 41,
20         68, 77, 44, 29, 72, 48, 71,
21         37, 48, 11, 69, 5, 65, 65
22     };
23
24     private Dao<Letter, Integer> letterDao;
25
26     public static void main(String[] args) {
27         new LetterQueryExample().run();
28     }
29
30     private void run() {
31         // Connect to the db server
32         if (connectToDB()) {
33             try {
34                 // DAO for the letters
35                 letterDao = DaoManager.createDao(connectionSource, Letter.class);
36
37                 // Take each char and build up the final word
38                 String solutionWord = findSolutionWord();
39                 System.out.println("Lösungswort: " + solutionWord);
40
41             } catch (SQLException e) {
42                 logSQLException(e);
43             } finally {
44                 // Close the connection
45                 closeConnection();
46             }
47         }
48     }
49
50     private boolean connectToDB() {
51         try {
52             connectionSource = new JdbcConnectionSource(DATABASE_URL, DATABASE_USER, DATABASE_PASSWORD);
53             return true;
54         } catch (SQLException e) {
55             logSQLException(e);
56             return false;
57         }
58     }
59
60     private String findSolutionWord() throws SQLException {
61         StringBuilder word = new StringBuilder();
62         for (int id : arrayIndexes) {
63             Letter letter = letterDao.queryForId(id);
64             if (letter != null) {
65                 word.append(letter.getLetter());
66             }
67         }
68         return word.toString();
69     }
70
71     private void closeConnection() {
72         try {
73             if (connectionSource != null) {
74                 connectionSource.close();
75             }
76         } catch (Exception e) {
77             LOGGER.log(Level.SEVERE, "Fehler beim Schließen der Verbindung: " + e.getMessage());
78         }
79     }
80
81     private void logSQLException(SQLException e) {
82         LOGGER.log(Level.SEVERE, "SQL Fehlercode: " + e.getErrorCode());
83         LOGGER.log(Level.SEVERE, "SQL Fehlermeldung: " + e.getMessage());
84     }
85 }
86 }

```

b)
IDs for 'V': [52, 78]
IDs for 'B': [9, 32, 58]
IDs for 'T': [50, 76]



c)



```

pom.xml (untitled) | Letter.java | LetterQueryExample.java
// Collect IDs from the result
for (Letter letter : letters) {
    idsList.add(letter.getId());
}

// Print the IDs for the current symbol
System.out.println("IDs for " + upperSymbol + ": " + idsList);
// Clear the list for the next character
idsList.clear();
}

return new int[0]; // Return an empty array or modify as needed
}

1 usage
private void summary () throws SQLException {
    List<Letter> allLetters = letterDao.queryForAll();
    int size = allLetters.size();
    int sumOfIds = allLetters.stream()
        .map(Letter::getId).mapToInt(id -> id).sum();

    System.out.println("Summe = " + sumOfIds);
    System.out.println("Durchschnittswert = " + sumOfIds/size);
}

1 usage
private boolean connectToDB() {
    try {
        connectionSource = new JdbcConnectionSource(DATABASE_URL, DATABASE_USER, DATABASE_PASSWORD);
    } catch (SQLException e) {
        logSQLException(e);
        return false;
    }
}

1 usage
private String findSolutionWord() throws SQLException {
    StringBuilder word = new StringBuilder();
    for (int id : arrayIndexes) {
        Letter letter = letterDao.queryForId(id);
    }
}

```

Output : **Summe** = 4167 , **Durchschnittswert** = 50

Aufgabe 3. HTTP und REST

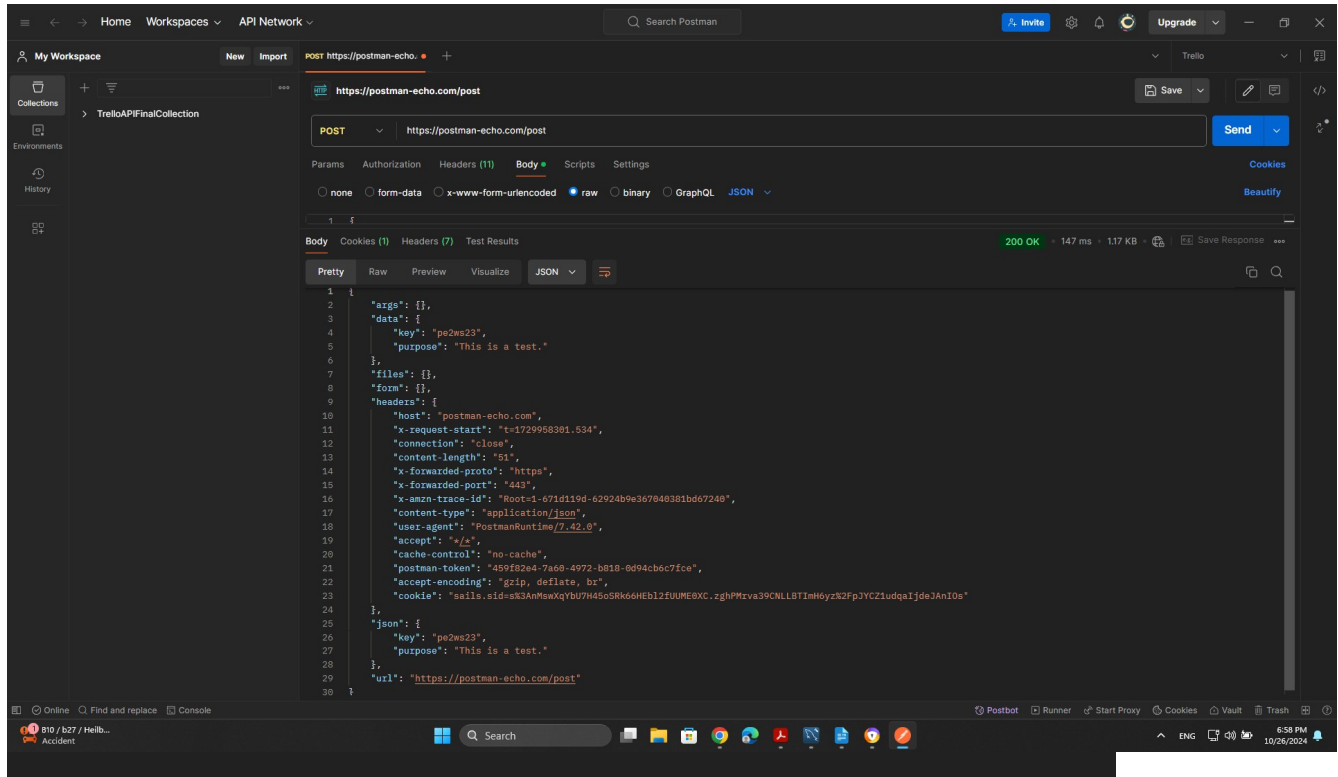
a) Wenn wir den GET-Request erfolgreich ausgeführt haben, erhalten wir ein JSON-Objekt mit verschiedenen Feldern, einschließlich des Witzes selbst.

```

GET https://api.chucknorris.io/jokes/random?category=history
200 OK - 105 ms - 1.3 KB
{
  "categories": [
    "history"
  ],
  "created_at": "2020-01-05 13:42:19.576075",
  "icon_url": "https://api.chucknorris.io/img/avatars/chuck-norris.png",
  "id": "9cvedgqgqamony3nngba",
  "updated_at": "2020-01-05 13:42:19.576075",
  "url": "https://api.chucknorris.io/jokes/9cvedgqgqamony3nngba",
  "value": "In the words of Julius Caesar, 'Veni, Vidi, Vici, Chuck Norris'. Translation: I came, I saw, and I was roundhouse-kicked inthe face by Chuck Norris."
}

```

b) Wir erhalten folgendes body :



c)

1. Abruf aller DVDs

- HTTP- Method : GET
- Pfad : /dvds/all
- Beschreibung : Diese Ressource ermöglicht das Abrufen aller verfügbaren DVDs. Optional können Filterparameter angesetzt werden (Kategorie-String , Titel, Restriciton). Dann sehen z.B die Queries so aus : *GET /dvds/all?category=action* , *GET /dvds/all?title=Adventure* , *GET /dvds?isAgeRestricted=true*

2. Abrufen einer DVD nach ID

- HTTP- Method : GET
- Pfad : /dvds/{id}
- Beschreibung: Diese Ressource ermöglicht das Abrufen einer spezifischen DVD anhand ihrer ID.

3. Erstellen einer neuen DVD via JSON body

- HTTP-Method : POST
- Pfad : /dvds/post
- Beschreibung : Diese Ressource ermöglicht das Erstellen einer neuen DVD. Der Request-Body enthält die Details der DVD (Titel, Kategorie, Altersfreigabe).

4. Löschen einer DVD

- HTTP- Method : DELETE
- Pfad : /dvds/{id}
- Beschreibung : das Löschen einer DVD anhand ihrer ID . Man kann auch ein Query-Parameter *keepInArchive=true* hinzugefügt werden, um eine Archivkopie der DVD zu behalten.

5. Aktualisieren einer DVD

- HTTP-Method : PUT
- Pfad : /dvds/id
- Beschreibung: Dadurch kann man die Info des existierenden Ressource aktualisieren anhand der ID.
- Der Request-Body enthält die aktualisierten DVD-Daten.