

AGH

AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

Wydział Fizyki i Informatyki Stosowanej

Praca inżynierska

Damian Łączak

kierunek studiów: **informatyka stosowana**

Aplikacja internetowa do nauki języka

Opiekun: **dr hab. inż. Tomasz Bołd**

Kraków, styczeń, 2018

Oświadczam, świadomy odpowiedzialności karnej za poświadczanie nieprawdy, że niniejszą pracę dyplomową wykonałem osobiście i samodzielnie i nie korzystałem ze źródeł innych niż wymienione w pracy.

.....
(czytelny podpis)

Merytoryczna ocena pracy przez opiekuna:

Merytoryczna ocena pracy przez recenzenta:

Spis treści

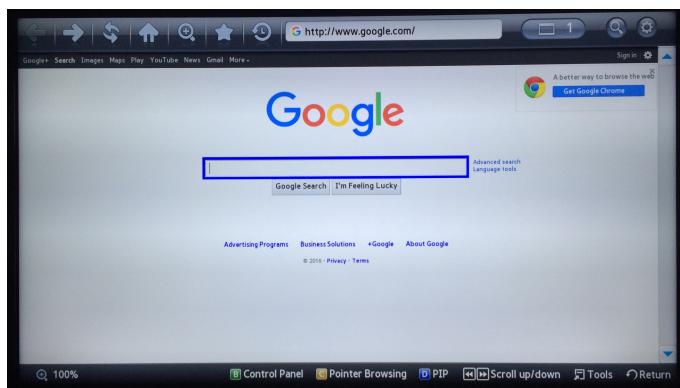
1. Wstęp	1
1.1. Cel aplikacji	2
2. Przyswajanie wiedzy	3
2.1. Interwały potwórzeń	3
2.2. Efekt przewagi obrazów	4
3. .NET	5
3.1. Implementacje .NET	5
3.1.1. .NET Core.....	6
3.1.2. .NET Framework.....	6
3.1.3. Mono.....	6
3.1.4. Universal Windows Platform (<i>UWP</i>)	6
3.2. C#	6
4. ASP.NET MVC	8
4.1. Model-Widok-Kontroler	8
4.2. Razor.....	9
4.3. Wzorzec Repozytorium	9
4.4. Jednostka Pracy.....	11
4.5. Mapowanie obiektowo-relacyjne.....	12
4.6. Wstrzykiwanie zależności	13
5. Microsoft SQL Server	15
5.1. Język zapytań	15
6. Testy	16
6.1. Testy Jednostkowe	16
6.2. Testy Integracyjne.....	16
7. Kaskadowe Arkusze Styli	18
7.1. Syntaktyczne Doskonałe Arkusze Styli	18
8. Scala	20

8.1. Javascript	20
8.2. Scala.js	22
8.2.1. sbt.....	22
8.2.2. Proces budowania projektu.....	22
8.2.3. Hello World.....	25
9. HTML.....	28
10. Opis działania aplikacji.....	29
10.1. Proces nauczania.....	29
10.1.1. Trening	29
10.1.2. Powtórka	30
10.1.3. Siła	32
10.1.4. Interwały czasowe	33
10.1.5. Wykrywanie poprawności odpowiedzi.....	34
10.1.6. Możliwe odpowiedzi	35
10.1.7. Obrazki	35
10.1.8. Dostosowanie wzorów	35
10.1.9. Wirtualny uczeń	36
10.1.10. Wyniki symulacji	37
10.2. Narzędzia administratorskie.....	38
10.3. Diagram przypadków użycia	39
11. Kwestie techniczne.....	40
11.1. Struktura programu	40
11.1.1. Repozytoria	40
11.1.2. Kontrolery.....	43
11.1.3. Serwisy.....	46
11.2. Struktura rozwiązania	48
11.2.1. Management	48
11.3. Baza danych.....	49
11.3.1. Informacje dodane przez ASP.NET MVC	51
11.3.2. Obsługa treningu.....	53
11.3.3. Obsługa powtórki	53
11.3.4. Informacje o Fiszach	53
11.3.5. Upload plików	54
11.4. Integracja Scala.js z ASP.NET MVC	54

11.4.1. Uruchamianie skryptu	55
11.5. Implementacje niektórych rozwiązań	56
11.5.1. Sesja użytkownika.....	56
11.5.2. Kolejka pytań	57
11.5.3. Podobieństwo wyrazów.....	58
11.5.4. Atrypty obiektów	58
12.Podsumowanie	63
12.1. Wnioski	63
12.1.1. Scala.js.....	63
12.2. Możliwe ulepszenia.....	64

1. Wstęp

Coraz większą popularność w dzisiejszym świecie zdobywają aplikacje internetowe. Ich nie-wątpliwą zaletą jest możliwość uruchomienia na dowolnym urządzeniu posiadającym internet i przeglądarkę internetową. Jeszcze do niedawna były to jedynie komputery i urządzenia mobilne, lecz wraz ze wzrostem popularności intelligentnych technologii coraz więcej produktów codziennego użytku potrafi łączyć się z internetem. Do takich urządzeń zaliczają się kuchenki mikrofalowe, lodówki, telewizory i wiele innych.



Rys. 1.1. Przeglądarka internetowa wbudowana w telewizor.

Środowisko internetowe gwarantuje nie tylko większą ilość użytkowników, lecz także mniejsze koszty uruchomieniowe danej aplikacji po stronie klienta. Aplikacje internetowe nie wymagają żadnego dodatkowego oprogramowania, ponieważ większość dzisiejszych urządzeń ma już je wbudowane. Dodatkowo są one uruchamiane na zewnętrznym serwerze co powoduje, że obciążenie obliczeniowe aplikacji jest przenoszone na dostawcę usługi, a nie na użytkownika.

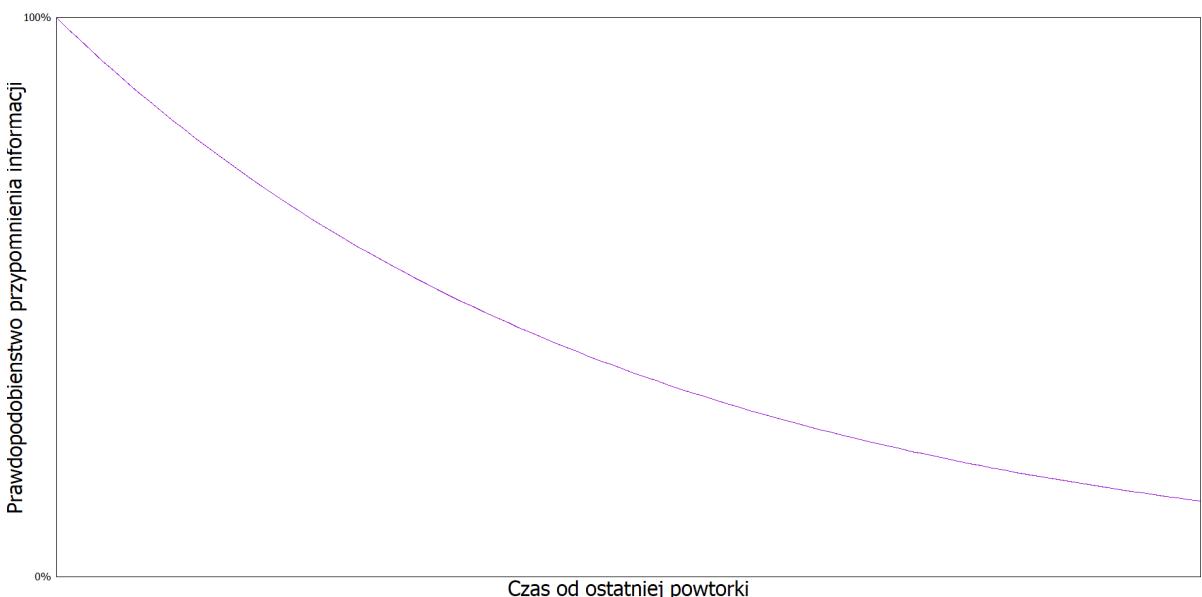
Dzięki temu klient nie ponosi większych kosztów wynikających z użycia serwisu. Nie musi wymieniać sprzętu, instalować sterowników, pobierać aktualizacji.

1.1. Cel aplikacji

W niniejszej pracy został ukazany projekt aplikacji internetowej, której zadaniem jest umożliwienie nauki wybranych słówek danego języka obcego za pomocą wirtualnych fiszek, które będą prezentowane użytkownikowi w nieregularnych odstępach czasowych obliczanych na podstawie poprawności udzielonych odpowiedzi. Program podczas odpytywania użytkownika o dane słowo używa obrazów powiązanych z omawianym wyrażeniem. Ważnym założeniem aplikacji w trakcie procesu nauczania było jak najmniejsze wykorzystanie innego języka niż nauczany.

2. Przyswajanie wiedzy

Hermann Ebbinghaus[1] (1850-1909) był niemieckim psychologiem, który jako jeden z pierwszych zajmował się tematyką eksperimentalnej psychologii związanej z przyswajaniem wiedzy. Jednym z jego pierwszych eksperymentów było stworzenie testu, podczas którego uczył się zestawu 20 sylab, które były bezsensowne ze względu na fakt, iż w jego języku nie występowały żadne słowa, które ich używały. Eksperiment ten pozwolił mu skonstruować pierwszą na świecie krzywą zapominania. Miała ona charakter eksponencjalny. Możemy z niej wywnioskować, iż podczas początkowego okresu zapominania tracimy najwięcej zapamiętanych informacji.



Rys. 2.1. Eksponencjalna krzywa zapominania.

Trzeba także zwrócić uwagę na fakt, iż w rzeczywistości[2] mózg nie jest w stanie przyswoić danych informacji w 100% po zakończeniu nauki.

2.1. Interwały powtórzeń

W celu jak najlepszego zapamiętania wyuczonych informacji należy, poza odpowiednim programem nauczania, zwrócić uwagę na fakt, w jakich interwałach czasowych powtarzany jest

przerabiany materiał[2]. Interwał musi być wystarczająco krótki, aby zapobiec zapominaniu i wystarczająco długi, aby zbyt często nie przyswajać powtarzanego materiału.

2.2. Efekt przewagi obrazów

Efekt przewagi obrazów odnosi się do zjawiska, w którym zdjęcia i obrazy mają większą szansę na bycie zapamiętanymi niż ich zapisane odpowiedniki.

Istnieje wiele badań [3][4], które ukazują pozytywny wpływ metod nauczania z wykorzystaniem obrazków.

Istnieją także badania, które zjawisko pokazują w zupełnie innym świetle, pokazując brak pozytywnego wpływu prezentacji obrazków na nauczanie języków obcych[5].

Wewnątrz projektu starano się wykorzystać powyższy efekt, wymagając na użytkowniku zapamiętania, a następnie przypomnienia słowa w języku obcym wykorzystując do tego celu obrazki. Takie postępowanie w najgorszym wypadku będzie na równi efektywne z tradycyjnym.

3. .NET

.NET jest platformą programistyczną umożliwiającą pisanie nowoczesnych aplikacji w językach wysokiego poziomu, do których zalicza się m.in C#, VB oraz F#. Platforma ta wyróżnia się tym iż:

- Pozwala na użycie wielu języków programowania podczas pisania naszych programów.
- Ma zaimplementowane mechanizmy do obsługi operacji asynchronicznych i współbieżnych.
- Można ją stosować na różnych platformach, które posiadają środowisko wykonywalne .NET.

Wszystkie języki używane w platformie .NET kompilowane są do Wspólnego Języka Pośredniego (po ang. *Common Intermediate Language*), który następnie jest tłumaczony na kod bajtowy i wykonywany za pomocą środowiska wykonywalnego danej implementacji .NET. CIL wyglądem przypomina język Assembler. Jest on od niego o wiele bardziej rozbudowany i łatwiejszy w zrozumieniu.

```
.assembly HelloWorld
.class auto ansi HelloWorldApp
{
    .method public hidebysig static void Main() cil managed
    {
        .entrypoint
        .maxstack 1
        ldstr "Hello world."
        call void [mscorlib]System.Console::WriteLine(string)
        ret
    }
}
```

Listing 3.1. Przykładowy kod aplikacji "Hello World" w języku CIL

3.1. Implementacje .NET

Każda aplikacja .NET jest uruchamiana na jednej z implementacji .NET.

Od roku 2016 wprowadzono .NET Standard - wspólny zestaw API, który każda z implemen-

tacji musi posiadać. Pozwala to na pisanie i używanie bibliotek programistycznych w różnych środowiskach .NET.

Istnieją aktualnie 4 główne implementacje .NET:

3.1.1. .NET Core

Został napisany z myślą o tworzeniu aplikacji cross-platformowych, które mogą zostać uruchomione na serwerach, jak i środowiskach chmurowych. Potrafi działać na platformie Windows, macOS oraz Linux. Jest to pierwsza implementacja .NET, która została zaprojektowana przez Microsoft z myślą o wieloplatformowości.

3.1.2. .NET Framework

Pierwsza, oryginalna implementacja .NET, która istnieje od roku 2002. Składa się ze środowiska uruchomieniowego Common Language Runtime (CLR) oraz biblioteki standardowej zwanej jako Framework Class Library (FCL). CLR zapewnia aplikacjom wirtualną maszynę, na której wykonywany jest kod bajtowy skompilowany z języka CIL. Ta implementacja jest używana w tej pracy inżynierskiej.

3.1.3. Mono

Darmowy projekt open-source prowadzony przez firmę Xamarin. Powodem stworzenia tego produktu była możliwość uruchamiania aplikacji napisanych w językach .NET na wielu platformach, jak i dostarczenie użytkownikom Linuxa narzędzi pozwalających na pisanie aplikacji w rodzinie języków .NET.

3.1.4. Universal Windows Platform (*UWP*)

Implementacja, która umożliwia tworzenie aplikacji dla wszystkich platform używających Windows 10, Xboxa, niektórych urządzeń stworzonych przez Microsoft i dostosowanych urządzeń IoT.

3.2. C#

C# jest językiem programowania trzeciej generacji. Został opublikowany w roku 2001 przez firmę Microsoft. Jest on silnie typowanym językiem wielo-paradygmatowym umożliwiającym programowanie imperatywne, deklaratywne, generyczne, funkcjonalne, komponentowe i zorientowane obiektowo. Najnowszy jego standard został wydany w listopadzie 2017 roku i jest oznaczony

jako wersja 7.2. Wchodzi on w grupę języków .NET, przez co jest kompilowalny do języka CIL. [6]

```
using System;
namespace HelloWorld
{
    class Hello
    {
        static void Main()
        {
            Console.WriteLine("Hello World!");
            Console.ReadKey();
        }
    }
}
```

Listing 3.2. Przykładowy kod aplikacji "Hello World" w języku C#

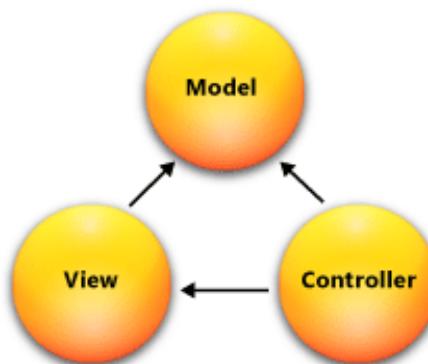
4. ASP.NET MVC

ASP.NET MVC jest frameworkm do budowania aplikacji internetowych w oparciu o wzorzec architektoniczny Model-View-Controller (MVC). Wykorzystuje implementacje .NET Framework do uruchamiania skompilowanego kodu źródłowego.

4.1. Model-Widok-Kontroler

Większość dzisiejszych systemów komputerowych działa na zasadzie wyświetlania danych, które aktualnie znajdują się w bazie danych i ewentualnie ich modyfikacji. W celu ujednolicenia tych systemów stosowany jest wzorzec Model-Widok-Kontroler(ang. Model-View-Controller), który rozdziela logikę aplikacji na 3 główne segmenty:

1. Model - Służy do pobierania, przechowywania i zamiany danych.
2. Kontroler - przetwarza zapytania użytkownika.
3. Widok - Służy do wyświetlania informacji .



Rys. 4.1. Model MVC. Źródło: msdn.microsoft.com

Cała aplikacja została skonstruowana zgodnie z tym wzorcem projektowym. ASP.NET MVC posiada wiele narzędzi, które ułatwiają zastosowanie tego wzorca. Dostarczana jest klasa bazowa **Controller**, która posiada wszystkie podstawowe metody do wyświetlania odpowiednich treści danych i zarządzania zapytaniami internetowymi użytkownika.

Wzorzec MVC niesie za sobą korzyści związane z warstwą wyświetlania danych. Jest to warstwa aplikacji, w której bardzo często dochodzi do zmian. Dzięki odseparowaniu danych od widoku jesteśmy w stanie tworzyć, jak i zmieniać widoki, bez wpływu na kod biznesowy aplikacji.

4.2. Razor

W celu stworzenia strony przez ASP.NET MVC należy napisać kod strony internetowej, który potrafi zostać skompilowany przez określony renderer do kodu HTML. Aktualnie dwoma najpopularniejszymi opcjami jest aspx i Razor[7]. Pierwszy z nich istnieje od samego początku ASP.NET MVC. Ma bardzo ciężką i niewygodną składnię, przez co jest trudny w nauce, obsłudze i utrzymaniu.

Razor natomiast został wprowadzony w wersji 3.0 ASP.NET MVC w celu eliminacji wielu problemów związanych z trudnością używania swojego poprzednika. Wprowadza całkowicie nową składnię, która jest łatwiejsza w użyciu i zrozumieniu.

Dzięki rendererom można używać języka C# w celu tworzenia stron internetowych. Pozwala to na używanie wszystkich możliwych konstrukcji językowych w trakcie procesu tworzenia stron : pętli, instrukcji warunkowych, instrukcji switch, metod i wielu innych.

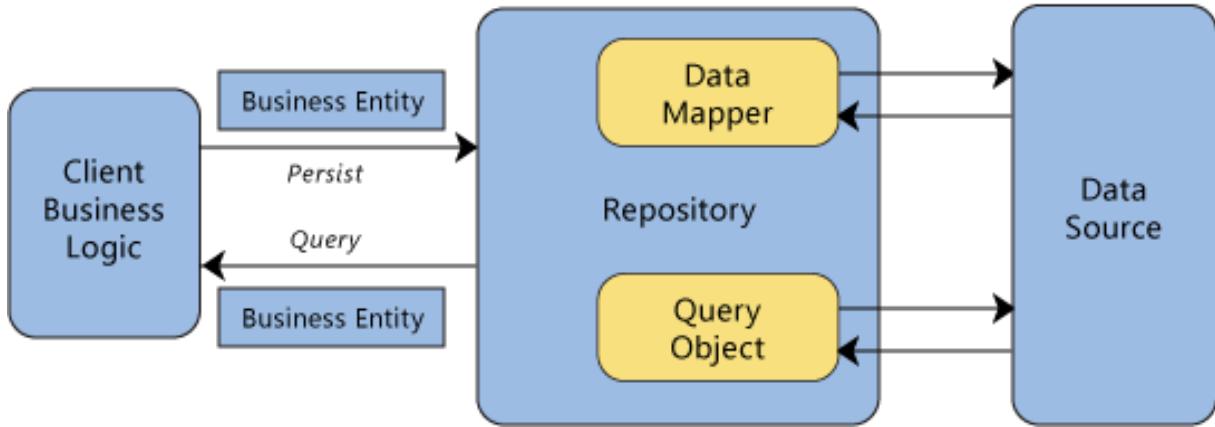
```
<h1>
    Witaj @ViewBag.Name ! Aktualna data to @DateTime.Now
</h1>
```

Listing 4.1. Przykładowy kod kompatybilny z Razorem.

4.3. Wzorzec Repozytorium

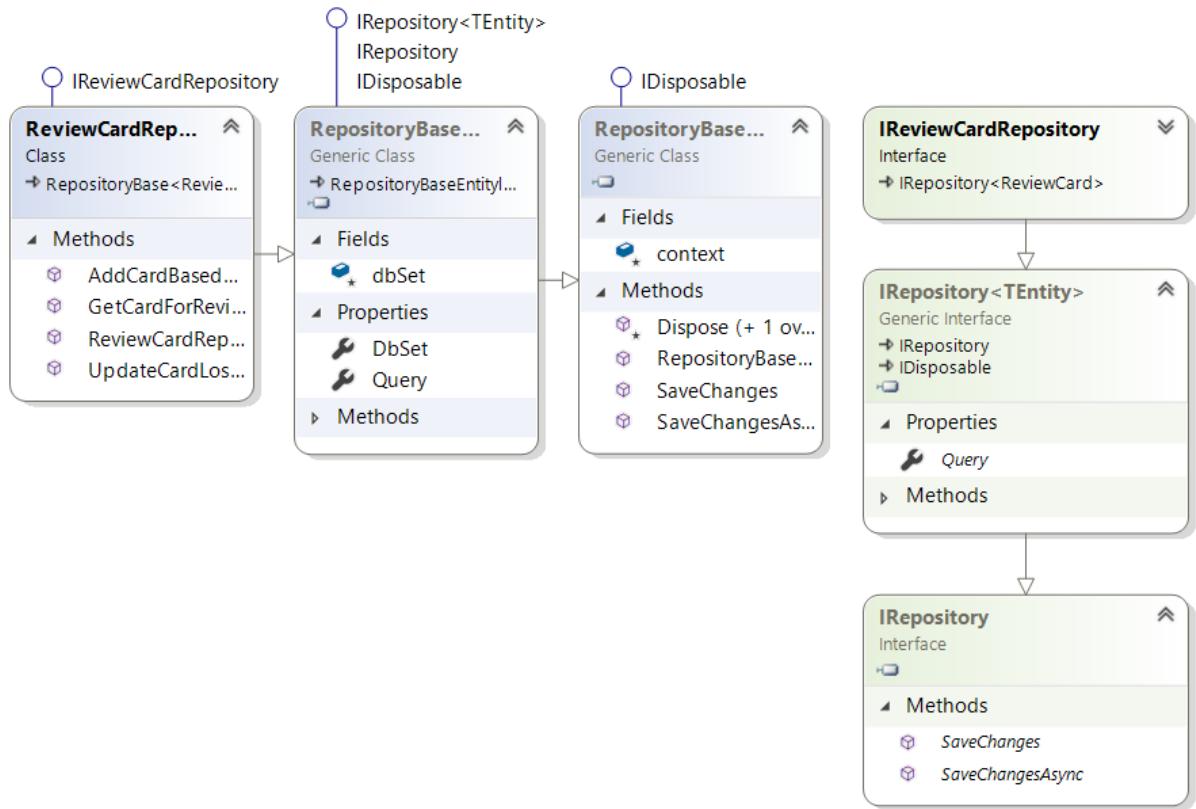
Wewnątrz aplikacji używana jest baza danych Microsoft SQL. Do komunikacji z bazą bardzo często jest tworzony kod, który zwraca podobne dane. W celu zmniejszenia redundancji kodu, jak i odseparowania zależności i odpowiedzialności wykorzystany został wzorzec Repozytorium (z ang. Repository Pattern)[8].

Wzorzec ten wykorzystuje obiekty, zwane repozytoriami, których zadaniem jest pobieranie i modyfikowanie danych po stronie serwera SQL. Nie zawierają żadnej logiki biznesowej i są niezwiązane z resztą kodu danej aplikacji.



Rys. 4.2. Wzorzec repozytorium. Źródło: msdn.microsoft.com

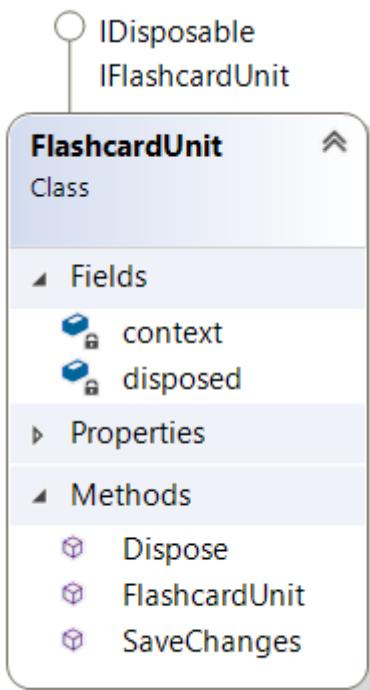
Wewnątrz projektu cały wzorzec repozytorium został oparty na interfejsach **IRepository** i **IRepository<T>** (który implementuje **IRepository**). W założeniu te interfejsy definiują operacje, które można zrealizować na danym obiekcie klasy 'T'. Informacja o sposobie realizacji operacji jest zdefiniowana w klasie **RepositoryBase<T>**. Wszystkie kolejne stworzone repozytoria dziedziczą po **RepositoryBase<T>** a ich interfejsy implementują **IRepository<T>**. Takie działanie pozwala nam na separację reprezentacji persystentnej (baza danych) i reprezentacji bieżącej (repozytoria). Wystarczy, iż zmienimy **RepositoryBase<T>** na jakikolwiek inną klasę implementującą **IRepository<T>**. Dzięki temu możemy zastąpić serwer Microsoft SQL inną implementacją.



Rys. 4.3. Repozytorium ReviewCardRepository wraz z implementowanymi interfejsami i odziedzicznymi klasami.

4.4. Jednostka Pracy

Wzorzec Jednostki Pracy (z ang. Unit of Work)[8] ma na celu uporządkowanie pracy z repozytoriami za pomocą umieszczenia ich wszystkich w jednej klasie. Dodatkowo dzięki temu rozwiązaniu wszystkie repozytoria współdzielą kontekst dostępu do bazy Microsoft SQL. Zapisanie danych odbywa się poprzez wywołanie metody **SaveChanges** wewnętrz Jednostki Pracy (**SaveChanges** jest konwencją wewnątrz projektu).



Rys. 4.4. Jednostka pracy wykorzystywana w projekcie.

4.5. Mapowanie obiektowo-relacyjne

Wewnątrz projektu zostało wykorzystane narzędzie Entity Framework pozwalające na korzystanie z mapowania obiektowo-relacyjnego. Dzięki tej technice można uprzednio zdefiniowane dane tabelaryczne odwzorować za pomocą klas, które zostaną stworzone na podstawie definicji tabel i relacji między nimi. Powstałe klasy są używane wewnątrz kolekcji, implementujących interfejs **IQueryable<T>**, na których możemy wykonywać zapytania z pomocą **LINQa**, które następnie zostaną przetłumaczone na zapytania SQL.

Takowe rozwiązanie ma wiele zalet :

1. Dane z danej tabeli można otrzymać zawsze w tym samym formacie.
2. Zmniejsza to zakres potrzebnych umiejętności w celu pobrania danych. Dany programista nie musi znać języka SQL, aby w sposób bezproblemowy pobrać interesujące go dane, nawet gdy tworzy bardzo skomplikowane zapytania.
3. Łatwiejsza nawigacja po zależnościach między tabelami. Na danym typie można wykorzystać operacje **Find All References**, która wskaże miejsca jego użycia w kodzie.
4. Operacje na interfejsie **IQueryable<T>** pozwalają na pobranie danych dopiero, gdy będą nam potrzebne za pomocą techniki lazy loading.

4.6. Wstrzykiwanie zależności

Wstrzykiwanie zależności (po ang. Dependency Injection) jest wzorcem projektowym, którego działanie polega na tym, iż obiekty aplikacji nie muszą tworzyć zależnych od siebie obiektów, lecz są tworzone przez obiekt nadzędny dla użytku tych klas.

Informacja o tym, jakie obiekty powinny zostać dostarczone danej instancji jest przekazywana najczęściej poprzez listę argumentów konstruktora bądź przy wywołaniu specjalnych metod.

Wewnątrz projektu w celu realizacji tego wzorca wykorzystana została biblioteka Ninject[9]. Z pomocą dodatkowej biblioteki **Ninject.Web.Common** jest ona w stanie w łatwy sposób zintegrować się z aplikacją ASP.NET MVC wykorzystując interfejs **IDependencyResolver**.

Każde utworzenie kontrolera wewnątrz aplikacji ASP.NET MVC wiąże się wpierw z odpytaniem aktualnie używanego **IDependencyResolver**a o próbę utworzenia takowego kontrolera. W tym miejscu zaczyna działać Ninject, który stara się utworzyć obiekt. Gdy konstruktor danej klasy będzie zawierał jakiekolwiek parametry to Ninject będzie starał się je utworzyć na podstawie zdefiniowanych uprzednio bindingów. Jeśli okaże się, że stworzenie obiektu kontrolera jest niemożliwe, to zwracany jest błąd wewnętrzny 500.

Stworzenie bindingów, które będą informować, na jakich zasadach odbywa się tworzenie żądanych obiektów, jest wykonywane za pomocą metody **Bind** znajdującej się w Kernelu Ninjeta.



Rys. 4.5. Proces bindowania z wyszczególnionymi częściami składowymi.

Pierwsza część bidowania odpowiada za przekazanie informacji o tym jakiego **Providera** należy użyć w celu zwrócenia instancji danego typu. Metody **ToSelf()** i **To<T>()** używają wbudowanego **StandardProvidera**. Pierwsza z metod zawsze będzie informowała o tym, iż należy zwrócić obiekt tego samego typu, gdzie druga konkretnie informuje jaki typ powinien zostać zwrócony w wyniku danego zapytania.

Ninject oferuje także opcje stworzenia własnych providerów. Można to osiągnąć za pomocą stworzenia klasy dziedziczącej po **Provider<T>** lub wykorzystania funkcji wewnętrznej metody **ToMethod(Func<IConext, T>)**.

Ninject w swoim domyślnym zachowaniu zawsze tworzy nowy obiekt danej klasy, gdy zostanie poproszony o instancje danego typu. Nie zawsze jest to jednak sytuacja pożądana. Każdy bind posiada możliwość zdefiniowania osobnego zasięgu, w którym będzie on obowiązywał. Umożliwi to współkorzystanie z danego obiektu wewnątrz danego zakresu. W wypadku środowiska internetowego bardzo często stosowanym zasięgiem jest **InRequestScope()**, które powoduje, iż obiekty danego typu stworzone przez Ninject są współdzielone w trakcie danego zapytania. Na końcu zapytania wszystkie obiekty są usuwane i ewentualnie jest wywoływana metoda **Dispose**(jeśli dany obiekt implementuje interfejs **IDisposable**), która zwalnia zasoby, które w normalnych warunkach nie zostałyby zwolnione. Takimi zasobami mogą być otwarte pliki, połączenia do bazy danych i tym podobne.

Takie działanie niesie za sobą szereg korzyści. Nie tylko nie ponosimy kosztów tworzenia nowych obiektów, lecz także jesteśmy w stanie współdzielić dane informacje pomiędzy różnymi serwisami. (Np. repozytoria nie muszą wykonywać dodatkowych zapytań SQL, gdyż część wyników znajduje się ich w cache'u). Dodatkową zaletą jest to, iż użycie tego wzorca powoduje zwiększenie czytelności kodu.

5. Microsoft SQL Server

Microsoft SQL Server[10] jest serwerem bazodanowym, który umożliwia przechowywanie, odczytywanie oraz modyfikowanie zapisanych w nim danych. Został stworzony przez firmę Microsoft Corporation. Jego pierwsza edycja ukazała się w roku 1989. Wszystkie przechowywane na nim dane są podzielone na tabele, które składają się z rekordów o ścisłe określonej strukturze, wewnętrz których przechowywane są zapisane informacje. Taki sposób zapisu danych znany jest też pod nazwą modelu relacyjnego. [11]

W celu integracji serwera Microsoft SQL z projektem napisanym w języku C# należy pobrać paczkę NuGet¹ o nazwie **System.Data.SqlClient**.

5.1. Język zapytań

Wewnątrz Microsoft SQL Server do operacji na danych używany jest język Transact-SQL (T-SQL). Stosuje się go w celu tworzenia zapytań T-SQL, które są komendami potrafią побiegać, modyfikować oraz zapisywać informacje w bazie danych. T-SQL jest rozszerzeniem języka SQL(Structured Query Language) i między innymi wprowadza:

1. Lokalne zmienne.
2. Blok **Try Catch**, wspierający obsługę wyjątków.
3. Dodatkowe funkcjonalności związane z obsługą tekstu, kalendarza i matematyki.
4. Możliwość tworzenia pętli oraz instrukcji warunkowych.
5. Procedury, funkcje składowane i wyzwalacze.

```
select flashcard.ID, trans.Translation from Flashcards flashcard  
inner join FlashcardTranslations trans on flashcard.id = trans.FlashcardID
```

Listing 5.1. Przykładowe zapytanie stworzone w języku T-SQL.

¹NuGet jest menadżerem paczek w środowisku .NET.[12]

6. Testy

Aplikacje tworzone w dzisiejszych czasach wymagają dostarczenia zestawu testów, których zadaniem jest sprawdzenie poprawności działania aplikacji. Z tego też powodu napisany został zestaw testów, które sprawdzają, czy poszczególne komponenty aplikacji działają poprawnie.

6.1. Testy Jednostkowe

Testy jednostkowe mają za zadanie sprawdzić poprawność działania pojedynczego modułu. Wszelkie odwołania do innych komponentów zostają zastąpione przez atrapy obiektów (po ang. mock objects), które symulują działanie ich prawdziwych odpowiedników. Wewnątrz projektu do operacji związanych z atrapami wykorzystany został framework Moq, który zawiera także dodatkowe funkcjonalności pozwalające ocenić czy dany test przebiegł poprawnie. Testy zostały ułożone zgodnie ze wzorcem AAA[13] - Aranżacja (po ang. Arrange) Akcja (po ang. Act) Asercja (po ang. assert). Dzięki temu testy są konsystentne i czytelne, przez co nie potrzeba dużej ilości czasu w celu zaznajomienia się z nimi.

```
public void StopLastTraining_assert_tests()
{
    //Aranżacja - stworzenie wewnętrznej atrapy obiektu dla atrapy serwisu
    mockTraining();
    //Akcja
    trainingReviewService.StopLastTraining();
    //Asercja - sprawdzenie poprawności wykonania
    trainingRepository.Verify(x => x.Remove(It.IsAny<long>()), Times.Once);
    unit.Verify(x => x.SaveChanges(), Times.Once); //wykorzystanie ←
    funkcjonalności frameworku Moq w celu sprawdzenia czy dana metoda ←
    została wywołana.
    Assert.AreEqual(null, sessionService.Object.UserInfo.TrainingInfo);
}
```

Listing 6.1. Przykładowy test jednostkowy wykorzystujący wzorzec AAA

6.2. Testy Integracyjne

Testy integracyjne mają za zadanie pokazać poprawną komunikację pomiędzy modułami w projekcie. Modułem może być każdy komponent zaprogramowany na potrzeby projektu jak i

zewnętrzny system (np. system obsługi plików). Wszystkie moduły, które nie są testowane w danym teście powinny zostać zastąpione przez odpowiednie atrapy.

```
[TestMethod]
public void ExistTest()
{
    using (var temp = new WindowsTempFile())
        Assert.IsTrue(File.Exists(temp.Path));
}
```

Listing 6.2. Test integracyjny wykorzystany w projekcie.

7. Kaskadowe Arkusze Styli

Kaskadowe Arkusze Styli (po ang. Cascading Style Sheets, w skrócie CSS)[14] wykorzystywane są w celu opisu sposobu wyświetlania elementów stron WWW. Pozwalają zdefiniować między innymi takie właściwości elementów jak: kolory, czcionki, położenie. Dzięki oddzieleniu warstwy danych od warstwy prezentacji możliwe jest wykorzystanie tych samych arkuszów stylu na wielu stronach. W celu określenia wyglądu danego elementu należy zdefiniować odpowiednią regułę, na którą składa się :

1. Selektor, który określa dla jakich elementów przeznaczona jest dana reguła.
2. Zestaw właściwości wraz z przypisanymi do nich wartościami

```
p //selektor p
{
//Właściwość color do której przypisano wartość orange
    color: orange;
}
```

Listing 7.1. Przykład prostej reguły, dzięki której wszystkie paragrafy będą miały domyślnie kolor pomarańczowy.

7.1. Syntaktycznie Doskonałe Arkusze Styli

W dzisiejszym świecie doświadczeni programiści nie używają już samego języka CSS do tworzenia arkuszy stylów. Wynika to z faktu występowania zbyt dużej redundancji kodu, która jest związana z tym, iż niektóre fragmenty strony są formatowane zawsze w ten sam sposób (np. motyw kolorystyczny dla większości elementów jest ten sam).

Jest to spowodowane brakiem zmiennych, szablonów, oraz innych konstrukcji, które pomogły w ograniczeniu powyższego problemu.

Z tego też powodu powstały rozwiązania niedogodności występujących w języku CSS. Jednym z nich jest język SASS - Syntactically Awesome Style Sheets (z ang. Syntaktycznie Doskonałe Arkusze Styli)[15]. Jest on kompatybilny ze wszystkimi wersjami CSS, przez co każdy kod napisany w CSS jest z nim zgodny. Oferuje bardzo dużo funkcjonalności, między innymi:

1. Zmienne, które pozwalają wielokrotnie używać danych wartości

2. Możliwość zagnieżdżania reguł w innych regułach.
3. Możliwość załączania plików pozwalająca na podział arkuszy stylów.
4. Mixin - szablony reguł, które można ponownie wykorzystywać.
5. Dziedziczenie reguł za pomocą słówka kluczowego **@extend**
6. Za pomocą operatorów `+`, `-`, `*`, `/`, `%` można wykonywać operacjach na liczbach.

```
@mixin border-radius($radius) {  
    -webkit-border-radius: $radius;  
    -moz-border-radius: $radius;  
    -ms-border-radius: $radius;  
    border-radius: $radius;  
}  
  
.box { @include border-radius(10px); }
```

Listing 7.2. Przykładowy kod SASS wykorzystujący mixin.

8. Scala

Scala[16] jest językiem programowania ogólnego zastosowania, który został wprowadzony na rynek przez Laboratorium "École Polytechnique Fédérale de Lausanne". Jest to język kompilowany bezpośrednio do kodu bajtowego Javy, przez co programy w nim napisane z łatwością uruchamiają się w środowisku wykonywalnym maszyny wirtualnej Javy. Scala jest językiem wielo-paradygmatowym[17]. Korzysta z dobrodzieństw programowania funkcjonalnego i obiektowego.

```
object HelloWorld {  
    def main(args: Array[String]): Unit = {  
        println("Hello, world!")  
    }  
}
```

Listing 8.1. Hello world napisany w języku Scala.

8.1. Javascript

Javascript jest głównym językiem programowania wykorzystywanym na stronach internetowych. Może być on interpretowany bądź komplikowany metodą Just In Time, która kompiluje kod tuż przed jego wykonaniem. Jest to dynamiczny język wieloparadygmatowy bazujący na prototypach. Wspiera programowanie obiektowe, imperatywne i deklaratywne.[18]

JavaScript został stworzony w ciągu 10 dni [19] przez firmę Netscape. Był stworzony z myślą o wzbogacaniu stron WWW o proste skrypty dodające elementy interakcji ze stroną. Prawdopodobnie krótki czas powstawania wpłynął negatywnie na język. JavaScript posiada wiele problemów, które utrudniają pisanie programów. Między innymi takich jak:

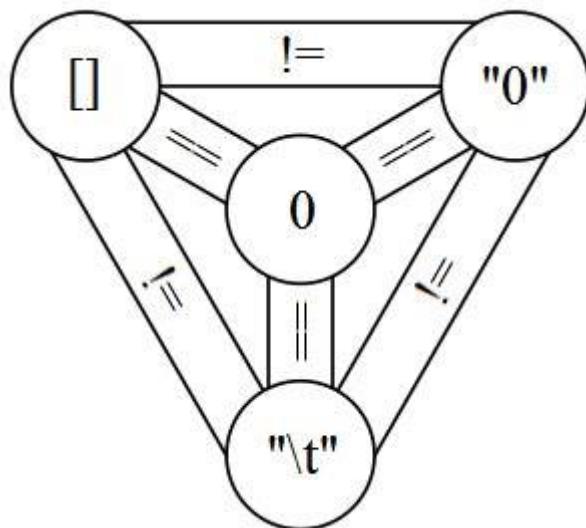
1. Niejawne rzutowania, które objawiają się przy użyciu operatora `==`. Są przyczyną wielu trudnych do odkrycia błędów
2. Automatyczne wstawianie średników, co może spowodować, iż program działa inaczej niż zamierzał to programista. Czasami potrafi to uratować program napisany przez programistę, lecz często prowadzi do dziwnych i niezrozumiałych błędów, takich jak w przypadku poniższego listingu: 8.2

3. Niezrozumiałe zachowana różnych funkcji. Metoda sortująca domyślnie sortuje tablice liczb w porządku alfabetycznym nie zwracając uwagi na fakt, iż w tablicy nie występuje ani jeden ciąg znakowy.
4. Zmienne, które w każdej chwili mogą zmienić swój typ. Pomimo tego, iż jest to błąd programistyczny to w wyniku tej niedogodności powstają trudne do namierzenia błędy w kodzie.

```
function foo() {
    return // Tutaj zostanie wstawiony średnik.
    {
        bar : "test"
    };
}
```

Listing 8.2. Przykład niepoprawnego kodu Javascript wynikłego z automatycznego wstawienia średnika

Z tego też powodu powstały języki, które starają się być lepsze i przyjaźniejsze dla programisty niż Javascript. Jest to między innymi: Typescript, Coffescript, Dart czy Scala.js. Takie języki programowania nie są wykonywane/kompilowane bezpośrednio w przeglądarce, lecz wpierw są kompilowane do kodu Javascript. Spowodowane jest to używaniem przez przeglądarki jedynie języka Javascript do wykonywania kodu¹.



Rys. 8.1. Przykład dziwnego zachowania języka Javascript. Źródło: devrant.com

¹Wyjątkiem jest tutaj przeglądarka Dartium, która posiada maszynę wirtualną języka Dart[20]

8.2. Scala.js

Scala.js jest narzędziem umożliwiającym komplikacje kodu Scali do języka Javascript. Umożliwia to pozbycie się niedogodności wymienionych powyżej. Jest to możliwe dzięki:

1. Silnemu typowaniu, dzięki któremu pisany kod nie zawiera bardzo prostych błędów.
2. Nie skupianiu się na dziwnych i frustrujących aspektach języka Javascript.
3. Środowiskom programistycznym wspierającym pracę programisty przez podpowiadanie składni, łatwą nawigację po kodzie źródłowym oraz inne udogodnienia.
4. Możliwości wykrycia błędów w trakcie kompilowania kodu źródłowego.

8.2.1. sbt

sbt (Simple Build Tool) jest narzędziem o otwartym kodzie źródłowym, pozwalającym na zarządzanie procesem budowania aplikacji napisanej w języku Scala lub Java. Jest to narzędzie bardzo rozbudowane, które umożliwia m.in.:

1. Współpracę z wieloma narzędziami testującymi dla scali.
2. Inkrementacyjne kompilowanie i testowanie.
3. Zarządzanie zależnościami przy pomocy menadżera paczek Ivy.[21]
4. Ciągłe wdrażanie, kompilowanie i testowanie kodu.

8.2.2. Proces budowania projektu

Proces kompilacji projektu jest podzielony na 3 etapy[22]:

1. Początkową kompilację
2. Szybką optymalizację
3. Pełną optymalizację (Opcjonalny)

8.2.2.1. Początkowa kompilacja

W trakcie kompilacji pliki .scala są kompilowane do plików .class i .sjsir. Pliki .class nie biorą udziału w tworzeniu kodu javascript. Ich zadaniem jest współpraca z innymi narzędziami, które być może będą ich używać. Przykładem takiego narzędzia może być **IntelliJ** lub **Eclipse**, które plików .class używają w celu wspomagania programisty w trakcie pisania kodu. Pliki .sjsir (Nazwa rozszerzenia jest skrótem od „ScalaJS Intermediate Representation”)[23] zawierają kod przejściowy między Scalą a Javascriptem. Większość konstrukcji została zastąpiona przez

ekwiwalenty z języka Javascript. Gdybyśmy połączyli wszystkie pliki .sjsir, wyprodukowane przez sbt to ich wynikiem byłby plik większy niż 20 MB. Wynika to z faktu, iż w tym pliku nadal znajduje się wiele niepotrzebnych bibliotek i konstrukcji. Jak na przykład **cała** biblioteka standardowa Scali.

```
module class Ltutorial_webapp_TutorialApp$ extends O {
  def main__AT__V(args: T[]) {
    this.appendPar__Lorg_scalajs_dom_raw_Node__T__V
    (mod:Lorg_scalajs_dom_package$.document__Lorg_scalajs_dom_raw_HTMLDocument
     () ["body"], "Hello World")
  }
  def appendPar__Lorg_scalajs_dom_raw_Node__T__V(targetNode: any, text: T) {
    val parNode: any = ←
      mod:Lorg_scalajs_dom_package$.document__Lorg_scalajs_dom_raw_HTMLDocument()
      ["createElement"]("p");
    val textNode: any = ←
      mod:Lorg_scalajs_dom_package$.document__Lorg_scalajs_dom_raw_HTMLDocument()
      ["createTextNode"](text);
    parNode["appendChild"] (textNode);
    targetNode["appendChild"] (parNode)
  }
  def init__() {
    this.O::init__();
    mod:Ltutorial_webapp_TutorialApp$<-this
  }
}
```

Listing 8.3. Przykładowy plik .sjsir dla projektu wyświetlającego HelloWorld na ekranie.

8.2.2.2. Szybka optymalizacja

W celu optymalizacji poprzedniego kroku stosuje się szybką optymalizację kodu .sjsir, która jako rezultat swej pracy stworzy kod Javascript. W tym celu należy użyć optymalizatora FastOptJS poprzez wpisanie komendy **FastOptJS** w sbt. Optymalizacja ma na celu:

1. Wyeliminowanie fragmentów kodu, które są nieużywane. Na przykład kod biblioteki standardowej, który nie zostanie wywołany.
2. Inline'owanie małych funkcji. Zmniejsza to koszt wywołań i wielkość kodu.
3. Zmiana zmiennych na stałe, jeśli ich wartość jest znana w trakcie kompilacji.

Dzięki tej operacji kod wykonywalny zmniejszy się z 20 MB do 1.5-2.5MB[23].

8.2.2.3. Pełna optymalizacja

Kod, który jest tworzony przez Scala.js jest zgodny z restrykcjami narzuconymi przez kompilator Closure[24]. Jest to narzędzie stworzone przez firmę Google, które potrafi optymalizować

kod Javascript.[25] Celem tego kroku jest zmniejszenie rozmiaru pliku Javascript i uczynienie konstrukcji w nim występujących bardziej wydajnymi. W wyniku tego procesu otrzymywany jest plik wykonywalny o rozmiarze między 150KB do kilkuset KB[23]. W celu użycia pełnej optymalizacji należy wywołać polecenie **FullOptJS** w sbt.

8.2.2.4. Pliki javascript

W wyniku działań optymalizatorów tworzone są poniższe pliki javascript. Należy mieć na uwadze fakt, iż pliki bibliotek muszą zostać dołączone do kodu strony przed plikiem z kodem wykonywalnym.

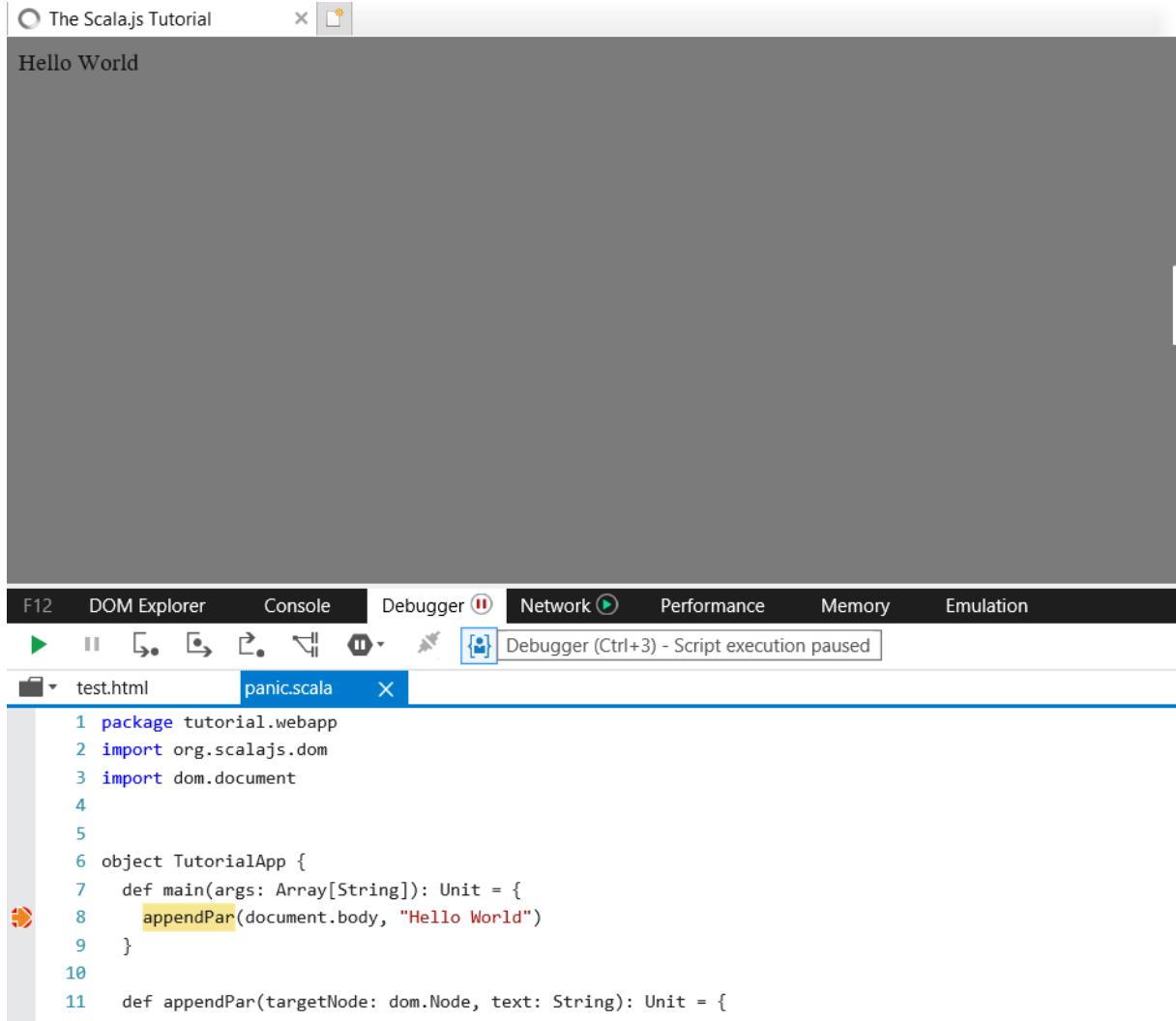
Typ komplikacji	Wytwarzony plik	Zawartość pliku
FastOptJS	scala-js-[Nazwa-Projektu]-fastopt.js	Plik z kodem wykonywalnym
FastOptJS	scala-js-[Nazwa-Projektu]-fastopt.js.map	Plik mapujący kod Javascript do kodu Scali.
FastOptJS	scala-js-[Nazwa-Projektu]-jsdeps.js	Plik z kodem zewnętrznych bibliotek użytych w procesie tworzenia aplikacji
FullOptJS	scala-js-[Nazwa-Projektu]-opt.js	Plik z kodem wykonywalnym
FullOptJS	scala-js-[Nazwa-Projektu]-opt.js.map	Plik mapujący kod Javascript do kodu Scali.
FullOptJS	scala-js-[Nazwa-Projektu]-jsdeps.min.js	Plik z kodem zewnętrznych bibliotek użytych w procesie tworzenia aplikacji.

8.2.2.5. Debugowanie

Jednym z najważniejszych procesów, które na celu mają znalezienie błędów w powstałym kodzie jest debugowanie. Proces debugowania aplikacji powinien umożliwiać programistę przy najmniej możliwość zatrzymania kodu w dowolnym miejscu, podejrzenia kodu źródłowego jak i możliwość odczytu i modyfikacji zmiennych.

Kod stworzony za pomocą kompilatora scala.js jest bardzo łatwy w debugowaniu. Przeglądarki posiadają możliwość dołączenia mapy kodów źródłowych, które do danych linijek kodu javascript przypisują ich odpowiedniki w pliku źródłowym. Umożliwia to prace z oryginalnym kodem źródłowym podczas używania przeglądarki internetowej.

Pliki z mapami źródłowymi mają format **{SkompilowanyPlik}.map**, gdzie Skompilowany Plik jest artefaktem naszego procesu komplikacji, np. **scala-js-[Nazwa-Projektu]-fastopt.js**.



Rys. 8.2. Przykład procesu debugowania kodu. Program zostaje zatrzymany na linijce przed dodaniem napisu Hello World.²

8.2.3. Hello World

Za pomocą powyższych metod i narzędzi można stworzyć w bardzo prosty sposób przykładową aplikację wykorzystującą język Scala.js. W tym celu należy użyć polecenia **sbt new sbt/scala-seed.g8**, które spowoduje utworzenie minimalnego projektu Scali. Aby projekt wykorzystywał Scala.js należy w nim wykonać parę zmian:

²Na obrazku na stronie widnieje już napis Hello World. Jest to artefakt, który został stworzony przy poprzednim uruchomieniu strony i jeszcze się nie odświeżył.

1. Należy dodać plik ./project/plugins.sbt³, wewnątrz którego znajdzie się instrukcja `addSbtPlugin("org.scala-js" % "sbt-scalajs" % "0.6.20")`, która poinformuje sbt o konieczności użycia pluginu Scali.js.
2. W pliku ./project/build.properties musi zostać określona wersja narzędzia sbt, którym będzie komplikowany projekt.
3. Plik build.sbt, który zawiera ustawienia związane z budową projektu, musi zostać zmodyfikowany w celu użycia pluginu Scali.js jak i pomocniczych bibliotek. Przykładowy plik, który jest używany w projekcie można zobaczyć na listingu 8.4.

Po takim zabiegu należy dodać kod źródłowy Scali do folderu ./src/main/scala. Przykładowy kod można znaleźć na listingu 8.5. Po wykonaniu powyższych czynności uruchomienie komendy **FastOptJs** w sbt powinno wygenerować pliki wynikowe(8.2.2.4), które można użyć na stronie WWW.

```
enablePlugins(ScalaJSPlugin)
libraryDependencies += "org.scala-js" %%% "scalajs-dom" % "0.9.1"
libraryDependencies += "be.doeraene" %%% "scalajs-jquery" % "0.9.1"

name := "Flashcards"
scalaVersion := "2.12.2"

// Informacja o tym, iż chcemy aby nasz kod zawsze miał uruchamianą metodę main ←
// po wczytaniu witryny.
scalaJSUseMainModuleInitializer := true

skip in packageJSDependencies := false
jsDependencies += "org.webjars" % "jquery" % "2.1.4" / "2.1.4/jquery.js"
jsDependencies += "org.webjars.bower" % "jsrender" % "1.0.0-rc.70" / ←
  "1.0.0-rc.70/jsrender.js"
```

Listing 8.4. Plik .sbt, który jest wykorzystywany w projekcie.

³./ jest katalogiem głównym projektu w tym przypadku.

```
package helloworld
import org.scalajs.dom
import dom.document

object TutorialApp {
    def main(args: Array[String]): Unit = {
        appendPar(document.body, "Hello World")
    }

    def appendPar(targetNode: dom.Node, text: String): Unit = {
        val parNode = document.createElement("p")
        val textNode = document.createTextNode(text)
        parNode.appendChild(textNode)
        targetNode.appendChild(parNode)
    }
}
```

Listing 8.5. Przykładowy projekt w Scali.js wypisujący napis Hello World na stronie.

9. HTML

Głównym językiem służących do opisu budowy stron internetowych jest HTML (Hypertext Markup Language). Jest to język deklaratywny, którego początki sięgają lat osiemdziesiątych. Został zapoczątkowany przez fizyka Tima Bernersa-Lee, który pracował w CERNie. Jego aktualną wersją (W dniu 2017-01-05) jest 5.1.2 wydana 1 listopada 2016 roku. Wewnątrz dokumentu HTML można zamieszczać pliki multimedialne, skrypty pisane w języku JavaScript oraz dokumenty CSS.

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
</head>
<body>
  Pusta strona html
</body>
</html>
```

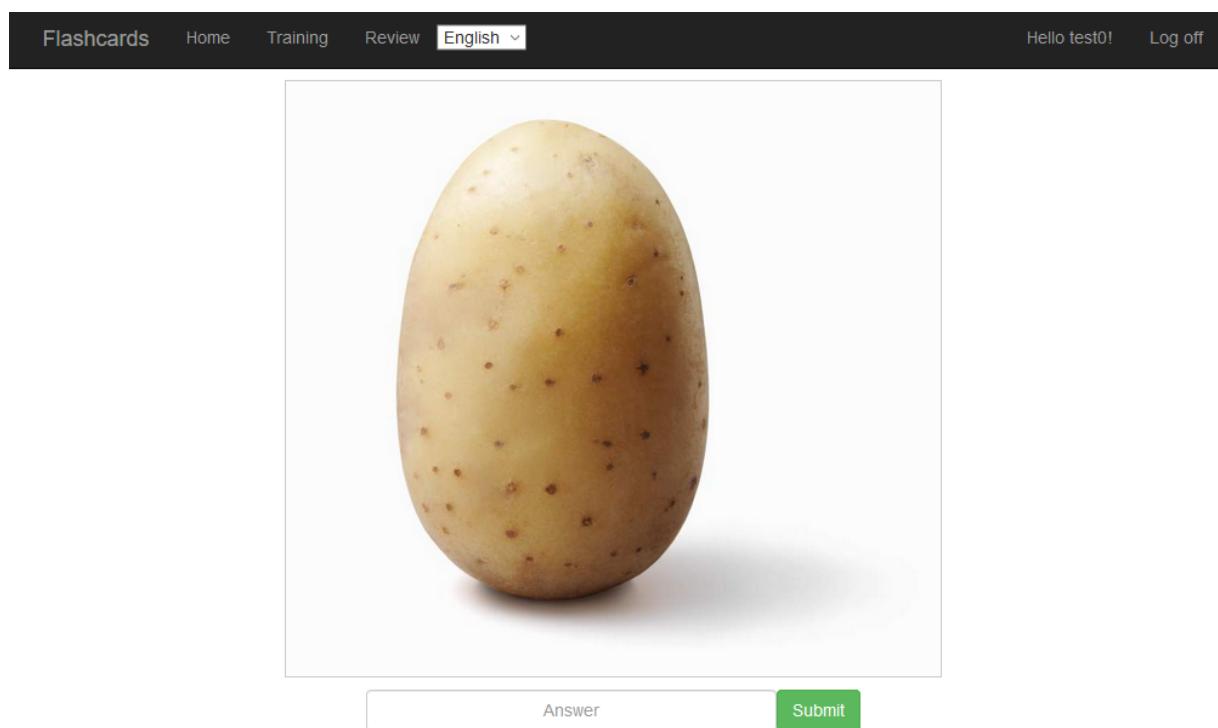
Listing 9.1. Przykładowa strona internetowa zapisana z użyciem języka HTML.

10. Opis działania aplikacji

10.1. Proces nauczania

Proces nauczania podzielony jest na 2 etapy :

- Trening
- Powtórkę



Rys. 10.1. Widok pytania w projekcie. Jest on wspólny dla każdego z etapów

10.1.1. Trening

W trakcie treningu użytkownik po raz pierwszy może spotkać się z danym słowem występującym w procesie nauki. Celem tego etapu jest zapoznanie użytkownika z nowym słowem, zanim zacznie go używać w ramach powtórki. Trening korzysta z kolejki, wewnętrznej której znajduje się maksymalnie 5 fiszek. Tak mała ilość ma na celu łatwiejsze przyswojenie nowego materiału.

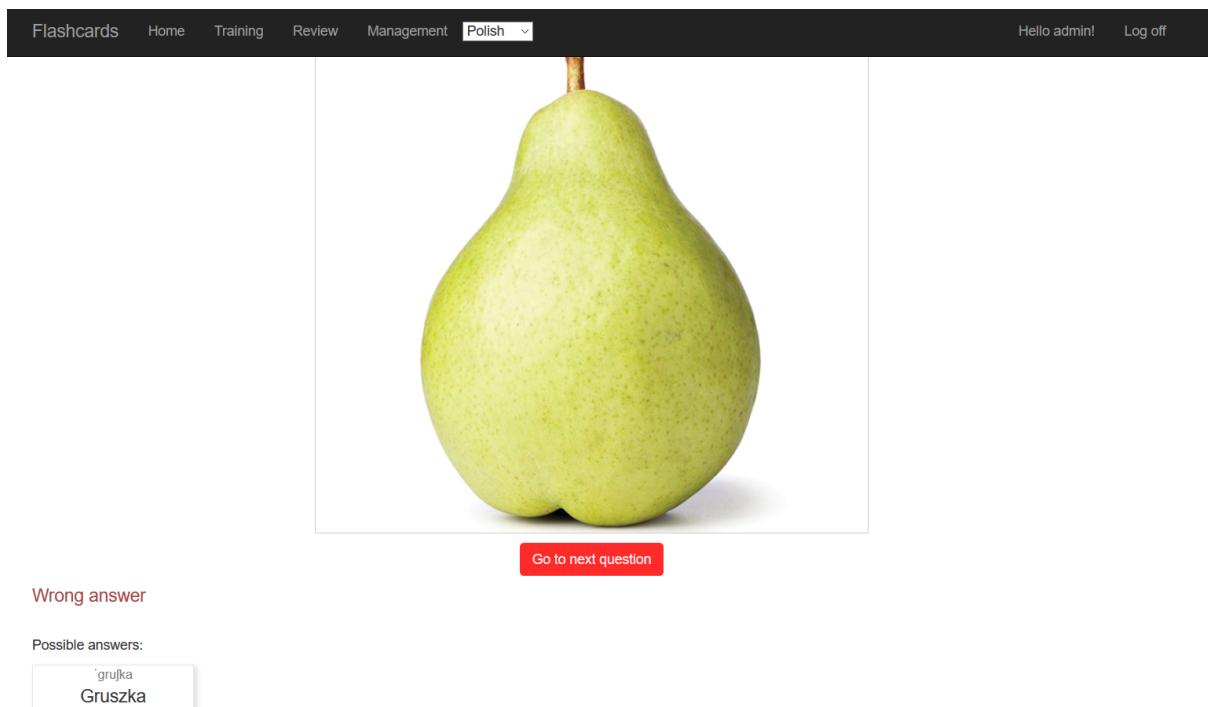
W przypadku poprawnej odpowiedzi na pytanie fiszka jest usuwana z kolejki i dodawana do zapamiętyanych fiszek, w przeciwnym wypadku przechodzi na koniec kolejki zwiększając ilość niepoprawnych odpowiedzi o 1.

Ilość niepoprawnych odpowiedzi będzie decydować o wskaźniku siły dla danej fiszki po jej wstępnym nauczeniu.

10.1.2. Powtórka

W powtórce mogą wystąpić jedynie te fiszki, na które użytkownik odpowiedział poprawnie w trakcie treningu i ich interwał czasowy od ostatniego powtórzenia już minął. Ten etap także składa się z kolejki, lecz jest ona maksymalnie 30 elementowa.

W przypadku poprawnej odpowiedzi na pytanie fiszka jest usuwana z kolejki i obliczany dla niej jest nowy interwał oraz siła z jaką jest zapamiętana. W przeciwnym wypadku fiszka jest przesuwana w pewne miejsce kolejki obliczane na podstawie poprzednich niepoprawnych odpowiedzi na tą fiszkę w danej powtórce.

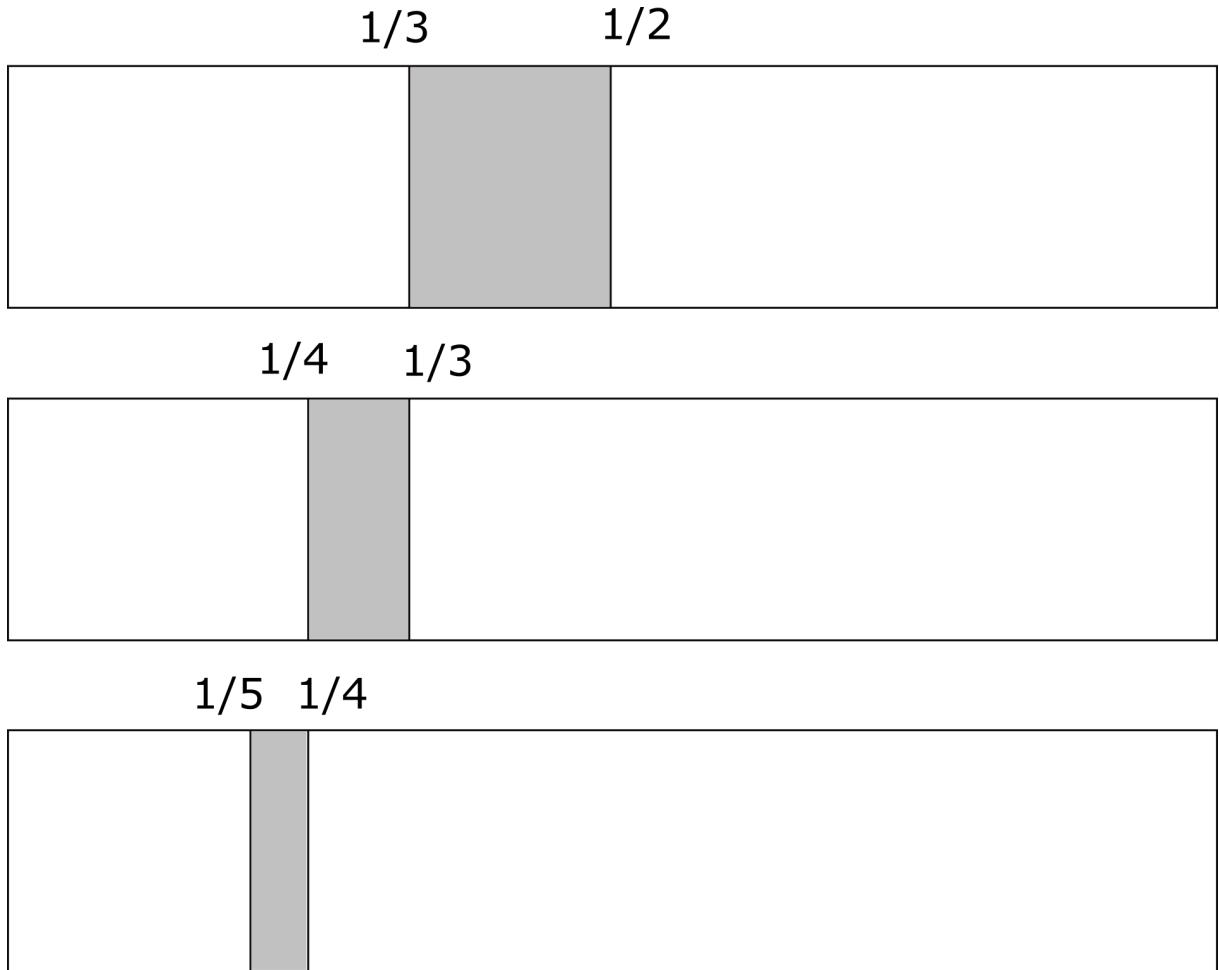


Rys. 10.2. Widok niepoprawnej odpowiedzi.

Na poniżej tabeli i obrazku można zobaczyć wewnętrz jakich zakresów znajdzie się wstawiana fiszka. Warto tutaj zaznaczyć iż 0 będzie oznaczała tutaj pierwsze miejsce w kolejce, zaś 1 będzie ostatnim miejscem w kolejce. Dla przykładu jeśli fiszka ma zostać wstawiona między pozycją $\frac{1}{3}$ a $\frac{1}{2}$ w kolejkę, która liczy aktualnie 20 kart. To zostanie ona wstawiona na pozycje 6,7,8,9, lub 10. Miejsce zostanie wybrane za pomocą liczby pseudolosowej.

Ilość złych odpowiedzi	Minimalna pozycja w kolejce	Maksymalna pozycja w kolejce
1	$\frac{1}{3}$	$\frac{1}{2}$
2	$\frac{1}{4}$	$\frac{1}{3}$
3 lub więcej	$\frac{1}{5}$	$\frac{1}{4}$

Tabela 10.1. Miejsce w którym znajdzie się fiszka po niepoprawnej odpowiedzi w trybie powtórki



Rys. 10.3. Zobrazowanie informacji zawartych w tabeli 10.1

10.1.3. Siła

Siła jest atrybutem zapamiętanych fiszek dla danego użytkownika. Jest liczbą stałoprzecinkową z zakresu 0-1, gdzie 0 oznacza kompletnie nie zapamiętaną fiszkę, a 1 bardzo dobrze zapamiętaną.

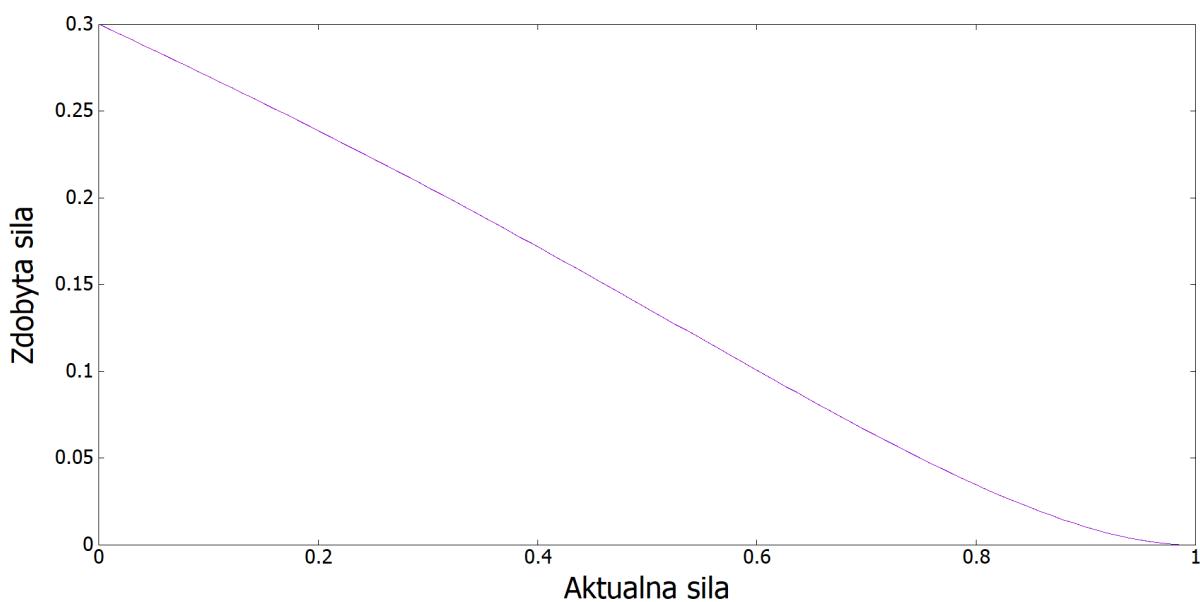
W przypadku poprawnej odpowiedzi na daną fiszkę siła jest obliczana na podstawie poniższych wzorów:

$$r(x) = (1 - x^3) * corr \quad (10.1)$$

$$\text{Nowa siła} = \frac{2.33x + r(x)}{2.33 + r(x)} \quad (10.2)$$

Gdzie:

- x - aktualna ilość siły
- corr - Wskaźnik ważności danego tłumaczenia pomnożony przez poprawność wpisanego słowa.

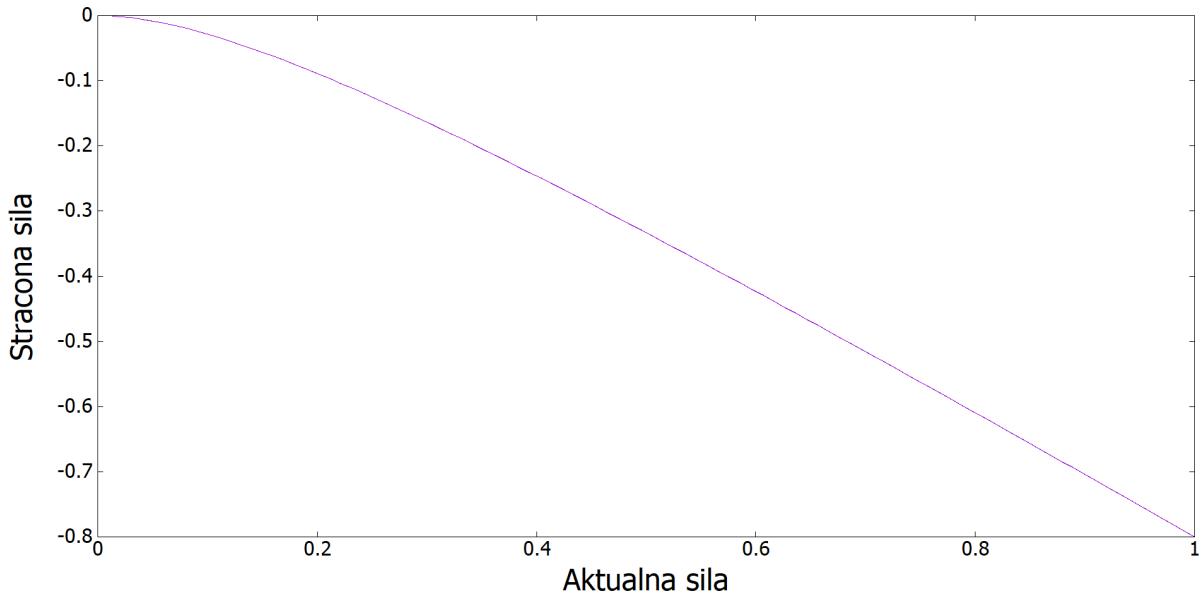


Rys. 10.4. Wykres przedstawia ilość zdobytej siły po poprawnej odpowiedzi. Powstał na podstawie różnicy między nową siłą a jej poprzednią wartością.

W przypadku poprawnej odpowiedzi na pytanie, które miało już przynajmniej 1 niepoprawną odpowiedź wzór na nową wartość siły wygląda następująco:

$$\text{Nowa siła} = 0.25 * \frac{x}{0.25L + x} \quad (10.3)$$

Gdzie L oznacza ilość niepoprawnych odpowiedzi.



Rys. 10.5. Wykres przedstawia ilość straconej siły dla $L = 1$. Wzór 10.3.

Wzór na siłę po zakończonym treningu dla danej fiszki wygląda następująco:

$$\text{Nowa siła} = \frac{0.5}{L + x} \quad (10.4)$$

10.1.4. Interwały czasowe

Pomiędzy kolejnymi wystąpieniami danej fiszki w trybie powtórki musi minąć ścisłe określony interwał czasowy, który w zależności od poprawności odpowiedzi może maleć lub rosnąć. Interwał czasowy zawsze wyrażany jest w dniach. Pierwsze dwa interwały mają odpowiednio wielkość 1 i 6 dni.

W przypadku poprawnej odpowiedzi na dane pytanie nowy interwał jest liczony według następującego wzoru:

$$I(n) = \begin{cases} 1 & n = 1 \\ 6 & n = 2 \\ [I(n-1) * (1 + x * \text{factor})] & n \geq 3 \end{cases} \quad (10.5)$$

$$\text{factor} = 1 - \frac{I(n-1)}{26} \quad (10.6)$$

Gdzie:

- $I(n)$ - n-ty interwał
- x - Siła obliczona ze wzoru 10.2

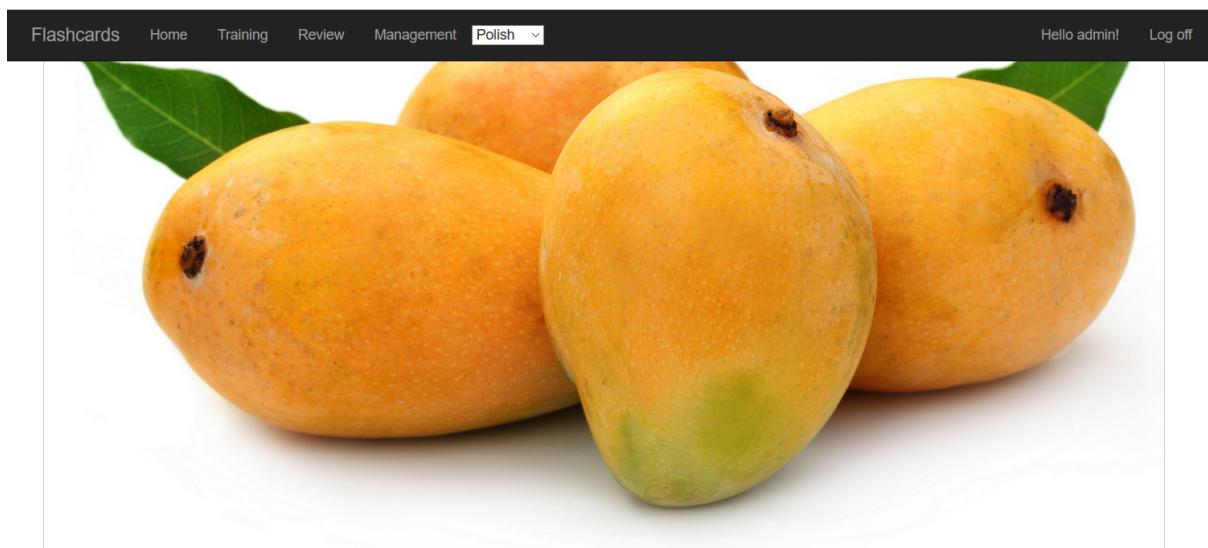
W przypadku poprawnej odpowiedzi poprzedzonej serią niepoprawnych odpowiedzi interwał jest liczony według następującego wzoru:

$$I(n) = \left\lfloor I(n-1) * \frac{1-x}{0.77} \right\rfloor \quad (10.7)$$

Gdzie x oznacza siłę liczoną według wzoru 10.3.

10.1.5. Wykrywanie poprawności odpowiedzi

W celu określenia poprawności odpowiedzi używa się metryki, która w przedziale 0-1 określa podobieństwo dwóch wyrazów. Używana jest w tym celu odległość Levenshteina określająca odległość między dwoma wyrazami na podstawie najmniejszej ilości operacji prostych jakie są potrzebne, aby przekształcić jeden wyraz w drugi. [26]



Rys. 10.6. Widok poprawnej odpowiedzi.

Operacja prosta może być:

- Usunięciem danego znaku
- Wstawieniem nowego znaku
- Zamianą danego znaku na inny

W celu uzyskania miary podobieństwa wyrazów używa się wzoru:

$$L = \frac{|s_1| + |s_2|}{2}^{1.1} \quad (10.8)$$

$$\text{Podobieństwo} = \frac{L - D}{L} \quad (10.9)$$

Gdzie :

- s_1, s_2 - wyrazy które porównujemy
- D - dystans Levenshteina

Odpowiedź jest poprawna, jeśli:

- Ma podobieństwo większe niż 0.82 dla trybu treningowego.
- Ma podobieństwo większe niż 0.7 w trakcie powtórki.

10.1.6. Możliwe odpowiedzi

Każda fiszka dla danego języka ma zestaw tłumaczeń, które definiują możliwy zbiór poprawnych odpowiedzi. Każde tłumaczenie składa się z:

- Poprawnej odpowiedzi.
- Zapisu fonetycznego.
- Ważności tłumaczenia, która jest używana we wzorze 10.1.

W trakcie udzielania odpowiedzi na pytanie, wybierane jest to tłumaczenie, które ma największą miarę podobieństwa (10.9) wśród tłumaczeń dla danej fiszki.

10.1.7. Obrazki

Każda fiszka musi zostać zaopatrzona w odpowiedni zestaw obrazów, z których jeden z nich zostanie wylosowany do procesu sprawdzenia wiedzy użytkownika. Ważne jest, aby obrazki jak najlepiej oddawały przedstawiane słowo i były jak najbardziej zróżnicowane.

10.1.8. Dostosowanie wzorów

W poprzednich podrozdziałach pojawiły się wzory związane z interwałami oraz siłą. Wzory 10.2, 10.3, 10.6 i 10.7 powstały na zasadzie zdefiniowania sparametryzowanych równań, których parametry były dobierane na zasadzie znalezienia najlepszego ich zestawu na podstawie symulacji. Do symulacji były dobierane skończone zestawy wartości parametrów z przestrzeni rozwiązań.

Na potrzeby symulacji powstały następujące równania:

$$\text{Nowa siła} = \frac{Ax + r(x)}{A + r(x)} \quad (10.10)$$

$$\text{Nowa siła} = B * \frac{x}{C * L + x} \quad (10.11)$$

$$I(n) = \left\lfloor I(n-1) * \frac{1-x}{D} \right\rfloor \quad (10.12)$$

$$\text{factor} = 1 - \frac{I(n-1)}{E} \quad (10.13)$$

Wzory 10.10, 10.11, 10.12 i 10.13 są odpowiednio odpowiednikami wzorów 10.2, 10.3, 10.7 i 10.6. Dla każdej ze zmiennych został określony przedział zmienności w jakim mogą się one zmieniać. Został on określony następująco:

$$1 \leq A \leq 9 \quad (10.14)$$

$$0.25 \leq B \leq 12 \quad (10.15)$$

$$0.25 \leq C \leq 10 \quad (10.16)$$

$$0.25 \leq D \leq 5 \quad (10.17)$$

$$20 \leq E \leq 50 \quad (10.18)$$

Zmienne A,B,C i D przyjmują zestaw wartości ze swoich przedziałów. Wartości, które przyjmują są od siebie równoodległe. T.j każda następna wartość była osiągana poprzez zwiększenie poprzedniej o stałą wartość dla danej zmiennej.

Zmienna E zawierała wszystkie liczby całkowite ze swojego przedziału.

Dla każdego zestawu zmiennych została przeprowadzona symulacja, która polegała na stworzeniu wirtualnego ucznia, który przez 360 dni poświęcał 60 minut na naukę. Każda próba odpowiedzi na daną fiszkę powodowała utratę 1 minuty na naukę co w efekcie dawało uczniowi możliwość przeglądnięcia 60 fiszek danego dnia. Następnie uczeń wracał do nauki dokładnie po 24 godzinach. Z powyższego wynika fakt, iż między dwoma kolejnymi rozpoczęciami powtórki mijało 25 godzin.

Co więcej, uczeń każdego dnia dostawał 5 nowych fiszek do swojego zestawu fiszek, których się uczył.

Wynikiem symulacji jest ilość nauczonych fiszek, które miały przynajmniej 5 powtórzeń i są zapamiętane przez ucznia na poziomie przynajmniej 75% (10.19). Celem jest znalezienie takiego zestawu wartości zmiennych, który zmaksymalizuje ten wynik.

10.1.9. Wirtualny uczeń

Wirtualny uczeń potrafi zapamiętywać informację zgodnie z krzywą zapominania. Uczeń wraz z bieżącym czasem traci szansę na poprawną odpowiedź na daną fiszkę. Procentowa szansa na odpowiedź jest obliczana na podstawie wzoru:

```
D:\programowanie\C#\Flashcards\Tweaker\bin\Debug\Tweaker.exe
(1, 0.25, 0.25, 0.25, 20) - 152
(1, 0.25, 0.25, 0.25, 22) - 152
(1, 0.25, 0.25, 0.25, 26) - 153
(1, 0.25, 0.25, 0.25, 34) - 155
(1, 0.25, 0.25, 0.25, 36) - 155
(1, 0.25, 0.25, 0.25, 40) - 162
(1, 0.25, 0.25, 0.77777777777778, 20) - 966
(1, 0.25, 0.25, 0.77777777777778, 21) - 977
(1, 0.25, 0.25, 0.77777777777778, 27) - 991
```

Rys. 10.7. Okno programu do znajdywania wartości parametrów. W nawiasie są zawarte wartości parametrów, zaś po myślniku ilość nauczonych fiszek. Program informuje nas o nowej kombinacji wartości parametrów, jeśli symulacja ma większy wynik niż jej poprzedniczki.

$$R = e^{\frac{-t}{s}} \quad (10.19)$$

Gdzie:

- t - czas od ostatniego zobaczenia danej fiszki.
- s - Ilość zobaczeń danej fiszki + 1.

Uznajemy, iż uczeń po zobaczeniu fiszki w odstępie czasu wynoszącym $t = 0$ ma 100% szansę na poprawną odpowiedź. Zmienna czasowa zawiera tutaj ilość dni, które upłynęły od ostatniego powtórzenia. Więc $t = 1$ oznacza, iż od ostatniego powtórzenia upłynął 1 dzień.

10.1.10. Wyniki symulacji

W wyniku przeprowadzonych symulacji udało się uzyskać poniższy zestaw zmiennych:

$$A = 2.33 \quad (10.20)$$

$$B = 0.25 \quad (10.21)$$

$$C = 0.25 \quad (10.22)$$

$$D = 0.77 \quad (10.23)$$

$$E = 26 \quad (10.24)$$

10.2. Narzędzia administratorskie

Administrator posiada dodatkowe narzędzia, które pozwalają mu na :

- Przeglądanie wszystkich fiszek.
- Dodawanie nowych fiszek do systemu.
- Edycja danej fiszki - dodanie lub edycja tłumaczeń, dodanie lub usunięcie powiązanych obrazków.

Prawa administracyjne nadaje się poprzez nadanie roli administratora w tabeli **AspNetUser-Roles**. Aktualnie należy użyć zapytania SQL, aby takowe uprawnienia nadać.

Return

English ▼

Connected	Image	Translation	Pronunciation	Significance
		Grape	grieip	1 ▼
		Grapes	grieips	1 ▼
		Fruit	fuu:t	0.01 ▼

Add new image

No file selected.
Send

Images

Remove

Original filename: Grapes11.jpg
Uploaded at date : 12/30/2017
AuthorID : ba2688ed-bc1a-4172-8b66-adfac7f8028b

Remove

Original filename: green-grapes.jpg
Uploaded at date : 12/30/2017
AuthorID : ba2688ed-bc1a-4172-8b66-adfac7f8028b

Remove

Original filename: Table_grapes_on_white.jpg
Uploaded at date : 12/30/2017
AuthorID : ba2688ed-bc1a-4172-8b66-adfac7f8028b

New translation

Translation

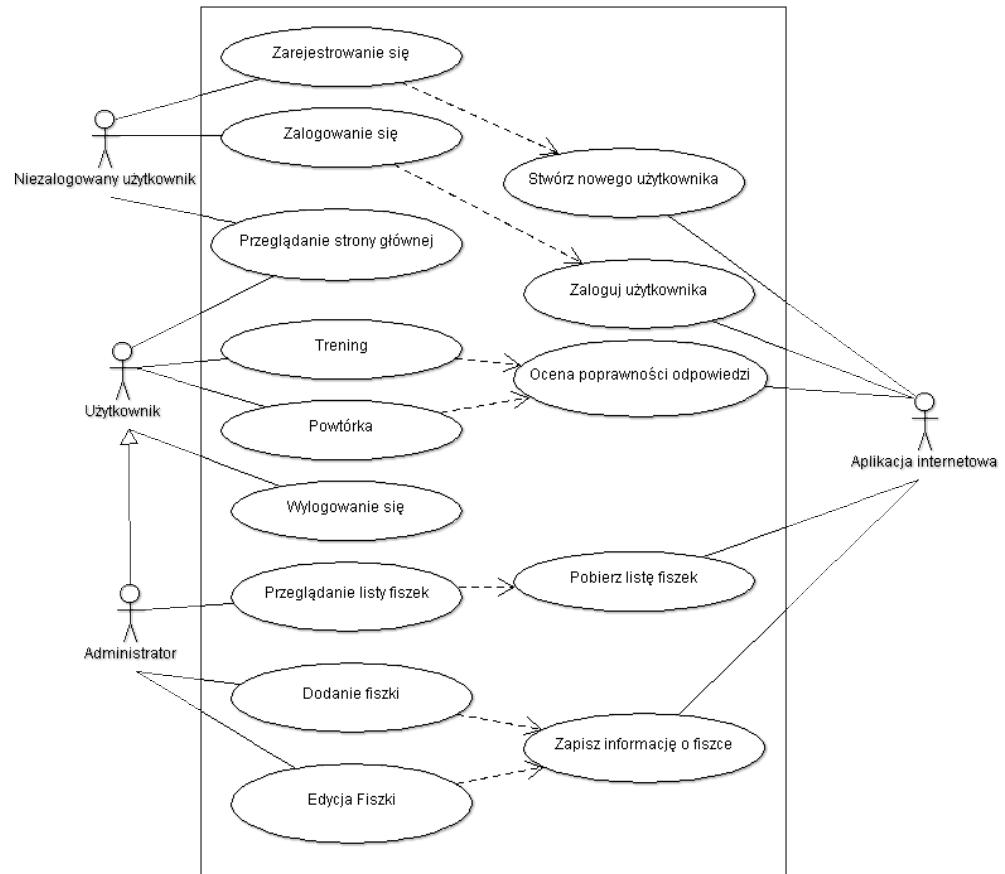
Pronunciation

Significance

Add

Rys. 10.8. Widok edycji fiszki.

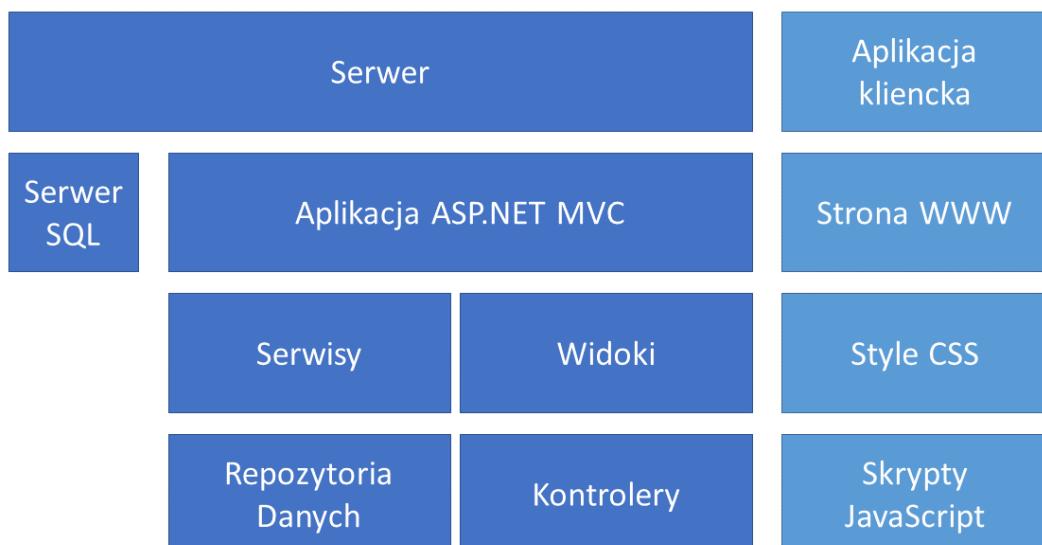
10.3. Diagram przypadków użycia



Rys. 10.9. Diagram przypadków użycia.

11. Kwestie techniczne

11.1. Struktura programu



Rys. 11.1. Diagram przedstawiający strukturę programu.

11.1.1. Repozytoria

Repozytoria odpowiadają za dostęp do bazy danych(4.3). Zawierają wewnątrz siebie kontekst do bazy danych, który pochodzi z frameworka Entity Framework. Dostarcza on metody operacji na bazie poprzez dostarczenie obiektu typu **DbSet<T>**, który umożliwia wykonanie zapytań na bazie danych. RepositoryBase implementuje wszystkie operacje, które są potrzebne w trakcie pracy z bazą danych. Klasy po nim dziedziczące wykorzystują metody bazowe w celu stworzenia bardziej skomplikowanych zapytań skonstruowanych pod daną tabelę zgodnie z wymaganiami biznesowymi.

```
public class RepositoryBase<TEntity, TContext> : ↵
    RepositoryBaseEntityless<TContext>, IRepository<TEntity>, IDisposable
{
    where TEntity : class, new()
    where TContext : DbContext, new()

    protected DbSet<TEntity> dbSet;

    public DbSet<TEntity> DbSet { get { return dbSet; } }

    public virtual IQueryable<TEntity> Query { get { return dbSet.AsQueryable(); } }

    public RepositoryBase(TContext context) : base(context)
    {
        dbSet = context.Set<TEntity>();
    }
    /*
    Reszta metod wewnętrz których realizowany jest dostęp do danych
    */
}
```

Listing 11.1. Część implementacji RepositoryBase



Rys. 11.2. Diagram klas wszystkich repozytoriów w projekcie.

11.1.2. Kontrolery

Kontrolery odpowiadają za wygenerowanie odpowiedzi dla danych zapytań. Wszystkie kontrolery wewnętrz aplikacji dziedziczą po klasie **ControllerBase**, która kolej dziedziczy po klasie **Controller**. ControllerBase odpowiada za zdefiniowanie wspólnych akcji jakie mogą być wykonywane przez wszystkie kontrolery. Ma to na celu zmniejszenie redundancji kodu jak i zwiększenie jego czytelności.

```
public class ControllerBase : Controller
{
    private readonly IPopupService popupService;
    protected readonly ISessionService sessionService;

    public ControllerBase(IPopupService popupService, ISessionService ←
        sessionService)
    {
        this.popupService = popupService;
        this.sessionService = sessionService;
    }

    public ActionResult RedirectToAction(string errorMessage = "Error")
    {
        AddError(errorMessage);
        return RedirectToAction();
    }

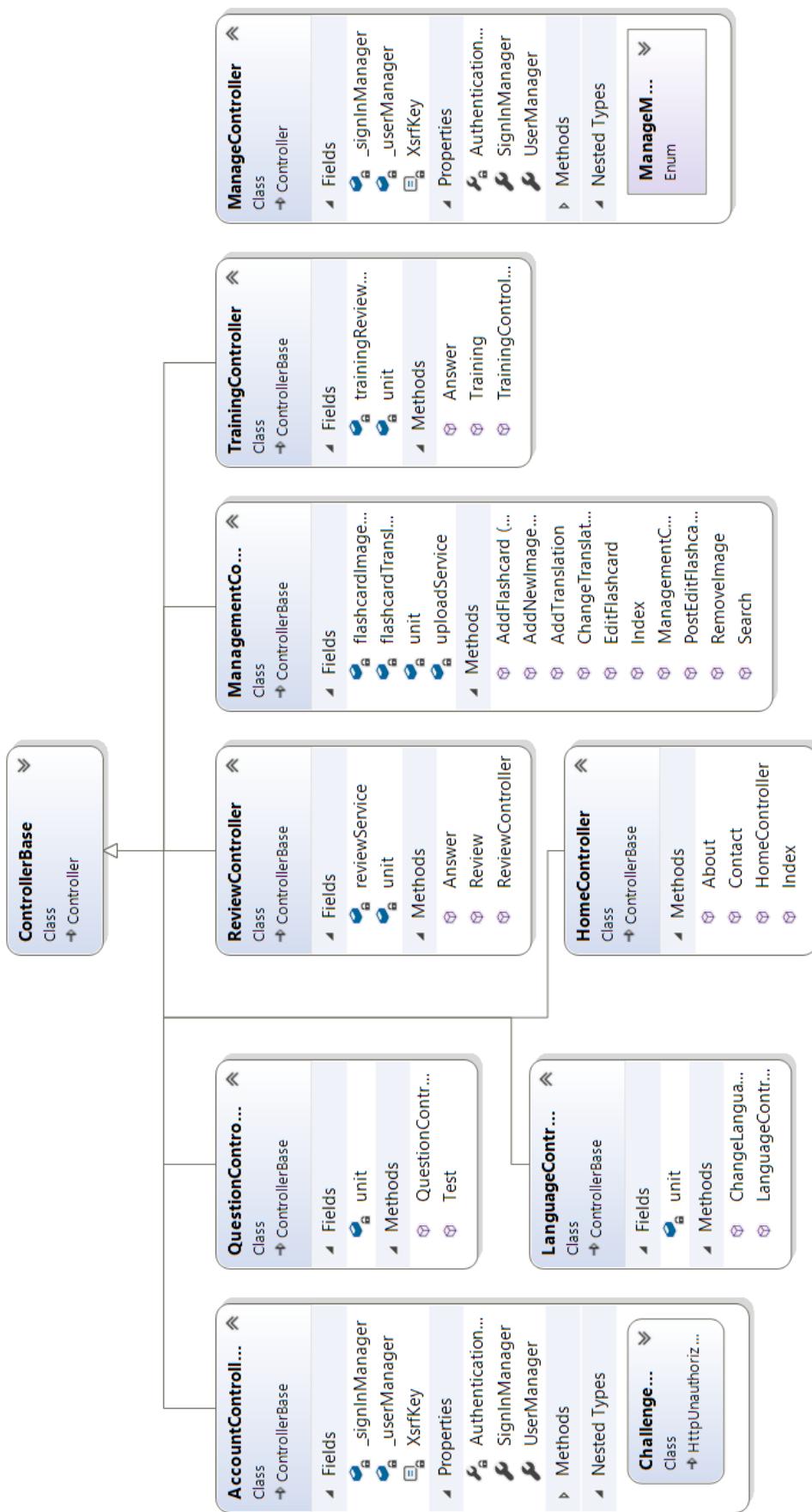
    /*Reszta metod zaimplementowanych przez ControllerBase*/
}
```

Listing 11.2. Część implementacji ControllerBase

Każdy kontroler zawiera zestaw akcji, które wykonywane są w wyniku zapytań użytkownika i zwracają obiekt typu **ActionResult**. W zależności od tego jaki typ ma obiekt zwracany dana akcja potrafi:

- wyrenderować stronę HTML.
- zwrócić skrypt JavaScript.
- Zwrócić informację zapisane w formacie JSON.

Dodatkowo, jeśli dana akcja ma zwrócić widok HTML to musi być dla niej zdefiniowany specjalny plik .cshtml, w którym znajduje się kod możliwy do wyrenderowania przez aktualny silnik renderujący. Metody dla akcji potrafią przyjmować argumenty, do których wartości przesypane są na podstawie parametrów przekazanych w zapytaniu od użytkownika.

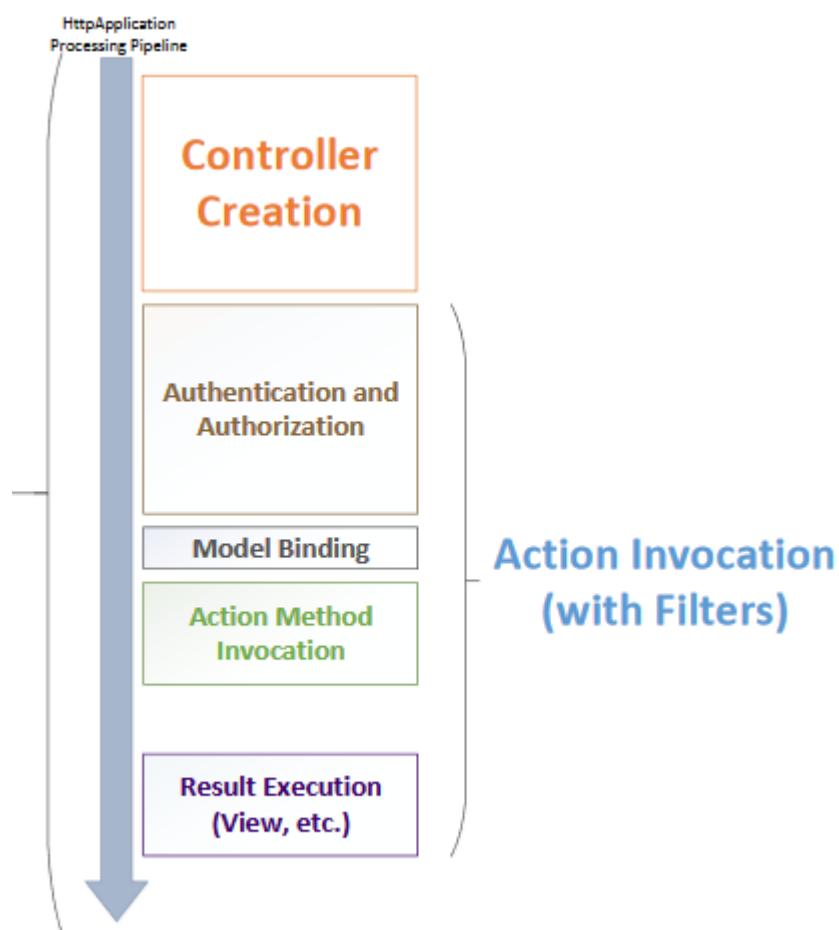


Rys. 11.3. Diagram klas wszystkich kontrolerów w projekcie

11.1.2.1. Cykl życia kontrolera

Cykl życia kontrolera składa się z 5 etapów:

- Kreacji
- Autoryzacji i autentykacji.
- Bindowania wartości do modelu.
- wywołania odpowiedniej akcji
- zwrócenie rezultatu działania (**ActionResult**)



Rys. 11.4. Cykl życia kontrolera. Źródło: docs.microsoft.com.

Na początku aplikacja musi określić jaki kontroler należy powołać do życia w celu obsługi zapytania. W tym celu są wykorzystywane tabele routingu, które mapują dane adresy URI na odpowiedni kontroler.

Dla przykładu: Domyślnie adres URI localhost:25388/Home/Index zostanie zmapowany na HomeController, który wykonuje akcję Index.

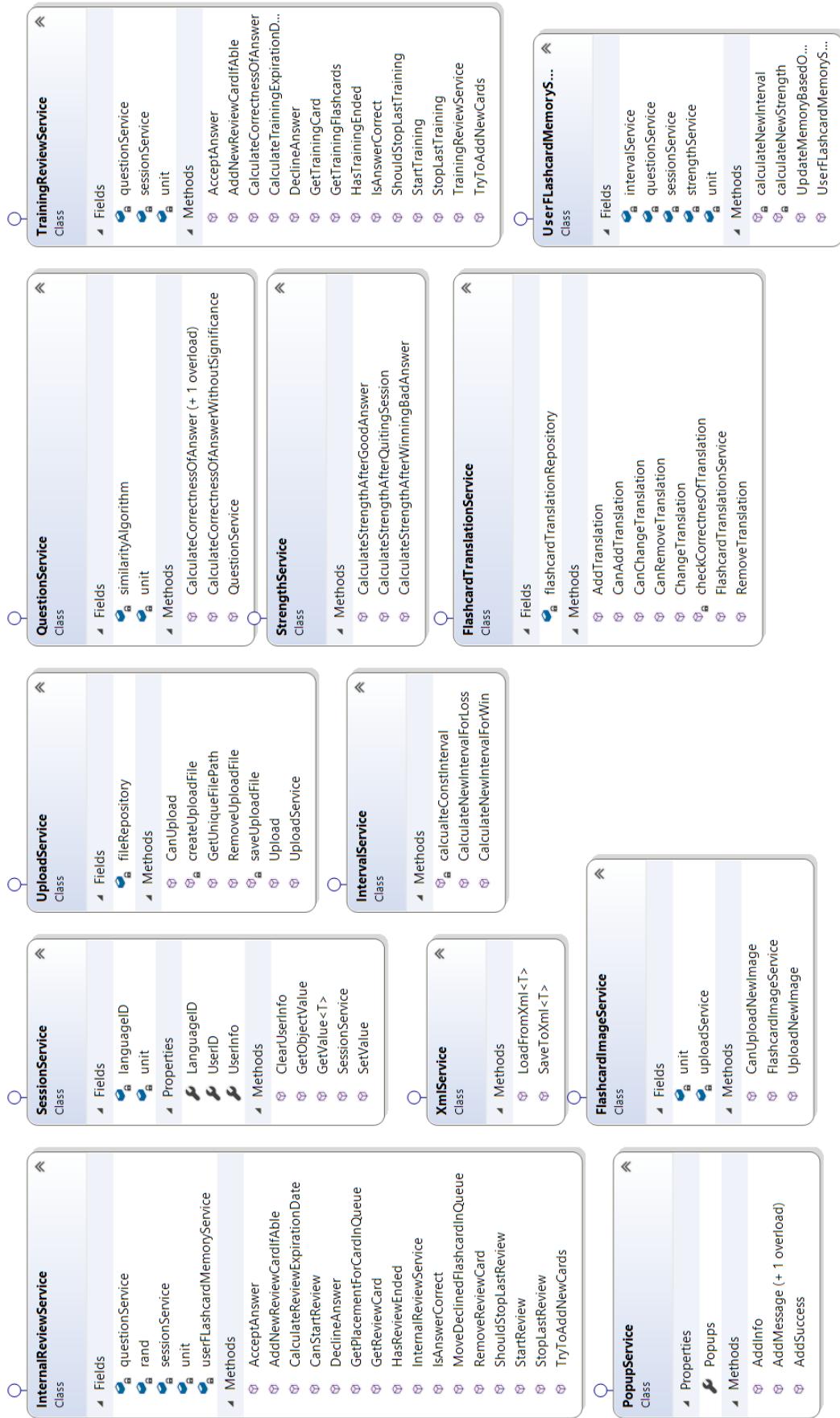
Następnie **DependencyResolver** stara się utworzyć instancje danego kontrolera. Gdyby nie było DependencyResolvera, bądź zwróciłby on null w wyniku swojego działania to zostaje stworzony kontroler na podstawie jego bezparametrowego publicznego konstruktora. Jeśli takowego by nie było to rzucany jest wyjątek. Po utworzeniu kontrolera aplikacja stara się zbadać prawa do danego zasobu na podstawie wewnętrznych reguł i atrybutów przypisanych do danej akcji i kontrolera. Jeśli zdarzy się sytuacja w której użytkownik nie ma praw do danego zasobu to zostaje przekierowany na odpowiednią stronę błędu 401 lub 403.

Kolejnym krokiem jest zbindowanie możliwie jak największej ilości parametrów przekazanych w zapytaniu do argumentów metody, która po tym zabiegu zostanie wywołana.

Na samym końcu następuje zwrócenie rezultatu działania akcji, który zostaje zwrócony użytkownikowi.

11.1.3. Serwisy

Serwisy zawierają w sobie implementacje całej logiki biznesowej. Zostały podzielone w celu logicznego pogrupowania różnych operacji wykonywanych przez aplikację. Dla przykładu **StrongthService** służy jedynie do obliczania siły na podstawie wzorów 10.3 i 10.2.



Rys. 11.5. Diagram klas wszystkich serwisów w projekcie

11.2. Struktura rozwiązania

Rozwiązanie jest podstawową jednostką pracy wewnątrz programu Visual Studio. Zawiera w sobie wszystkie projekty, ich ustawienia, informację o kolejności ich komplikacji oraz dodatkowe pliki niezwiązane z żadnym projektem. Projekt posiada solucję, w której w skład wchodzi:

- Projekt Common - zawierający metody ogólnego przeznaczenia.
- Projekt Entities - wykorzystujący wzorzec repozytorium. Służy do łączenia się z bazą danych.
- Projekt FlashcardCommon - zawierający metody związane z całym projektem.
- IntegrationTests - zawiera testy integracyjne.
- UnitTests - zawiera testy jednostkowe.
- TestSuite - zawiera zestaw narzędzi wykorzystywany przez testy.
- Utilities - zawiera metody ogólnego przeznaczenia.
- WebUtils - zawiera metody ogólnego przeznaczenia dla środowiska ASP.NET MVC.
- Management - zawiera porzuconą okienkową aplikację administratora.
- Services - zawiera wszystkie serwisy wykorzystywane przez projekt.
- Flashcards - zawiera właściwą aplikację internetową.
- Tweaker - zawiera program do znajdowania najefektywniejszego zestawu parametrów dla procesu nauczania.
- wirtualny folder git - zawiera w sobie pliki związane z repozytorium git¹.

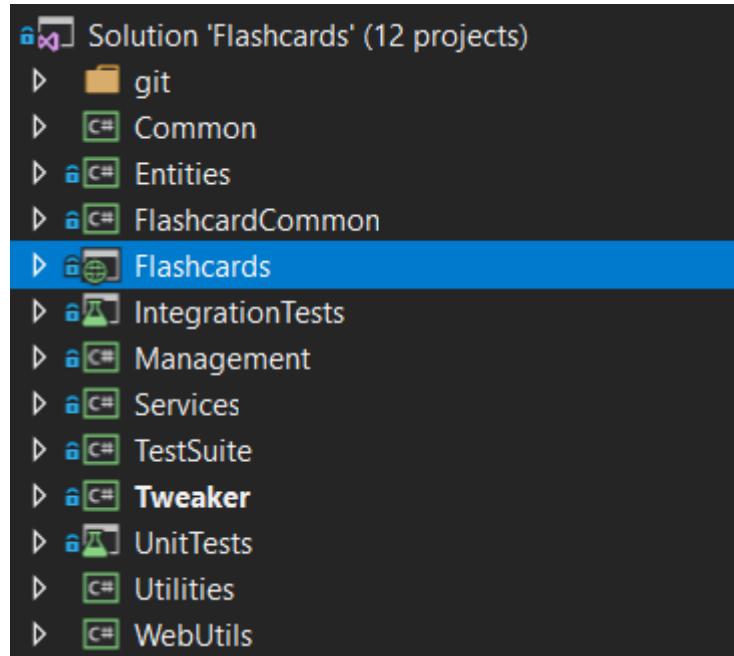
11.2.1. Management

Jest to jeden z projektów, który zasługuję na większą uwagę. Początkowym założeniem aplikacji było stworzenie dodatkowego projektu okienkowego, który byłby narzędziami administratora. Projekt został bardzo szybko porzucony. Wynikało to z dwóch faktów :

- Konieczności utrzymania dwóch różnych względem siebie projektów front-endowych.
- Łatwiejsza w użytku i utrzymaniu jest jedna aplikacja internetowa.

Projekt nie został skasowany ze względu na to, iż nadal posiada działający interfejs. Gdyby w przyszłości wynikła potrzeba użytkowania aplikacji okienkowej to jest już ona gotowa.

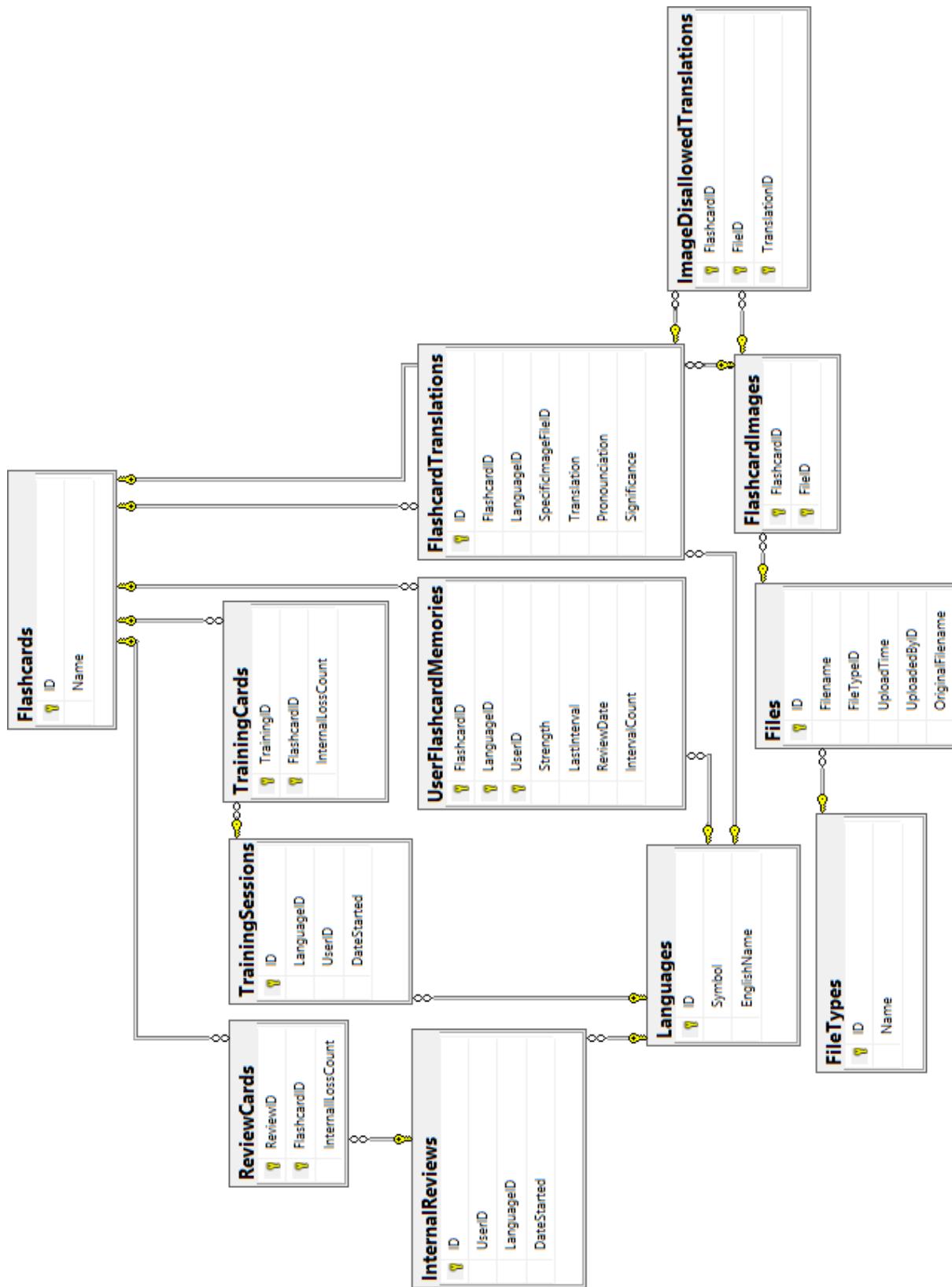
¹Repozytorium git służy do wersjonowania kodu.



Rys. 11.6. Rysunek przedstawiający rozwiązanie projektu.

11.3. Baza danych

W projekcie zastosowano bazę danych Microsoft SQL Server. Jest to najlepiej wspierana baza danych przez C#. Dodatkową zaletą jest jej łatwość integracji z Entity Framework.



Rys. 11.7. Diagram ERD bazy danych.

11.3.1. Informacje dodane przez ASP.NET MVC

ASP.NET dostarcza bibliotekę Identity, za której pomocą można łatwo wdrożyć operacje logowania, rejestracji oraz obsługi sesji dla użytkowników. Domyślnie Identity używa lokalnej bazy sqlite² w celu zapisu informacji o użytkownikach. Aby używać bazy danych SQL należy zmienić connection string, który przechowuje informację o tym jaką bazą danych powinna zostać użyta. Po takiej zmianie zostaną utworzone automatycznie tabele odpowiedzialne za przechowywanie informacji o użytkownikach na nowej bazie danych.

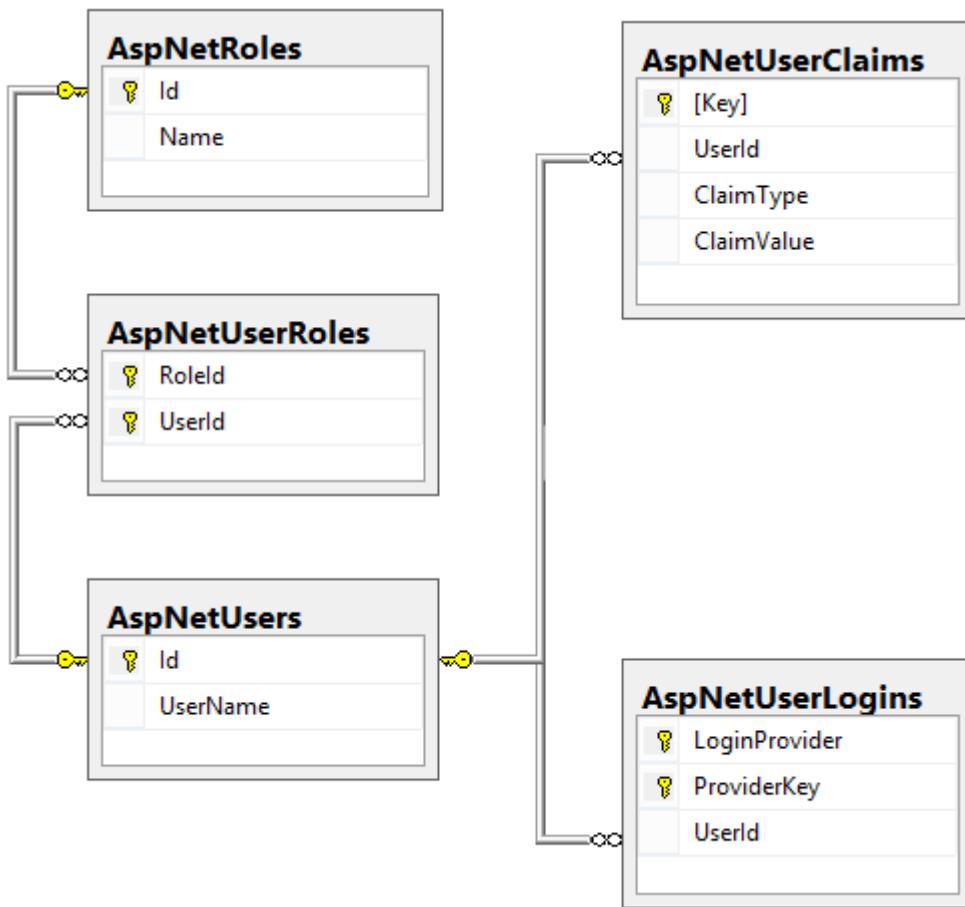
```
<add name="IdentityConnection" connectionString="Data ←
    Source=DESKTOP-8H0EDBL\DAMIANSQL;Initial Catalog=Flashcards;Integrated ←
    Security=True;MultipleActiveResultSets=True;Connect Timeout=120;" ←
    providerName="System.Data.SqlClient" />
```

Listing 11.3. Connection string dla biblioteki Identity.

Identity dodaje następujące tabele do projektu:

- __MigrationHistory
- AspNetRoles
- AspNetUserClaims
- AspNetUserLogins
- AspNetUserRoles
- AspNetUsers

²Baza danych, która operuje na danym pliku binarnym w celu używania bazy danych bez konieczności posiadania dodatkowego serwera.



Rys. 11.8. Diagram ERD ASP.NET Identity

MigrationHistory zawiera informacje stworzone w wyniku migracji z danego rozwiązania bazodanego do innego. W przypadku tego projektu są tutaj zawarte informacje o zmianie domyślnej bazy sqlite na Microsoft SQL Server.

AspNetRoles przechowuje dane o tym, jakie role mogą zostać przypisane do użytkowników.

AspNetUserClaims zawiera żądania dostępu do konta danego użytkownika. Nie jest ona wykorzystywana w tym projekcie.

AspNetUserLogins zawiera informacje o loginach dla danych użytkowników, za pomocą których mogą się logować na swoje konto.

Wewnątrz tabeli AspNetUserRoles znajdują się informacje o przypisaniu użytkownika do danej roli.

AspNetUsers przechowuje informacje o użytkowniku.

Wartym uwagi jest fakt, iż biblioteka Identity przechowuje identyfikatory w postaci ciągu 128 znakowego.

11.3.2. Obsługa treningu

Za obsługę treningu odpowiadają tabele:

- TrainingSessions
- TrainingCards

TrainingSessions przechowuje informacje o danej sesji treningowej pod którą podpięte są aktualnie nauczane karty poprzez tabelę TrainingCards. Połączenie odbywa się za pomocą relacji jeden do wielu.

11.3.3. Obsługa powtórki

Tabele wewnętrznych których znajdują się informacje o powtórce to:

- InternalReviews
- ReviewCards

Internal Reviews zawiera informacje o aktualnej powtórce dla danego użytkownika podczas gdy ReviewCards zawierają fiszki przynależące do danej powtórki. Połączenie odbywa się za pomocą relacji jeden do wielu.

11.3.4. Informacje o Fiszках

Informacje o fiszkach są przechowywane w tabelach:

- Flashcards
- FlashcardTranslations
- FlashcardImages
- ImageDisallowedTranslations

Pierwsza z tabel zawiera Identyfikator danej fiszki wraz z jej nazwą angielską, która jest wykorzystywana w narzędziach administracyjnych. Dla tej tabeli został stworzony unikalny indeks dla pola Name. Za pośrednictwem tego pola jest zrobione wyszukiwanie wewnętrznych narzędzi administracyjnych, więc przyczynia się to do zwiększenia wydajności zapytań. Dodatkowo, ważne jest to, aby fiszki miały unikalną nazwę systemową, aby mogły być rozróżnialne przez administratora.

FlashcardsTranslations zawiera informacje o tłumaczeniach dla danej fiszki w danym języku. Posiada jeden indeks dla pól FlashcardID i LanguageID. Ta decyzja umotywowana jest faktem

zwiększenia szybkości zwieracanych zapytań. Program bardzo często będzie wyszukiwał translacji dla danej fiszki według języka i jej identyfikatora.

FlashcardImages zawiera informacje o tym jaki zuploadowany plik jest dopisany dla danej fiszki. Informacja o tym jaki to jest plik i gdzie się znajduje jest zawarta w tabeli Files.

Czasami może się zdarzyć, iż dany obrazek nie oddaje w sposób precyzyjny danego tłumaczenia. Z tego powodu powstała tabela ImageDisallowedTranslations, która określa dla jakiego tłumaczenia dany obrazek nie może zostać użyty.

11.3.5. Upload plików

Wszystkie informacje o zuploadowanych plikach na serwer są określone przez tabele:

- Files
- FileTypes

Tabel Files zawiera informacje o zuploadowanym pliku. Warto tutaj zwrócić uwagę na to, iż nie jest tutaj przechowywana bezwzględna ścieżka do pliku, lecz jedynie jego nazwa wewnętrz kolumny Filename. Określenie dokładnej ścieżki odbywa się z pomocą typu uploadu, który jest określony przez FileTypeID.

Tabela FileTypes jest słownikiem zawierającym nazwę i identyfikator danego typu pliku.

11.4. Integracja Scala.js z ASP.NET MVC

Visual studio aktualnie nie posiada żadnego wsparcia dla scali.js[**ScalaNoSupport**]. Dodatkowo wyniki wyszukiwania w google pod hasłem "Visual studio scala.js"[**ScalaNoSupportGoogle**] nie zwracają żadnych zapytań. Można więc domniemywać, iż ta praca przeciera niejako szlak podczas prób połączenia tych 2 technologii.

W związku z powyższym pierwszą decyzją jaka powinna zostać podjęta podczas tworzenia takiego połączenia to nie używanie środowiska Visual Studio do pisania kodu Scali. Jest ono całkowicie niedostosowane do tego celu. Wewnątrz tej pracy jest używane środowisko **IntelliJ IDEA**, jednakże można w tym celu użyć każdego innego dowolnego programu (Przy mniejszych programach wystarczy nawet sam edytor tekstowy i linia poleceń).

Projekt Scali.js został utworzony w katalogu ./Scripts/scala³ z ustawienia build.sbt takimi jak na listingu 8.4.

Mając tak przygotowany projekt oraz zakładając, iż mamy już kompilowalny kod, który może zostać użyty na stronie, musimy przygotować go do wdrożenia w kod .cshtml naszej strony. Wpierw w tym celu przygotujemy paczkę skryptową (po ang. ScriptBundle), która będzie dodana przez ASP.NET MVC na stronie. Należy to zrobić wewnątrz pliku **BundleConfig.cs** w projekcie ASP.NET MVC. Kod zastosowany w projekcie jest widoczny na listingu 11.5.

³./ jest katalogiem głównym projektu ASP.NET MVC

```
bundles.Add(new ScriptBundle("~/bundles/scalaDependencies").Include(
    "~/Scripts/scala/target/scala-2.12/flashcards-jsdeps.js"
));

bundles.Add(new ScriptBundle("~/bundles/scala").Include(
    "~/Scripts/scala/target/scala-2.12/flashcards-fastojs.js"
));
```

Listing 11.4. Dodanie paczek ze skryptami dla BundleConfig.

Następnie możemy użyć tych paczek wewnątrz kodu .cshtml strony za pomocą instrukcji **Scripts.Render("~/bundles/scalaDependencies")** i **@Scripts.Render("~/bundles/scala")**.

11.4.1. Uruchamianie skryptu

Cały kod Scala.js został podzielony logicznie na pojedyncze klasy, gdzie każda z klas odpowiada danemu widokowi bądź małej grupce widoków. Po stworzeniu takowej struktury nasuwa się następujące pytanie: jak należy zrobić logikę, która jest odpowiedzialna za uruchamianie tych klas?

Wewnątrz projektu zastosowano bardzo prostą metodę, która polegała na utworzeniu klasy Main, która zawierała metodę Init możliwą do uruchomienia z poziomu kodu JavaScript. Funkcja ta przyjmowała zmienną znakową, która zawierała nazwę modułu, który powinien zostać uruchomiony w ramach danej strony. Wykorzystano tutaj instrukcję switch, która w zależności od parametru powodowała uruchomienie metody run obiektu danej klasy.

```
@JSExportTopLevel("InitFlashcard")
//JsExport w tym przypadku informuje o tym, iż ta funkcja powinna być możliwa do ←
//wywołania z poziomu JavaScript pod nazwą InitFlashcard.
def Init(moduleName: String): Unit =
{
    moduleName match
    {
        case "Management.Index" => new Flashcards.Management.Index().Run()
        case "Management.EditFlashcard" => new ←
            Flashcards.Management.EditFlashcard().Run()
        case "Training.Question" => new Flashcards.Questions.TrainingQuestion().Run()
        case "Review.Question" => new Flashcards.Questions.ReviewQuestion().Run()
    }
}
```

Listing 11.5. Dodanie paczek ze skryptami dla BundleConfig.

Nie jest to rozwiązanie satysfakcjonujące. Rozwiązanie idealne polegałoby na użycie mechanizmu refleksji (który jest niedostępny wewnątrz Scala.js), bądź podobnym, który na podstawie

ciągu znakowego byłby w stanie uruchomić daną klasę bez potrzeby pisania przez programistę osobnej logiki uruchamiania kodu dla każdej z klas.

11.5. Implementacje niektórych rozwiązań

11.5.1. Sesja użytkownika

Po stronie aplikacji ASP.NET MVC dla danego użytkownika zawsze jest podpięta sesja użytkownika, którą identyfikuje się po ciasteczkowi umieszczonym po stronie przeglądarki. Cały mechanizm zarządzania sesją został umieszczony wewnątrz serwisu **SessionService**. Wewnątrz sesji zostały umieszczone dane o:

- Aktualnie wybranym języku.
- Aktualnie zalogowanym użytkowniku. (Jeśli użytkownik był zalogowany)
- Aktywnym treningu. (Jeśli użytkownik był zalogowany)
- Aktywnej powtórce. (Jeśli użytkownik był zalogowany)

Informacja o wybranym języku decydowała o tym w jakim języku powinny odbywać się treningi i powtórzenia. Dane aktualnego użytkownika obejmowały jego ID oraz nazwę użytkownika. Były to dane, które bardzo często były używane, dlatego umieszczone je wewnątrz sesji, aby nie pobierać ich z bazy danych.

Dane o aktywnym treningu oraz powtórce umożliwiały sprawne zarządzanie kolejką bez potrzeby dokładnego odwzorowania kolejki po stronie bazy danych. Zmniejszyło to poziom skomplikowania trybu nauczania.

```
public static void SetValue(object value, [CallerMemberName] string name = "")  
{  
    HttpContext.Current.Session[name] = value;  
}  
  
public static T GetValue<T>([CallerMemberName] string name = "")  
    where T : class  
{  
    if (HttpContext.Current.Session[name] == null)  
        return null;  
  
    return (T)HttpContext.Current.Session[name];  
}
```

Listing 11.6. Metody użyte do zapisu i odczytu danych wewnątrz sesji.

11.5.2. Kolejka pytań

Przechowuje informację, w jakiej kolejności pojawiać się będą fiszki, z których użytkownik ma być odpytywany w trakcie nauki. Należy rozróżnić 2 różne kolejki pytań znajdujące się w aplikacji dla każdego z etapów nauki. Jednym z nich jest kolejka znajdująca się wewnątrz sesji użytkownika, nad którą aplikacja ma najłatwiejszą kontrolę i druga, przechowywana po stronie bazy danych, do której poszczególnych elementów aplikacja odwołuje się poprzez repozytoria.

11.5.2.1. Trening

Algorytm na samym początku treningu pobiera 5 losowych kart i wstawia je do kolejki wewnątrz sesji użytkownika jako Listę obiektów **TrainingCardInfo**. Te same fiszki zostają wstawione także po stronie bazy danych do tabeli **TrainingCards**.

W przypadku niepoprawnej odpowiedzi licznik przegranych jest tak samo uaktualniany wewnątrz sesji, jak i wewnątrz bazy danych. Karta na początku kolejki wewnątrz sesji jest przesuwana na jej koniec.

Usunięcie karty na skutek poprawnej odpowiedzi usuwa informację o obiekcie z sesji, jak i z bazy danych. Dodawana jest przy tym nowa fiszka do sesji, jak i do bazy danych, tak aby w kolejce znajdowało się maksymalnie 5 kart. Jeśli nie ma żadnych nowych przedmiotów do nauczenia się to do kolejki nie są dodawane nowe karty.

W przypadku utraty sesji (na przykład na skutek przelogowania się użytkownika) program stara się przywrócić sesję treningową dla użytkownika. Dokładna kolejność kolejki nie zostanie zachowana. Karty w kolejce zostaną ułożone rosnąco według ilości niepoprawnych odpowiedzi, które zostały zapamiętane.

11.5.2.2. Powtórka

Powtórka działa analogicznie do treningu z kilkoma różnicami:

- Przedmiotów wewnątrz kolejki jest maksymalnie 30, a nie 5.
- Używana jest lista obiektów **ReviewCardInfo** i tabela **ReviewCards**.
- W wypadku niepowodzenia fiszka nie jest wstawiana na koniec kolejki, lecz wewnątrz danego przedziału, który został opisany w sekcji 10.1.2.

Utrata dokładnej kolejności fiszek w przypadku straty sesji jest bardziej bolesna dla trybu powtórki. Pomimo tego nie jest to problem dla aplikacji, ponieważ utrata sesji odbywa się bardzo rzadko. Dodatkowo użytkownik prawdopodobnie nie zorientowałby się, że kolejność została wymieszana, ponieważ interfejs nie informuje o aktualnym stanie kolejki.

11.5.3. Podobieństwo wyrazów

Podobieństwo wyrazów opisane wewnętrz sekcji 10.1.5 zostało zaimplementowane w klasie **LevenshteinSimilarity** implementującej interfejs **ISimilarityAlgorithm**. Klasa ta jest instancjonowana tylko w jednym miejscu w programie - wewnątrz klasy **QuestionService**, której zadaniem jest sprawdzenie poprawności odpowiedzi. Ma to na celu umożliwienie łatwej podmiany algorytmu, jeśli zaimplementowany zostanie interfejs **ISimilarityAlgorithm** przez inną klasę.

```
namespace Common.Words.Similiraties
{
    public interface ISimilarityAlgorithm
    {
        //Metoda powinna zwrócić liczbę z zakresu 0-1 jako miarę podobieństwa ←
        //między podanymi wyrazami.
        double CalculateSimilarity(string original, string compared);
    }
}
```

Listing 11.7. Interfejs dla klas obliczających podobieństwo wyrazów.

11.5.4. Atrypy obiektów

Wewnątrz programu zastosowano framework Moq do tworzenia atrap obiektów. Niestety to nie zawsze wystarcza. Szczególnie w wypadku gdy potrzebna jest atrapa obiektu bazodanowego, którego wartości są przynajmniej poprawne i wypełnione. Z tego też powodu powstał interfejs **IDummyCreator<T>**, którego zadaniem jest stworzenie atrapy obiektu "T". Odbiera się to za pomocą metody **Create**, która zawsze po wywołaniu ma zwrócić pełnoprawny obiekt.

```
public class LanguageDummyCreator : IDummyCreator<Language>
{
    private static UniqueIDGenerator uniqueID = new UniqueIDGenerator();
    private Language language;
    public LanguageDummyCreator() { language = create(); }
    {   return new Language()
        {
            ID = uniqueID,
            EnglishName = RandomGenerator.GenerateString(10),
            Symbol = RandomGenerator.GenerateString(2)
        };
    }
    public Language Create()
    {
        var temp = language;
        language = create();
        return temp;
    }
}
```

Listing 11.8. Przykład kreatora atrap na podstawie LanguageDummyCreator.

11.5.4.1. Unikalne identyfikatory

Powysze atrapy bazodanowe byłyby całkowicie nieprawidłowe, gdyby nie miały unikalnych identyfikatorów. Wiele metod bazuje na założeniu unikalności identyfikatorów. Nawet w testach jednostkowych brak tej unikalności może być powodem błędów.

W celu rozwiązania tego problemu stworzono **UniqueIDGenerator**, którego zadaniem jest wygenerować unikalny identyfikator dla danej instancji tej klasy. Jego działanie jest bardzo proste i polega na inkrementowaniu licznika.

Przykład użycia tej klasy można zobaczyć na listingu 11.8.

```
public class UniqueIDGenerator
{
    private int _uniqueID = 0;
    public int UniqueID { get { return _uniqueID++; } }
    public static implicit operator int(UniqueIDGenerator generator)
    {
        return generator.UniqueID;
    }
}
```

Listing 11.9. Implementacja UniqueIDGenerator.

11.5.4.2. Enumerator

W projektach bazodanowych bardzo często dochodzi do sytuacji w których są używane tabele słownikowe. Przykładem takiej tabeli może być tabela **FileTypes**, która zawiera identyfikator i nazwę pliku.

Bardzo wygodnym rozwiązaniem przy takich tabelach jest utworzenie typu enum, który zawiera wszystkie dozwolone wartości jakie mogą być wykorzystywane przez program dla danej tabeli słownikowej. Wprowadza to statyczne sprawdzenie czy użyliśmy dobrej wartości, ponieważ mamy tylko ich skromny zakres dostępny z enuma. Kompilator powiadomi nas o tym, jeśli chcemy użyć czegoś co nie istnieje.

Dodatkowo możemy wykorzystać Intellisense⁴ do podpowiadania nam dostępnych wartości.

Niestety takie rozwiązanie ma jeden i to ważny problem. Bardzo ciężko jest jednocześnie edytować tabelę i enuma występującego w kodzie. Bez żadnego narzędzia programista zawsze musi pamiętać, iż przy edycji jednego zawsze musi zostać zedytowana druga rzecz. Może to doprowadzić do trudnych do wykrycia błędów.

Z tego też powodu stworzyłem typ **Enumerator**, który na podstawie danego Enuma wypełnia daną tabelę za pomocą Entity Frameworka odpowiednimi wartościami. Typ ten potrafi za pomocą metody **CreateNewIfAble** stworzyć nieistniejące rekordy dla danych kluczowych w tabeli. W domyślnej wersji na podstawie wszystkich wartości z danego enuma tworzy wykryte nieistniejące rekordy przypisując im odpowiednie wartości poprzez właściwości **ID** i **Name**.

⁴Jest to narzędzie podpowiadające składnię podczas pisania kodu.

```
public Enumerator<TEntity, TEnum, TContext> CreateNewIfAble()
{
    var all = repository.GetAll();

    foreach (TEnum val in Enum.GetValues(typeof(TEnum)).Cast<TEnum>())
    {
        if (all.Any(entity => ((dynamic)entity).ID == (int)((dynamic)val)))
            continue;

        string name = val.ToString();
        int value = (int)((dynamic)val);

        TEntity newEntity = new TEntity();
        ((dynamic)newEntity).Name = name;
        ((dynamic)newEntity).ID = value;

        repository.Add(newEntity);
    }

    repository.SaveChanges();
    return this;
}
```

Listing 11.10. Metoda CreateNewIfAble.

11.5.4.3. Upload plików

Wewnątrz aplikacji zaimplementowano system uploadu plików oparty o rozwiązania dostarczane przez ASP.NET MVC.

Zostało to wykorzystane przy dodawaniu nowego obrazka do danej fiszki. W tym celu stworzono formularz, wewnątrz którego dodano dwa elementy **input**. Jeden ukryty informujący o ID karty i drugi, wewnątrz którego była przechowywana informacja o uploadowanym pliku. W celu poprawnego przesyłu danych przez formularz należy użyć elementu **form**, który do przesyłu danych używa formatu MIME. Można to osiągnąć poprzez ustawienie atrybutowi **enctype** wartości **multipart/form-data**.

Wewnątrz akcji odpowiedzialnej za upload pliku należy użyć typu **HttpPostedFileBase** jako argumentu, za którego pośrednictwem przychodzi do nas informacja o zuploadowanym pliku. Typ ten zawiera wszystkie informacje o zuploadowanym pliku.

Następnie wykorzystywany jest serwis **FlashcardImageService**, którego zadaniem jest sprawdzenie, czy możliwy jest upload danego obrazka. W przypadku, gdy serwis stwierdzi taką możliwość, jest uruchamiana procedura uploadu pliku za pośrednictwem serwisu **UploadService**.

Po stronie serwera zostanie utworzony unikalny plik o tym samym rozszerzeniu co rozszerzenie uploadowanego pliku. Nazwa tego pliku będzie generowana na podstawie metody **Path.GetRandomFileName**, która jest dostarczana przez .NET Framework. Plik zostanie zuploadedowany w miejscu wskazanym przez klasę **UploadLocation** na podstawie rodzaju uploadu (w tym wypadku typem uploadu jest upload obrazków).

Po zakończeniu procedury nowy obrazek może być od razu wykorzystywany przez użytkownika.

```
public void UploadNewImage(Flashcard flashcard, HttpPostedFileBase file, string ↵
    userID)
{
    var dbFile = uploadService.Upload(file, userID, ↵
        FileTypeEnum.FlashcardImage);

    unit.FlashcardImageRepository.Add(new FlashcardImage()
    {
        FileID = dbFile.ID,
        FlashcardID = flashcard.ID
    });
}

unit.FlashcardImageRepository.SaveChanges();
}
```

Listing 11.11. Metoda `UploadNewImage` wewnątrz `FlashcardImageService`, która uploaduje plik z obrazkiem.

12. Podsumowanie

12.1. Wnioski

Celem pracy było wytworzenie projektu aplikacji internetowej wykorzystującej fiszki w celu nauki języka obcego. Projekt został zrealizowany z użyciem najnowszych rozwiązań technologicznych w dziedzinie aplikacji internetowych takich jak ASP.NET MVC, C#, SASS, Scala.js.

Proces nauki został wykonany na podstawie samodzielnie stworzonych wzorów opartych o krzywą zapominania i system powtórzeń, których parametry były dostosowywane na podstawie przeprowadzonych simulacji. Dzięki tym rozwiązaniom proces nauki z wykorzystaniem tego projektu będzie efektywniejszy.

Aplikacja została stworzona zgodnie ze wzorcem MVC. Dzięki temu poszczególne warstwy aplikacji (przechowywania, przetwarzania, wyświetlanie danych) mogą być oddzielnie modyfikowane bez wpływu na inne. Rozwiązanie to wpływa korzystnie na czytelność kodu programu.

12.1.1. Scala.js

Integracja z językiem Scala.js w projekcie ASP.NET MVC została zakończona z powodzeniem. Jest to w pełni możliwe do wykonania pomimo braku jakiegoś narzędzia integrującego Visual Studio z tym językiem.

Rozwiązanie to działało bardzo dobrze. Pisanie w tym języku było bardzo przyjemne od strony programistycznej, dzięki możliwości skupienia się na kodzie i nie zwracaniu uwagi na dziwne zachowania języka, które występowały w kodzie JavaScript. Niestety to rozwiązanie ma poważną wadę. Z racji tego jak działa proces kompilacji Scala.js niemożliwym jest aby utworzyć kilka plików z kodem JavaScript. W dzisiejszym świecie działaniem pożądanym jest zmniejszenie rozmiaru i ilości zapytań HTTP jak to tylko możliwe. Niemożność rozdzielenia kodu na kilka modułów i wczytanie tylko jednej wymaganej części powoduje zwiększenie ilości pobieranych danych. Co więcej, należy mieć na uwadze fakt, iż to rozwiązanie jest niepopularne wśród programistów. W związku z tym, wszelkie napotkane problemy będą prawdopodobnie rozwiązywane w pojedynkę bez pomocy zewnętrznych źródeł.

12.2. Możliwe ulepszenia

Interfejs aplikacji jest napisany w języku angielskim. Pomimo wykorzystania jedynie języka obcego w trakcie nauki, interfejs mógłby być tłumaczony na język ojczysty danego użytkownika. Ułatwiłoby to posługiwaniu się aplikacją dla osób nieznających języka angielskiego.

Aktualnie dla opisu danej fiszki są używane obrazki oraz tłumaczenia, które wykorzystywane są w trakcie procesu nauczania. Należałyby rozważyć dodanie nowych form opisu fiszki, które pomogłyby w nauce. Przykładem nowej funkcjonalności mogłyby być dodanie lektora, który wypowiadałby dane tłumaczenia podczas nauki. Taki dodatek mógłby pozytywnie wpłynąć na efektywność zapamiętywania.

Bibliografia

- [1] Dr. Bruce Abbott's. *Human Memory*. URL: <http://users.ipfw.edu/abbott/120/Ebbinghaus.html> (term. wiz. 2017-12-31).
- [2] Dr. John Wittman. *The Forgetting Curve*. URL: https://www.csustan.edu/sites/default/files/groups/Writing%20Program/forgetting_curve.pdf (term. wiz. 2017-12-31).
- [3] Neda Karimi Seyed Jalal Abdolmanafi Rokni. „VISUAL INSTRUCTION: AN ADVANTAGE OR A DISADVANTAGE? WHAT ABOUT ITS EFFECT ON EFL LEARNERS' VOCABULARY LEARNING?” W: *Asian journal of social sciences & humanities* 2.4 (list. 2013).
- [4] Walling JR. Nelson DL Reed VS. „Pictorial superiority effect.” W: *Journal of Experimental Psychology: Human Learning and Memory* 2.5 (wrz. 1976).
- [5] Jeroen G.W. Raaijmakers. „Spacing and repetition effects in human memory: application of the SAM mode”. W: *Cognitive Science* (2002), s. 431–452.
- [6] *Introduction to the C# Language and the .NET Framework*. URL: <https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework> (term. wiz. 2017-01-03).
- [7] *Introduction to ASP.NET Web Programming Using the Razor Syntax (C#)*. URL: <https://docs.microsoft.com/en-us/aspnet/web-pages/overview/getting-started/introducing-razor-syntax-c> (term. wiz. 2017-01-04).
- [8] *Implementing the Repository and Unit of Work Patterns in an ASP.NET MVC Application (9 of 10)*. URL: <https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions/getting-started-with-ef-5-using-mvc-4/implementing-the-repository-and-unit-of-work-patterns-in-an-asp-net-mvc-application> (term. wiz. 2017-12-31).
- [9] *Ninject*. URL: <https://github.com/ninject/Ninject> (term. wiz. 2017-12-31).
- [10] *SQL Server 2016*. URL: <https://www.microsoft.com/pl-pl/sql-server/sql-server-2016> (term. wiz. 2017-01-03).
- [11] dr Artur Bartoszewski. *Wprowadzenie do baz danych*. URL: <http://www.bartoszewski.pradom.pl/bazy/Wprowadzenie%20do%20baz%20danych%20wyklad%2001.pdf> (term. wiz. 2017-01-03).

- [12] *What is NuGet?* URL: <https://www.nuget.org/> (term. wiz. 2017-01-03).
 - [13] Microsoft. *Unit Testing: Testing the Inside.* URL: <https://msdn.microsoft.com/en-us/library/jj159340.aspx> (term. wiz. 2017-01-01).
 - [14] *CSS.* URL: <https://developer.mozilla.org/pl/docs/Web/CSS> (term. wiz. 2017-01-08).
 - [15] *Syntactically Awesome Style Sheets.* URL: <http://sass-lang.com/> (term. wiz. 2017-01-01).
 - [16] *Scala (programming language).* URL: [https://en.wikipedia.org/wiki/Scala_\(programming_language\)](https://en.wikipedia.org/wiki/Scala_(programming_language)) (term. wiz. 2017-01-02).
 - [17] *Tour of Scala.* URL: <http://docs.scala-lang.org/tour/tour-of-scala.html> (term. wiz. 2017-01-02).
 - [18] *JavaScript.* URL: <https://developer.mozilla.org/pl/docs/Web/JavaScript> (term. wiz. 2017-01-05).
 - [19] Charles Severance. *Java Script: Designing a Language in 10 Days.* URL: <https://www.computer.org/csdl/mags/co/2012/02/mco2012020007.html> (term. wiz. 2017-01-09).
 - [20] *Dartium: Chromium with the Dart VM.* URL: <https://webdev.dartlang.org/tools/dartium> (term. wiz. 2017-01-03).
 - [21] *Apache IvyTM.* URL: <http://ant.apache.org/ivy/> (term. wiz. 2017-01-03).
 - [22] *Hands-on Scala.js : The Compilation Pipeline.* URL: <http://www.lihaoyi.com/hands-on-scala-js/#TheCompilationPipeline> (term. wiz. 2017-01-02).
 - [23] *Compilation and optimization pipeline.* URL: <https://www.scala-js.org/doc/internals/compile-opt-pipeline.html> (term. wiz. 2017-01-02).
 - [24] *Understanding the Restrictions Imposed by the Closure Compiler.* URL: <https://developers.google.com/closure/compiler/docs/limitations> (term. wiz. 2017-01-02).
 - [25] *Closure Compiler.* URL: <https://developers.google.com/closure/compiler/> (term. wiz. 2017-01-02).
 - [26] *Odległość Levenshteina.* URL: https://pl.wikipedia.org/wiki/Odleg%C5%82o%C5%9B%C4%87_Levenshteina (term. wiz. 2017-01-05).
-

Spis rysunków

1.1	Przeglądarka internetowa wbudowana w telewizor.	1
2.1	Eksponencjalna krzywa zapominania.	3
4.1	Model MVC. Źródło: msdn.microsoft.com	8
4.2	Wzorzec repozytorium. Źródło: msdn.microsoft.com	10
4.3	Repozytorium ReviewCardRepository wraz z implementowanymi interfejsami i odziedzicznymi klasami.	11
4.4	Jednostka pracy wykorzystywana w projekcie.	12
4.5	Proces bindowania z wyszczególnionymi częściami składowymi.	13
8.1	Przykład dziwnego zachowania języka Javascript. Źródło: devrant.com	21
8.2	Przykład procesu debugowania kodu. Nasz program zostaje zatrzymany na linijce przed dodaniem napisu Hello World.	25
10.1	Widok pytania w projekcie. Jest on wspólny dla każdego z etapów	29
10.2	Widok niepoprawnej odpowiedzi.	30
10.3	Zobrazowanie informacji zawartych w tabeli 10.1	31
10.4	Wykres przedstawia ilość zdobytej siły po poprawnej odpowiedzi. Powstał na podstawie różnicy między nową siłą a jej poprzednią wartością.	32
10.5	Wykres przedstawia ilość straconej siły dla $L = 1$. Wzór 10.3.	33
10.6	Widok poprawnej odpowiedzi.	34
10.7	Okno programu do znajdywania wartości parametrów. W nawiasie są zawarte wartości parametrów, zaś po myślniku ilość nauczonych fiszek. Program informuje nas o nowej kombinacji wartości parametrów, jeśli symulacja ma większy wynik niż jej poprzedniczki.	37
10.8	Widok edycji fiszki.	38
10.9	Diagram przypadków użycia.	39
11.1	Diagram przedstawiający strukturę programu.	40
11.2	Diagram klas wszystkich repozytoriów w projekcie.	42

11.3 Diagram klas wszystkich kontrolerów w projekcie	44
11.4 Cykl życia kontrolera. Źródło: docs.microsoft.com.	45
11.5 Diagram klas wszystkich serwisów w projekcie	47
11.6 Rysunek przedstawiający rozwiązanie projektu.	49
11.7 Diagram ERD bazy danych.	50
11.8 Diagram ERD ASP.NET Identity	52