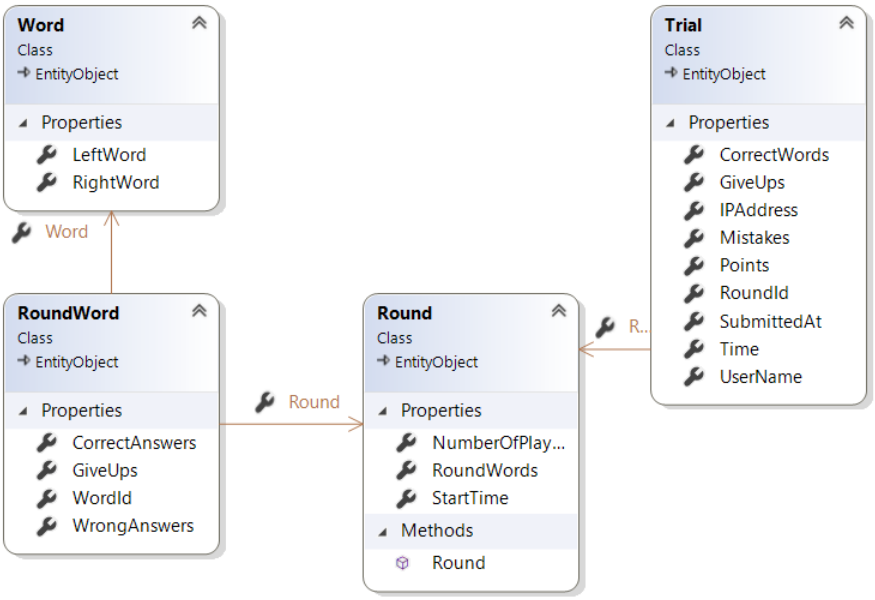
Angular - Vokabelquiz

In den nächsten Wochen soll eine größere Anwendung in Form eines Vokalquiz erstellt werden. Dabei sollen in der letzten Ausbaustufe mehrere „SpielerInnen“ in Vergleichswettkämpfen dieselben zufällig gewählten Vokabeln übersetzten.

1.) Dazu ist zunächst eine EF-Datenbankanwendung mit folgenden Entitäten zu erstellen:



Funktionsweise / Datenbank

Die Tabelle Word soll alle möglichen Vokabeln beinhalten (LeftWord = Deutsch und RightWord = Englisch).

Eine Runde beinhaltet immer eine bestimmte Anzahl von zu übersetzenden Vokabeln. In der Tabelle Round wird daher die Anzahl der Vokabeln und die Startzeit (=wann wurde die Runde erstellt) gespeichert.
Das Feld NumberOfPlayers soll später die Anzahl der Spieler beinhalten, welche diese Runde gespielt haben.

Wenn während des Spiels eine neue Runde erstellt wird und dabei die Anzahl der gewünschten Vokabeln eingegeben wurde, sollen aus der Word Tabelle zufällige Vokabeln für diese Runde ausgewählt werden.
Gespeichert werden die gewählten Vokabeln für diese Runde dann in der Tabelle **RoundWord**. In dieser Tabelle wird dann auch noch gespeichert, wie oft das jeweilige Wort in dieser Runde von allen Spielern in Summe richtig übersetzt wurde (CorrectAnswers), wie oft

es falsch übersetzt wurde (WrongAnswer) und wie oft die Spieler dieses Wort „übersprungen“ haben (GiveUps).

Wenn ein Spieler mitmachen möchte, werden diesem hintereinander alle Vokabeln der aktiven Runde (immer die zuletzt angelegte Runde), zur Übersetzung gegeben. Weiters wird die Zeit gestoppt, die der Spieler für die Übersetzung aller Vokabeln benötigt.
Wenn man keine Ahnung hat, wie die Übersetzung lautet, kann man ein Vokabel aus überspringen.

Das fertige Ergebnis wird in der Tabelle Trial gespeichert.

Implementierung EntityFramework-Projekt

Im IUnitOfWork und UnitOfWork soll eine Methode **FillDb** deklariert und implementiert werden. Diese Methode soll die Datenbank neu erstellen und alle möglichen Wörter aus der csv Datei „words.csv“ eingelesen. LeftWord und RightWord sind dabei mit Tabulator getrennt. Das Trennzeichen kann bei unserer bekannten Hilfsmethode „ReadStringMatrixFromCsv“ angegeben werden (\t = Tabulator)

```
string[][] csvWords = MyFile.ReadStringMatrixFromCsv("words.csv",false, '\t');
```

Beachte: Wenn es mehrere richtige Übersetzungen gibt, so sind diese mit Beistrich getrennt. In der Tabelle Word wird aber der gesamte Text (inkl. Beistrichen) für LeftWord und RightWord gespeichert. (Erst bei der Auswertung der Benutzereingabe wird dies dann relevant → es genügt dann, wenn der Benutzer eines der möglichen richtigen Wörter eingegeben hat)

Beispiel-Zeile der csv-Datei:
deren, dessen, wessen *which one's, of which, whose*

Einfach unterstrichener Text wird in LeftWord gespeichert. Doppelt unterstrichener Text wird in RightWord gespeichert.

Wenn der Benutzer diese Vokabel übersetzen soll. Wird ihm **LeftWord** angezeigt. Er muss jetzt **einen** der durch Beistrich getrennten Begriff von **RightWord** eingeben. D.h. wenn er „whose“ eingibt wäre es z.B. richtig.

Durch diese Vorgehensweise, kann später auch relativ einfach vom Englischen ins Deutsche übersetzt werden.

UnitTest:

Erstelle ebenfalls ein Unit-Test-Projekt in dem die Methode FillDb (Klasse UnitOfWork) überprüft wird. Im weiteren Verlauf, sollen auch UnitTests für die anderen Repository-Methoden erstellt werden (eine Testklasse je Repository).

Beachte, dass auch im Unit-Test Projekt appsettings.json Datei für den Connection String benötigt wird!

In jeder Unit-Test-Klasse kann zur Initialisierung der Tests eine ClassInitialize Methode gestartet werden (z.B. FillDb aufrufen, um die Datenbank auf den Startzustand zu setzen):

```
namespace Vocabulary.TestPersistence
{
    [TestClass]
    0 references
    public class UnitOfWorkTest
    {
        //
        //You can use the following additional attributes as you write your tests:
        //
        //Use ClassInitialize to run code before running the first test in the class
        [ClassInitialize()]
        0 references
        public static void MyClassInitialize(TestContext testContext)
        {
            using IUnitOfWork uow = new UnitOfWork();
            uow.FillDb();
        }
```

Für den Test von UnitOfWork.FillDb werden Sie eine Query-Methoden im Word-Repository benötigen.
Implementieren Sie dazu eine Methode „GetAll()“, welche alle Datensätze der Tabelle Word, sortiert nach der Id liefert.

Repositories

Folgende Methoden sollen in den Repositories implementiert werden (inkl. Deklarationen im entsprechenden Interface!)

Ergänze auch entsprechende Unit-Tests um die Methoden sinnvoll zu testen!

Word-Repository:

```
/// <summary>
/// Liefert alle Wörter sortiert nach Id
/// </summary>
/// <returns></returns>
public List<Word> GetAll()


/// <summary>
/// Liefert alle Vokabeln aus einer übergebenen Runde
/// </summary>
/// <param name="round_id"></param>
/// <returns></returns>
 public List<Word> GetWordsByRoundId(int round_id)


/// <summary>
/// Liefert eine Liste zufälliger Vokabeln (Word).
/// Achtung! Es wird davon ausgegangen, dass die Id´s in
/// der Datenbank lückenlos aufsteigend vergeben wurden!
/// Die gewünschte Anzahl an Vokabeln wird als Parameter übergeben.
/// </summary>
/// <param name="count"></param>
```

```
        /// <returns></returns>
        public List<Word> GetRandomWords(int count)
```

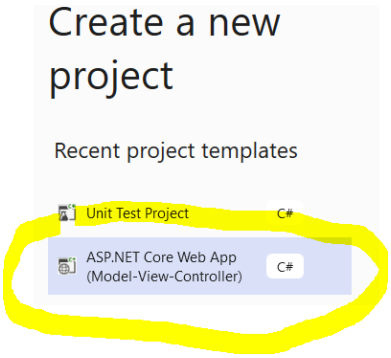
Round-Repository:

```
        /// <summary>
        /// Erstellt einen neue Runde und setzt die Startzeit auf die aktuelle Zeit
        /// Dann werden 10 zufällige Vokabeln gewählt (id´s zufällig wählen) und
        /// erstellt die entsprechenden Einträge in der RoundWords-Table
        /// </summary>
        public void GenerateNewRound()


        /// <summary>
        /// Liefert die aktuelle Runde (ist immer die zuletzt erstellt Runde)
        /// </summary>
        /// <returns></returns>
        public Round GetActualRound()
```

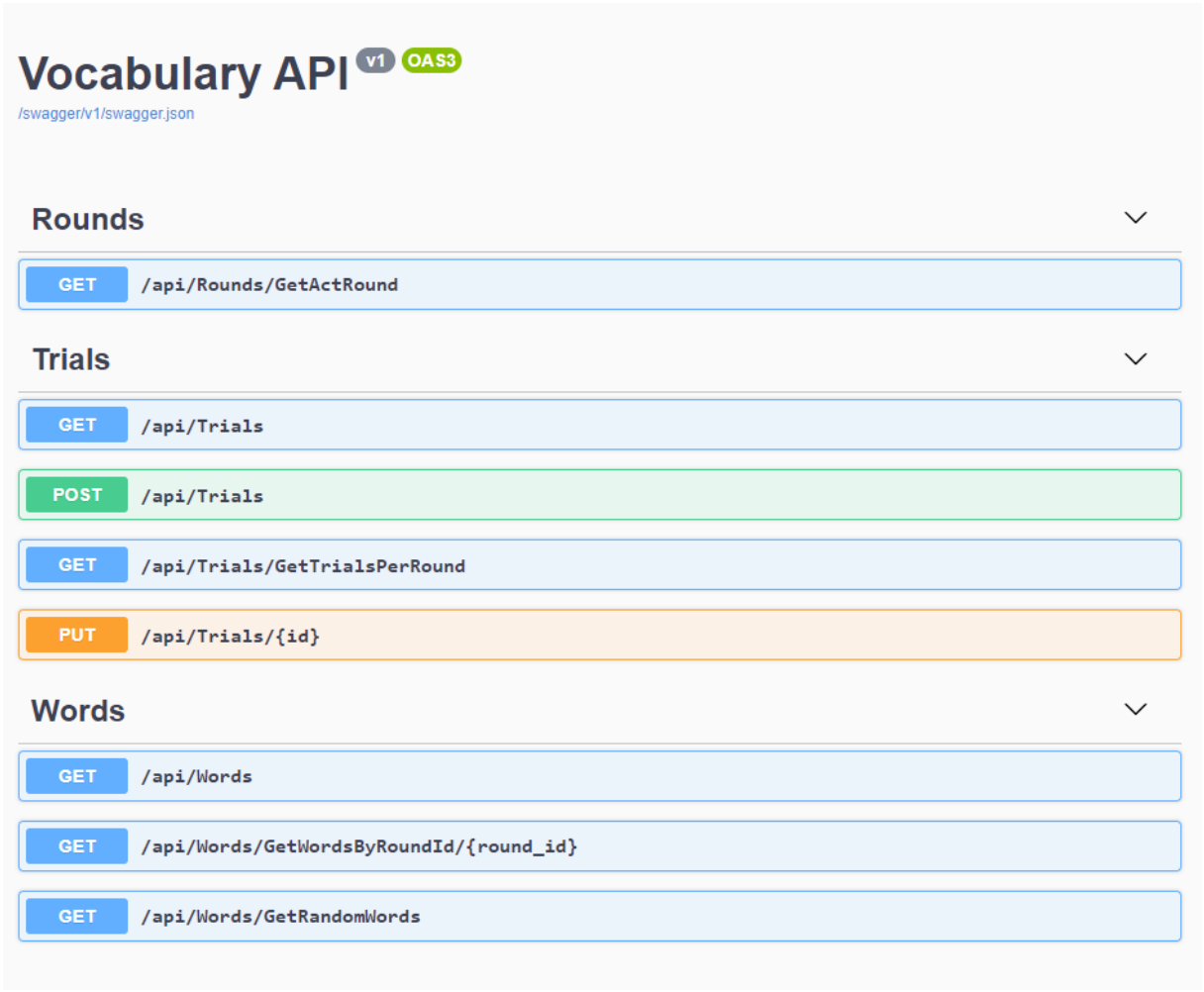
Trial-Repository:

```
        /// <summary>
        /// Liefert den Trial mit der übergebenen id
        /// </summary>
        /// <param name="id"></param>
        /// <returns></returns>
        public Trial GetById(int id)


        /// <summary>
        /// Speichert einen Spielergebnis-Datensatz eines Spieles
        /// (Ergebnis für eine Runde).
        /// </summary>
        /// <param name="trial"></param>
        /// <returns></returns>
        public bool Insert(Trial trial)


        /// <summary>
        /// Aktualisert einen Spielergebnis-Datensatz
        /// wenn dieser bereits existiert.
        /// </summary>
        /// <param name="trial"></param>
        /// <returns>true wenn erfolgreich, false wenn Datensatz nicht
       gefunden</returns>
        public bool Update(Trial trial)


        /// <summary>
        /// Liefert alle aktuellen Spielergebnisse = Trials
        /// einer bestimmten Runde
        /// </summary>
        /// <param name="roundId"></param>
        /// <returns></returns>

        public List<Trial> GetTrialsPerRound(int roundId)
```

Implementierung WebAPI Projekt



Implementiere einen API-Server, welcher folgende Service zur Verfügung stellt:



Binde auch den **Swagger** ein und teste die einzelnen Services (vgl. PowerPoint Präsentation „Swagger“).

Implementierung Angular-Projekt – Ausbaustufe 1

- Angular App fragt nach Namen
- 15 zufällige Vokabel werden geladen und deutscher Text angezeigt (je Vokabel eine Komponente)
- Neben jedem Vokabel kann die Übersetzung eingegeben werden
- Button für Übermitteln neben jeder Vokabel → wenn richtig → grünes Häkchen und Eingabefeld disabled
- Wenn falsch → rotes X (es kann dann erneut eingegeben werden oder auf eigenen „Aufgeben“ Button geklickt werden)
- Statistik: Anzahl Fehler, Anzahl gelöste Wörter, Anzahl nicht gelöste Wörter mitzählen und anzeigen