Git-Repository: [Individuelle Abschlussarbeit BLJ brejan](https://github.com/bretscherjan/Abschlussprojekt_2025_API-2_janbre_Board_FlashCardsNooThesi)

Bretscher Jan

jb@letsbuild.ch

PrimeSoft Group

Bahnhofstrasse 4

CH-8360 Eschlikon

Flashcard desktop app

Individuelle Abschlussarbeit BLJ

Inhalt

[1 Einleitung 2](#_Toc200694248)

[1.1 Änderungstabelle 2](#_Toc200694249)

[1.2 Aufgabenstellung 2](#_Toc200694250)

[1.3 Projektbeschreibung 3](#_Toc200694251)

[1.4 Risiken & Bedenken 3](#_Toc200694252)

[2 Planung 4](#_Toc200694253)

[2.1 Terminplan 4](#_Toc200694254)

[2.2 Entscheidungsmatrix 5](#_Toc200694255)

[**2.2.1** **Technologie** 5](#_Toc200694256)

[2.3 Datenbank 6](#_Toc200694257)

[3 Hauptteil 7](#_Toc200694258)

[3.1 Datenbank 7](#_Toc200694259)

[3.1.1 ERD und Struktur 7](#_Toc200694260)

[3.1.2 Datenbank aufsetzen 9](#_Toc200694261)

[3.1.3 Testdaten einfügen 9](#_Toc200694262)

[3.1.4 CRUD-Operationen & Joins 10](#_Toc200694263)

[3.2 WPF-Applikation und Serverbackend umsetzen 13](#_Toc200694264)

[3.2.1 Projekt aufsetzen & Codeumgebung 13](#_Toc200694265)

[3.2.2 Falscher Plan zur Verbindung mit der Datenbank 14](#_Toc200694266)

[3.2.3 Lösung, um mit Datenbank zu verbinden 14](#_Toc200694267)

[3.2.4 Codeerklärung Prototyp mit Authentifizierung 15](#_Toc200694268)

[3.2.5 Decks und Cards anzeigen V2.1 20](#_Toc200694269)

[3.2.6 Codeerklärung Decks und Cards anzeigen V2 (Final-Version) 20](#_Toc200694270)

[4 Arbeitsjournal 24](#_Toc200694271)

[4.1 Tag 1 - 04.06.25 24](#_Toc200694272)

[4.2 Tag 2 - 05.06.25 24](#_Toc200694273)

[4.3 Tag 3 - 06.06.25 24](#_Toc200694274)

[4.4 Tag 4 - 11.06.25 25](#_Toc200694275)

[4.5 Tag 5 - 12.06.25 25](#_Toc200694276)

[4.6 Tag 6 - 13.06.25 25](#_Toc200694277)

[4.7 Tag 7 - 18.06.25 26](#_Toc200694278)

[4.8 Tag 8 - 19.06.25 26](#_Toc200694279)

[4.9 Tag 9 - 20.06.25 26](#_Toc200694280)

[4.10 Tag 10 - 25.06.25 26](#_Toc200694281)

[4.11 Tag 11 - 26.06.25 26](#_Toc200694282)

[4.12 Tag 12 - 27.06.25 26](#_Toc200694283)

[5 Anhang 27](#_Toc200694284)

[5.1 Quellenangabe 27](#_Toc200694285)

[5.2 Tools 27](#_Toc200694286)

[5.3 Checkliste 27](#_Toc200694287)

# Einleitung

## 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 | C#/WPF/.Net-Projekt aufgesetzt  Fehlüberlegung dokumentiert | 1.3 |
| 11.06.25 | Sicherheitsschema gezeichnet  Datenbankverbindung hergestellt  Code und Sicherheitskonzept dokumentiert | 2 |
| 12.06.25 | Decks anzeigen  Cards anzeigen  Strukturänderung MainWindow.xaml 🡪 Index.xaml  Programmierschritte dokumentiert | 2.1 |
| 13.06.25 | Decks hinzufügen  Programmierschritte dokumentiert | 2.2 |
| 18.06.25 |  |  |
| 19.06.25 |  |  |
| 20.06.25 |  |  |
| 25.06.25 |  |  |
| 26.06.25 |  |  |
| 27.06.25 |  |  |

## 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.

## 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.

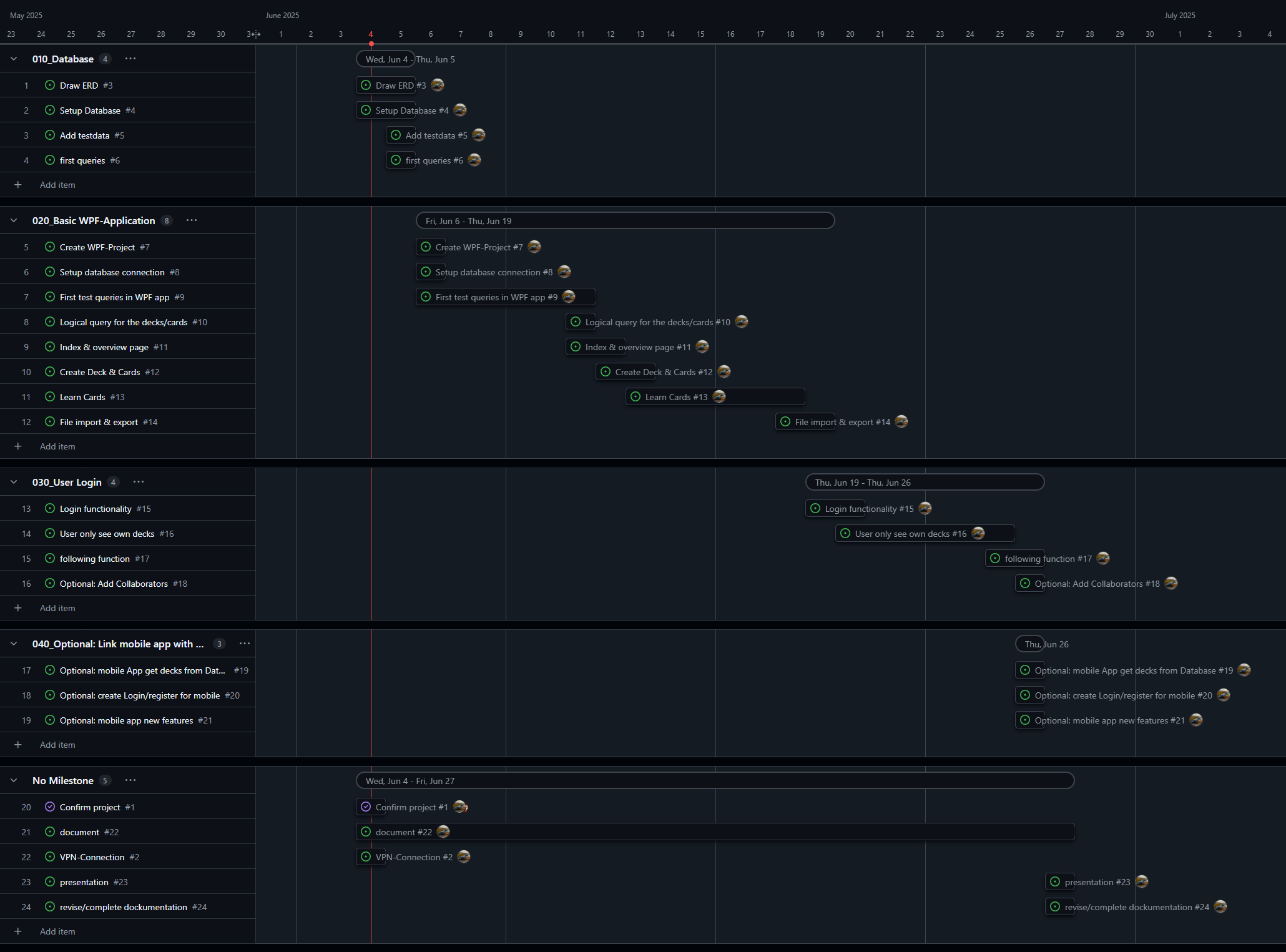
## Risiken & Bedenken

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.

# Planung

## 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.

## Entscheidungsmatrix

### **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).

## Datenbank

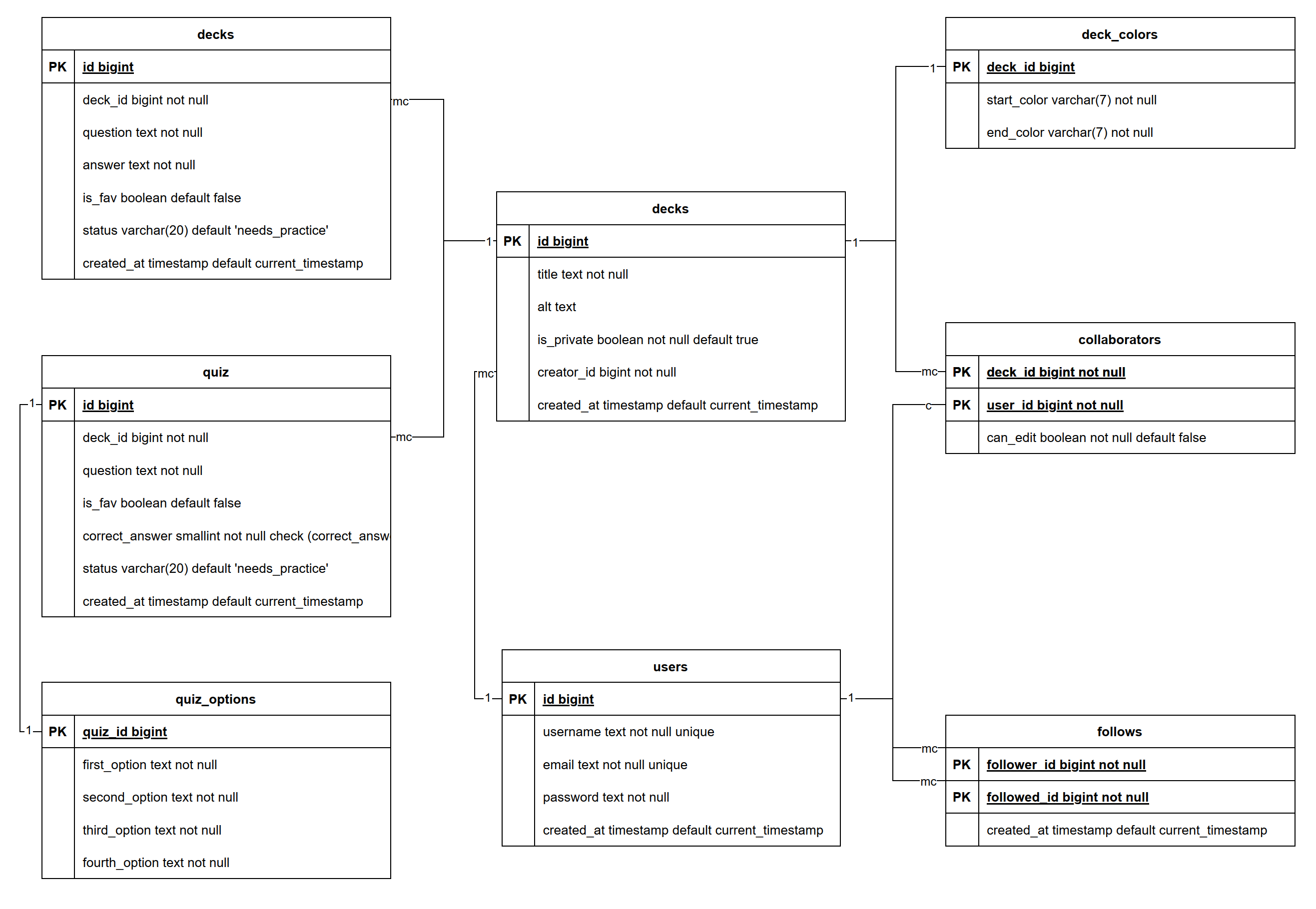
|  |  |  |
| --- | --- | --- |
| Kriterium | NoSQL Datenbank | Relationale Datenbak |
| Datenstruktur | 3/5 | 5/5 |
| Skalierbarkeit | 4/5 | 2/5 |
| Datenübertragungs  sicherheit | 2/5 | 5/5 |
| abfragekomplexität | 2/5 | 5/5 |
| Entwicklungskomfort | 4/5 | 4/5 |
| Zusammen | 15/25 | 21/25 |

# Hauptteil

## Datenbank

### ERD und Struktur

ERD der FlashCard Datenbank



**cards**

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**).

#### 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.

#### 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.

#### 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!

### 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.

### 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.

### 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

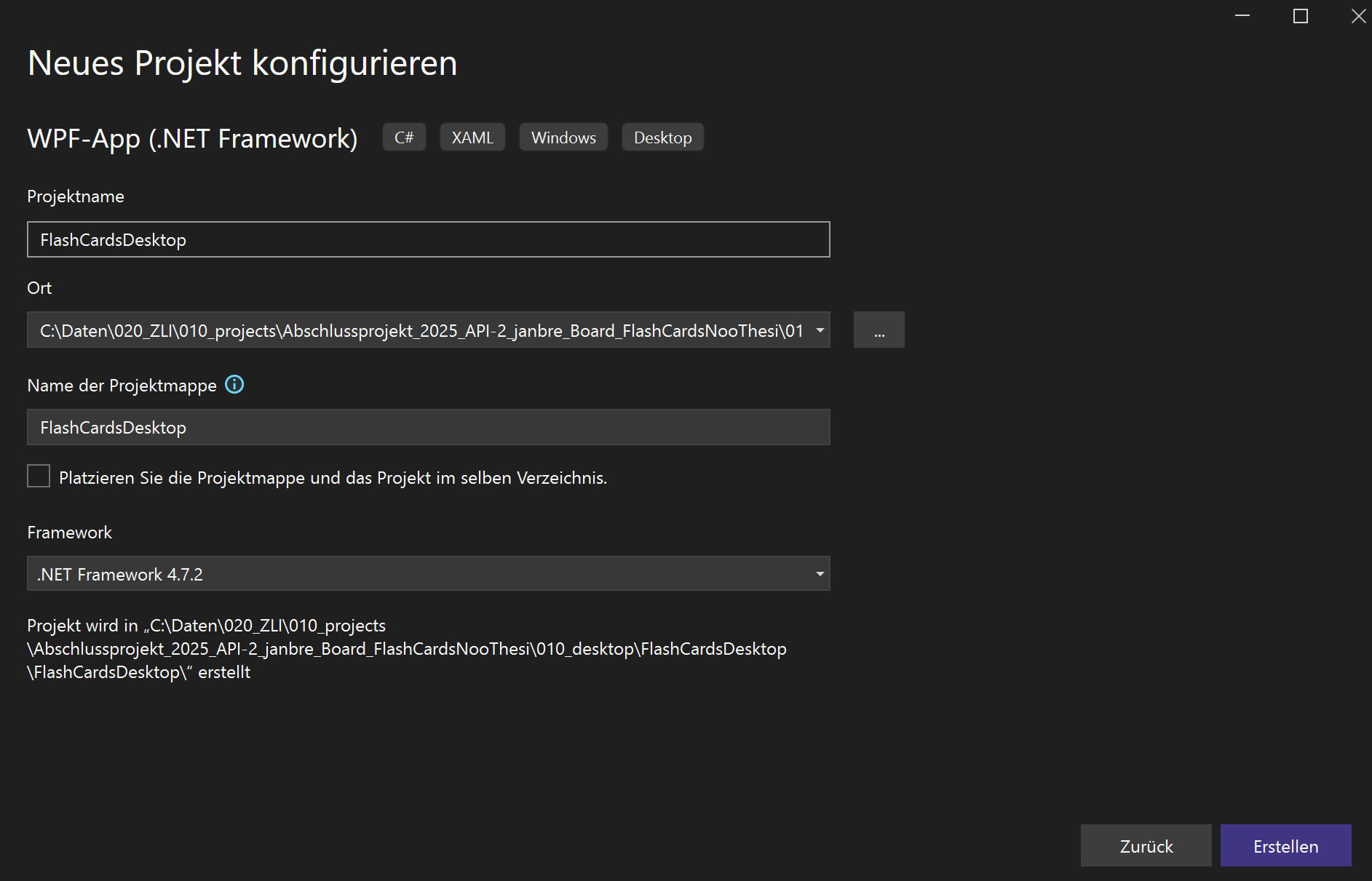
## WPF-Applikation und Serverbackend umsetzen

### 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 |

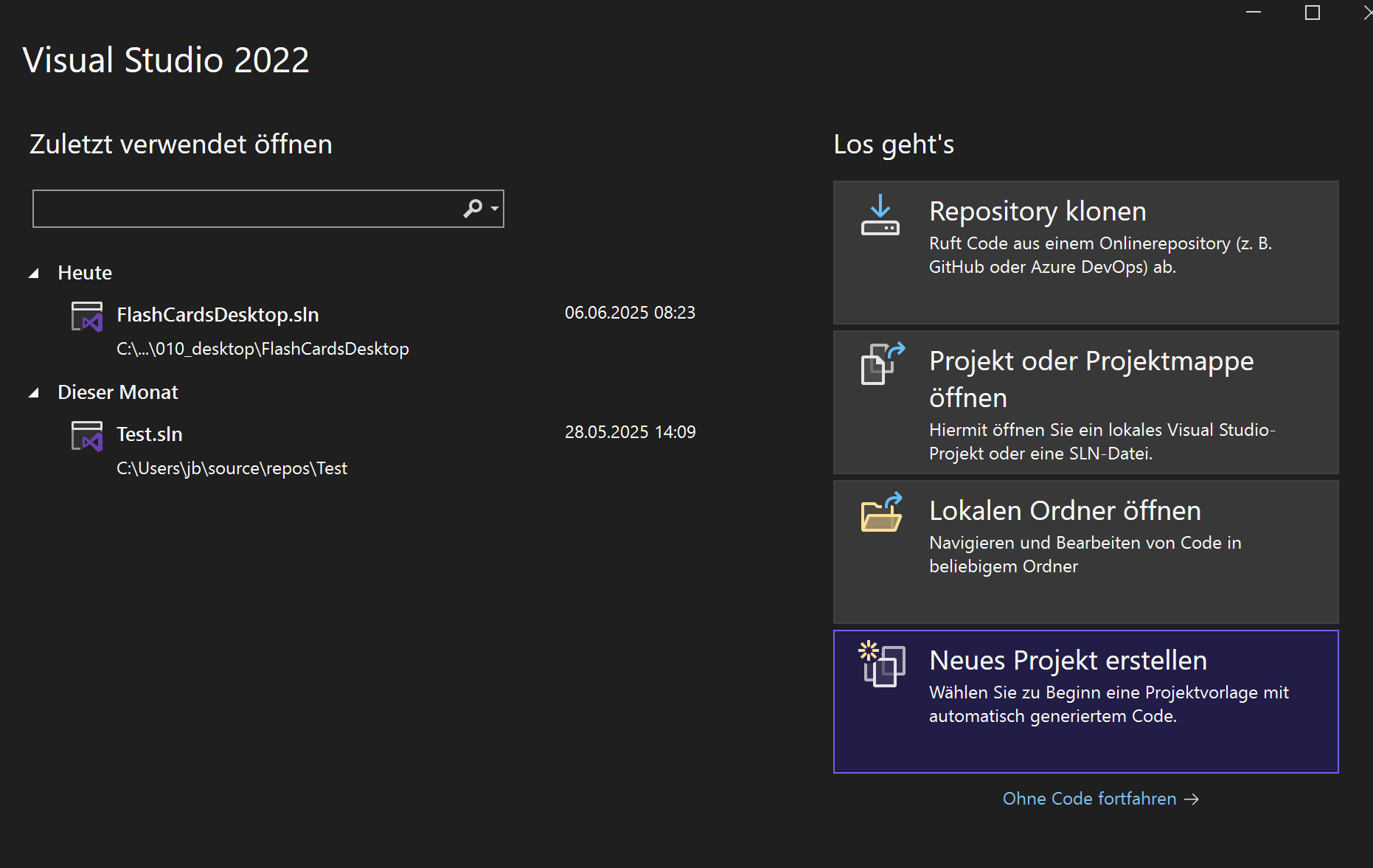
Da ich eine Windowsapplikation mit .Net, WPF und C# umsetzen möchte, eignet sich Visual Studio 2022 am besten geeignet.



Titel und Speicherort für Projektmappe



Die passende Projekt Vorlage auswählen



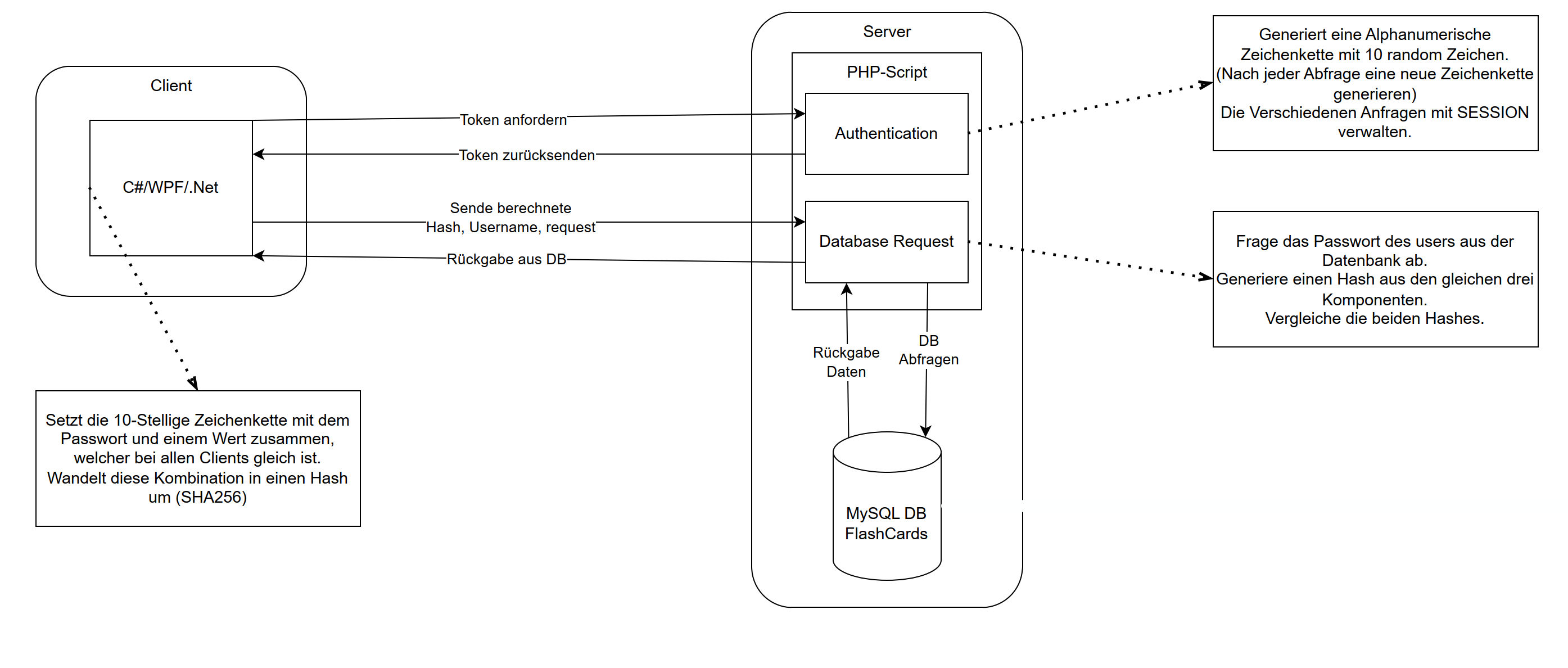
Neues Projekt wird erstellt

### Falscher Plan zur Verbindung mit der Datenbank

Ich wollte direkt von der C# App her Auf die Datenbank zugreifen. Es gibt eine Client-Datenbank-Bibliothek, um eine Verbindung zum MySql-Server aufzubauen. Ich habe es mit dieser Bibliothek versucht, es konnte jedoch keine Verbindung aufbauen. Mir war nicht direkt klar, woran dies liegt. Ich setzte deshalb in meinem Docker einen Mysql Container auf, um die Verbindung mit einem Lokalen Server zu testen. Auch dies hatte nicht funktioniert. Hier lag es wahrscheinlich daran, dass der Server nicht über Localhost:3306 lief, sondern über eine interne IP in Docker. Ich verfolgte dies jedoch nicht weiter, weil ich später bei der Applikation die Datenbankverbindung sowieso nicht so machen kann, weil der Client sonst immer im Server-Netzwerk oder mit einer VPN-Verbindung verbunden sein muss, was unrealistisch ist. Ich habe sehr viel Zeit in dieses Problem investiert. Nun habe ich jedoch eine Lösung, wie ich dieses Problem umgehen kann.

### Lösung, um mit Datenbank zu verbinden

Die Lösung ist über ein Serverseitiges PHP-Script. Der Client kann über http-Requests verschiedene Funktionen im PHP aufrufen. Ich kann hierfür die Library ‘System.Net.Http’ verwenden. Die System-Library ist eine Standardbibliothek für .Net. Diese Bibliothek ermöglicht mir sehr einfach eine http-Abfrage. Mit dieser Lösung muss ich mich nun jedoch um die Sicherheit kümmern, da nicht jeder, der auf das PHP-Script zugreifen kann auch die Datenbank Abfragen darf.



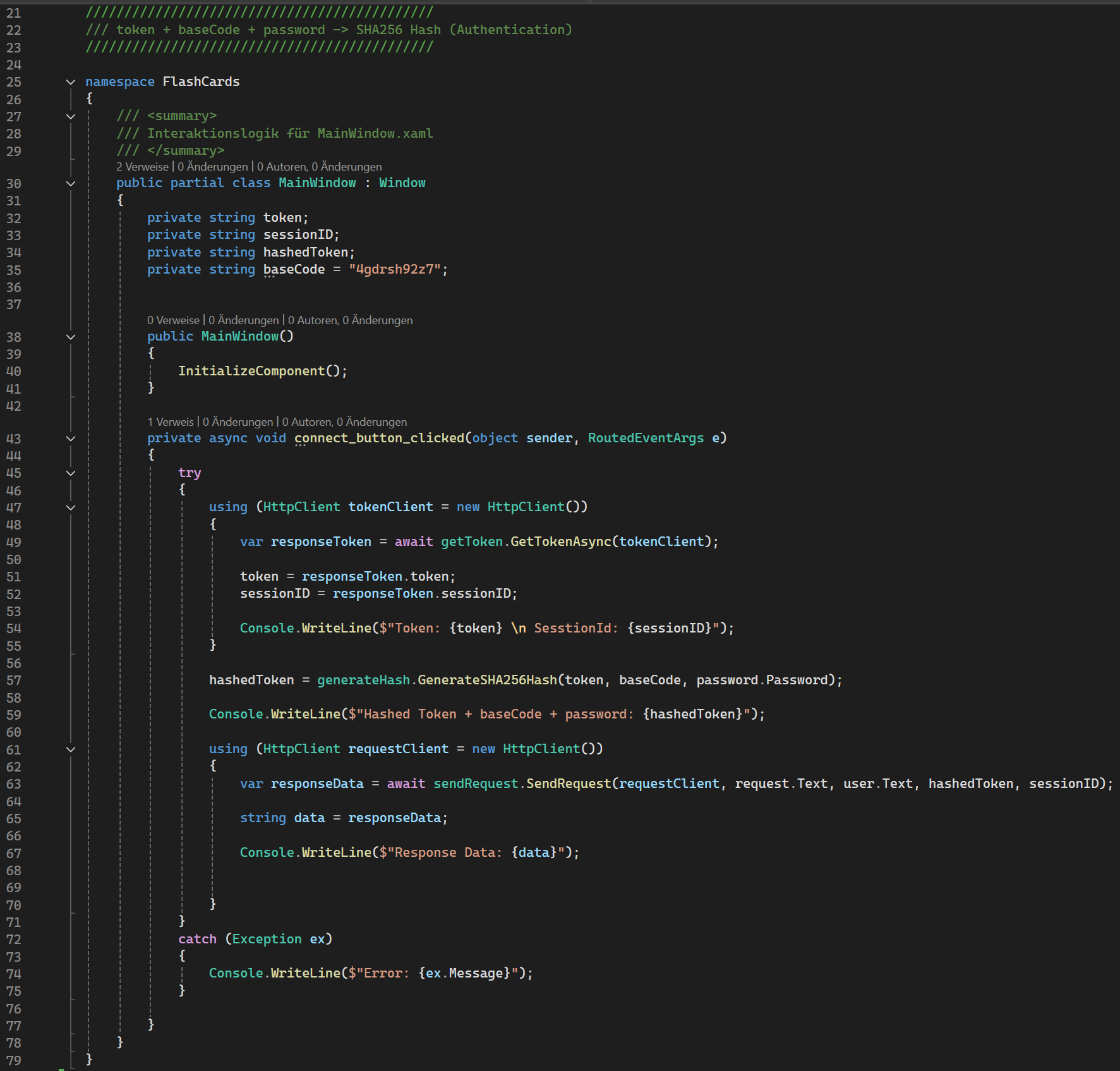
Datenbankabfrage Netzwerkschema

Bei diesem Schema habe ich kurz festgehalten, wie die Netzwerkkommunikation funktionieren sollte. Der Client fordert als erstes einen Token vom Server an. Der Server generiert eine Random Zeichenkette mit 10 Zeichen (gross-/klein-Buchstaben und Zahlen). Dieser Token wird dem Client zurückgesendet. Der Client generiert aus dem erhaltenen Token, dem Passwort des Benutzers und einem fixen 10-stelligen Code (‘Salt’), welcher bei allen Clients gleich ist, einen Hash (SHA256). Dieser Hash wird zusammen mit dem Benutzernamen und einer Aktion (z.B. getDecks) wieder an den Server gesendet. Nun weiss das PHP-Script den Benutzernamen und kann somit das Passwort dieses Benutzers aus der Datenbank abfragen. Der Server erstellt nun einen geleichenen Hash aus diesen drei Komponenten. Wenn der vom Client geschickte Hash mit dem Server-generierten übereinstimmt, wird die Aktion weiterverfolgt, die Datenbank abgefragt und in JSON-Format zurück an den Client gesendet. Wenn die Hashes nicht übereinstimmen, kommt direkt ein Error zurück.

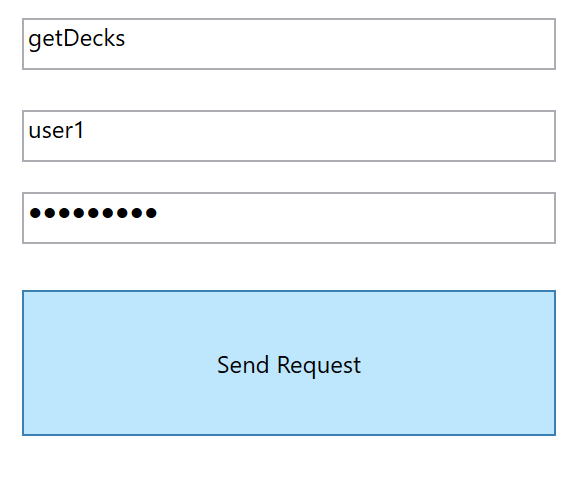
### Codeerklärung Prototyp mit Authentifizierung

#### MainWindow.xml.cs V1

Ich habe vier Files für die Hauptfunktion für die Datenbank Abfrage. Eine WPF-Applikation ist immer so aufgebaut, dass es ein MainWindow.xaml gibt, welches für das UI zuständig ist. XAML ist ähnlich wie HTML eine Markup Language für das Darstellen vom UI (Extensible Application Markup Language). Für die Logik gibt es dann ein MainWindow.xaml.cs. Hier kann man die ganze Logik verwalten.



MainWindow.xaml.cs



UI des Prototyps

Zuoberst in diesem Code sind die Instanzvariablen. Diese werden dort definiert, damit überall in dieser Klasse darauf zugegriffen werden kann. Der ‘baseCode’ ist der Salt Wert, welcher vor dem Hashen angehängt wird. Der Konstruktor ist auf Zeile 38. Hier wird das UI aus MainWindow.xaml geladen. Danach kommt ein Event-Listener. Wenn man auf den ‘Send-Request’-Button drückt, wird diese Methode aufgerufen. Auf der Zeile 47 wird mit der ‘System.Net.Http’ Bibliothek ein http-Client erstellt und dieser dann der GetTokenAsync-Methode mitgegeben. Diese wird weiter [unten](#_getToken.cs) genauer beschrieben (getToken.cs). Von dieser Methode kommt ein Array zurück, welches etwa so aussehen könnte:

{

"token": "DLpVFfdBBR",

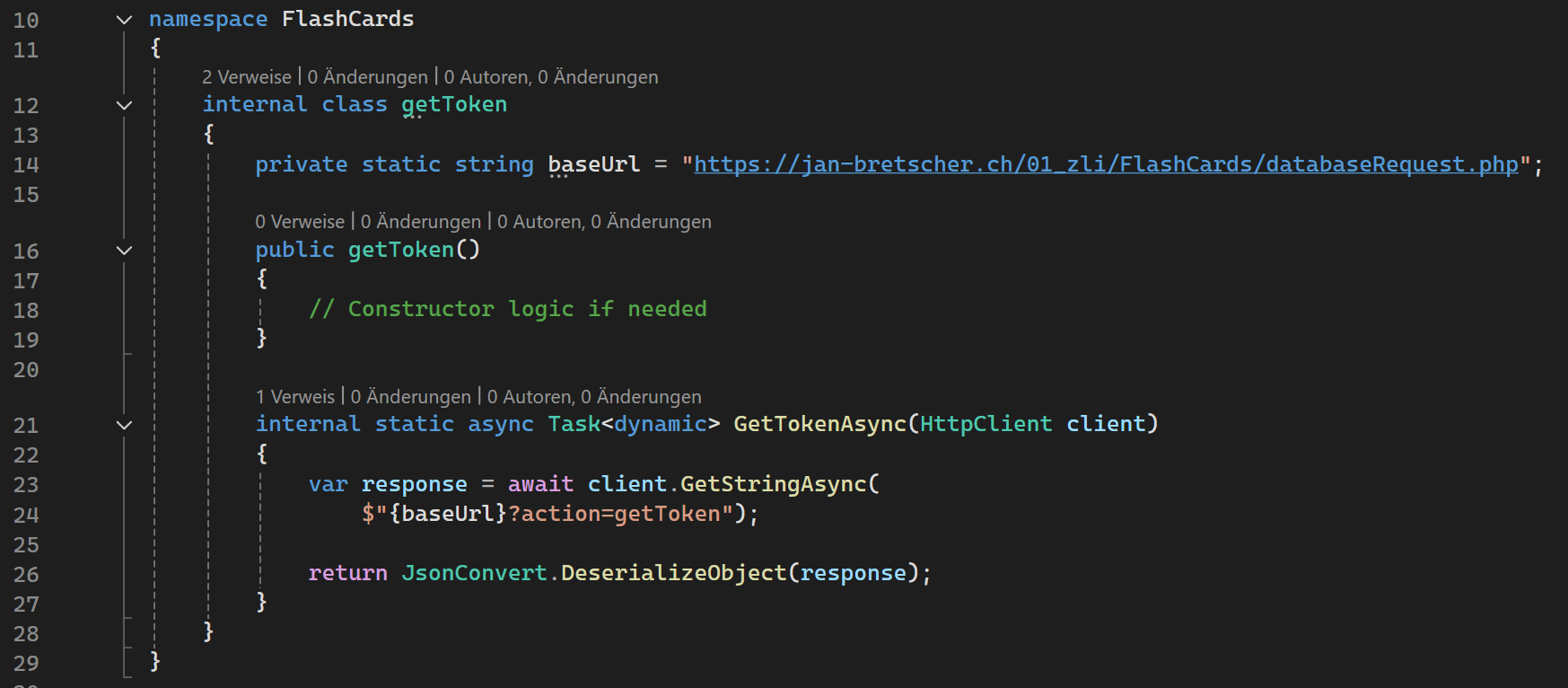
"sessionID": "qpp0t8srnbaueonl1iae9k8v06"

}

Der Token wird einzeln ausgelesen und dann fürs Debugging geprintet. Danach wird die GenerateHash-Methode ausgeführt, mit allen drei Komponenten als Parameter. Mehr dazu auch weiter [unten](#_generateHash.cs) (generateHash.cs). Auch hier wird alles in die Konsole geprintet zu Debugging Zwecken. Wenn man nun den Hash hat, wird die SendRequest-Methode aufgerufen, mit den Paramentern: http-Client, den request (z.B. getCards), den userName, den gehasheden Token und die SessionID. Mehr hierzu auch weiter [unten](#_sendRequest.cs) (sendRequest.cs).

All dies ist in einem Try. Wenn während einem Vorgang ein Fehler kommt, wird direkt abgebrochen und es geht in den catch, wo der Fehler noch genau in die Konsole geprintet wird.

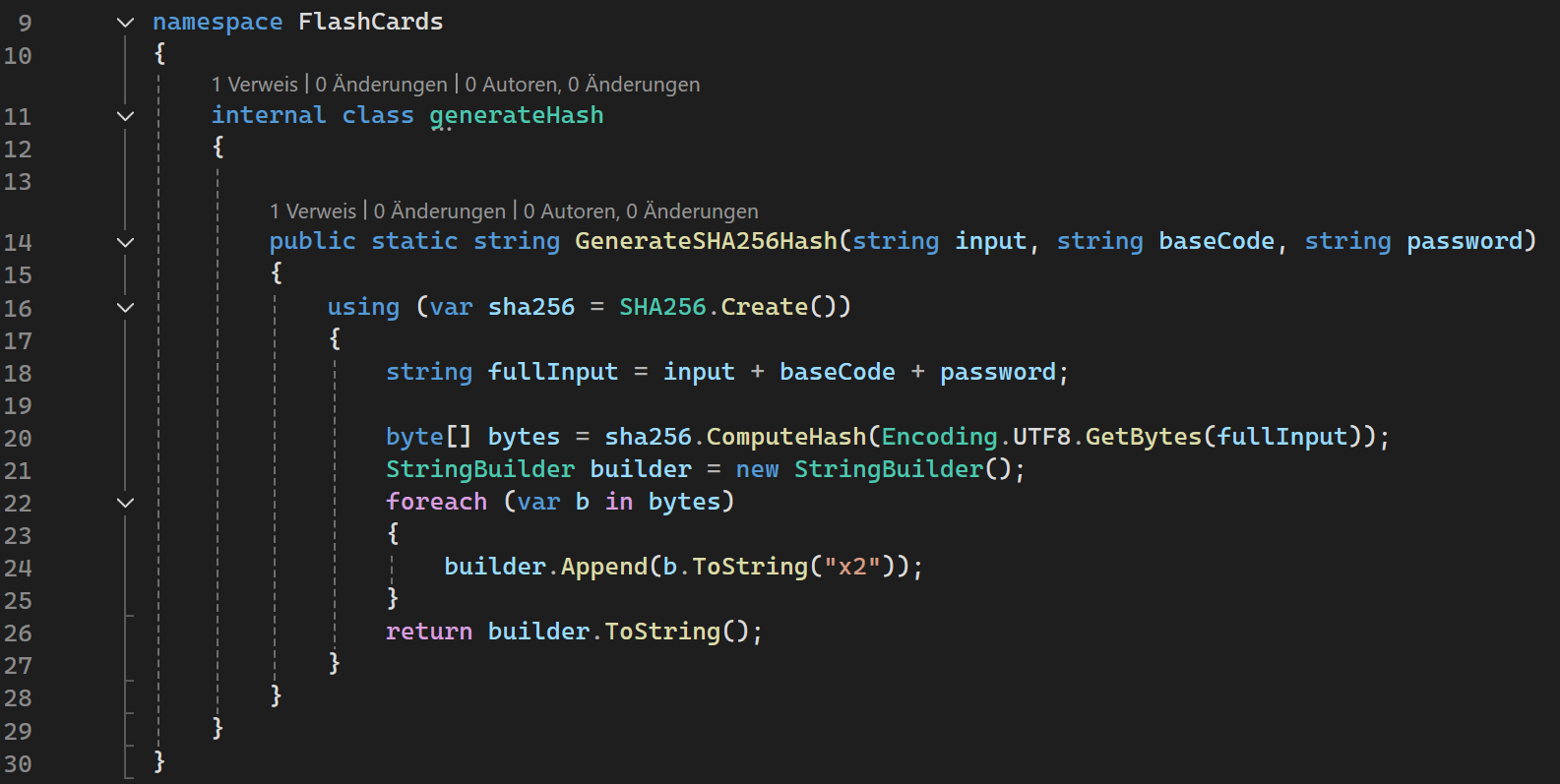
#### getToken.cs



getToken.cs

Die GetTokenAsync-Methode sendet eine http-Anfrage an die baseUrl. Hier liegt das PHP-Script, welches die Anfragen bearbeitet. Es wird noch der Parameter action=getToken angehängt. Hierzu später noch mehr. Es wird ein String erwartet, weshalb das Script einen String bekommt und diesen dann in JSON-Format umwandelt und zurückgibt.

#### generateHash.cs



generateHash.cs

In der GenerateSHA256Hash-methode wird die System.Security.Cryptography-Bibliothek verwendet. Als erstes wird ein neues Objekt, sha256 instanziiert. Danach alle Komponente zusammengenommen, welche gehashed werden. Danach wird die Zeichenfolge ‘fullInput’ mit dem SHA256-Algorythmus zu einer Bitfolge umgewandelt und danach zu einem String zusammengesetzt. Dieser string wird dann in einen HexaDezimalen, für uns ‘lesbaren’ String umgerechnet.

#### sendRequest.cs

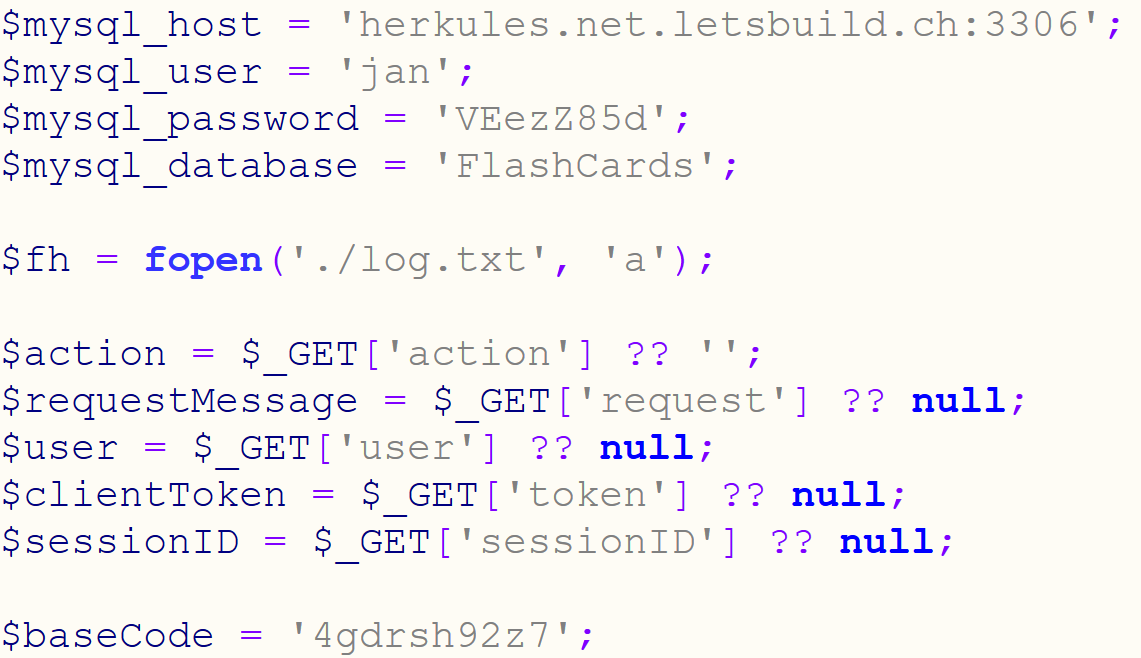


sendRequest.cs

Diese Klasse funktioniert sehr ähnlich, wie [getToken](#_getToken.cs). Der einzige Unterschied sind die vielen Parameter, welche hier mitgeschickt werden. Es werden die Anfrage, der Username, der gehasheden Wert und die SesstionID mitgegeben. Die SessionID ist wichtig, dass der Server denselben Token verwendet, wie er diesem Client zur Verfügung gestellt hat für die Berechnung des Hashs verwendet. Mehr hierzu weiter [unten](#_databaseRequest.php) (databaseRequest.php).

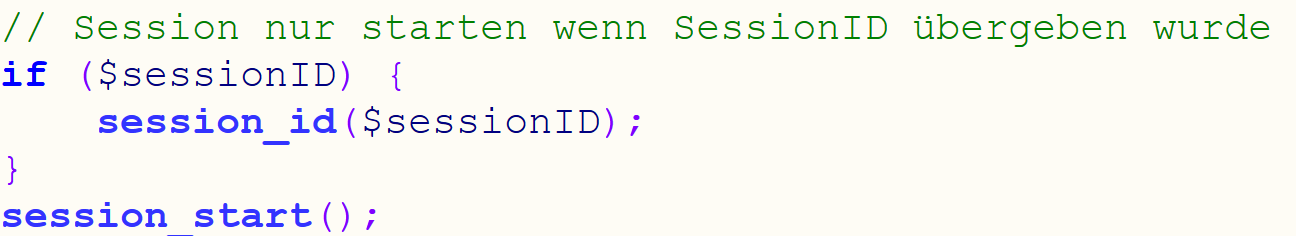
#### databaseRequest.php

Hier werden die allgemeinen Variablen definiert. Dies sind alle Credentials für die Datenbankverbindung, das errorhandling logfile, die verschiedenen Parameter, welche mit dem http-Request mitgegeben werden und dem baseCode (=Salt).



CodeSnippet databaseRequest.php allgemeine Variabeln

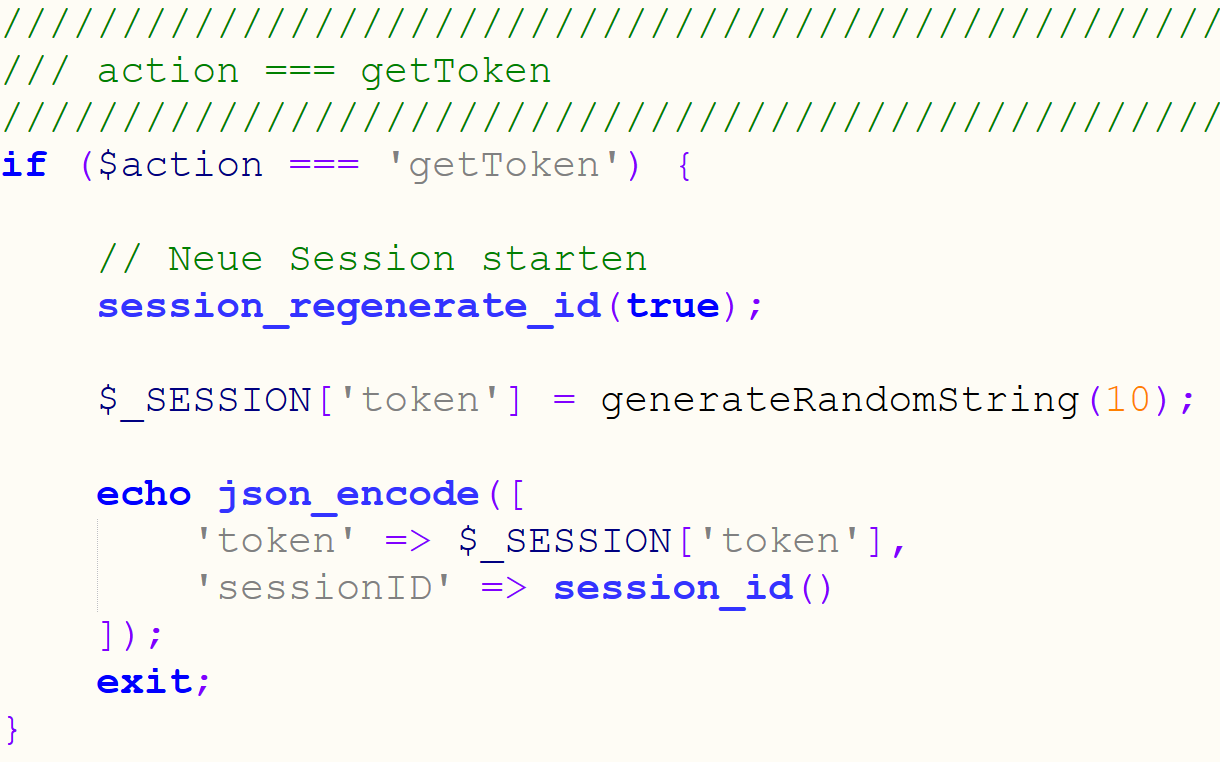
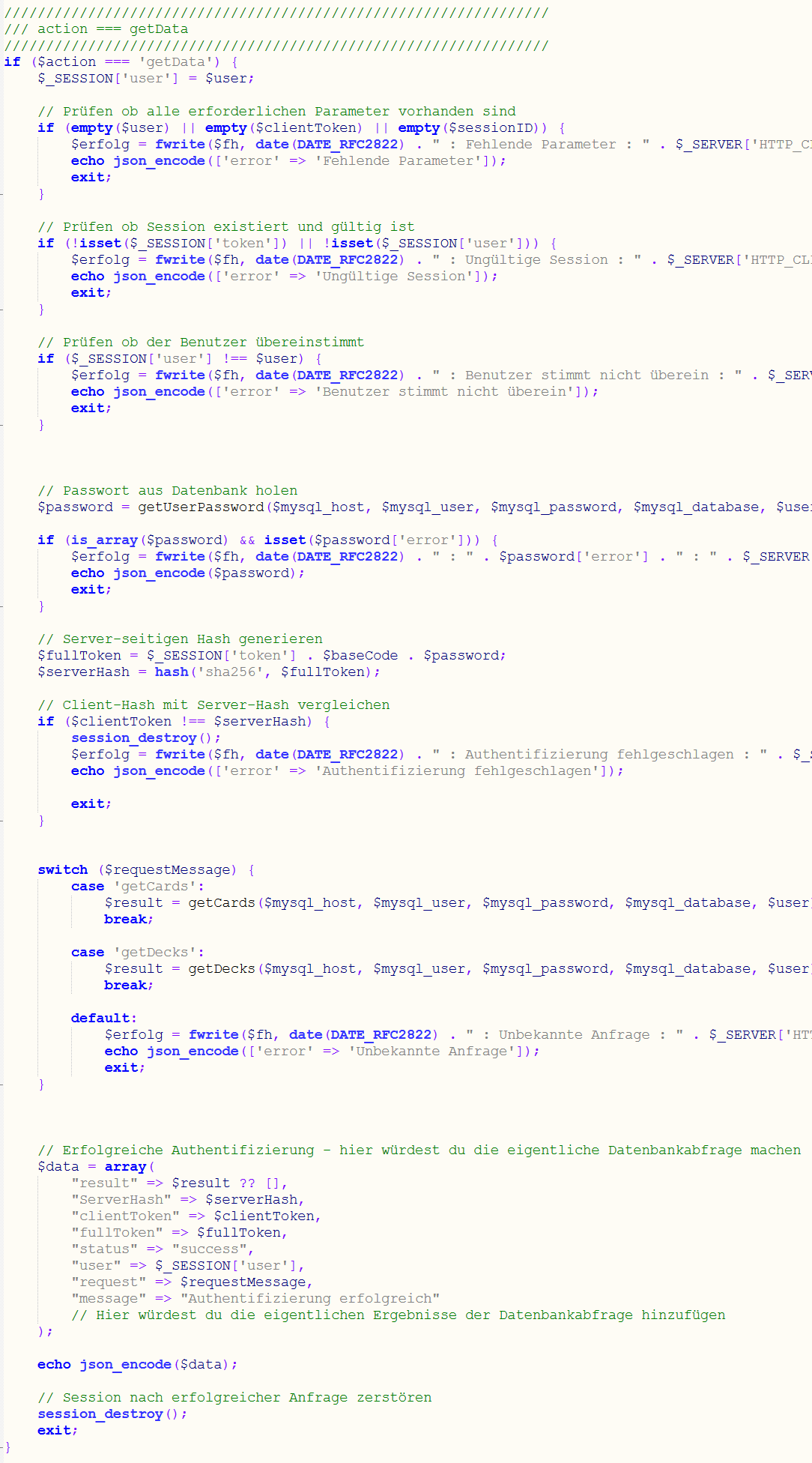
Der Client ruft das PHP-Script zweimal auf. Das erste mal um einen Token zu bekommen, das zweite mal, um den Request zu senden. Wenn er einen token anfordert, ist er noch in keiner SESSION, deshalb wird eine neue gestartet. Beim zweiten mal gibt es jedoch bereits eine SESSION für diesen Client und deshalb wird die SESSION mit der alten ID wieder geöffnet.



CodeSnippet databaseRequest.php Sessionverwaltung

Wenn der Parameter vom http-Request ‘action=getToken’ lautet, wird dieser If ausgeführt. Die aktuelle Session, welche oben gestartet wurde, wird nun mit session\_regenerate\_id(true); neu erstellt. Dies dient als Sicherheitsvorkehrung, dass nicht die wiedergeöffnete session verwendet wird, sondern nur ein Klon. Danach wird die Funktion generateRandomString(10); ausgeführt. Diese gibt einen Wert von 10 Zeichen zurück, welche in der Session-Variablen ‘token’ gespeichert wird. Als Rückgabewert wird der Token und die sessionID zurückgegeben.

CodeSnippet databaseRequest.php ‘action===getData’



CodeSnippet databaseRequest.php action === ‘getToken’

Wenn die action===getData lautet, wird dieser Codeteil ausgeführt. Als erstes wird die sessionvariable ‘user’ mit dem mitgegebenen username befüllt. Danach wird überprüft, ob der Client alle erfoderlichen Parameter mitgeschickt hat, ob die Session existiert und gültig ist und ob der Session-user mit dem mitgegeben benutzernamen übereinstimmt. Falls etwas nicht stimmt, wird ein Fehler zurück gesendet, ein Kommentar ins Logfile geschrieben und der Code wird unterbrochen. Danach wird die Funktion getUserPassword aufgerufen, welche das Passwort des Benutzers aus der Datenbank ausliest, welcher als Parameter mitgegeben wurde. Auch hier wird die Passwortvariable geprüft, dass kein Fehler entsteht. Mit dem Schlüsselwort ‘hash’ kann ein Hash erstellt werden. Wenn der ClientToken nicht mit dem vom Sever berechneten Hash überein stimmt, wird ein Fehler zurückgegeben und der Code unterbrochen. Falls diese jedoch gleich sind, kommt der switch zum Zug. Der Client schickt als Parameter einen request-Schlüsselwort mit, welches in diesem Switch vermerkt sein muss. Fürs Testing gibt es erst zwei Möglichkeiten. Entweder ‘getCards’ oder ‘getDecks’. Jenachdem wird eine andere Funktion aufgerufen, welche unten erläutert wird. Das Ergebnis wird als Array zurückgegeben ($data). In diesem JSON-Array $data enthält noch viele weitere Ausgaben, welche jedoch nur für Debugging Zwecke mitgesendet wurden. Wichtig ist hier ‘result’. Zum Schluss wird die Session restlos gelöscht, sodass sich niemand mehr mit dieser id anmelden kann.

Alle Funktionen, welche die Datenbank abfragen, sind sehr ähnlich aufgebaut. Ich habe deshalb versucht hier möglichst viel zusammen zu nehmen. Dies ist jedoch nur die Verbindung zum Server herstellen. Deshalb wird nun bei allen DatenbankabfrageFunktionen (z.B. getUserPassword, getCards, getDecks) als erstes createDatabaseConnection aufgerufen. In sql wird die individuelle SQL-Query gespeichert, welche dann mit mysqli\_stmt\_execute ausgeführt wird. Wenn etwas zurückgekommen ist, wird jeder Datensatz in zusammengenommen in einem Array, welches dann zurückgegeben wird. Die mysql-Verbindung wird nach der Abfrage wieder geschlossen.



CodeSnippet databaseRequest.php getCards

Mysqli ist die neuere Syntax für mysql. Das ‘i’ steht für Improved und die mysqli-Funktion steht seit PHP v5 zur verfügung. Diese neue Version erlaubt auch Objektorientierte Datenbankabfragen. Ich verwende dies in meinem Projekt jedoch nicht, weil ich denke, dass es nur unnötig kompliziert wird.

### Decks und Cards anzeigen V2.1

Bei dieser Funktionalität hat die Applikation einen grossen Schritt nach vorne gemacht mit dem Design. Nun kommt man beim Start der App auf die Indexseite, wo man alle Decks sieht, welche dem angemeldeten User gehören oder bei denen er Mitarbeiter ist. Auf dieser Seite gibt es eine Suchfunktion, um nach den Deck-Titeln zu suchen. Wenn man aufs Deck drückt, werden alle zugeordneten Karten und Quizze in einer Übersichtsliste angezeigt. Auch hier kann man die Karten mit einer Suchfunktion nach den Fragen filtern. Weiter ist auf der Indexseite auch die Löschenfunktionalität vorhanden, mit der man ein ganzes Deck mit mit allen weiteren Entitäten, wie den Karten löschen kann.

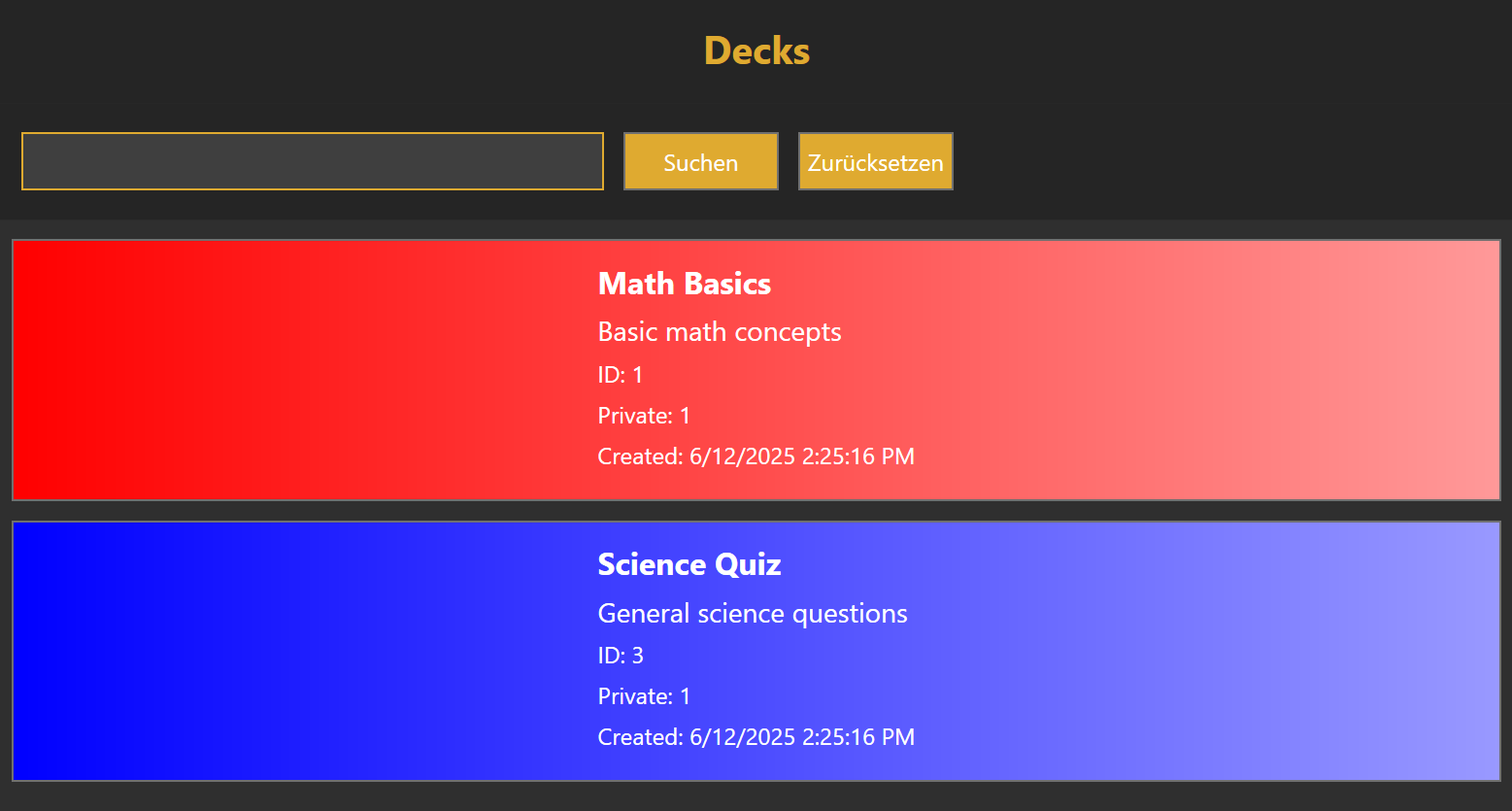
### Codeerklärung Decks und Cards anzeigen V2.1 (Final-Version)

#### Index.xaml & Index.xaml.cs

Das MainWindow.xaml hat nun keinen Nutzen mehr. Die gesamte Funktionalität ist in Index.xaml(.cs) übergegangen. Dies aus benennungsgründen, damit alles etwas übersichtlicher wurde. In diesem File hat es drei grosse Erweiterungen gegeben, sodass es nun auf der Endgüldigen Version steht. Die Funktionen sind: Alle decks, welche dem Benutzer gehören oder bei denen er mitarbeitet im UI darstellen, in den angezeigten Decks suchen und einzelne Decks vollständig löschen.

Darstellen der Decks im UI

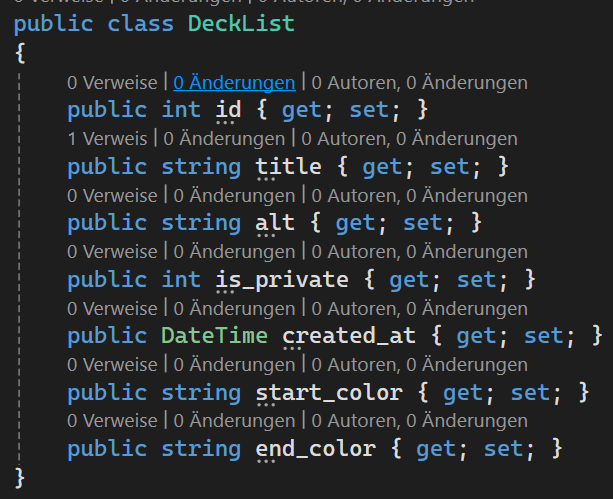
Das UI sieht wie im Bild rechts dargestellt aus. Die Logik und Funktionalität ist grundsätzlich das selbe, wie beim [Prototyp](#_MainWindow.xml.cs_V1). Der Client schickt via http-Request einen Token anfordert. Mit diesem Token berechnen beide einen Authentifizierungshash. Danach sendet der Client eine Anfrage an den Server, welcher die Datenbank abfragt und in JSON-Format zurück gibt.



Index.xaml Final-UI

Das ganze wird wie folgt ins UI-übertragen:

Die Funktionalität der Datenbankabfrage entspricht im Wesentlichen dem Prototyp, wobei ich die JSON-Rückgabe nun nicht mehr in die Konsole ausgebe. Stattdessen wird die DeckList-Klasse verwendet, um die Rückgaben zu speichern. Mit DeserializeObject (Newtonsoft.Json) wird das JSON-Array in eine Sammlung von DeckList-Objekten umgewandelt, die in die Listen allDecks und filteredDecks geschrieben wird. Diese Listen dienen der Suchfunktion.



Index.xaml.cs getDecks()-Function & DeckList-Class

Im Gegensatz zum Prototyp habe ich auch die Datenbankabfrage angepasst. Als Parameter gibt man beim Methodenaufruf SendRequest den username mit, welcher hier für die Filterung der Daten benötigt wird. Es werden: id, titel, Alternativtext, ob das Deck privat ist, wann es erstellt wurde und die Beiden Farben abgefragt. Dies von den Decks, bei denen die userId mit der vom mitgegebenen User übereinstimmen. Es wird aber auch nach Collaborator gesucht, beiden Decks wo user\_id mit dem User überein stimmen.

**SELECT** d**.**id**,** d**.**title**,** d**.**alt**,** d**.**is\_private**,** d**.**created\_at**,** dc**.**start\_color**,** dc**.**end\_color

**FROM** decks d

**LEFT** **JOIN** deck\_colors dc **ON** d**.**id **=** dc**.**deck\_id

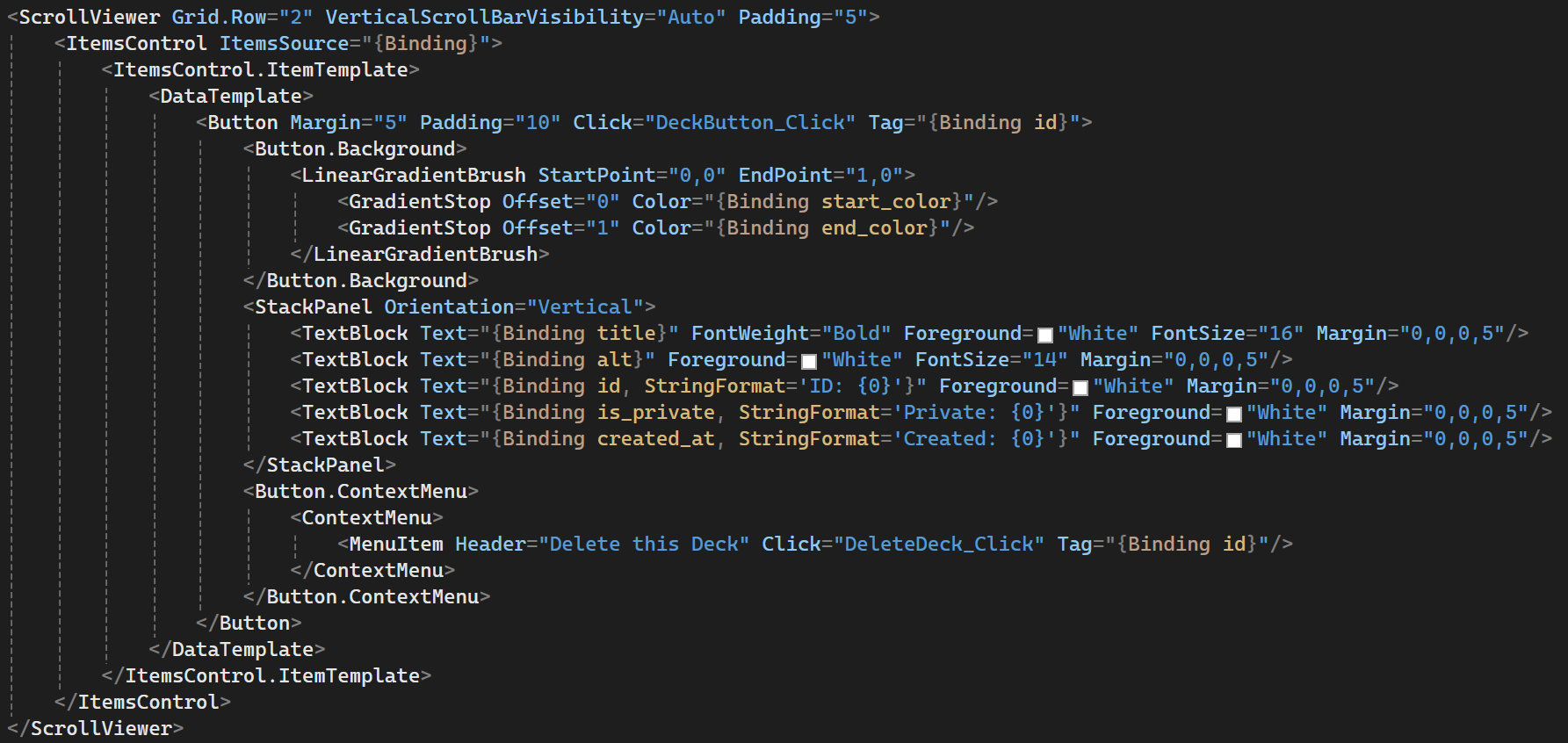
**WHERE** d**.**creator\_id **=** **(**

**SELECT** u**.**id **FROM** users u **WHERE** u**.**username **=** **?)** **OR** d**.**id **IN** **(**

**SELECT** deck\_id **FROM** collaborators **WHERE** **user\_id** **=** **(**

**SELECT** u**.**id **FROM** users u **WHERE** u**.**username **=** **?));**

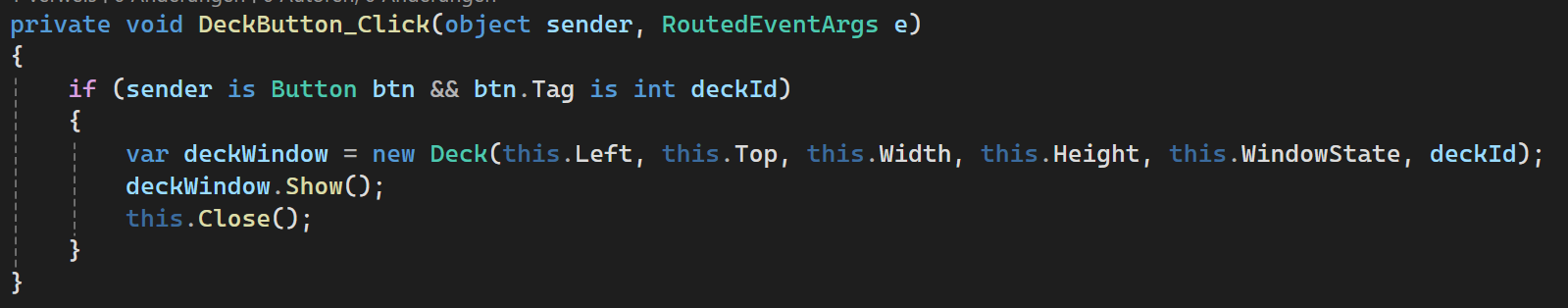
Um das ganze im UI darzustellen wird Binding verwendet. Binding ist das Schlüsselwort, um Daten aus dem Code-Behind ins Frontend zu bringen. Mit ItemsControll wird die Qelle der Daten so festgelegt, dass sie aus dem Code behind stammen. Das ganze liegt in einem ScrollViewer, damit man scrollen kann, wenn es zu viele Decks wären. Für jedes Element aus DeckList wird ein neuer Button erstellt, welche auf die Deckübersicht weiterleitet. Der button hat einen Farbverlauf, wie er in der Datenbank steht (start\_color, end\_color) und in TextBlöcken stehen die weiteren Informationen zum Deck mit verschiedenen Styles. Das ContextMenu wird fürs [Löschen](#_Decks_löschen) des Decks verwendet.



Index.xaml Ausschnitt von Decksanzeige

Routing zwischen den Seiten

Wenn man auf den Button eines Decks drückt, wird dieser EventListener ausgeführt. Er öffnet ein neues Window, nämlich [Deck.xaml](#_Deck.xaml). Bei Buttonclick wird dem Eventlistener den Parameter deckId mitgegeben. Dieser wird wiederum beim Instanziieren des neuen Fensters mitgegeben, mit weiteren Informationen zum aktuellen Fenster. Diese sind dazu da, dass das neue Fenster gleich gross und am gleichen Ort erscheint. Dieses neue Objekt von Deck wird dargestellt (.Show();) und das aktuelle Index-Fenster geschlossen (.Close();).



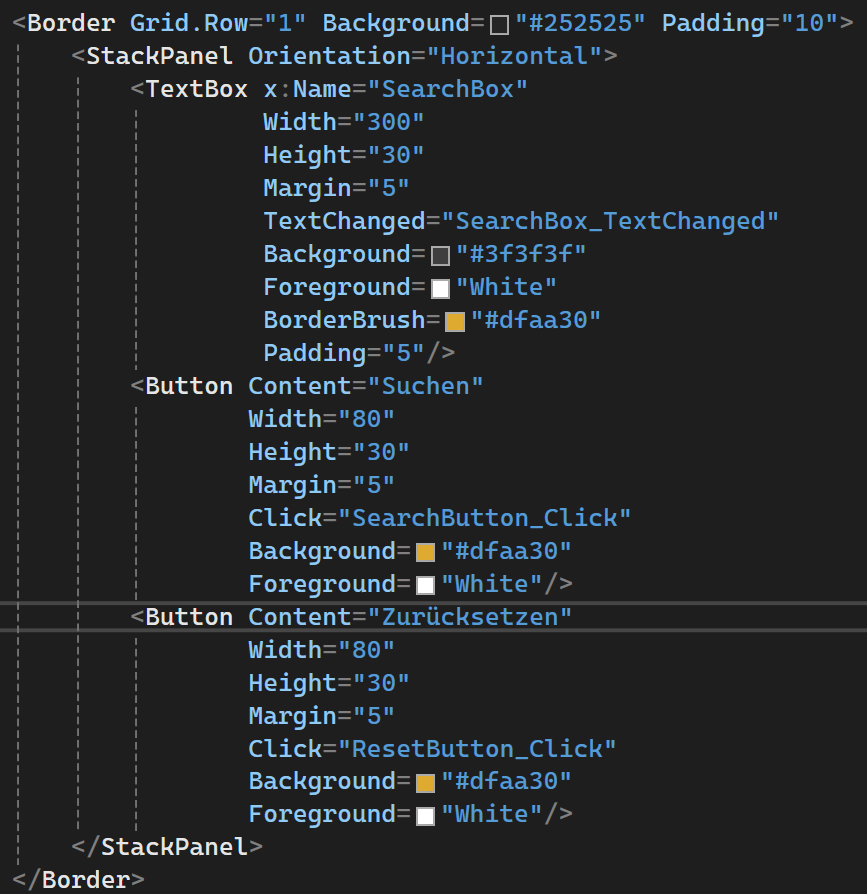
Index.xaml.cs Routing nach Deck.xaml

Nach Decks suchen

Im UI gibt es ein Inputfeld (TextBox), inwelcher man einen Text eingeben kann, um nach einem Deck zu suchen. Weiter sind die Buttons ‘Suchen’ und ‘Reset’ vorhanden. Bei jedem ButtonClick wird eine Methode im Code-Behind ausgeführt. Wenn sich in der TextBox etwas ändert, oder der Suchen-Button gedrückt wird, dann wird FilterCards(); aufgerufen. Hier wird als erstes überprüft, ob die Liste allCards Leer ist, welche bei getDecks befüllt wurde mit allen Ergebnissen aus der Datenbank. Danach wird der eingegebene Text von Leerzeichen befreit und alles wird klein geschrieben. Wenn das Textfeld keinen Text beinhaltet, die Liste mit allen Decks neu Instanziiert und als filteredDecks gespeichert. Wenn jedoch ein Suchbegriff eingegeben wurde, die Liste ‘filteredDecks’ mit den Elementen aus allDecks befüllt, welche den SuchText beinhalten. Wenn der Reset-Button gedrückt wurde, dann wird der inhalt der Textbox geleert und und die Liste filteredDecks mit allDecks befüllt.



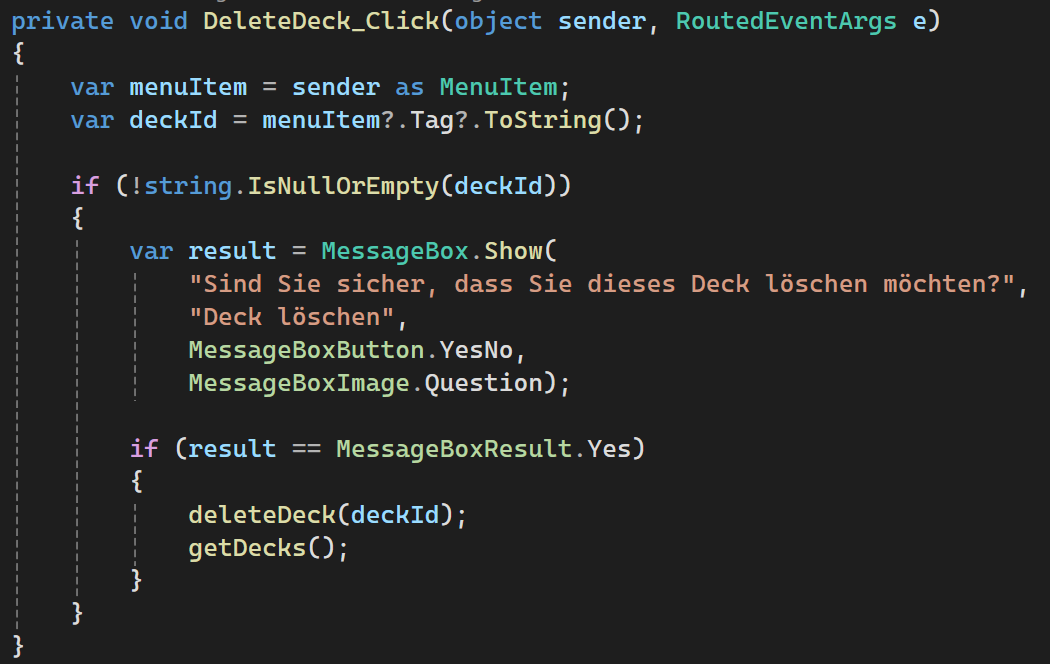
Index.xaml.cs Suchfunktion-Logik



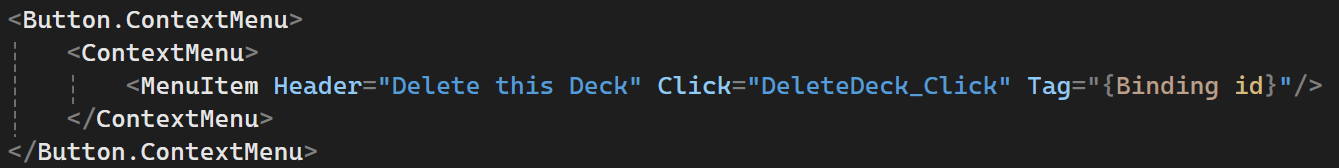
Index.xaml für die Suchfunktion

Decks löschen

Mit Rechtsclick auf ein Deck kann man es Löschen. Mit einem ContextMenu kommte bei Rechtsclick eine auswahl von den MenuItems. Bei mir ist es nur eine Möglichkeit, nämlich ‘Delete this Deck’. Wenn man dort draruf drückt, wird die DeleteDeck\_Click-Methode ausgeführt. Auch hier wird die DeckId als Parameter mitgegeben. Es kommt noch eine MessageBox, ob man wirklich löschen will und falls ja gedrückt wird, wird die deletDeck()-Methode ausgeführt. Diese ist sehr ähnlich wie getDecks. Der unterschied ist, dass der parameter request=deleteDeck lautet. Dies löst im PHP-Script eine andere Funktion aus.

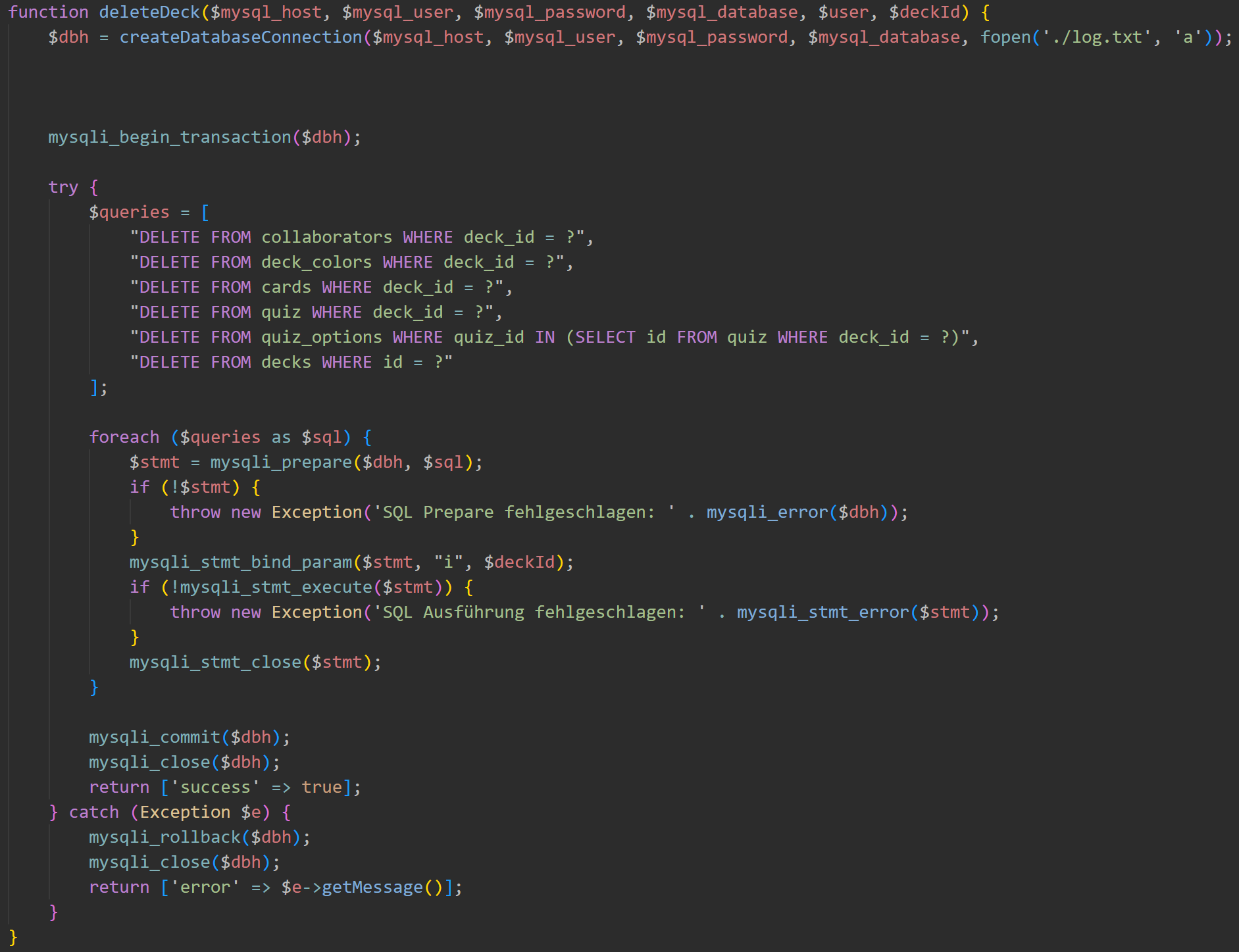


Index.xaml.cs Löschfunktion Event



Index.xaml Ausschnitt für Löschfunktion

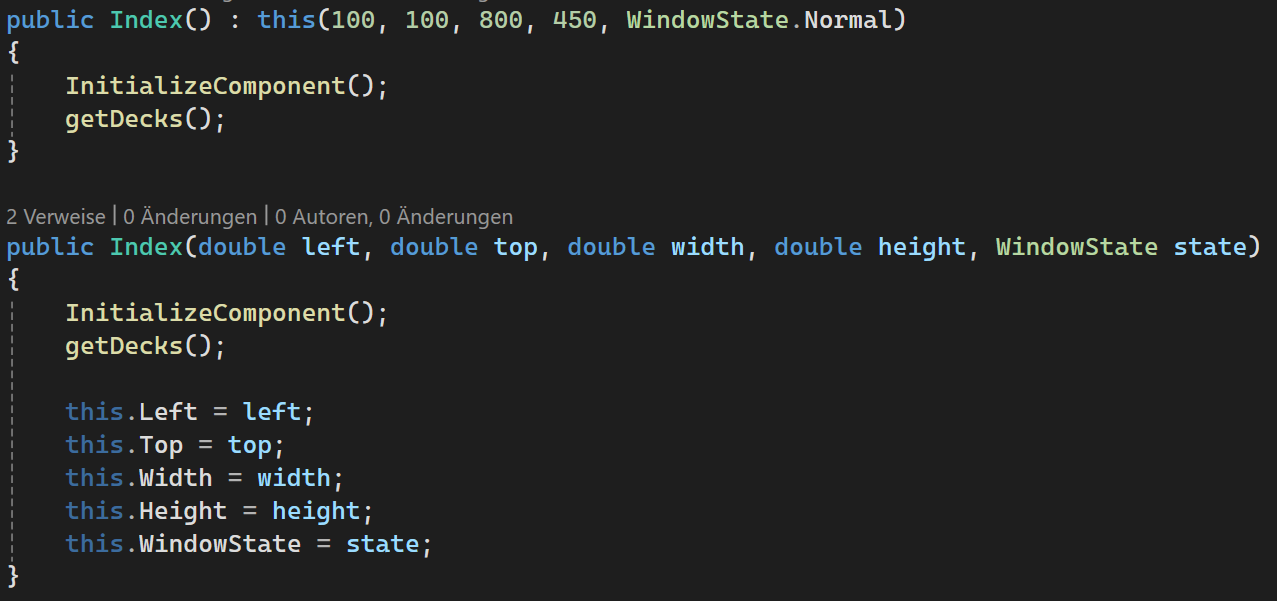
Die deleteDeck Funktion im PHP ist ähnlich aufgebaut, wie die Get-abfragen. Zuerst wird eine Verbindung zur Datenbank hergestellt. Danach beginnt eine transaction. Dies ermöglicht mehr als nur eine Abfrage. Da man hier mehrere Tabellen löschen muss, welche alle zu einem Deck gehören, eignet sich dies ganz gut. Im Array $queries werden alle Datenbankinteraktionen definiert. Und danach mit einer foreach-Schleife jede Query ausgeführt.



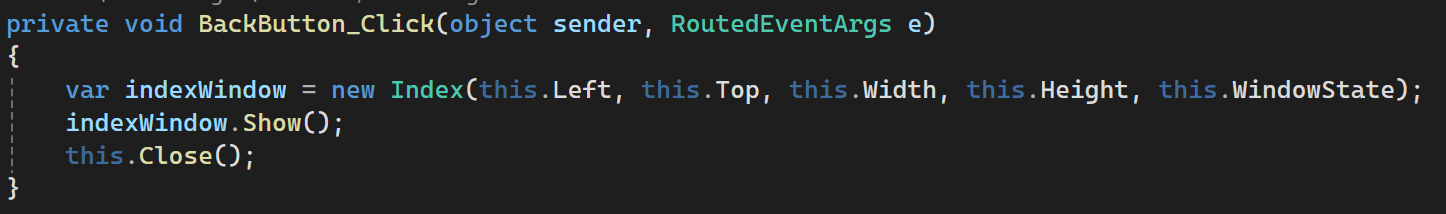
databaseRequest.php request===deleteDeck

#### Deck.xaml und Deck.xaml.cs

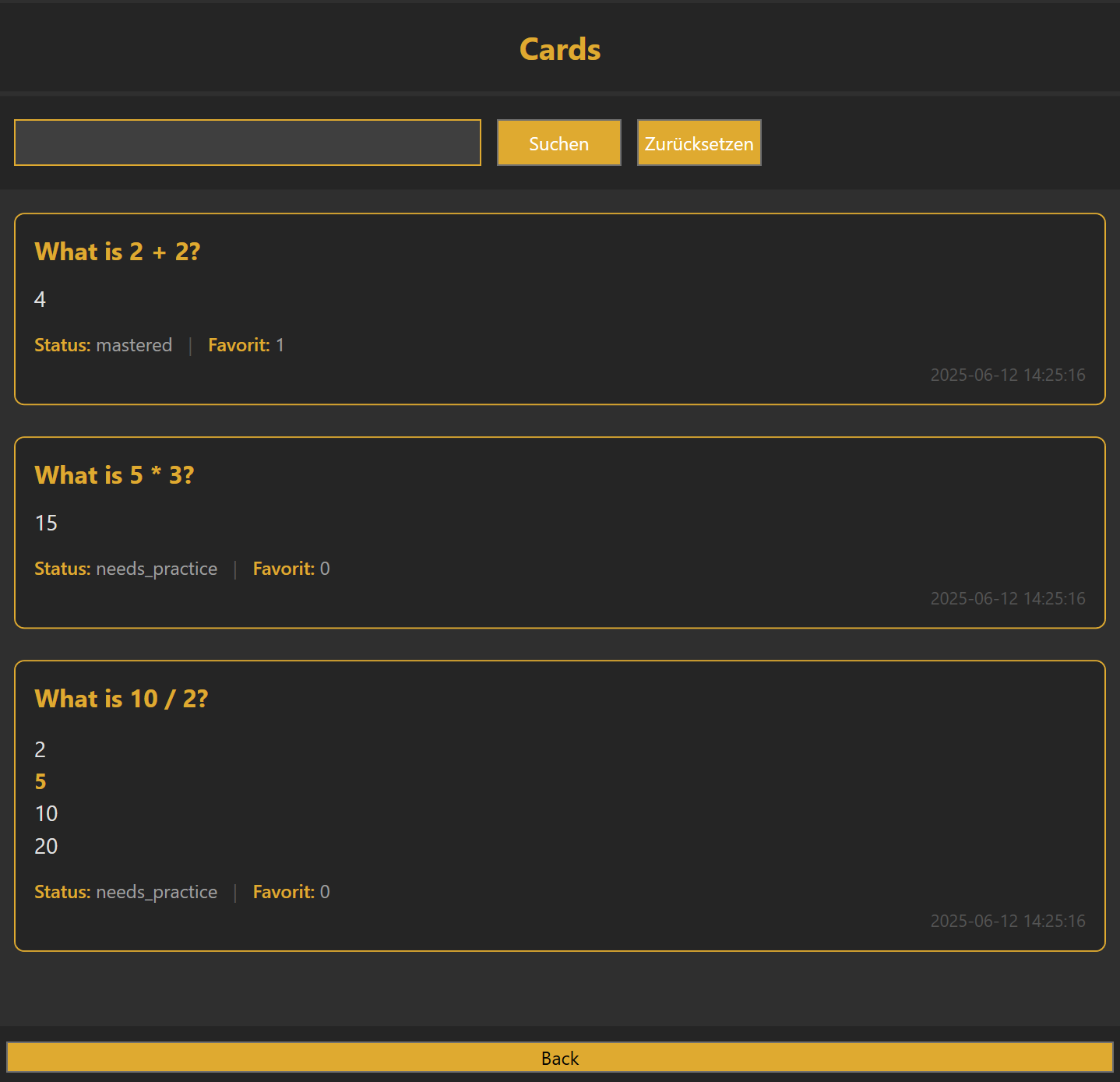
Im deck.xaml(.cs) ist es grundsätzlich das Selbe, wie im [Index.xaml](#_Index.xaml_&_Index.xaml.cs). Hier werden alle Karten eines Decks angezeigt, wessen Id per Parameter bei Seitenaufruf mitgegeben wurde. Hier sind noch einige Variablen mehr in der CardList klasse. Auch die SQL-Abfrage hierfür ist einiges komplexer, weil alle cards und alle quiz des Decks angezeigt werden. Es sind sozusagen zwei Queries und beide habe eine andere Ausgabe. Mit dem Schlüsselwort UNION ALL können zwei verschiedene Abfragen zu einer zusammengehängt werden. Die Suchfunktion und das Darstellen der Daten funktioniert grundsätzlich sehr ähnlich wie oben beschrieben. Neu ist auch, dass man mit dem Backbutton zurück kommt. Wie beim routing zur Deck Seite wird hier ein neues Window Instanziert, welches die Indexseite darstellt. Dies auch wieder mit den Parameter, welche die grösse des Fenster angeben. Hier war das Problem, dass beim ersten erstellen des Windows diese Parameter nicht erlaubt sind. Wenn ein Window ganz zu beginn gestartet wird, darf der Konstruktor keine Werte verlange, da das Objekt zu Beginn leer instanziiert wird. Die Lösung hierfür waren also 2 verschiedene Konstruktoren, einen mit den Parametern und einen ohne. Wenn die Applikation als erstes gestartet wird, werden die Parameter per default gesetzt. Wenn jedoch das Fesnster später während dem laufen der Applikation aufgerufen wird, dann werden die Konstruktorparameter verwendet, welche mitgegeben werden.



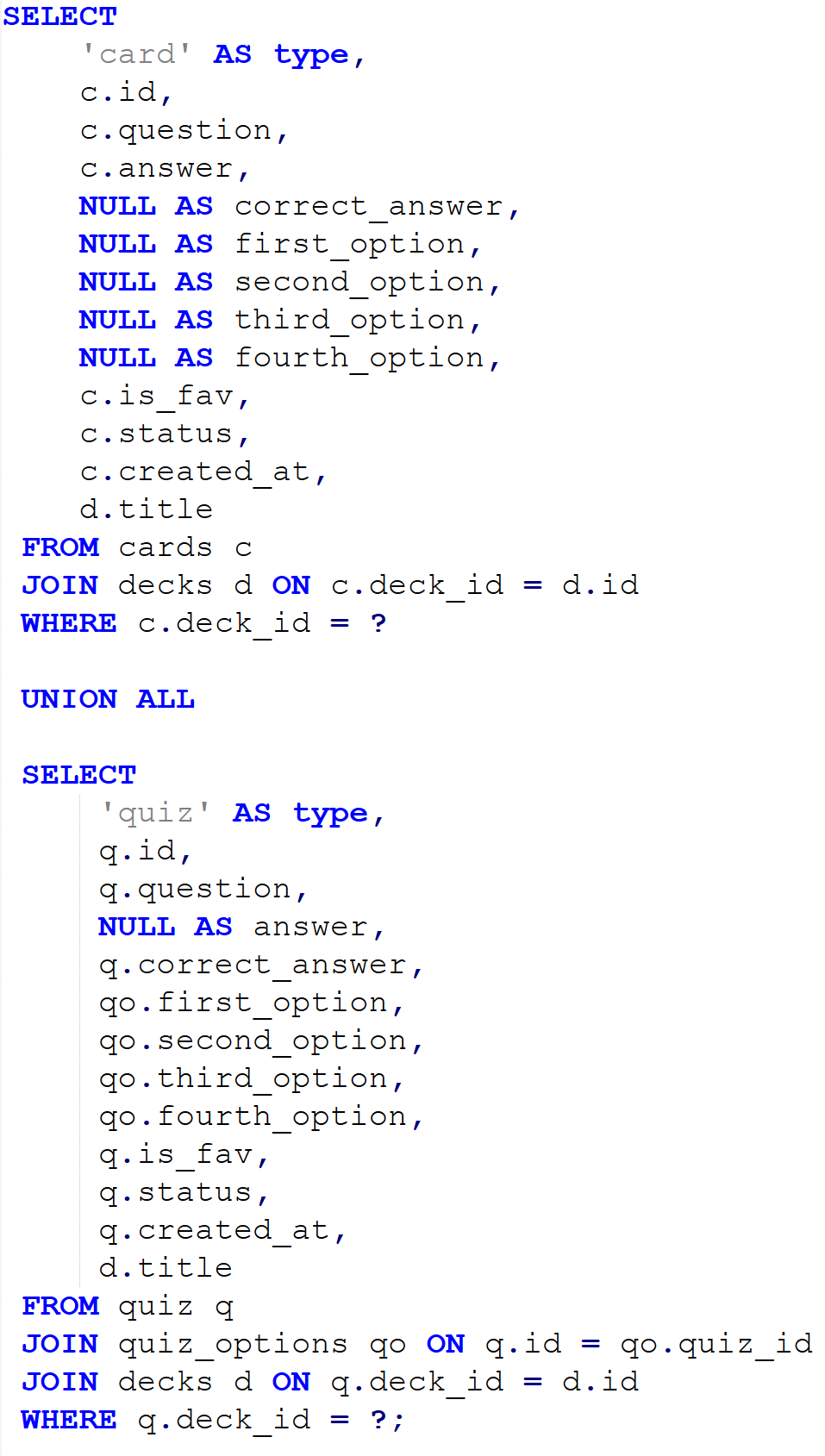
Index.xaml.cs Konstruktoren



Deck.xaml.cs routing auf Indexpage



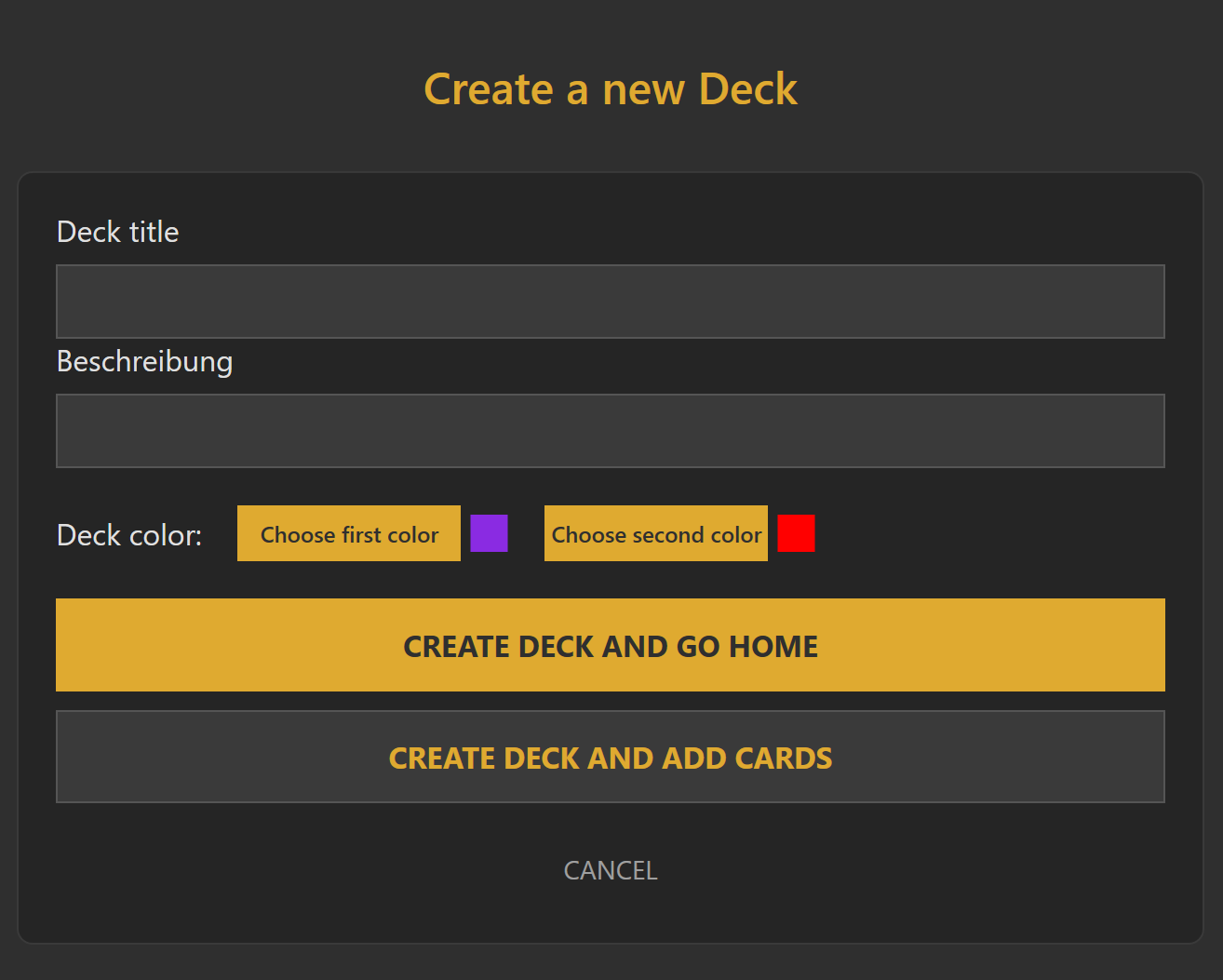
Deck.xaml UI mit dargestellten Karten



SQL-Abfrage für cards und quiz

### Codeerklärung Decks hinzufügen V2.2

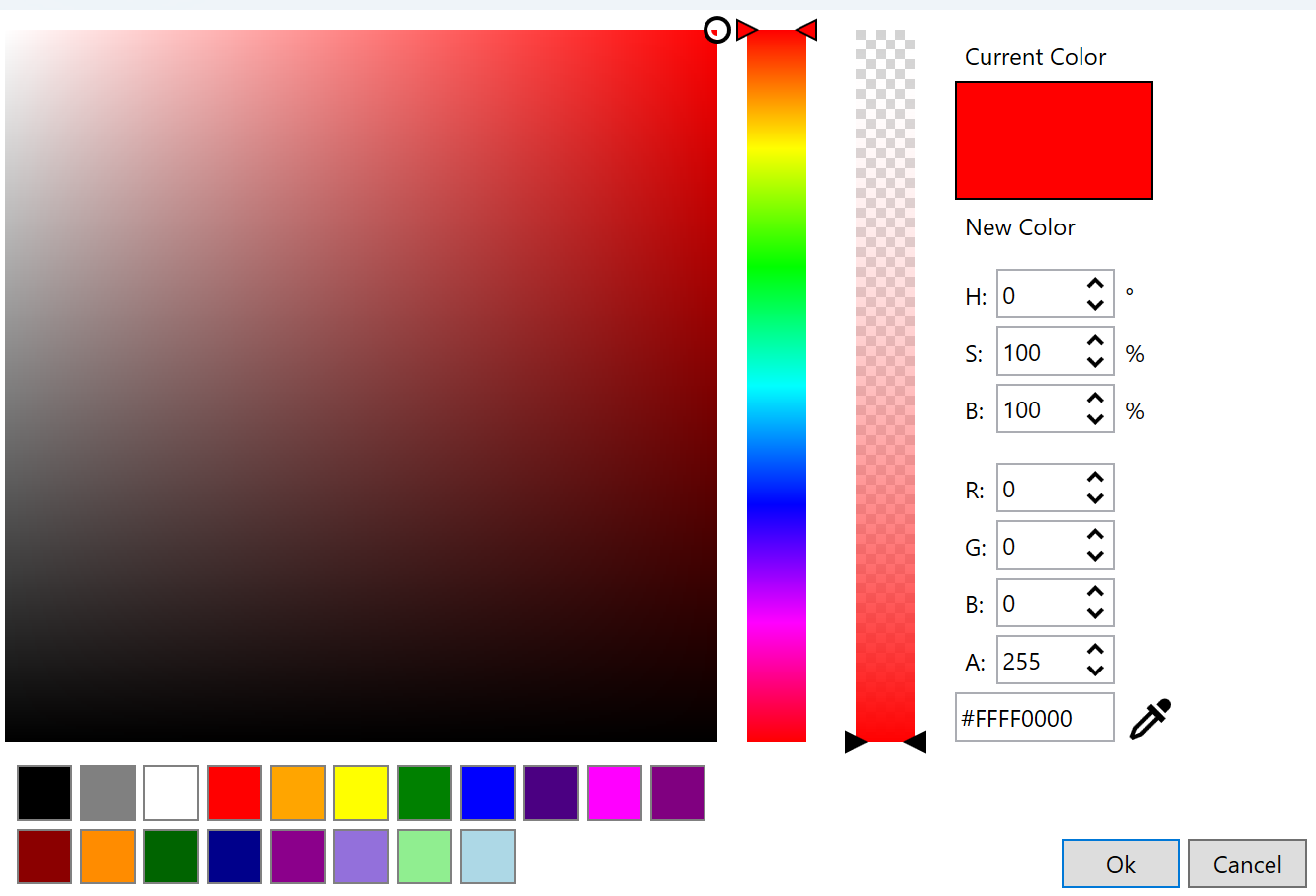
Auf einer weiteren zusätzlichen Seite können Decks hinzugefügt werden. Für die Decks können Farben, Titel und eine Beschreibung eingegeben werden.



Deck erstellen UI

#### CreateDeck.xaml und CreateDeck.xaml.cs

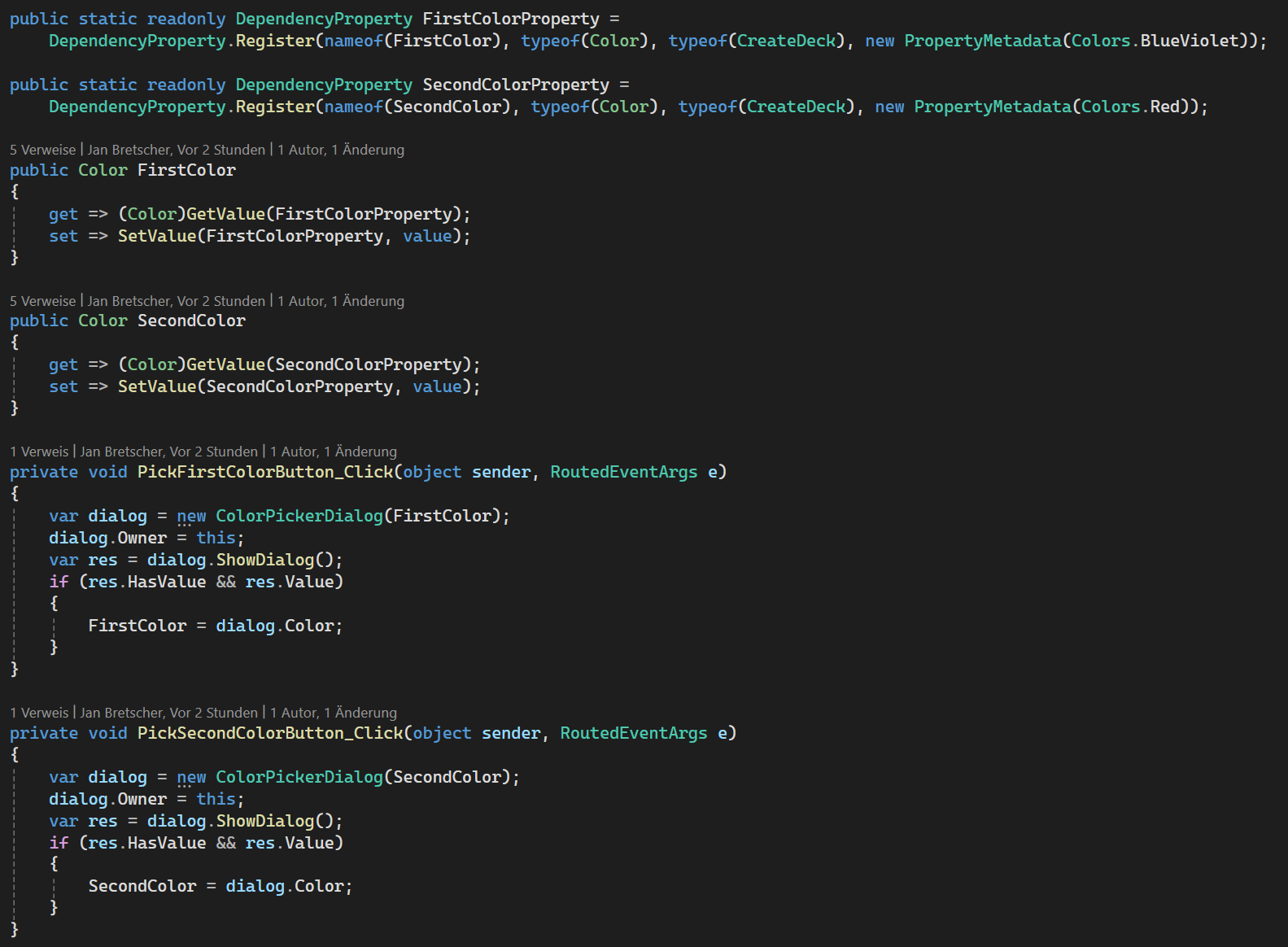
Farbauswahl



Colorpicker UI

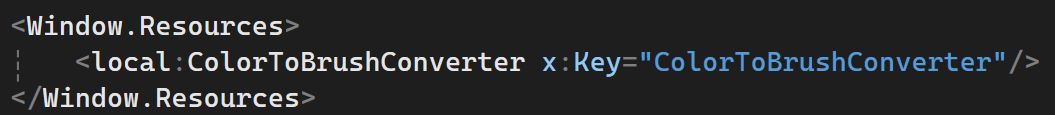
Ein Deck zu erstellen ist ziemlich simpel. Man muss einfach alle felder ausfüllen und Create Deck drücken. Jedes Deck hat zwei Farben, aus welchen ein Farbverlauf entsteht beim anzeigen auf der Indexseite. Für diese ColorPicker-funktion konnte ich einfach eine Library installieren. Ich verwende diese Bibliothek: <https://github.com/dsafa/wpf-color-picker>. Es war sehr einfach diese zu installieren, das sie mit dem NuGet-Packagemanager erhähltlich ist. Es hatte eine gute Dokumentation und ein gutes Beispiel, für die Implemenation.

Ich musste die Farbauswahl zweimal implementieren, da zwei separate Farben benötigt wurden. Beim Klicken auf den *Choose Color*-Button wird eine neue Instanz des ColorPickerDialog erstellt und das Dialogfenster geöffnet. Nach der Farbauswahl wird der entsprechende Setter mit dem gewählten Wert aktualisiert, der dann für weitere Verarbeitung genutzt werden kann. Als Standardwerte sind Violett für die erste und Rot für die zweite Farbe voreingestellt.

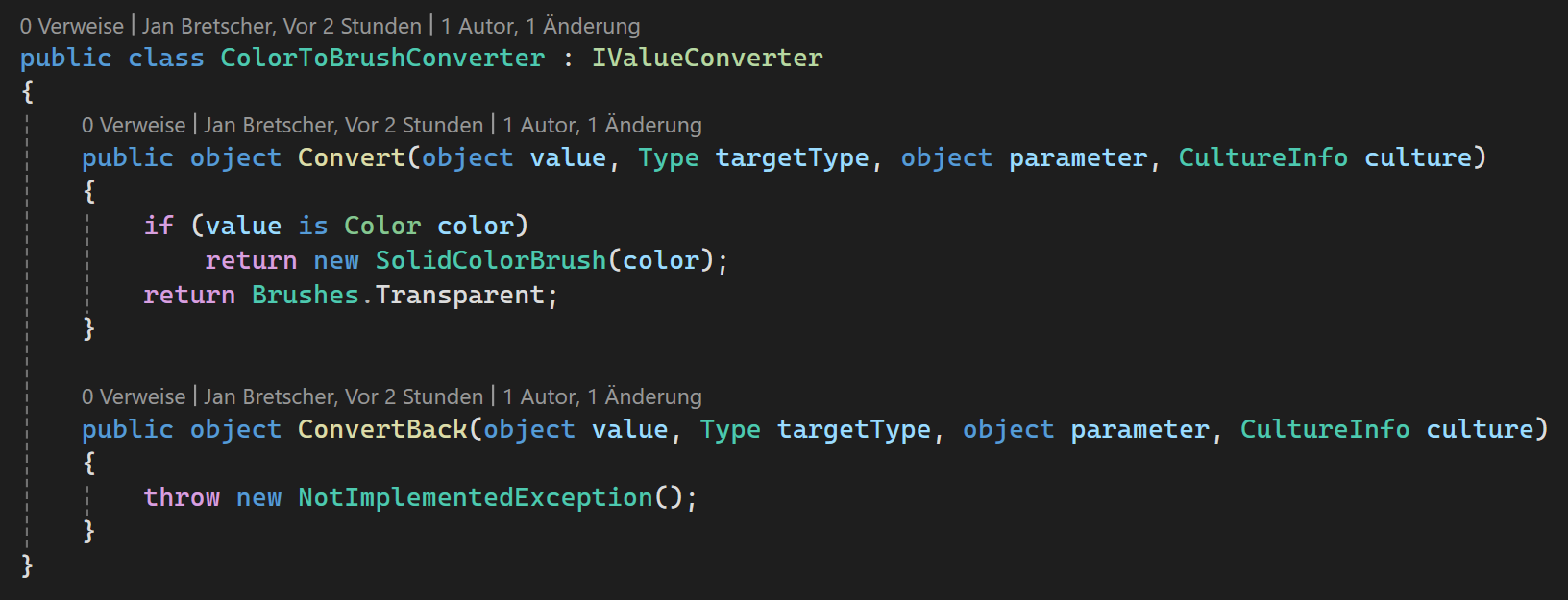


Code für colorpicker

Eine besondere Herausforderung bestand darin, die ausgewählte Farbe in der Benutzeroberfläche anzuzeigen. Dafür habe ich eine neue Klasse ColorToBrushConverter erstellt, die auf Elementen der Bibliothek basiert und von einem Beispiel in der Dokumentation inspiriert war. Im XAML-Frontend musste ich diese Klasse zunächst in den Window.Resources registrieren, um sie anschließend nutzen zu können. Damit ließ sich ein Rechteck mit der dynamisch ausgewählten Farbe füllen.



Xaml für Farbvorschau



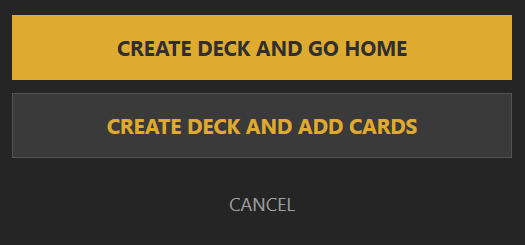
ColorToBrushConverter Klasse

Aktionsbuttons

Es gibt drei verschiedene Aktionsbuttons auf dieser Seite. Create & go home, create & add cards oder cancel. Wenn man das Deck created, wird die Methode createDeck aufgerufen, welche wie gewohnt einen Token anfordert, diesen Hashed und danach die AddDeck-Methode aufruft.

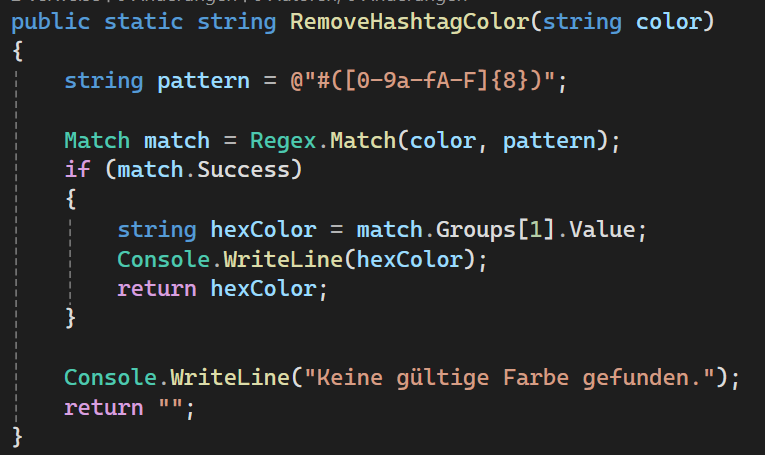


CreateDeck.xaml.cs createDeck()-Methode



Aktionsbuttons im UI

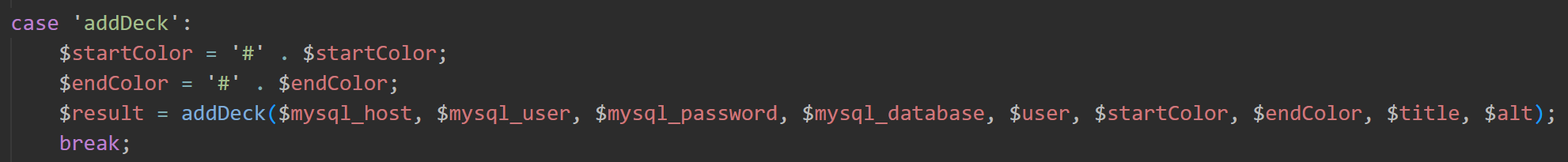
Ich habe hier sehr viele Parameter, welche ich mitgeben muss. Dies sind: addDeck, username, hashedToken, sessionID, firstColor, secondColor, title und Beschreibung. Es ist bestimmt nicht die sauberste Lösung mit diesen vielen Parametern, jedoch eine zuverlässige und sichere Möglichkeit. Mit tile.Text und alt.Text werden die Textfelder ausgelesen. Zu oberst im Try wird überprüft, ob das Textfeld title einen Inhalt hat. Wenn nicht, kommt ein Fehler-Window. Ich hatte grosse Probleme mit dem übertragen der Parameterwerte per http-Request. Dies lag daran, dass in der URL Hashtags (‘#’) und Underlines (‘\_’) nicht erlaubt waren. Da ich die Farben jedoch als Hexwert übertragen habe (Format: ‘#FFFFFFFF’) störte dies die URL. Bei einem # wird ein ‘anker’ (link) erwartet, wie er in HTML z.B. gegeben werden könnte. Ich musste also eine Lösung finden, um die Farben ohne den # zu übertragen, danach jedoch trotzdem richtig in der Datenbank abspeichern. Ich habe hierfür also eine neue Klasse erstellt, welche mit Regex den # vom String entfernt. Der Patern ist die Vorlage des Strings also zuerst ein # und danach 8 Zeichen von 0-9, a-f und A-F. Danach wird überprüft, ob der String mit der Vorlage übereinstimmt. Falls ja, dann wird das erste Zeichen entfernt (‘#’) und der neue String zurückgegeben.



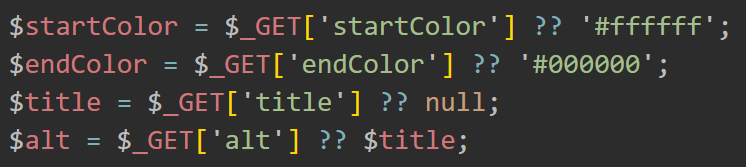
removeHashtagColor.cs Regex-Funktion

#### AddDecks in databaseRequest.php

Als die Farben noch mit einem # zuvor übergeben wurde, war die Ausgabe ab den Farben immer der Defaultwert. Wenn kein gültiger Parameter übergeben wurde, dann wird einfach der Defaultwert daneben verwendet. Da nun die Farben keine # mehr haben, müssen diese wieder angefügt werden. Diese mache ich mit einer einfachen Konkatination. Danach wird die Funktion addDeck aufgerufen, wo die SQL-Abfrage stattfindet.

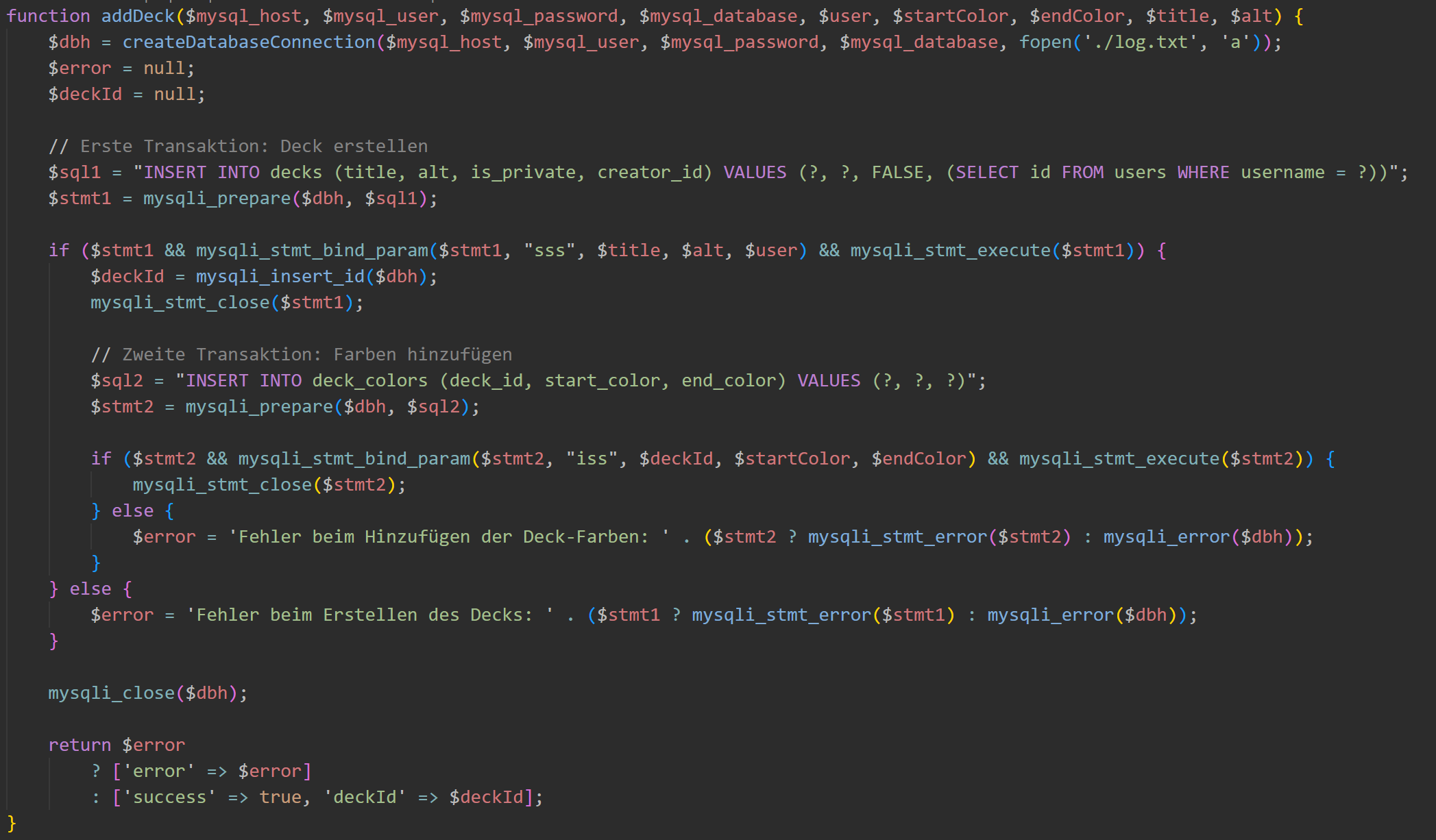


action = addDeck



Parametervariablen definieren

Um ein Deck hinzuzufügen müssen zwei Tabellen angepasst werden. Dies sind ‘decks’ und ‘deck\_colors’. Zuerst werden die Werte, welche als Parameter mitgegeben wurden, in decks hinzugefügt. Wenn dies erfolgreich ausgeführt werden konnte, werden die Werte in deck\_colors hinzugefügt. Wenn beides erfolgreich ablief, wird ein success und die deckId zurückgegeben.



databaseRequest.php addDeck-Funktion

# Arbeitsjournal

## 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](#_ERD_und_Struktur) 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.

## 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](#_CRUD-Operationen_&_Joins) 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.

## 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. Das Problem ist, dass ich meine Datenbank-Abfragen auf dem Client machen wollte. Hierfür gäbe es eine NuGet-Libaray, welche die Verbindung erleichtern würde. Da ich jedoch nur vom Lokalen Netzwerk auf die Datenbank zugreifen kann, ist dies keine Lösung, denn nicht jeder, der diese Applikation verwendet hat auch eine VPN-Verbindung in unser Heimnetzwerk. Ich habe heute sehr lange an diesem Problem getüftelt, jedoch ohne Erfolg. Als Test-Verbindung wollte ich einen Docker Container Aufsetzen und auf die Lokale Datenbank zugreifen. Auch dies ist jedoch gescheitert. Somit habe ich heute nichts Sinnvolles umsetzen können, was sehr frustrierend war. Mir kam dann jedoch die rettende Lösung. Es ist zwar etwas aufwändiger, jedoch ist es sauberer umgesetzt und funktioniert sicher. Ich werde nun ein Serverseitiges Script mit PHP schreiben, welches die Datenabfragen durchführt. Dieses File liegt auf dem Webserver, auf welchen man von extern zugreifen kann. Mit http-Requests werde ich dann die Daten von meiner WPF-Applikation an die Datenbank senden, und wieder zurück. Heute war kein guter Tag, und ich habe sehr viel Zeit verloren. Ich werde deshalb noch zu Hause weiter entwickeln, damit ich nicht zu sehr hinter den Zeitplan falle. Dass ich ihn nun einhalten kann, ist momentan jedoch sehr unwahrscheinlich.

## Tag 4 - 11.06.25

Ich Dokumentierte heute alles für die Client-Server-Kommunikation. Dies kostete mich einen hablen Tag, jedoch ist meine Dokumentatio nun auf einem ziemlich aktuellen stand. Das ganze Sicherheitskonzept und die Umsetzung des PHP-Scripts habe ich zu Hause begonnen und hier fertiggestellt. Es war Herausfordernd, ein gutes Sicherheitssystem zu entwickeln, damit nicht jeder die Datenbank abfragen kann. Ich bin nun jedoch sehr zufrieden mit meiner Lösung, welche ich gut ausgeklügelt finde. Genaueres zum Lösungsweg ist [oben](#_Lösung,_um_mit) beschrieben. Ich arbeitete danach an der Indexseite. Die Datenbankabfrage lief nun einwandfrei, was mir schonmal zeigt, dass mein Plan funktioniert hatte mit den Tokens und dem PHP-Script. Mein grosses Problem heute war das anzeigen der Decks im XAML-Window. Ich hatte die Ausgabe der Datenbank in einem JSON-String im Code-behind (MainWindow.xaml.cs) nun musste ich jedoch die Daten aus dem JSON im UI Darstellen. Dies war nicht ganz einfach und ich habe wieder einiges an Zeit zahlen müssen. Ich bin jedoch gut voran gekommen heute und konnte das Problem gut Lösen. Noch bin ich knapp im Zeitplan, ich habe dann jedoch nicht mehr viel Zeit für die nächsten Issues für morgen. Ich werde einfach weiter arbeiten und so gut wie möglich versuchen den Zeitplan wieder einzuholen.

## Tag 5 - 12.06.25

Ich habe heute viel für die logischen Abfragen für die Decks und Cards gemacht. Weiter habe ich auch die Projektbenennung etwas umgebaut. Nun hat das MainWindow.xaml keine Bedeutung mehr. Anstelle habe ich Index.xaml als erstes Window erstellt, das beim Applikaionsstart gestartet wird. Es gab heute einige Herausvorderungen, welche ich meistern musste. Das aufwändigste war, das JSON-Array richtig in die ListDecks Klasse zu schreiben und das ganze dann im UI Anzeigen zu können. Auch grosse Herausvorderungen waren die richtigen SQL-Queries um dir richtigen Daten abzufragen oder zu löschen. Ich denke, dass ich nun einiges neues lernen konnte und die Basis und das know how habe um die weiteren Seiten zu entwickeln. Ich habe beim Serverseitigen Script die delete-Funktion hinzugefügt, um Decks zu löschen. Ich hatte sehr lange einen Fehler mit Statuscode 500, ich wusste jedoch nicht genau wo im PHP-Code. Nach langem suchen sah ich meinen blöden fehler, dass ich einen Parameter beim Funktionsaufruf vergessen habe. Es war ein gemischtes Gefühl, als ich den Fehler erkannt habe. Zum einen war ich frustriert, dass ich so lange an etwas kleinem, einfachem gesucht habe, zum andern war ich erleichtert, dass ich nicht viel anpassen musste um dies zu fixen. Ich bin immernoch etwas hinter dem Zeitplan, denke jedoch, dass wenn ich so weiterarbeite und keine Zwischenfälle dazu kommen, werde ich nicht noch weiter nach hinten fallen

## Tag 6 - 13.06.25

Der heutige Tag war von Höhen und Tiefen geprägt. Ich bin zwar noch ein Stück weiter nach hinten gefallen, jedoch bin ich mit dem aktuellen Produkt bereits sehr zufrieden. Die neue Funktion createDeck funktioniert nun einwandfrei. Ich hatte hier einige Schwierigkeiten, welche ich lösen musste, wie mit dem HashTag, welcher die Parameter beim http-Request gestört hatte. Es dauerte sehr lange, um diesen Fehler zu erkennen, nun bin ich jedoch umso glücklicher, dass dies nun gut funktioniert und ich nun fürs nächste mal weiss worauf ich bei http-requests achten muss. Ich bin auch froh, dass der ColorPicker problemlos implementiert werden konnte, und dass dieser wunderbar funktioniert. Am Ende vom heutigen Tag habe ich viel erreicht und einige Probleme behoben. Backendseitig im PHP ist es nun kein grosser Aufwand, um weitere Datenbankabfragen-Funktionen zu erstellen. Ich kann die bestehenden verwenden und etwas anpassen. Von der Dokumentation bin ich auch auf einem sehr aktuellen Stand. Ich denke, dass ich meine Zeit gut einteile, dass ich immer wieder dokumentieren kann. Nun bin ich engültig hinter dem Zeitplan. Ich bin bei einem Issue erst in der Hälfte, welchen ich heute Mittag eigentlich abschliessen sollte. Ich Arbeite einfach normal weiter und werde gegen Schluss einpaar Issues aus zeitlichen Gründen streichen müssen. Ich werde auf jeden Fall darauf achten, dass die wichtigen Funktionen damit das Programm seinen Zweck erfüllt, abzuschliessen.

## Tag 7 - 18.06.25

## Tag 8 - 19.06.25

## Tag 9 - 20.06.25

## Tag 10 - 25.06.25

## Tag 11 - 26.06.25

## Tag 12 - 27.06.25

# Anhang

## Quellenangabe

Gemini

Deepseek

W3Schools

## Tools

MySQL

Visual Studio Code

Visual Studio 2022

Microsoft Word

Notepad++

Microsoft PowerPoint

DrawIO

OpenVPN

C#/WPF/.Net

## Checkliste