FLASHCARD DESKTOP APP

INDIVIDUELLE ABSCHLUSSARBEIT BLJ

PrimeSoft Group
Bahnhofstrasse 4
CH-8360 Eschlikon

Git-Repository: Individuelle Abschlussarbeit BLJ brejan

Bretscher Jan
jb@letsbuild.ch

04.06.2025 Abgabedatum: 27.06.25 Version: 1

Inhalt

1 Einleitung

1.1 Änderungstabelle

Datum	Was?	Version
04.06.25	Git-Repository erstellt Git-Project mit Milestones & Issues erstellt Projekt bestätigen lassen Dokumentation erstellt Einleitung der Dokumentation geschrieben	1
05.06.25	Datenbank aufgesetzt Testdaten eingefügt Im Hauptteil den Abschnitt Datenbank abgeschlossen	1.1
06.06.25		2
11.06.25		
12.06.25		
13.06.25		
18.06.25		
19.06.25		
20.06.25		
25.06.25		
26.06.25		
27.06.25		

1.2 Aufgabenstellung

Das Ziel von der individuellen Abschlussarbeit des BLJ's ist es, dass man dem Lehrbetrieb und den Coaches zeigen kann, was man im Basislehrjahr gelernt hat. Es ist empfohlen, dass man ein Thema oder Themenbereich bearbeitet, welcher einem im Lehrbetrieb später auch erwartet. Die PrimeSoft arbeitet viel mit Microsoft zusammen, weshalb ich später viel mit C# machen werde. Es macht deshalb nur Sinn, dass ich eine Windows Applikation schreiben werde, welche ich mit C# umsetzen werde.

1.3 Projektbeschreibung

Mein Projekt ist eine Windows Desktop-Applikation: eine Flashcard-App. Die Grundidee ist ähnlich wie bei Quizlet: Man kann verschiedene Decks erstellen und darin Karten speichern, um diese zu lernen. Zudem kann man eigene Benutzerkonten anlegen, sodass jeder Nutzer nur auf seine eigenen Decks zugreifen kann.

Als Backend dient eine relationale Datenbank auf einem MySQL-Server, den mein Vater hostet. Ich habe Administratorzugriff auf diesen Server, was mir ermöglicht, das Projekt selbstständig umzusetzen.

Wir haben im ZLI bereits an einer mobilen Version einer Flashcard-App gearbeitet. Mein Ziel ist es, diese später mit derselben Datenbank zu verknüpfen, um die Decks sowohl auf dem Laptop als auch auf dem Handy lernen zu können. Dies ist jedoch ein optionales Ziel, da es wahrscheinlich den zeitlichen Rahmen sprengen würde.

Der Fokus meines Projekts liegt grösstenteils auf der Datenbank, der Datenbankverbindung aus der Applikation und den SQL-Abfragen. Ein weiterer Schwerpunkt ist die gesamte WPF-Applikation (Windows Presentation Foundation Application), die ich mit C# und dem .NET-Framework umsetze.

Da ich später bei meinem Betrieb grösstenteils mit C# arbeiten werde, ist dieses Projekt ideal, um praktische Erfahrungen zu sammeln. Ich habe bisher wenig bis keine Erfahrung mit C#, daher kann ich viel Neues lernen. Im ZLI haben wir relationale Datenbanken nicht sehr detailliert behandelt, weshalb ich umso interessierter bin, mein Wissen in diesem Bereich zu vertiefen. Ich finde Datenbanken einen sehr interessanten und wichtigen Fachbereich der Informatik und schätze die Möglichkeit, mein Wissen darin erweitern zu können.

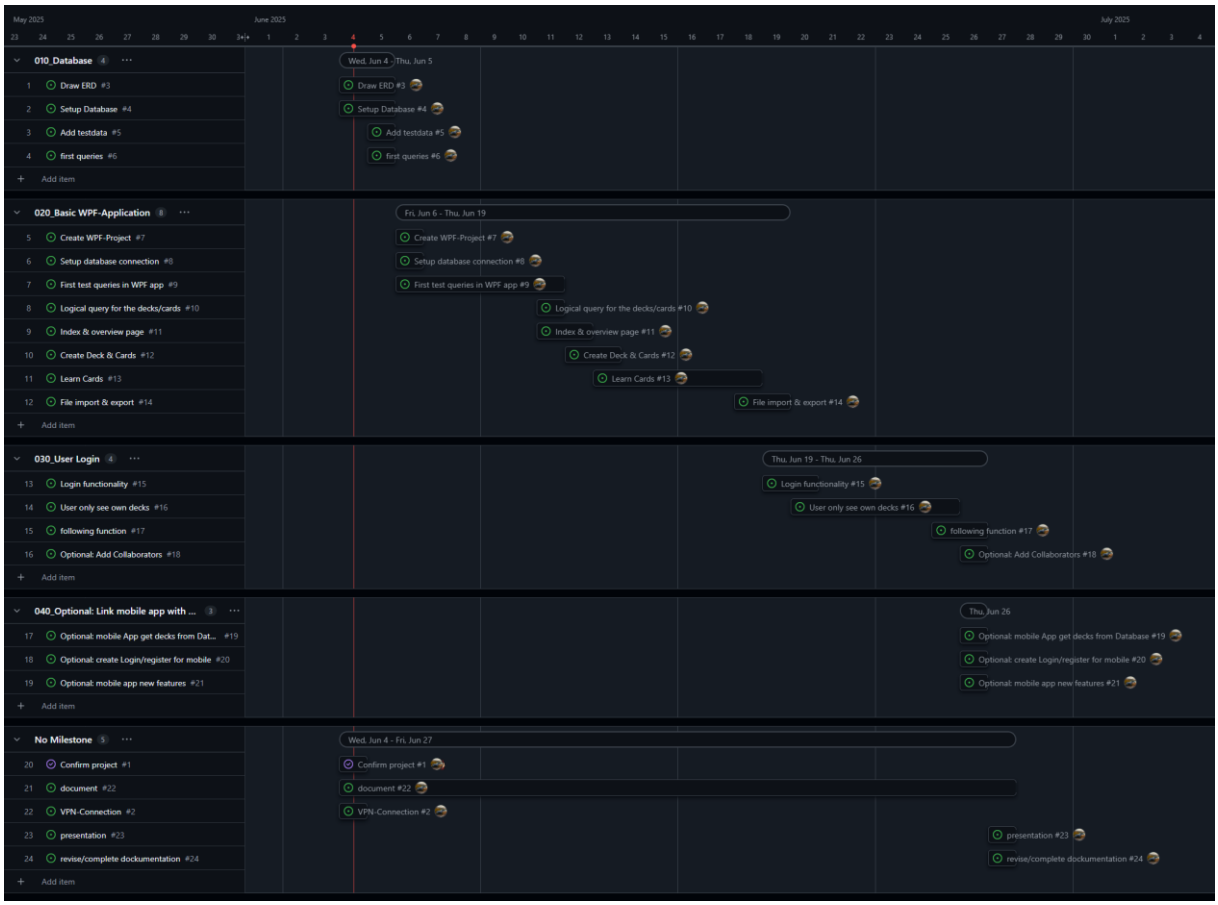
1.4 Risiken & Bedenken

Auf den MySQL-Server kann ich nur vom gleichen Netzwerk zugreifen, indem auch der Server ist. Ich muss jedoch vom ZLI ausarbeiten, was bedeutet, dass ich entweder eine VPN-Verbindung ins lokale Netzwerk zu Hause herstellen muss oder ich muss alles, was ich im ZLI mache in einem Lokalen Docker-Container umsetzen und danach zu Hause auf den Richtigen Server übertragen muss. Ich würde die erste Variante bevorzugen, weil ich so flexibler und einfacher arbeiten könnte. Ich bin jedoch froh, dass es sicher einen Plan B gibt, auch wenn dieser etwas umständlicher ist.

Was ich auch noch nicht genau weiss, wie ich es umsetzen kann ist die Verbindung zur Datenbank aus der WPF-Applikation heraus. Ich habe beim Frühlingsferienprojekt bereits eine WPF-Applikation erstellen und mit C# arbeiten dürfen. Ich arbeite einfach nach meinem Zeitplan und schaue vorzu, wie ich vorankomme. Ich bin jedoch sehr zuversichtlich, was das Projekt und die Datenbankverbindung angehen.

2 Planung

2.1 Terminplan



Roadmap Flashcard App

Dies ist der geplante Zeitplan für das Projekt. Ich habe drei Milestones hinzugefügt, welche ich sicher erreichen möchte/werde. Der Vierte habe ich als optional drin, weil ich denke, dass es zeitlich nicht ganz für diesen reichen wird. Die drei Milestones beinhalten einige Issues, welche ich zu erledigen habe. Ich habe pro Issue zwischen ¼ und ¾ Tag gerechnet.

Der **erste Milestone** dreht sich um die **Datenbank**. Ich werde zuerst ein ERD-Zeichnen, welches ich danach 1:1 übernehmen kann und die Datenbank damit umsetzen. In der Datenbank werden alle Decks und Karten verwaltet und die Userprofile. Wenn ich die Datenbank umgesetzt habe, werde ich einige Testdaten einfügen und grundlegende SQL-Abfragen durführen, um zu testen, ob alles wie erwartet funktioniert. Diesen Milestone möchte ich am zweiten Tag bereits erledigt haben.

Der **zweite Milestone** ist die **Grundsätzliche WPF-Applikation**. Dies ist zum einen ein WPF-Projekt zu erstellen, zum einen auch bereits in C# zu coden. Ich habe für diesen Milestone am meisten Issues erstellt, und auch am meisten Zeit eingerechnet. Ich habe mir hierfür 6 Tage Zeit gegeben. Mit ‘Grundsätzliche WPF-Applikation’ ist gemeint, dass man die App theoretisch bereits verwenden könnte. Die Grundfunktionen sind:

- Alle Kartenstapel anzeigen
- In Kartenstapeln suchen
- Alle Karten/Quiz anzeigen
- In Karten/Quiz suchen

- Kartenstapel erstellen
- Kartenstapel bearbeiten
- Karten/Quiz erstellen
- Karten/Quiz bearbeiten
- Karten lernen
- Lieblingskarten markieren
- Karten als JSON oder CSV exportieren
- Kartenstapel als JSON oder CSV exportieren
- Karten als JSON oder CSV importieren
- Kartenstapel als JSON oder CSV importieren

Der **dritte Milestone** ist die **Login-Funktion**. Hier werde ich die Benutzerverwaltung hinzufügen. Als Benutzer kann man dann sich anmelden oder registrieren, sodass man danach nur noch auf seine eigenen Decks zugriff hat. Eine weitere Funktion wird Folgen sein. In diesem Milestone werde ich die follow-Funktion hinzufügen, dass verschiedene User einander folgen können und auch zusammen als Gemeinschaftswerk an einem Deck arbeiten können. Diesen Milestone möchte ich in 4 Tagen erledigt haben. Ich denke, dass dies auch eher sportlich und wahrscheinlich knapp machbar sein wird.

Als Optionaler Milestone habe ich mir gesetzt, dass ich bei meiner Mobile Applikation auch ein Login erstelle, sodass man Geräte übergreifend auf seine Decks zugreifen kann. Ich bin mir jedoch ziemlich sicher, dass dies leider zeitlich nicht mehr drin liegt.

2.2 Entscheidungsmatrix

2.2.1 Technologie

KRITERIUM	C#/WPF/.NET	JAVA/JAVAFX	ELECTRON/JS	PYTHON/PYQT
WINDOWS-INTEGRATION	5/5	3/5	2/5	3/5
GUI-KOMFORT & LOOK	4/5	3/5	4/5	3/5
LERNAUFWAND	3/5	3/5	4/5	4/5
PERFORMANCE	5/5	4/5	2/5	3/5
COMMUNITY/SUPPORT	5/5	4/5	4/5	3/5
LOKALE DATENHALTUNG	5/5	4/5	3/5	4/5
BENUTZERVERWALTUNG	5/5	4/5	4/5	3/5
PERSÖNLICHER NUTZEN	5/5	3/5	4/5	4/5
ZUSAMMEN	37/40	28/40	27/40	27/40

- C# mit WPF ist ideal für eine Windows-native App.
- .NET bietet umfassende Bibliotheken für Benutzerverwaltung, Dateispeicherung, UI-Design und Sicherheit.
- WPF (Windows Presentation Foundation) ist etabliert und leistungsfähig für moderne UIs.
- Perfekte Integration in das Windows-Ökosystem (Look & Feel, Registry, Dateisystem).

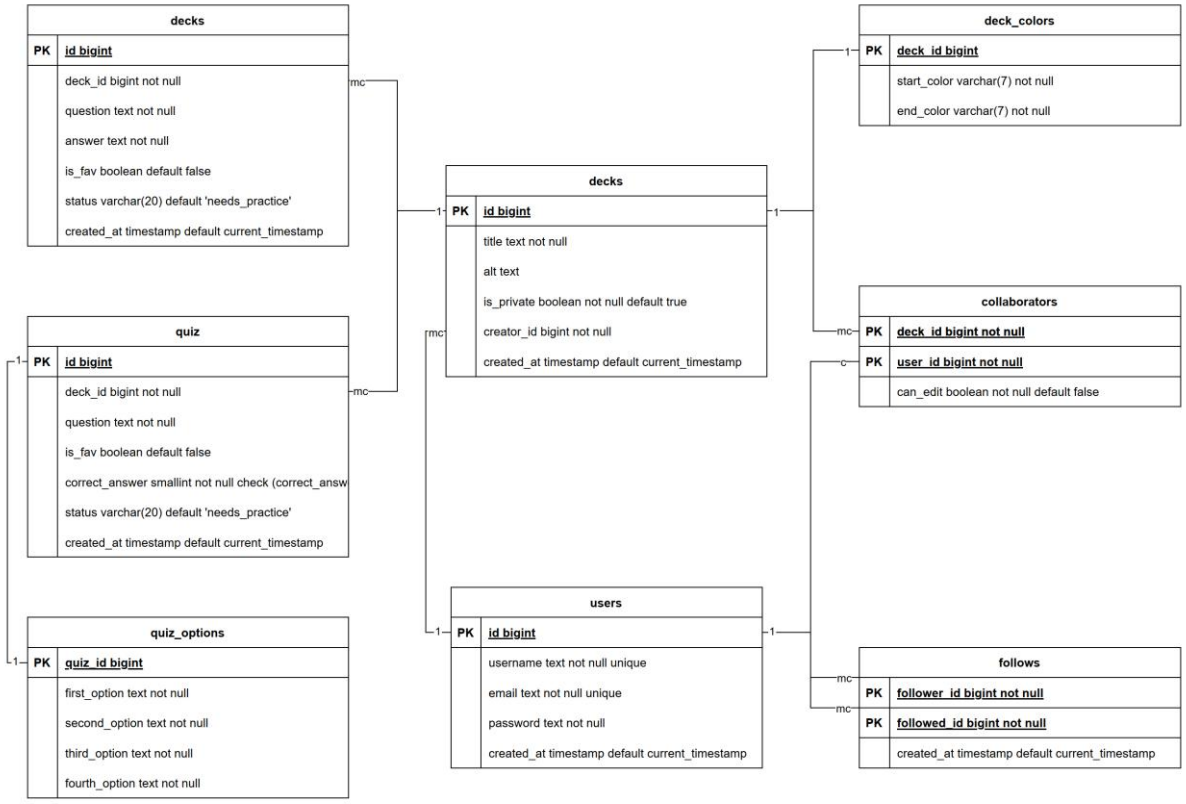
2.3 Datenbank

KRITERIUM	NOSQL DATENBANK	RELATIONALE DATENBAK
DATENSTRUKTUR	3/5	5/5
SKALIERBARKEIT	4/5	2/5
DATENÜBERTRAGUNGS SICHERHEIT	2/5	5/5
ABFRAGEKOMPLEXITÄT	2/5	5/5
ENTWICKLUNGSKOMFORT	4/5	4/5
ZUSAMMEN	15/25	21/25

3 Hauptteil

3.1 Datenbank

3.1.1 ERD und Struktur



ERD der FlashCard Datenbank

Dieses ERD beschreibt die Struktur einer Datenbank für eine Lernkarten-Anwendung. Die Kernentitäten sind **users** (Benutzerverwaltung) und **decks** (Lernkartenverwaltung). Ein Benutzer kann Decks erstellen, entweder alleine oder in Zusammenarbeit mit anderen Nutzern (**collaborators**). Jedes Deck enthält Karten, die entweder als einfache Frage-Antwort-Paare (**cards**) oder als Multiple-Choice-Quiz (**quiz** mit **quiz_options**) gestaltet sein können. Zusätzlich können Benutzer einander folgen (**follows**), und Decks haben individuelle Farben (**deck_colors**).

3.1.1.1 Überlegungen zu den Entitäten & Attributen

Ein User muss sich mit einem username, einer email und einem passwort registrieren. Diese Daten werden im entsprechenden Datensatz gespeichert. Die id und der Zeitstempel (created_at) werden dabei automatisch bei der Registrierung gesetzt.

Ein User kann beliebig vielen anderen Nutzern folgen. Das läuft über die follows-Tabelle, in der jeweils die follower_id und followed_id eingetragen werden.

Die Hauptfunktion für Users ist das Erstellen von Decks. Dabei gibt es zwei Möglichkeiten: Entweder man erstellt ein Deck alleine oder gemeinsam mit einem anderen User. Wenn man jemanden zur Zusammenarbeit einlädt, wird in der collaborators-Tabelle die deck_id des entsprechenden Decks sowie die user_id des eingeladenen Users gespeichert. Ob ein Deck Mitarbeitende hat oder nicht, erkennt man

daran, ob es einen passenden Eintrag in der collaborators-Tabelle gibt. Wenn nicht, ist es einfach ein privates Deck ohne Mitwirkende.

Ein Deck besitzt verschiedene Attribute: title, alt (als Alternativtext), creator_id, ein card_type, ein is_private-Schalter (ob das Deck privat ist) sowie ein created_at-Zeitstempel. Das Attribut is_private habe ich bisher nicht aktiv eingeplant – es ist daher standardmässig auf true gesetzt. Es könnte aber gut für spätere Erweiterungen genutzt werden.

Jedes Deck hat ausserdem zwei Farben (Start- und Endfarbe), die in der Tabelle deck_colors gespeichert sind. Diese Farben werden über die deck_id dem jeweiligen Deck zugeordnet.

Innerhalb eines Decks befinden sich Karten – entweder als klassische Frage-Antwort-Karten (cards) oder als Multiple-Choice-Quizfragen (quiz). Der Unterschied liegt in der Struktur: Bei cards gibt es genau eine Frage und eine Antwort. Bei einem quiz gibt es eine Frage mit vier Antwortmöglichkeiten und einem Index, der die korrekte Antwort angibt. Die vier Optionen habe ich bewusst in eine eigene Tabelle (quiz_options) ausgelagert, um keine Array- oder Listenstruktur im quiz selbst zu verwenden.

3.1.1.2 Die drei Normalformen

1NF (Erste Normalform):

- Alle Attribute enthalten atomare Werte (keine Listen/Mehrfachwerte)
- Keine sich wiederholenden Gruppen

2NF (Zweite Normalform):

- Alle Teile eines Eintrags, die nicht der Schlüssel sind, müssen vom *ganzen* Schlüssel abhängen und nicht nur von einem Teil davon.

3NF (Dritte Normalform):

- Keine Information darf von einer anderen Nicht-Schlüssel-Information abhängen; alles hängt direkt vom Hauptschlüssel ab.

3.1.1.3 Meine Umsetzung der Normalformen

1NF:

Es gibt keine wiederholten Gruppen von Attributen innerhalb einer Tabelle. Ich habe z.B. die Tabelle quiz_options erstellt, dass ich kein Array aus den Optionen in quiz habe. Dasselbe gilt bei den Farben (deck_colors).

2NF:

Die 2. Normalform ist nur relevant, wenn eine Tabelle einen zusammengesetzten Primärschlüssel hat. Z.B. hat die Tabelle follows einen zusammengesetzten Primärschlüssel aus follower_id und followed_id. Das Attribut created_at hängt hier von beiden Attribuen ab.

Die meisten anderen Tabellen haben einen normalen int als PK und betreffen deshalb diese Normalform nicht.

3NF:

Die transitiven Abhängigkeiten werden vermieden durch z.B. eine separate Tabelle deck_colors statt der Farben direkt in decks zu schreiben.

Jedes Nicht-Schlüsselattribut bezieht sich direkt auf den Primärschlüssel!

3.1.2 Datenbank aufsetzen

Nachdem ich das ERD gezeichnet habe, schrieb ich einen passenden SQL-Code, welcher die Datenbank 1:1 so umsetzte. Dieser Teil war eine reine Fliessarbeit, weil ich mir die gesamte Struktur, die Beziehungen und Flags beim ERD bereits überlegt habe.

```
create table decks (
  id bigint primary key,
  title varchar(55) not null,
  alt varchar(55),
  card_type smallint not null,
  is_private boolean not null default true,
  creator_id bigint not null,
  created_at timestamp default current_timestamp,
  foreign key (creator_id) references users (id) on delete cascade
);
```

Hier ist ein kurzes Snippet aus dem Code. Die Tabelle decks wird erstellt mit allen Attributen. In der zweitletzten Zeile wird ein Fremdschlüssel gesetzt, welcher auf den User verweist, der das Deck erstellt hat.

In der CMD konnte ich mit `mysql -u jan -h herkules.net.letsbuild.ch -p --ssl-verify-server-cert=FALSE` mit dem MySQL-Server verbinden. Dies funktioniert jedoch nur, wenn ich im lokalen Netzwerk bin (gleiches wie der Server). Hierfür musste ich meinen Laptop über eine VPN-Verbindung im Netzwerk zu Hause verbinden. Dies funktionierte einwandfrei. Zuerst musste ich mit

```
CREATE DATABASE FlashCards;
```

eine neue Datenbank erstellen und danach mit

```
USE FlashCards;
```

diese DB auch verwenden. Danach konnte ich mein gesamtes SQL-Script einfügen.

3.1.3 Testdaten einfügen

Ich liess mit AI Testdaten in Form von SQL-Coder erstellen und konnte diese auch einfach einfügen.

```
INSERT INTO cards (id, deck_id, question, answer, is_fav, status) VALUES
(1001, 101, 'Hello', 'Hallo', FALSE, 'learned'),
(1005, 102, 'Fläche eines Kreises?', 'pi * r^2', TRUE, 'learned'),
(1009, 105, 'Être', 'Sein', FALSE, 'learned'),
(1010, 106, 'H2O', 'Wasser', FALSE, 'learned'),
(1011, 108, 'Was ist ein "loop"?', 'eine Wiederholung', TRUE, 'learned'),
(1012, 101, 'Water', 'Wasser', FALSE, 'needs_practice');
```

Hier ist ein einfaches INSERT INTO Beispiel für die Tabelle ‘cards’. Ich hatte hier das Problem, dass Gemini zuerst den einen Datensatz so einfügen wollte:

```
(1011, 108, 'Was ist ein 'loop'?', 'eine Wiederholung', TRUE, 'learned'),
```

Das Problem sind die Singlequotes ('). Umgekehrt ist das Problem, wenn ich aussen Doublequotes (") habe, dann darf ich innen nur Singelquotes haben. Mein Fazit also:

Es müssen innen und aussen unterschiedliche Anführungszeichen sein.

```
-- Funktioniert nicht!
(1011, 108, 'Was ist ein 'loop'?', 'eine Wiederholung', TRUE, 'learned')
(1011, 108, "Was ist ein "loop"?", "eine Wiederholung", TRUE, "learned")
-- Funktioniert
(1011, 108, 'Was ist ein "loop"?', 'eine Wiederholung', TRUE, 'learned')
(1011, 108, "Was ist ein 'loop'? ", "eine Wiederholung", TRUE, "learned")
```

Ich denke, dass die einzige Lösung ist, dass ich Strings in Doublequotes schreibe und man dann nur einfache Anführungszeichen verwenden darf. Ansonsten lief alles reibungslos ab und die Datenbank ist nun mit Testdaten befüllt.

3.1.4 CRUD-Operationen & Joins

Ich habe einige testabfragen gemacht, um die Funktionalität der Datenbank zu testen. Die Abfragen, die ich gemacht habe, kann ich wahrscheinlich später in meiner Applikation gut verwenden.

Wichtig: die 'id' ist in den meisten Fällen Auto Increment.

```
-- Benutzer erstellen
INSERT INTO users (username, email, password)
VALUES ('max_mustermann', 'max@example.com', 'secure123');

-- Benutzer Login
SELECT id, username FROM users
WHERE email = 'max@example.com' AND password = 'secure123';

-- Benutzer aktualisieren
UPDATE users
SET username = 'new_username', email = 'new@example.com'
WHERE id = 1;

-- Benutzer löschen
DELETE FROM users WHERE id = 1;
```

Hier sind userinteraktionen gemacht. Mit INSERT INTO kann man einen neuen User erstellen. Hier werden diese Daten in die Datenbank hinzugefügt.

Fürs Login kann man grundsätzlich beim User den benutzername und das Passwort abfragen und mitgeben. Somit kann man nicht nur wie hier die id und den username zurück geben, sondern auch die Decks.

Mit UPDATE kann ein User-Datensatz aktualisiert werden. Hier wird z.B. der bestehende username durch new_username und die email durch new@example.com erstetzt beim Datensatz mit der id 1.

DELETE löscht den benutzer ganz normal aus der Datenbank.

```
-- Deck erstellen
INSERT INTO decks (title, alt, is_private, creator_id)
VALUES ('Mathe Basics', 'Deck für Grundrechenarten', false, 1);

-- Farben hinzufügen
INSERT INTO deck_colors (deck_id, start_color, end_color)
VALUES (
  (SELECT id FROM decks ORDER BY id DESC LIMIT 1),
  '#FF0000',
  '#00FF00'
);
```

Das Deck erstellen läuft gleich ab. Beim hinzufügen der Farben ist es jedoch etwas komplizierter. In der Datenbank ist das Attribut 'deck_id' als Primary Key eingetragen. Dies bedeutet, dass dieser Wert Unique sein muss. Dies ist auch gut so, weil ein Deck nur zu genau einer Farbkombination gehören kann. Hier ist es also vorteilhaft, mit einer Subquery zu arbeiten. Die Subquery selektiert die 'id' des letzten Datensatzes aus 'decks'. Wenn ich also ein neues der erstelle, bekommt es eine id, welche vom Auto Increment immer die höchste sein muss. Wenn ich dann gleich eine Farbe hinzufügen möchte, dann wird automatisch die letzt Zahl ausgesucht.

```
-- alle Decks eines Benutzers anzeigen
SELECT d.id, d.title, d.alt, dc.start_color, c.end_color
FROM decks d
LEFT JOIN deck_colors c ON d.id = dc.deck_id
WHERE d.creator_id = 1 OR d.id IN (
    SELECT deck_id FROM collaborators WHERE user_id = 1
);

-- Deck aktualisieren
UPDATE decks
SET title = 'Updated Title', is_private = true
WHERE id = 1;

-- Deck löschen
DELETE FROM decks WHERE id = 1;
```

Bei der ersten Abfrage, werden alle Decks eines Benutzers mit id, titel, alternativ text und den beiden Fraben dargestellt. Es wird nach der id in deck.creator_id und in collaborators.user_id gesucht. Somit werden auch diese Decks angezeigt, die man nicht selbst erstellt hat, sonder bei denen man 'nur' Mitarbeiter ist.

Das Deck aktualisieren und löschen funktioniert grundsätzlich gleich wie bei den Usern.

```
-- Karte hinzufügen
INSERT INTO cards (deck_id, question, answer, is_fav)
VALUES ((SELECT id FROM decks ORDER BY id DESC LIMIT 1), 'Was ist 2+2?',
'4', true);

-- Quiz hinzufügen
-- Quiz-Frage erstellen
INSERT INTO quiz (deck_id, question, correct_answer)
VALUES ((SELECT id FROM decks ORDER BY id DESC LIMIT 1), 'Was ist die
Hauptstadt von Deutschland?', 1);
-- Optionen hinzufügen
INSERT INTO quiz_options (quiz_id, first_option, second_option,
third_option, fourth_option)
VALUES ((SELECT id FROM quiz ORDER BY id DESC LIMIT 1), 'Berlin',
'München', 'Hamburg', 'Köln');
```

Die Karte fügt man gleich wie wie ein User hinzu.

Das Quiz ist interessanter aber auch sehr ähnlich, wie decks und deren Farben. Hier ist 'quiz_id' auch der Primary Key. Es darf also auch wieder nur ein quiz_options zu einem quiz gehören. Dies ist eigentlich das Selbe, wie den decks.

```
-- alle Karten eines Decks anzeigen
-- Einfache Karten
SELECT id, question, answer, is_fav FROM cards WHERE deck_id = 1;

-- Quizze
SELECT q.id, q.question, qo.first_option, qo.second_option,
qo.third_option, qo.fourth_option, q.correct_answer
FROM quiz q
JOIN quiz_options qo ON q.id = qo.quiz_id
WHERE q.deck_id = 1;
```

Hier werden einzeln von Type cards und quiz die zum deck 1 gehören dargestellt, mit Frage und Antwort/en. Die Abfrage so ist noch ziemlich simpel. Das Ziel später ist dann jedoch dies zu kombinieren und alle Karten, egal ob card oder quiz, in einer Tabelle darzustellen.

```
-- Favoriten markieren
UPDATE cards SET is_fav = true WHERE id = 1;

-- karte/quiz löschen
DELETE FROM cards WHERE id = 1;
DELETE FROM quiz WHERE id = 1;
```

Dies ist wieder weniger interessant. Hier werden cards/quiz geändert und gelöscht.

Wenn man einen Datensatz aus quiz löschen möchte, muss man gut aufpassen. Quiz hängt über einen FK mit quiz_options zusammen. Wenn man nun zuerst den Datensatz in quiz löscht, kommt ein Fehler. Theoretisch müsste man also immer zuerst den betreffenden Datensatz in quiz_options löschen, sodass der Foreign Key gelöscht wird, und erst danach in quiz löschen.

Bei dieser Datenbank ist dies jedoch kein Problem, denn ich habe beim Installieren der Datenbank mit folgendem erweitert:

```
FOREIGN KEY (quiz_id) REFERENCES quiz (id) ON DELETE CASCADE
```

Das 'ON DELETE CASCADE' macht, dass wenn eine Komponente der Verbindung (hier: entweder 'quiz_id' oder 'id') gelöscht wird, wird das Gegenstück auch gelöscht.

```
-- mitarbeiter hinzufügen
INSERT INTO collaborators (deck_id, user_id, can_edit)
VALUES (1, 2, true);

-- Folgen
INSERT INTO follows (follower_id, followed_id)
VALUES (1, 2);
```

Beim Mitarbeiter hinzufügen und Folgen muss man etwas beachten. Hier setzt sich der Primary Key aus den Values von deck_id und user_id oder follower_id und followed_id. Das heisst es darf kein Datensatz bereits vorhanden sein, bei dem deck_id = 1 und user_id = 2 ist. Das selbe gilt für die Followers.

```
-- alle mitarbeiter eines Decks anzeigen
SELECT
    collaborator.username AS username_collaborator,
    d.title AS deck_title,
    creator.username AS username_creator
FROM collaborators c
JOIN users collaborator ON c.user_id = collaborator.id
JOIN decks d ON c.deck_id = d.id
JOIN users creator ON d.creator_id = creator.id
WHERE c.deck_id = 1;
```

Hier wird der Benutzername der Collaborators, der Decktitel und der Benutzername des erstellers vom deck mit der id = 1 angezeigt.

Mit JOIN users collaborator und JOIN users creator kann man Fehler vermeiden, da die Namen gut gewählt sind und klar welcher user es ist. Unübersichtlicher wäre es gewesen, wenn man es zweimal mit JOIN users … geschrieben hätte.

```
-- Anzahl Follower pro Benutzer
SELECT u.username, COUNT(f.follower_id) AS follower_count
FROM users u
LEFT JOIN follows f ON u.id = f.followed_id;
```

Diese SQL-Abfrage gruppiert die Datensätze der follows-Tabelle nach dem Benutzer, dem gefolgt wird (followed_id). Für jeden dieser Benutzer wird dann die Anzahl der Zeilen (d.h. die Anzahl der Follower) ermittelt und als Ergebnis zurückgegeben. Das Ergebnis ist eine Liste von Benutzern und der jeweiligen Anzahl ihrer Follower

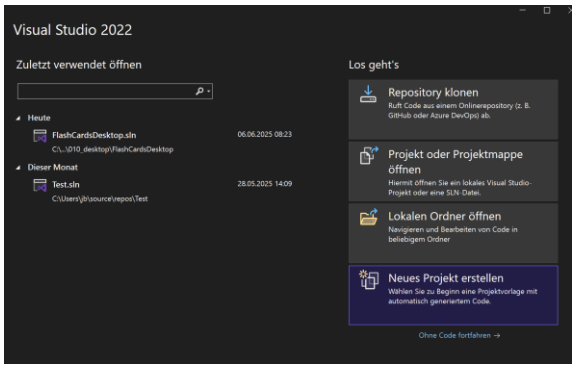
3.2 WPF-Applikation umsetzen

3.2.1 Projekt aufsetzen & Codeumgebung

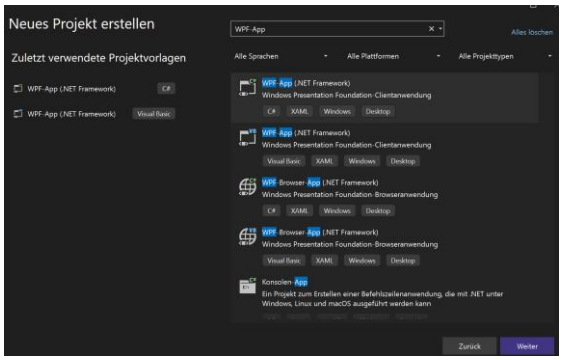
Ich werde dieses Projekt mit Visual Studio 2022 umsetzen. Folgend habe ich ein paar wichtige unterschiede gegenüber VS Code aufgelistet.

Kriterium	VS Code	Visual Studio 2022
Plattform	Windows, macOS, Linux	Nur Windows
Entwicklungsumgebung	Editor	Komplette IDE
Sprachen	Viele, per Erweiterung	Fokus auf C#, C++, .NET
Zielgruppe	Webdevs, DevOps	.NET-/Windows-Stack
Projekttypen	Web, Skripte, leichtgewichtig	Desktop, Mobile, Enterprise

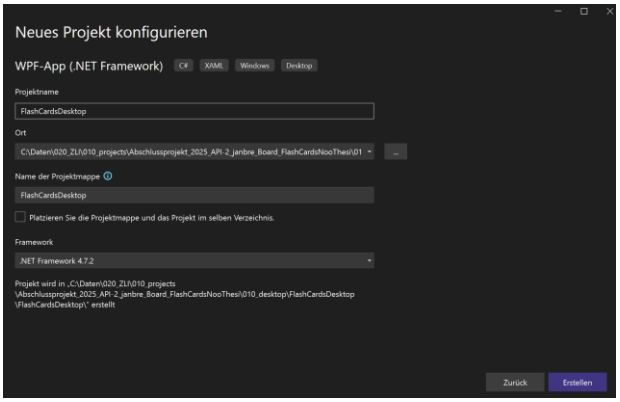
Da ich eine Windowsapplikation mit .Net, WPF und C# umsetzen möchte, eignet sich Visual Studio 2022 am besten geeignet.



Neues Projekt wird erstellt



Die passende Projekt Vorlage auswählen



Titel und Speicherort für Projektmappe

3.2.2 Verbindung mit der Datenbank

4 Arbeitsjournal

4.1 Tag 1 - 04.06.25

Wir starteten heute mit der Individuellen Abschlussarbeit des BLJ's. Jörg führte uns gut und ausführlich in dieses Thema ein und erklärte uns die wichtigsten Details. Danach konnte ich direkt mit der Planung beginnen, weil ich mir voraus bereits überlegt habe, was ich machen möchte. Ich überlegte mir auch schon vor dem Beginn des Projektes, wie ich das ganze Projekt umsetzen kann. Somit konnte ich zielstrebig meine Milestones und Issues Definieren. Nach dem Mittag wurde mein Projekt freigegeben und ich konnte starten. Ich fokussierte mich heute auf die Dokumentation. Ich habe ein Dokument erstellt, mit den nötigen Untertiteln befüllt und alles beschrieben, was ich bis jetzt konnte. Dies sind die gesamte Einleitung und Planung. Weiter habe ich ein erster Entwurf von einem ERD für die Datenbank gezeichnet. Ich finde es ziemlich gut und denke, dass es meinen Anforderungen an die Datenbank entspricht. Der MySQL-Server, der als Host meiner Datenbank dient, befindet sich bei mir zu Hause. Um darauf zugreifen zu können, muss ich mich normalerweise im lokalen Netzwerk befinden. Dieses Problem konnte ich mit einer VPN-Verbindung umgehen, um auch vom ZLI aus darauf zugreifen zu können. Obwohl die VPN-Verbindung einwandfrei funktionierte, konnte ich zunächst nicht auf den MySQL-Server zugreifen. Das Problem lag an den Firewall-Regeln des MySQL-Servers, die das Gateway von OpenVPN (dem VPN-Server-Dienst) nicht zuliessen. Ich musste daher eine neue Regel hinzufügen, um den Zugriff auf die Datenbank auch über den VPN-Server zu ermöglichen.

4.2 Tag 2 - 05.06.25

Ich startete heute mit dem Aufsetzen der Datenbank. Das ERD, welches ich gestern gezeichnet habe, habe ich heute noch vervollständigt und dann begonnen einen passenden SQL-Code dazu zu schreiben. Den SQL-Code zu schreiben war nicht sehr anspruchsvoll, da ich alle Details der Relationen, Entitäten, Attribute und Flags bereits gestern im ERD umgesetzt und mir überlegt habe. Die einzige Schwierigkeit war die Syntax. Ich konnte den SQL-Code problemlos auf den MySQL-Server laden und meine Datenbank so aufsetzen. Mit AI habe ich dann Testdatensätze erstellt, um nun alles gut testen zu können. Weiter habe ich einige nützliche CRUD-Operationen mit der Datenbank getestet, damit ich sicherstellen konnte, dass meine Überlegungen funktionieren. Alles hat so funktioniert, wie ich es erwartet und geplant habe. Ich habe nun auch bereits einige nützliche SQL-Queries, welche ich danach für die Applikation richtig verwenden kann. All diese Schritte habe ich auch sauber dokumentiert. Ich bin nun mit meiner Dokumentation auf gleichem stand, wie ich auch beim Projekt weit bin. Dies möchte ich übers ganze Projekt hindurch so beibehalten. Ich bin sehr erleichtert, dass die Datenbank im Backend so unkompliziert aufgesetzt werden konnte und ich mit dieser nun gut weiterarbeiten kann. Es hat mich auch erleichtert, dass die VPN-Verbindung nun auch einwandfrei funktioniert und ich aus dem ZLI aus mit der richtigen Datenbank arbeiten kann.

4.3 Tag 3 - 06.06.25

Ich habe zuerst die Falsche Projektvorlage gewählt. Ich habe 'WPF-App (.Net Framework) Visual Basic' gewählt. Ich möchte mein Projekt jedoch nicht mit Visual Basic, sondern mit C# umsetzen. Ich habe den Fehler jedoch schnell bemerkt, weil es .vb und .cs Dateien waren. Ich muss mich nun noch etwas an diese Entwicklungsumgebung gewöhnen, weil ich bisher immer mit VS Code gearbeitet habe. Dies ist jedoch gut so, weil ich später im Betrieb dann auch mit Visual Studio 2022 arbeiten werde.

Das Hauptziel, welches ich mir heute gestellt habe, konnte ich leider nicht erreichen. Ich wollte eine Verbindung zur Datenbank aus dem C#-Projekt herstellten.

4.4 Tag 4 - 11.06.25

4.5 Tag 5 - 12.06.25

4.6 Tag 6 - 13.06.25

4.7 Tag 7 - 18.06.25

4.8 Tag 8 - 19.06.25

4.9 Tag 9 - 20.06.25

4.10 Tag 10 - 25.06.25

4.11 Tag 11 - 26.06.25

4.12 Tag 12 - 27.06.25

5 Anhang

5.1 Quellenangabe

Gemini

Deepseek

W3Schools

5.2 Tools

MySQL

Visual Studio Code

Visual Studio 2022

Microsoft Word

Notepad++

Microsoft PowerPoint

DrawIO

OpenVPN

C#/WPF/.Net

5.3 Checkliste