

Pflichtenheft

1 Softwarevoraussetzungen

- NuGet Packages:
 - Newtonsoft.Json (13.0.3)
 - Serilog (4.0.0)
 - Serilog.Sinks.Console (5.0.1)
 - Serilog.Sinks.File (5.0.0)
- .NET 6.0

2 Architektur

2.1 Klassen

- MultipleChoice
- YesNo
- TextInput
- FlashCard
- User
- Usermanager
- QuestionList
- Question
- Flashcards
- FlashcardService

2.1.1 Klassenhierarchie

Alle Frage-Klassen (MultipleChoice, YesNo, TextInput) haben eine Parent-Klasse "Question" von welcher sie einige Properties und Methoden erben.

2.1.2 Question

```
@startuml
class Question{
+ Type: string { get; set; }
+ Answerpressed: bool { get; set; }
+ btnNextQuestion: Button { get; set; }
+ textblockQuestion: TextBlock { get; set; }
- subject: string
+ Subject: string { get; set; }
+ Text: string { get; set; }
- string[2]: subjects
+ CorrectAnswer: string { get; set; }
+ LastUsed: DateTime { get; set; }
+ Counter: int { get; set; }
+ abstract Draw(Grid grid): void
```

```

+ CheckAnswer(string answer): void
+ abstract Copy(): void
}
@enduml

```


 Klassendiagram Question

2.1.3 Multiple Choice

```

@startuml
class MultipleChoice{
+ Answers: string[4] { get; set; }
+ ans1: Button { get; set; }
+ ans2: Button { get; set; }
+ ans3: Button { get; set; }
+ ans4: Button { get; set; }
+ MultipleChoice()
+ MultipleChoice(string text, string ans1, string ans2, string ans3, string ans4,
string correct, string subject)
+ EVENT Click(object sender, RoutedEventArgs e): void
}
@enduml

```

 Klassendiagram MultipleChoice

2.1.4 YesNo

```

@startuml
class YesNo{
+ yes: Button { get; set; }
+ no: Button { get; set; }
+ YesNo()
+YesNo(string text, string subject, string correctAnswer)
+ EVENT Click(object sender, RoutedEventArgs e): void
}
@enduml

```

 Klassendiagram YesNo

2.1.5 TextInput

```

@startuml
class TextInput{
+ WrongAnswer: string { get; set; }
+ submit: Button { get; set; }
+ textBoxAnswer: RichTextBox { get; set; }


```

```
+ TextInput()  
+ TextInput(string text, string subject, string answer, string wrongAnswer)  
+ EVENT Submit_Click(object sender, RoutedEventArgs e): void  
}  
@enduml
```

 Klassendiagram TextInput

2.1.6 FlashCard

```
@startuml  
class Flashcard{  
  English: string { get; set; }  
  German: { get; set; }  
}  
@enduml
```

 Klassendiagram Flashcard

2.1.7 User


```
@startuml  
class User{  
  + Username: string { get; set; }  
  + Password: string { get; set; }  
  + filepathvocable: string { get; set; }  
  + filepathuser: string { get; set; }  
  + Profilpicture: string { get; set; }  
  + QuestionsAnsweredMath: int { get; set; }  
  + QuestionsAnsweredGerman: int { get; set; }  
  + QuestionsAnsweredCorrectMath: int { get; set; }  
  + QuestionsAnsweredCorrectGerman: int { get; set; }  
  + StatisticsDraw()  
}  
@enduml
```

 Klassendiagram Flashcard


2.1.8 QuestionList

```
@startuml  
class QuestionList{  
  - questions: List<Question>  
  - random: Random  
  + CurrentSubject: string  
  + FilterBySubject(string subject): QuestionList  
}
```


```
+ Add(Question question): void
+ Remove(Question question): void
+ DeserializeFromJSON(): void
+ GetRandomQuestion(): Question
+ SerializeToJSON(): void
+ Shuffle(): void
}
@enduml
```

 Klassendiagramm QuestionList

```
@startuml
class UserManager{
- filePath: string
- filepathquestions: string
+ Questionlist: QuestionList
+ CurrentUser: User
+ UserManager(): void
+ SavedUsers(List<User> users): void
+ LoadUsers() List<User>
+ AuthenticateUser(string username, string password): User
+ RegisterUser(string username, string password): bool
+ logout(): void
+ ChangeProfilpicture(string newProfilepicture): void
+ ChangeUsername(string newUsername): void
- HashPassword(string password): string
}
@enduml
```

 Klassendiagramm UserManager

```
@startuml
class FlashcardService{
- filePath: string
+ FlashcardService(): void
+ LoadFlashcards(): List<Flashcard>
+ SaveFlashcards(List<Flashcard> flashcards): void
}
@enduml
```

 Klassendiagramm FlashcardService

2.

2.2 Aufbau der UI

- Ein MainWindow, welches immer angezeigt wird. Über dieses werden dann die jeweiligen Pages angezeigt.

- MainWindow:
 - Window, mit einem Menü um die einzelnen Pages anzeigen zu lassen, ist dauerhaft ersichtlich
- WindowAddQuestion:
 - Window, um Fragen zu erstellen/editieren
- PageHome:
 - Page, Titelseite, auf welcher der User eine kurze Beschreibung bekommt, was er tun kann/soll
- PageAufgabe:
 - Page, um die Aufgaben anzuzeigen und zu beantworten
- PageEnglisch:
 - Page, auf welcher der user Vokabeln üben/anzeigen/manipulieren kann
- PageLogin:
 - Page, wenn der User sich einloggen will
- PageSign:
 - Page, wenn der User sich registrieren will
- PageNewQuestion:
 - Page, um die derzeitige Fragenliste des Users anzuzeigen und zu manipulieren (Fragen hinzufügen, editieren, löschen)
- PageEinstellungen:
 - Page, auf welcher der User sein Profil anpassen kann und auch das Theme (Darkmode/Lightmode) ändern kann
- PageProfile:
 - Page, auf welcher der user sein Profil und seine Statistiken einsehen kann

3 Umsetzung der Anforderungen

3.1 Organisatorisch

- zweier Gruppe: Bilal Ensar Bugday, Marlon Pichler
- Arbeit im Unterricht und zubhause (siehe GIT-Repository)

3.2 Technischer Inhalt

3.2.1 Must Haves

- Klassen:

- Wir haben Klassen verwendet (siehe Punkt 2.1 für weitere Informationen)
- Collections:
 - Wir haben in vielen Klassen/Funktionen mit Listen/Arrays gearbeitet.
 - Wir haben das JSON-Format für das Speichern von Daten verwendet. Dies basiert auf Dictionaries.
- Graphische Darstellung von Objekten:
 - Wir haben, um unser Programm so intuitiv und effizient wie möglich zu gestalten, in verschiedensten Pages/Windows Objekte graphisch dargestellt.
- Serialisierung:
 - Wir haben unsere Daten lokal im JSON-Format abgespeichert.
 - Dadurch konnten wir:
 - den Usern das permanente Verändern ihres Benutzernamen oder des Profilbildes ermöglichen
 - den Usern das Erstellen ihrer eigenen, individuellen Frageliste und Vokabelliste ermöglichen
- Unterformulare:
 - Durch unsere permanente *Verwendung von Unterformularen und Pageswitches* konnten wir den Usern die intuitivste und bedienungsleichteste Benutzeroberfläche bieten.
- Logging:
 - Wir haben unser Projekt so effizient und zielorientiert wie möglich geloggt um eine gute Debug-Möglichkeit und Problembehandlung zu sichern
 - Dabei haben wir unseren Logger so designed, dass dieser täglich neue Logging-Dateien erstellt, diese an diesem Tag befüllt und nach sieben Tagen wieder löscht.

3.2.2 Nice To Haves

- Menüzeile:
 - Wir haben aufgrund der Intuitivität in unserer Anwendung eine selbstprogrammierte Menüspalte, welche immer sichtbar ist und auf welcher man einfach zwischen den Pagen navigieren kann
- Verwendung von externen Bibliotheken:
 - Wir haben in unserem Programm (siehe Punkt 1) sowohl für das Loggen, als auch für die Serialisierung NuGet Packages verwendet.
 - Dies hat uns sehr weitergeholfen, da wir nun nicht selbst die Serialisierung in das JSON-Format übernehmen mussten
- Vererbung:
 - Um unsere Klassenstruktur (siehe Punkt 2.1) so effizient wie möglich zu gestalten, haben wir die "Vererbung von Klassen" gelernt/benutzt.
 - Dies hat uns massiv weitergeholfen, da wir dadurch alle unsere Fragen in eine große Fragenliste speichern konnten. Ohne Vererbung wäre das ganze Projekt sehr viel komplexer und unübersichtlicher geworden

3.3 Beschreibung der Anwendung

Erstmal haben wir ein "MainWindow", welche das Grundkonstrukt unserer Anwendung bildet. Auf dieser ist eine "Menüspalte", über welche man in der gesamten Anwendung navigieren kann. Auf diesem "MainWindow" werden dann die einzelnen Pages geladen. Man startet immer auf der PageHome, auf welcher

man aufgefordert wird, sich anzumelden oder sich zu registrieren. Wenn versucht wird zu diesem Zeitpunkt eine andere Page zu öffnen, bekommt man eine Mitteilung, dass man sich anmelden/registrieren muss. Bei einem Klick auf "Anmelden"/"Registrieren" wird man auf die jeweilige Page weitergeleitet und kann sich anmelden/registrieren.

Sobald man dies getan hat, stehen einem alle Wege offen. Man kann mit nur einem einzigen Klick in der Menüspalte seine Fähigkeiten in Deutsch/Mathe durch Fragen von verschiedenen Fragetypen verbessern, seine Englischkenntnisse mit Vokabeln verbessern, seine Fragenliste anpassen, sein Benutzerkonto verändern, seine Statistiken einsehen,...

4 Mögliche Probleme und ihre Lösung

- Natürlich hat bei unserem Projekt nicht alles auf Anhieb geklappt und wir hatten allfällige Schwierigkeiten. Spezifische Beispiele und deren Lösungen werden hier näher besprochen

4.1 Serialisierung - Einheitliches Laden/Speichern von Fragen

- Speziell bei der Serialisierung hatten wir eine recht große Schwierigkeit, welche uns einige Gehirnzellen und Arbeitszeit abverlangt hat.
- Wir konnten zwar recht schnell Fragen Laden, jedoch wurde es beim Speichern um einiges schwerer:
 - Wir konnten die Fragen zwar einmal Deserialisieren, jedoch nach dem Serialisieren nicht mehr Deserialisieren
 - Dies lag daran, dass unser Serialisierter Text eine andere Struktur benutzte, welche wir nicht deserialisieren konnten
 - Lösung:
 - Man kann dem Serializer von Newtonsoft.Json, welchen wir verwendet haben, genau sagen, was er von einem Objekt serialisieren soll und was nicht

4.2 Userspezifische Fragedateien

- Anfangs hatten wir eine Frageliste, welche dann bearbeitet werden konnte.
- Das Problem:
 - Jeder User bearbeitet dieselbe Frageliste.
- Die Lösung:
 - Beim erstellen/registrieren eines Benutzers wird automatisch eine eigene JSON-Datei erstellt, welche anfangs die originale Frageliste enthält.
 - Zudem wird der derzeit angemeldete User als statische Variable gespeichert
 - Dadurch kann die Frageliste des Users und die dazugehörige Datei angepasst werden, ohne die anderen Fragedateien der anderen User zu manipulieren
 - Zudem kann auch jederzeit der Dateipfad bzw. der Dateiname, in welchem die Fragen des Benutzers liegen, verändert werden, falls der Username geändert wird