Aplikacja do nauki języków

Prezentacja UML

Marcin Kondrat, Jowita Ochrymiuk, Marcin Roszkowski

Wykorzystane Technologie

- **ASP.NET Core MVC** — Struktura aplikacji zapewniająca rozdzielenie logiki, widoków i danych
- **Entity Framework Core** — ORM do zarządzania bazą danych
- **Razor Views** — Silnik widoków generujący dynamiczne strony HTML

Wykorzystane Wzorce

01 MVC

Wzorzec **architekturalny**

W projekcie wzorzec **MVC** (Model-View-Controller) został użyty do rozdzielenia logiki aplikacji, danych i interfejsu.

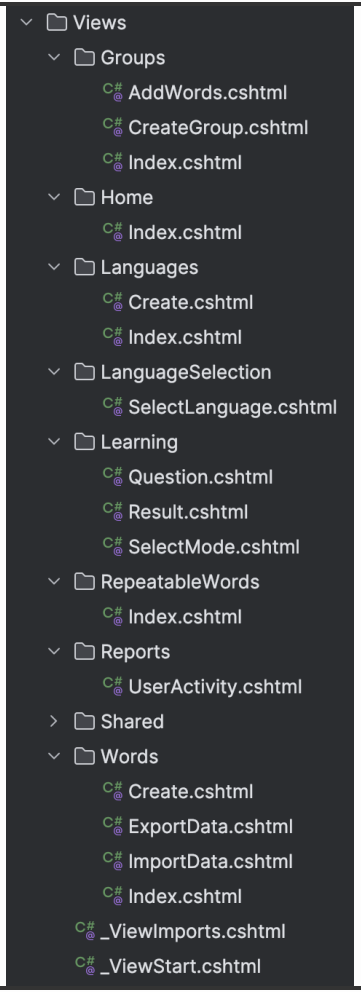
Modele, takie jak *Word* i *Group*, zarządzają danymi przez *Entity Framework Core*.

Kontrolery, np. *LearningController*, obsługują żądania i kierują je np. do fasady *LearningFacade*.

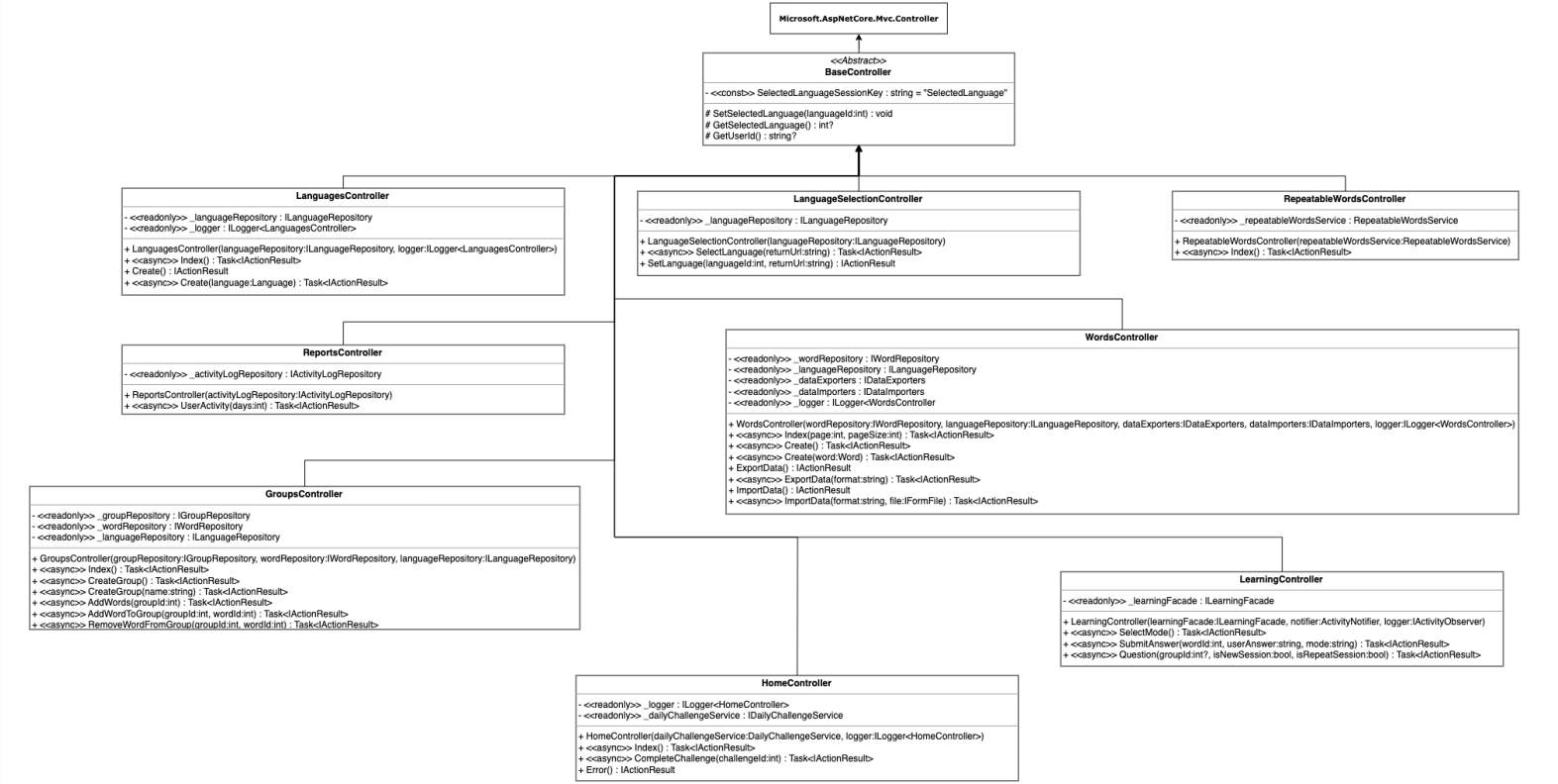
Widoki Razor generują dynamiczne strony HTML, zapewniając interakcję z użytkownikiem. Dzięki MVC aplikacja jest modularna i łatwa w utrzymaniu.

Models

Views


Views
Groups
AddWords.cshtml
CreateGroup.cshtml
Index.cshtml
Home
Index.cshtml
Languages
Create.cshtml
Index.cshtml
LanguageSelection
SelectLanguage.cshtml
Learning
Question.cshtml
Result.cshtml
SelectMode.cshtml
RepeatableWords
Index.cshtml
Reports
UserActivity.cshtml
Shared
Words
Create.cshtml
ExportData.cshtml
ImportData.cshtml
Index.cshtml
_ViewImports.cshtml
_ViewStart.cshtml

Controllers



02

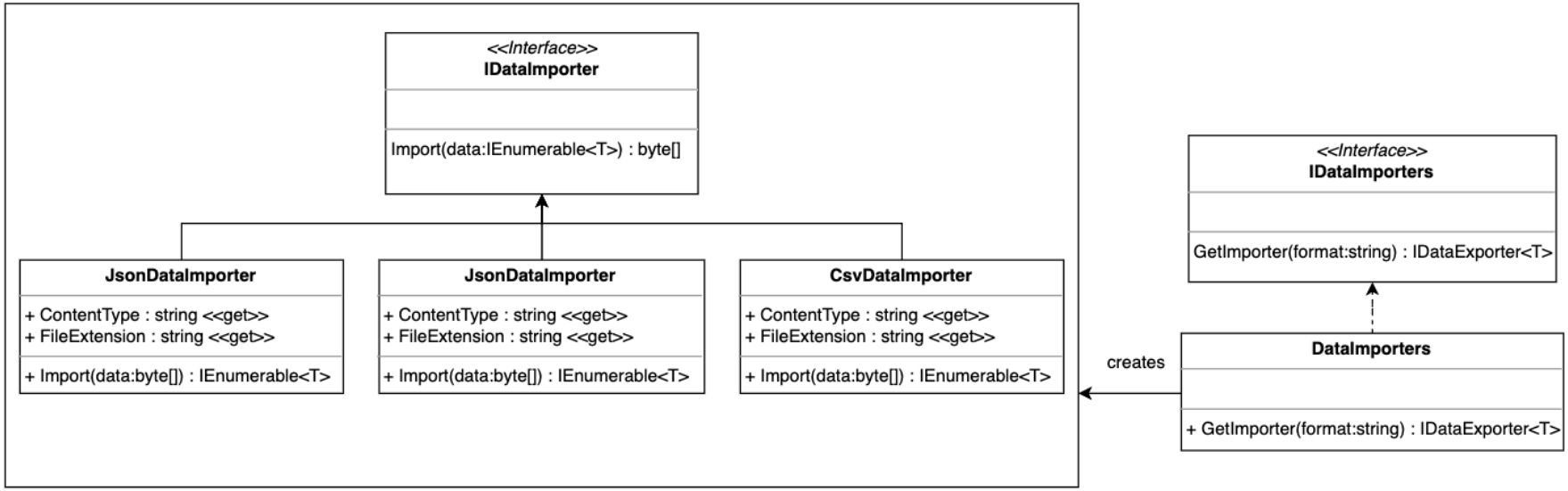
Factory

Wzorzec **kreacyjny**

W projekcie wzorzec **Factory** został zastosowany np. w klasie *DataImporters*, która dynamicznie zwraca odpowiedni importer danych w zależności od formatu, np. *XmlDataImporter* dla XML, *JsonDataImporter* dla JSON czy *CsvDataImporter* dla CSV.

Każdy importer implementuje interfejs *IDataImporter<T>,* co zapewnia spójność obsługi różnych formatów.

Dzięki temu dodanie nowego formatu wymaga jedynie implementacji odpowiedniego importera i rejestracji go w fabryce, co upraszcza zarządzanie kodem i zwiększa elastyczność aplikacji.


<<Interface>>
IDataImporter
Import(data:IEnumerable<T>) : byte[]
JsonDataImporter
+ ContentType : string <<get>>
+ FileExtension : string <<get>>
+ Import(data:byte[]) : IEnumerable<T>
JsonDataImporter
+ ContentType : string <<get>>
+ FileExtension : string <<get>>
+ Import(data:byte[]) : IEnumerable<T>
CsvDataImporter
+ ContentType : string <<get>>
+ FileExtension : string <<get>>
+ Import(data:byte[]) : IEnumerable<T>
creates
<<Interface>>
IDataImporters
GetImporter(format:string) : IDataExporter<T>
DataImporters
+ GetImporter(format:string) : IDataExporter<T>

03 Facade

Wzorzec **strukturalny**

W projekcie wzorzec **Facade** został zastosowany w klasie *LearningFacade*, która pełni rolę uproszczonego interfejsu dla złożonych operacji związanych z procesem nauki.

LearningFacade integruje różne elementy systemu, takie jak strategie nauki, zarządzanie grupami słów czy logowanie aktywności użytkownika, umożliwiając kontrolerom dostęp do tych funkcji w sposób spójny i przejrzysty.

Dzięki temu fasada ukrywa złożoność implementacyjną i ułatwia utrzymanie oraz rozszerzanie kodu aplikacji.

04 Proxy

Wzorzec **strukturalny**

W projekcie wzorzec **Proxy** został zastosowany w klasie *CachedWordRepositoryProxy*, która dodaje mechanizm pamięci podręcznej do repozytorium *IWordRepository*.

Klasa najpierw sprawdza, czy dane są w cache, a w razie braku deleguje zapytanie do repozytorium, zapisując wynik w pamięci podręcznej. Dzięki temu aplikacja działa szybciej, zmniejszając liczbę zapytań do bazy danych.

05 Repository

Wzorzec **strukturalny**

W projekcie wzorzec **Repository** został zastosowany do oddzielenia logiki biznesowej od warstwy dostępu do danych.

Klasa bazowa *Repository<T>* zapewnia uniwersalne metody zarządzania encjami, takie jak *GetByIdAsync* czy *AddAsync*.

Specyficzne implementacje, jak *GroupRepository*, rozszerzają ją o operacje domenowe, np. filtrowanie grup według języka.

Dzięki **Repository** warstwa dostępu do danych jest modularna, spójna i łatwa do rozszerzenia.

06 Observer

Wzorzec **czynnościowy**

W projekcie wzorzec **Observer** został zastosowany w klasach *ActivityNotifier* i *ActivityLogger*, aby umożliwić monitorowanie i reagowanie na zmiany w aktywności użytkownika.

Klasa *ActivityNotifier* pełni rolę podmiotu, który zarządza listą obserwatorów i powiadamia ich o zmianach, takich jak poprawne odpowiedzi użytkownika.

ActivityLogger działa jako obserwator, zapisując te zdarzenia w bazie danych.

Dzięki wzorcowi Observer aplikacja może łatwo rozszerzać funkcjonalność powiadamiania bez modyfikacji głównej logiki.

07 Strategy

Wzorzec **czynnościowy**

W projekcie wzorzec **Strategy** obsługuje różne tryby nauki dzięki interfejsowi *ILearningStrategy*, zaimplementowanemu przez strategie, takie jak *FlashcardsStrategy*, *MultipleChoiceStrategy*, czy *FillInTheBlankStrategy.*

Klasa LearningStrategies dynamicznie wybiera odpowiednią strategię na podstawie trybu (*LearningMode*).

Dzięki temu logika specyficzna dla trybów jest odseparowana, co ułatwia rozwój i utrzymanie aplikacji.