

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

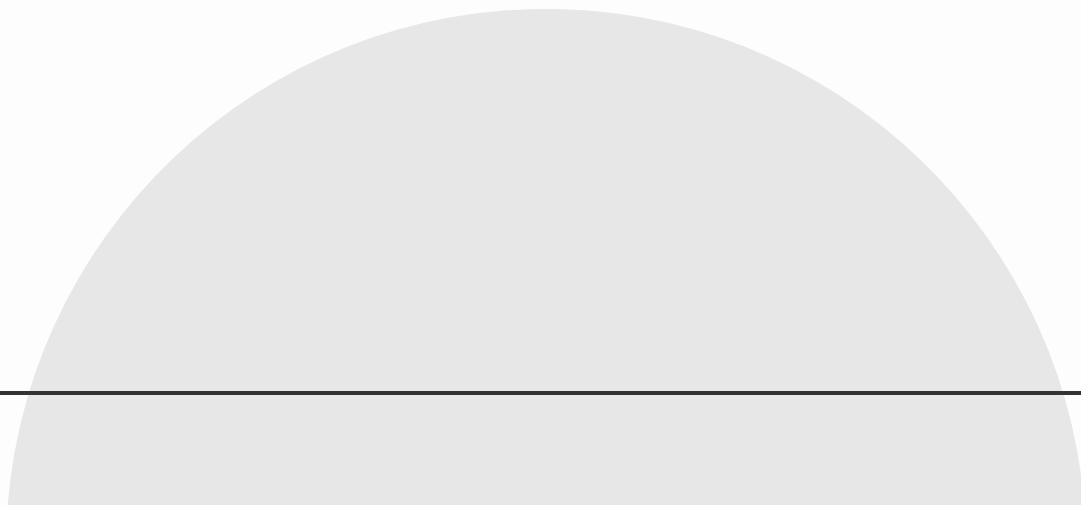
Entity Framework Core

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

ActivityLog

+ Id : int <<get>> <<set>>
+ UserId : string <<get>> <<set>>
+ WordId : int <<get>> <<set>>
+ IsCorrect : bool <<get>> <<set>>
+ Corrected : bool <<get>> <<set>>
+ Timestamp : DateTime <<get>> <<set>>

LearningResult

+ IsCorrect : bool <<get>> <<set>>
+ CorrectAnswer : string <<get>> <<set>>
+ UserAnswer : string <<get>> <<set>>

<<Enum>>

Learning Mode

Flashcards
MultipleChoice
FillInTheBlank

DailyChallenge

+ Id : int <<get>> <<set>>
+ UserId : string <<get>> <<set>>
+ LanguageId : int <<get>> <<set>>
+ Words : List<Word> <<get>> <<set>>
+ CreatedAt : DateTime <<get>> <<set>>
+ IsCompleted : bool <<get>> <<set>> = false

Word

+ Id : int <<get>> <<set>>
+ Original : string <<get>> <<set>>
+ Translation : string <<get>> <<set>>
+ LanguageId : int <<get>> <<set>>
+ Language : Language? <<get>> <<set>>

GroupWord

+ GroupId : int <<get>> <<set>>
+ Group : Group? <<get>> <<set>>
+ WordId : int <<get>> <<set>>
+ Word : Word? <<get>> <<set>>

Group

+ Id : int <<get>> <<set>>
+ Name : string <<get>> <<set>>
+ UserId : string <<get>> <<set>>
+ LanguageId : int <<get>> <<set>>
+ Language : Language <<get>> <<set>>
+ GroupWords : ICollection<GroupWord> <<get>> <<set>>

Language

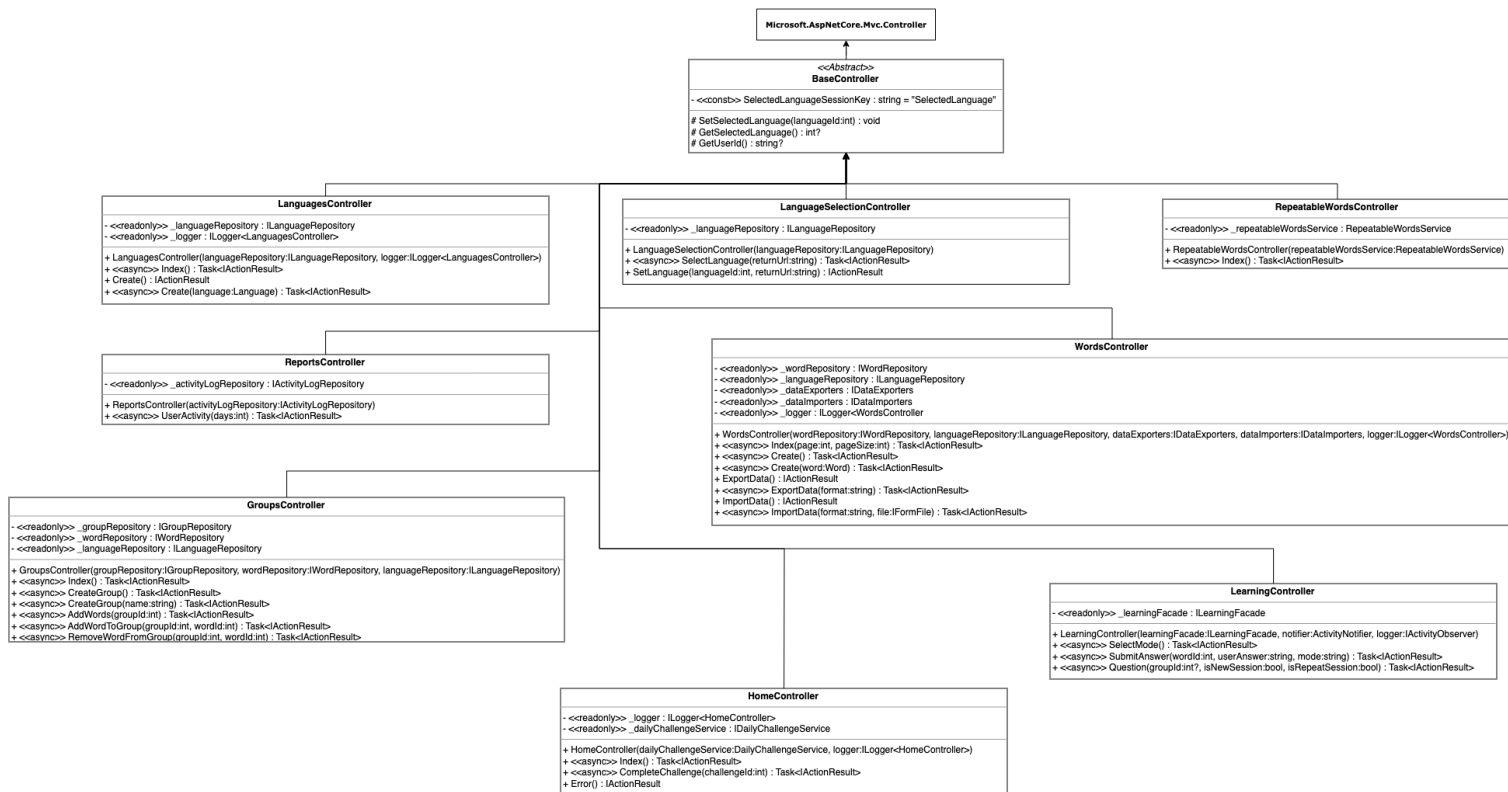
+ Id : int <<get>> <<set>>
+ Code : string <<get>> <<set>>
+ Name : string <<get>> <<set>>
+ Groups : ICollection<Group>? <<get>> <<set>>
+ Words : ICollection<Word>? <<get>> <<set>>

Views

```

  ▾ Views
    ▾ Groups
      C# AddWords.cshtml
      C# CreateGroup.cshtml
      C# Index.cshtml
    ▾ Home
      C# Index.cshtml
    ▾ Languages
      C# Create.cshtml
      C# Index.cshtml
    ▾ LanguageSelection
      C# SelectLanguage.cshtml
    ▾ Learning
      C# Question.cshtml
      C# Result.cshtml
      C# SelectMode.cshtml
    ▾ RepeatabeWords
      C# Index.cshtml
    ▾ Reports
      C# UserActivity.cshtml
    > ▾ Shared
    ▾ Words
      C# Create.cshtml
      C# ExportData.cshtml
      C# ImportData.cshtml
      C# Index.cshtml
      C# _ViewImports.cshtml
      C# _ViewStart.cshtml
```

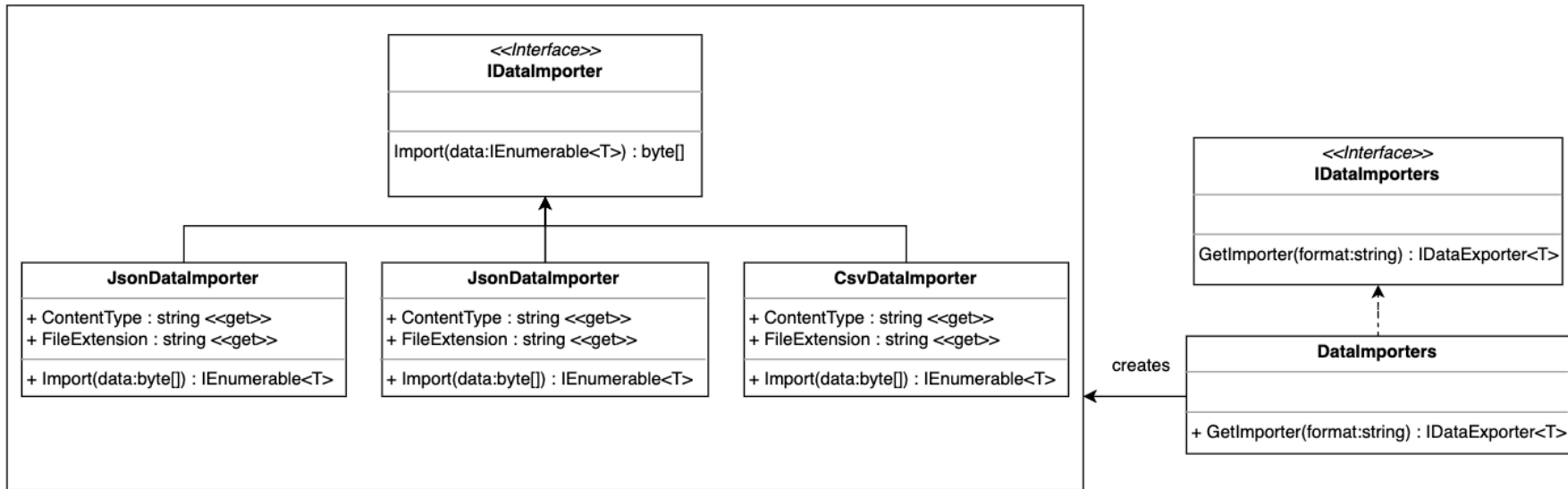
Controllers



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.

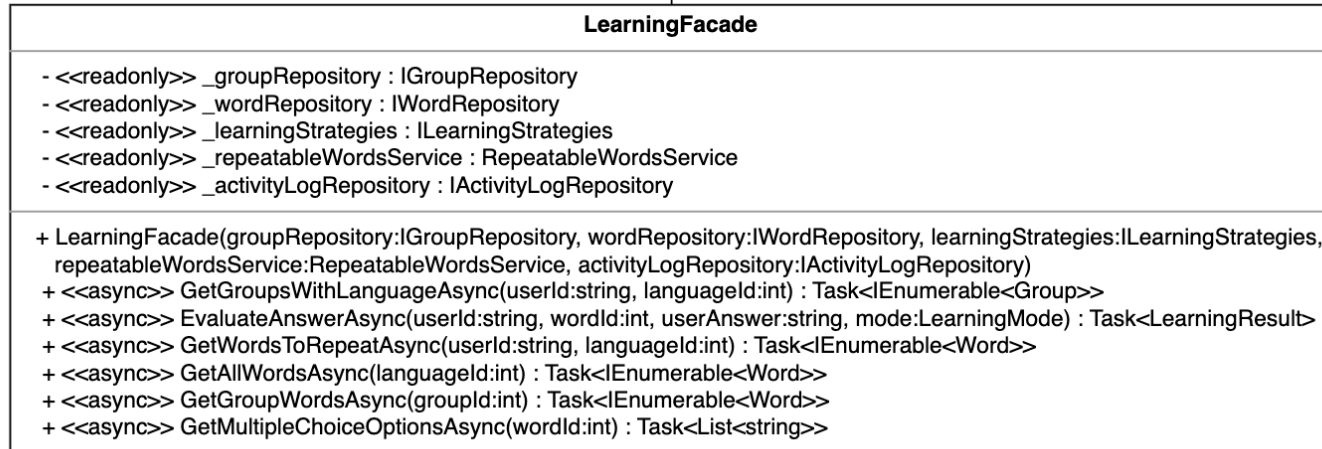
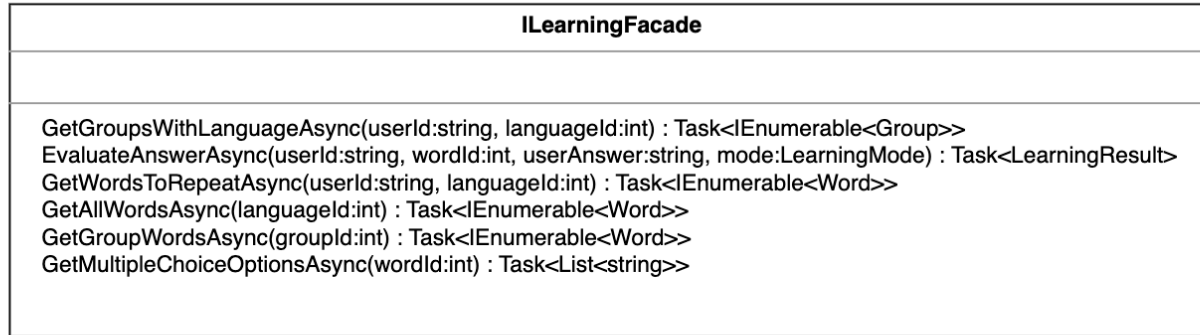
Dzięki temu dodanie nowego formatu wymaga jedynie implementacji odpowiedniego importera i rejestracji go w fabryce, co upraszcza zarządzanie kodem i zwiększa elastyczność aplikacji.



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

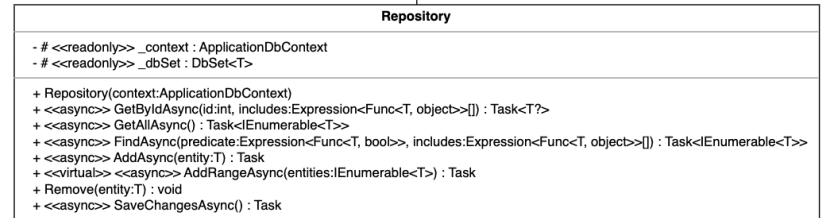
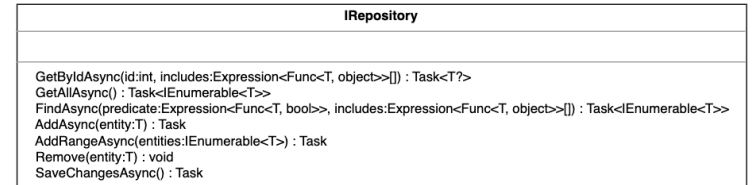
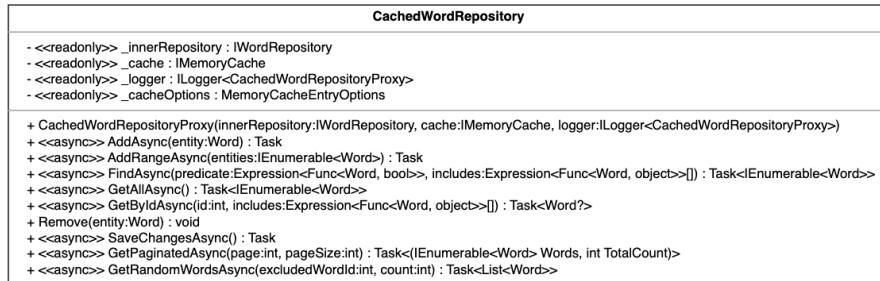
LearningFacade integruje różne elementy systemu, takie jak strategie nauki, zarządzanie grupami słów czy logowanie aktywności użytkownika, umożliwiając kontrolerom dostęp do tych funkcji w sposób spójny i przejrzysty.

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



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.

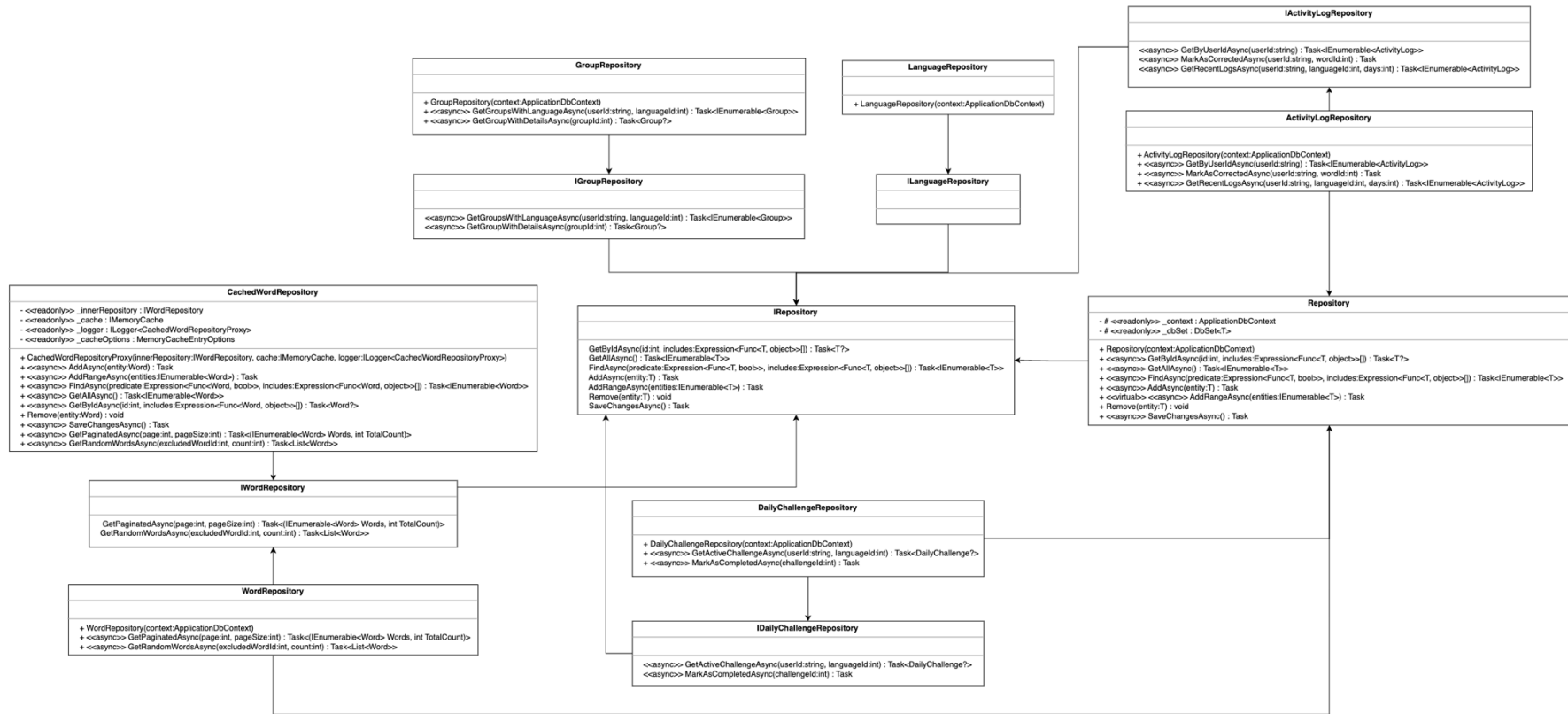


W projekcie wzorzec **Repository** został zastosowany do oddzielenia logiki biznesowej od warstwy dostępu do danych.

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.

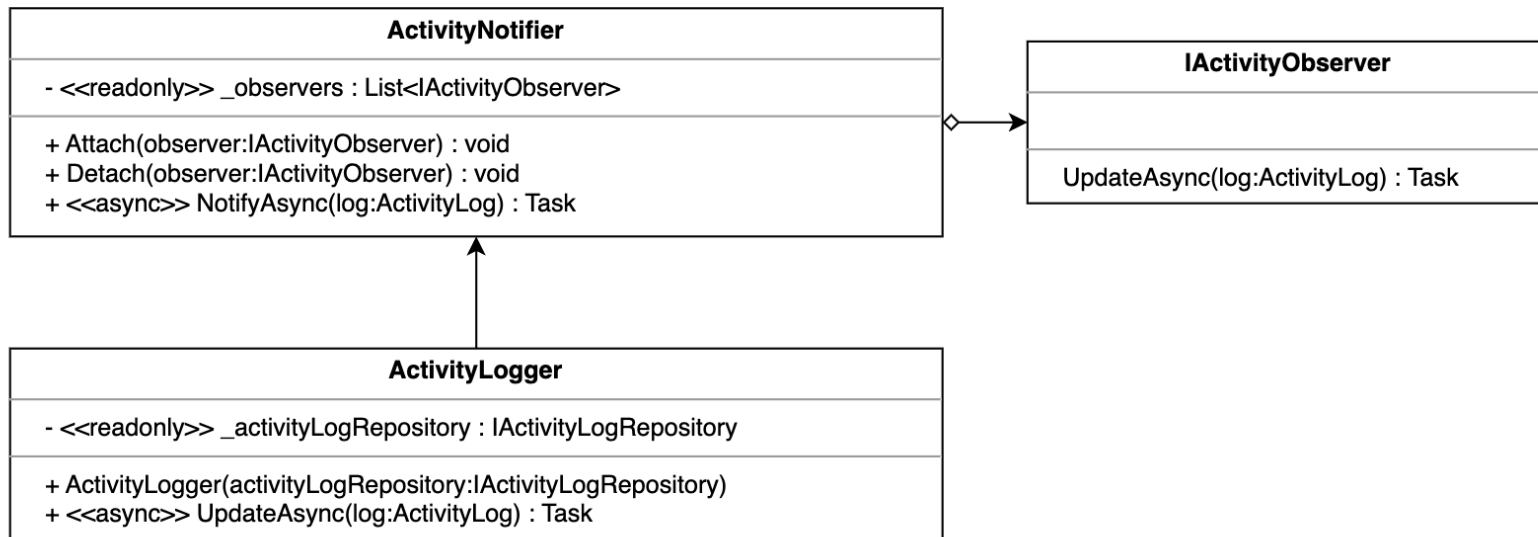


W projekcie wzorzec **Observer** został zastosowany w klasach *ActivityNotifier* i *ActivityLogger*, aby umożliwić monitorowanie i reagowanie na zmiany w aktywności użytkownika.

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.

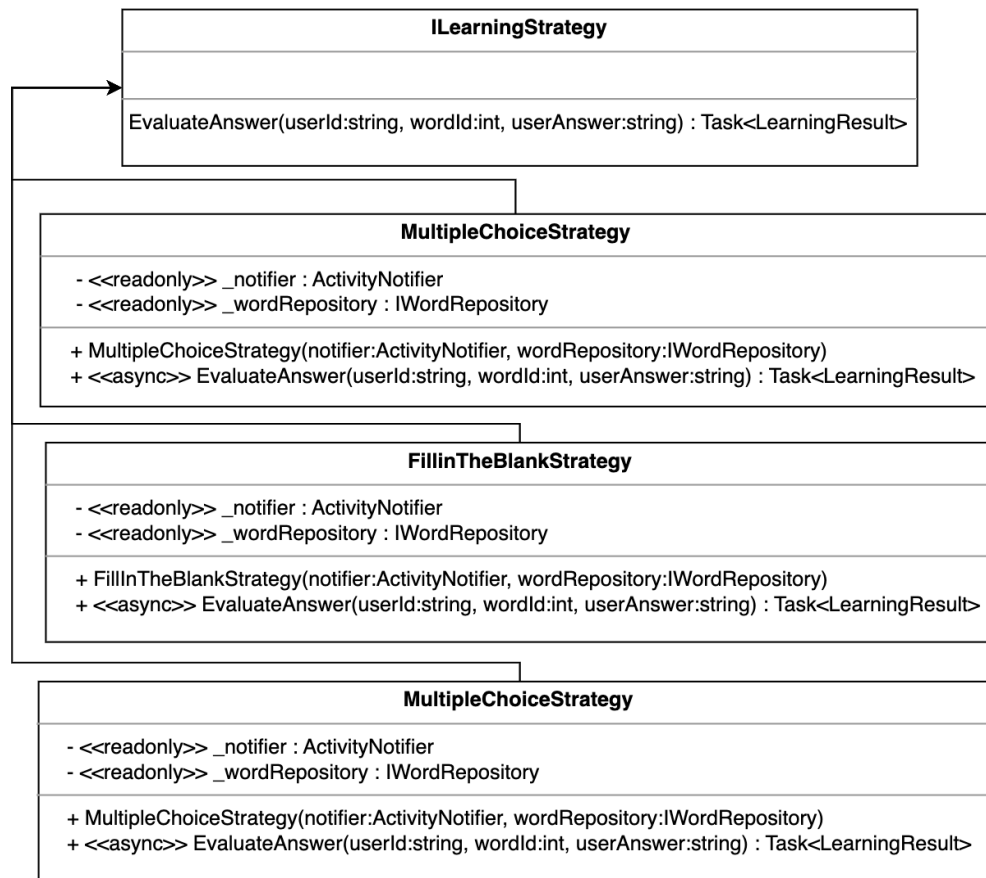
Dzięki wzorcowi Observer aplikacja może łatwo rozszerzać funkcjonalność powiadamiania bez modyfikacji głównej logiki.



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.



Schemat UML Projektu

