Politechnika Białostocka

Wydział Informatyki

Marcin Kondrat
nr albumu: 115530

Jowita Ochrymiuk
nr albumu: 115538

Marcin Roszkowski
nr albumu: 115607

Projekt zaliczeniowy

Aplikacja do nauki języków obcych

Przedmiot: Zaawansowane techniki programistyczne
Prowadzący: dr inż. Marek Tabędzki

Białystok 2025

Spis treści

Język i framework

Projekt został zrealizowany w języku **C#** na platformie **.NET Core 9.0,** z wykorzystaniem frameworka **ASP.NET Core MVC**. Dzięki wzorcowi MVC aplikacja zachowuje przejrzystą strukturę, oddzielając logikę biznesową, dane i interfejs użytkownika. Wykorzystanie **Entity Framework Core** umożliwia łatwe zarządzanie bazą danych poprzez mapowanie modeli na tabele oraz obsługę migracji. Mechanizm **Dependency Injection** wbudowany w **ASP.NET Core** pozwala na efektywne zarządzanie zależnościami, minimalizując konieczność ręcznego tworzenia obiektów i wspierając modularność.

Aplikacja korzysta z **Razor Pages**, co ułatwia generowanie dynamicznych widoków i integrację interfejsu z logiką. Wprowadzenie wzorców projektowych, takich jak Factory, Facade i Strategy, usprawnia rozwój i rozszerzanie funkcjonalności, np. zarządzania słownictwem, importu danych czy trybów nauki. W celu optymalizacji wydajności zastosowano pamięć podręczną, co zmniejsza obciążenie bazy danych i przyspiesza działanie aplikacji. Projekt wspiera również dynamiczną obsługę wielu języków, umożliwiając łatwą rozbudowę o nowe kultury i języki.

Instrukcja instalacji

Pobieranie .NET 9.0

Pobierz i zainstaluj .NET SDK 9.0:
https://dotnet.microsoft.com/en-us/download/dotnet/9.0

Pobieranie kodu

- Sklonuj repozytorium projektu z systemu kontroli wersji Git:
  git clone https://github.com/breftejk/ZTP-Project
- Przejdź do folderu projektu:
  cd ZTP-Project

Pobieranie dotnet-ef oraz pakietów NuGet

Użyj następującej serii poleceń w terminalu:

dotnet tool install --global dotnet-ef
dotnet add package Microsoft.AspNetCore.Identity.EntityFrameworkCore
dotnet add package Microsoft.AspNetCore.Identity.UI
dotnet add package Microsoft.EntityFrameworkCore
dotnet add package Microsoft.EntityFrameworkCore.Sqlite
dotnet add package Microsoft.EntityFrameworkCore.Tools
dotnet add package Microsoft.VisualStudio.Web.CodeGeneration.Design

dotnet add package Scrutor
dotnet restore

Przygotowanie bazy danych

Zbuduj i zainijalizuj bazę danych (musi być to wykonane tylko raz):

dotnet ef migrations add DatabaseInit
dotnet ef database update

Uruchomienie aplikacji

Wykonaj następującą komendę:

dotnet run

Dostęp do aplikacji

- Otwórz przeglądarkę i przejdź pod adres: http://localhost:5083
- Możesz zacząć korzystać z aplikacji!

Instrukcja użytkownika

Logowanie i rejestracja

Przy pierwszej wizycie na stronie aplikacji, poproszony zostaniesz o zalogowanie się. Wybierz „Register as a new user" i utwórz nowe konto. Od teraz będziesz mógł logować się nim do aplikacji.

Jeśli chcesz korzystać z funkcjonalności ograniczonych dla administratorów, takich jak eksport i import danych, użyć możesz testowego konta do zalogowania się:

- E-mail: test@example.com
- Hasło: StrongPassword123!

Wybór języka

Po zalogowaniu się do aplikacji, poproszony zostaniesz o wybór języka. Dostępne są języki: niemiecki, francuski, hiszpański oraz włoski.

Codzienne wyzwania

Przywitany zostaniesz stroną główną, na której możesz zobaczyć swoje codzienne wyzwanie. To grupa losowych słówek z wybranego przez Ciebie języka. Dzienne wyzwanie oznacza się jako ukończone, klikając przycisk „Mark as Completed".

Nauka słówek

W zakładce "Learning" możesz wybrać grupę słówek, którą chcesz opanować, oraz tryb nauki najlepiej odpowiadający Twoim potrzebom. Do wyboru są fiszki (Flashcards), quiz wielokrotnego wyboru (Multiple Choice) oraz uzupełnianie luk w zdaniach (Fill In The Blank).

Tworzenie i zarządzanie grupami słów

W zakładce „Your Groups" możesz tworzyć i zarządzać grupami słów. Po wybraniu języka wystarczy kliknąć „Create New Group", wpisać nazwę w polu „Group Name", a następnie potwierdzić, klikając „Create Group". Po utworzeniu grupy wyświetli się komunikat „Group created successfully".

Grupy można edytować, wybierając opcję „Manage Words", gdzie możesz dodawać nowe słowa lub usuwać te, które nie są już potrzebne.

Dzięki temu, w zakładce „Learning", możesz uczyć się wyłącznie słówek z wybranych przez siebie grup.

Powtórki

Po kliknięciu przycisku „Repeat Words" znajdziesz słowa, które sprawiły Ci trudność podczas nauki. System automatycznie dobiera słowa do powtórki na podstawie historii Twoich odpowiedzi.

Raporty i śledzenie postępów

W zakładce „User Activity" możesz sprawdzić swoje statystyki z określonego przedziału czasowego, takie jak liczba poprawnych i błędnych odpowiedzi.

Import i eksport danych

Dane możesz importować i eksportować, wybierając odpowiedni format za pomocą opcji „Select Format", gdzie dostępne są JSON, XML i CSV.

Wzorce projektowe

MVC

W projekcie wzorzec MVC (Model-View-Controller) został zastosowany jako architektoniczne podejście do rozdzielenia logiki aplikacji, interfejsu użytkownika i warstwy dostępu do danych. Modele, takie jak *Word* czy *Group*, reprezentują dane i są zarządzane przez *Entity Framework Core*. Kontrolery, na przykład *LearningController* i *WordsController*, obsługują żądania użytkownika, delegując logikę do odpowiednich serwisów lub repozytoriów, takich jak *LearningFacade*. Widoki, zbudowane w technologii

Razor Pages, generują dynamiczne strony HTML, które umożliwiają interakcję użytkownika z aplikacją.

Kod związany z wzorcem MVC znajduje się w różnych pakietach aplikacji:

- **Modele**: Zdefiniowane w pakiecie ***ZTP_Project.Data.Models***, zawierają encje takie jak *Word* czy *Group*.
- **Kontrolery**: Znajdują się w pakiecie ***ZTP_Project.Controllers***, gdzie klasy takie jak *GroupsController* i *ReportsController* obsługują funkcjonalności związane z grupami i raportami.
- **Widoki**: Umieszczone w folderze ***Views***, zawierają dynamicznie generowane szablony HTML dla różnych stron aplikacji.

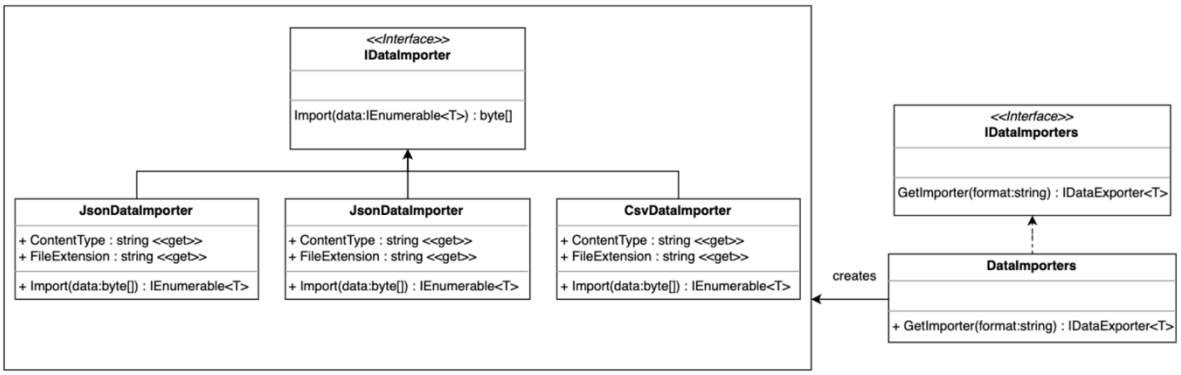
Zastosowanie wzorca MVC zapewnia elastyczność i modularność aplikacji. Dzięki wyraźnemu podziałowi odpowiedzialności między modelami, widokami i kontrolerami łatwo można dodawać nowe funkcjonalności, takie jak nowe strony czy rozszerzone operacje na danych, bez modyfikacji istniejących komponentów. Wzorzec ten wspiera również testowalność, umożliwiając testowanie logiki aplikacji w izolacji od interfejsu użytkownika.

Factory

W projekcie wzorzec Factory został zastosowany w celu dynamicznego zarządzania m.in. importem danych w różnych formatach, takich jak JSON, XML czy CSV. Klasa DataImporters pełni rolę fabryki, która na podstawie przekazanego formatu zwraca odpowiednią implementację importera danych (*JsonDataImporter, XmlDataImporter, CsvDataImporter*). Każdy importer implementuje interfejs *IDataImporter<T>*, zapewniając jednolity sposób obsługi różnych formatów plików.

Kod związany z fabryką znajduje się w pakiecie ***ZTP_Project.Data.Import***. Definicja klasy *DataImporters* znajduje się w pliku *DataImporters.cs*, natomiast przykładowa implementacja importera danych, jak *JsonDataImporter<T>*, znajduje się w pliku *JsonDataImporter.cs*. Wywołanie fabryki odbywa się poprzez metodę *GetImporter*, która w zależności od przekazanego formatu tworzy odpowiedni obiekt.

Zastosowanie wzorca Factory pozwala na łatwe rozszerzenie funkcjonalności aplikacji o obsługę nowych formatów danych. W przypadku dodania nowego formatu wystarczy zaimplementować nową klasę importera, dziedziczącą po *IDataImporter<T>*, oraz dodać obsługę tego formatu w metodzie *GetImporter*. Dzięki temu kod pozostaje modularny i elastyczny.

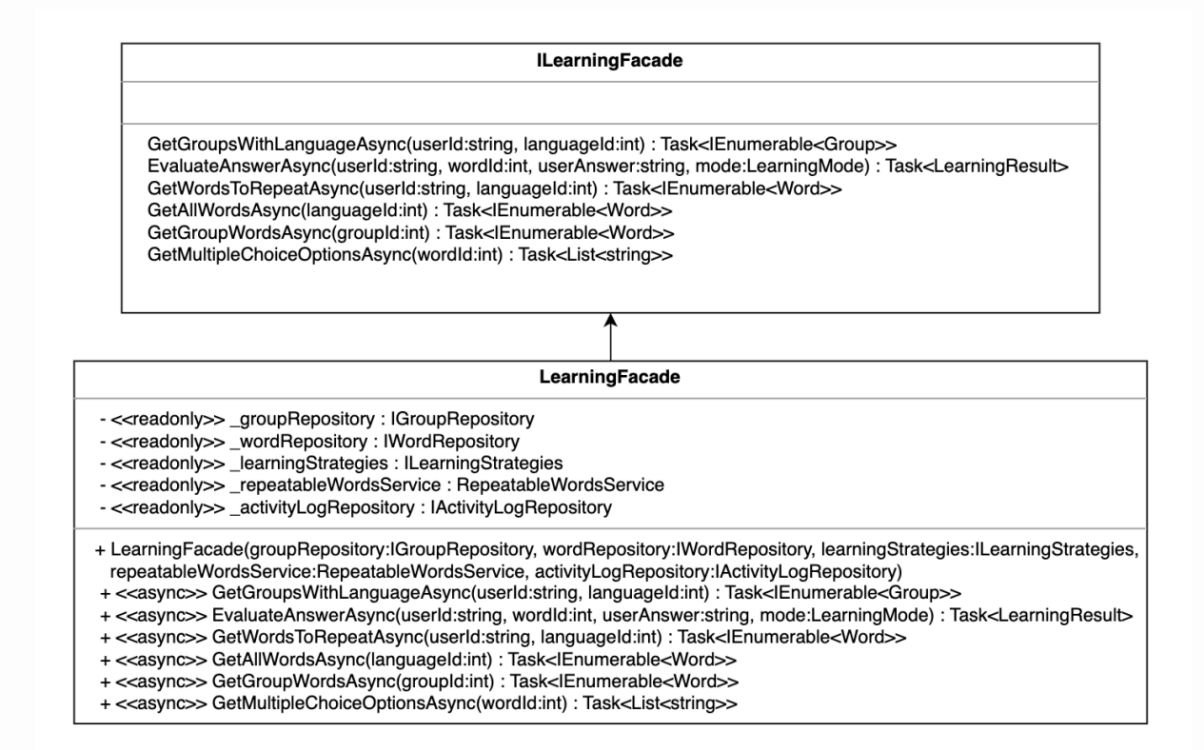


Facade

W projekcie wzorzec Facade został zastosowany w celu uproszczenia dostępu do złożonych operacji związanych z procesem nauki, takich jak zarządzanie grupami, słowami, strategiami nauki, obsługą powtórek oraz logowaniem aktywności użytkownika. Klasa *LearningFacade* pełni rolę fasady, integrując różne komponenty systemu (*IGroupRepository, IWordRepository, ILearningStrategies, RepeatableWordsService, IActivityLogRepository*) i zapewniając jednolity interfejs dla kontrolerów.

Kod fasady znajduje się w module zarządzania logiką aplikacji, a jej definicja jest w pliku *LearningFacade.cs*. Przykładowe metody, takie jak *EvaluateAnswerAsync* i *GetGroupsWithLanguageAsync*, pozwalają na korzystanie z funkcji systemu bez potrzeby bezpośredniego odwoływania się do poszczególnych komponentów. Wywołania fasady występują w miejscach, gdzie kontrolery korzystają z operacji związanych z nauką.

Zastosowanie wzorca Facade ułatwia rozszerzanie aplikacji, np. przez dodanie nowych strategii nauki czy funkcji logowania, bez konieczności modyfikacji kodu kontrolerów. Dzięki temu system jest bardziej przejrzysty, łatwiejszy w utrzymaniu i elastyczny w rozwoju.

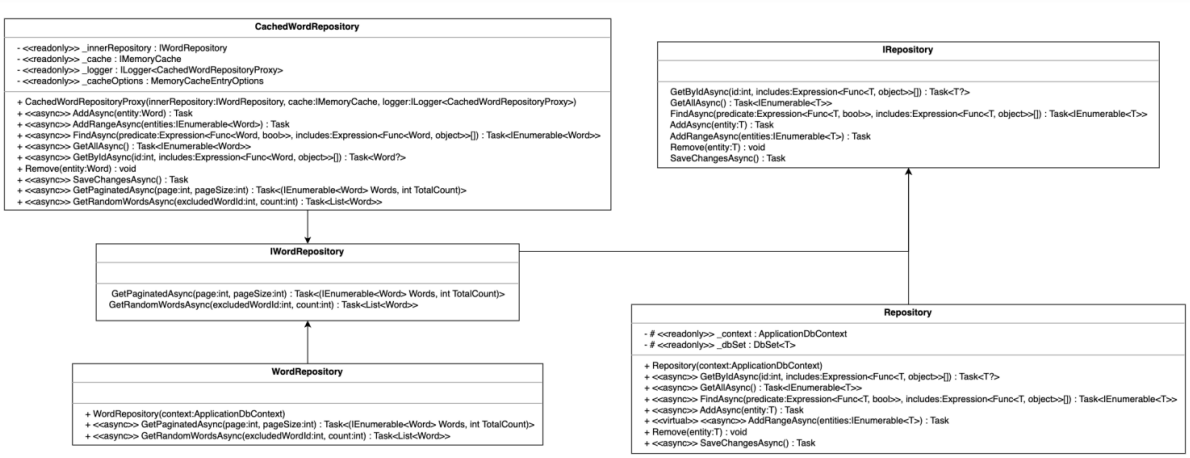


Proxy

W projekcie wzorzec Proxy został zastosowany w klasie *CachedWordRepository*, która dodaje mechanizm pamięci podręcznej do repozytorium *IWordRepository*. Klasa ta sprawdza, czy dane są dostępne w cache, a w przypadku ich braku deleguje zapytanie do właściwego repozytorium (*WordRepository*) i zapisuje wynik w pamięci podręcznej. Dzięki temu zmniejsza się liczba zapytań do bazy danych, co przyspiesza działanie aplikacji.

Kod związany z wzorcem Proxy znajduje się w pakiecie ***ZTP_Project.Data.Repositories***. Definicja klasy *CachedWordRepository* znajduje się w pliku *CachedWordRepository.cs*, a wywołania jej metod, takich jak *GetByIdAsync* czy *GetAllAsync*, zapewniają korzystanie z pamięci podręcznej podczas pracy z danymi.

Zastosowanie wzorca Proxy umożliwia łatwe rozszerzenie funkcjonalności aplikacji, na przykład przez dodanie nowych mechanizmów zarządzania cache (np. rozproszona pamięć podręczna) lub rozszerzenie logiki cache dla innych repozytoriów. Dzięki temu kod pozostaje modularny, a zmiany w pamięci podręcznej nie wpływają na logikę samego repozytorium.

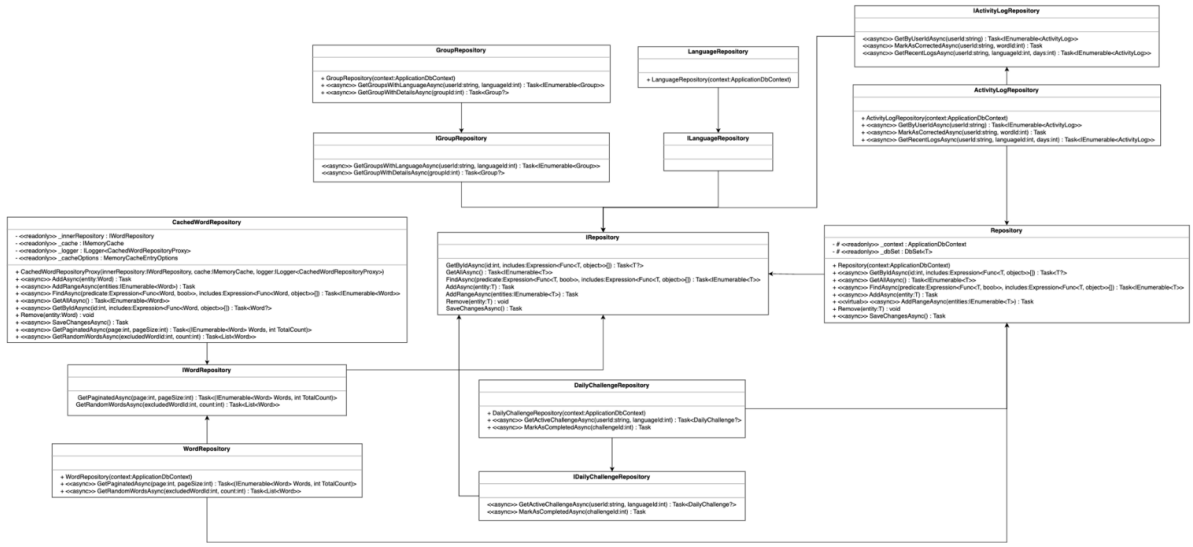


Repository

W projekcie wzorzec Repository został zastosowany w celu oddzielenia logiki biznesowej od warstwy dostępu do danych, co zwiększa modularność i ułatwia utrzymanie kodu. Klasa bazowa *Repository<T>* definiuje podstawowe operacje na encjach, takie jak *GetByIdAsync, AddAsync* czy *Remove*, zapewniając jednolite zarządzanie danymi. Specyficzne implementacje rozszerzają tę funkcjonalność o operacje domenowe, np. *WordRepository* (plik *WordRepository.cs*, pakiet ***ZTP_Project.Data.Repositories***) dodaje paginację za pomocą metody *GetPaginatedAsync* oraz możliwość losowego wybierania słów dzięki *GetRandomWordsAsync*.

Podobnie, *GroupRepository* (plik *GroupRepository.cs*) obsługuje filtrowanie grup według języka użytkownika za pomocą *GetGroupsWithLanguageAsync*, a *DailyChallengeRepository* (plik *DailyChallengeRepository.cs*) zarządza wyzwaniami dziennymi, oferując metodę *GetActiveChallengeAsync* do pobierania aktualnego wyzwania dla użytkownika.

Kod odpowiedzialny za wzorzec znajduje się w pakiecie ***ZTP_Project.Data.Repositories***. Definicja klasy bazowej *Repository<T>* umożliwia rozszerzanie aplikacji poprzez dodawanie nowych repozytoriów i metod specyficznych dla domeny, bez modyfikacji istniejącej logiki. Dzięki temu aplikacja pozostaje elastyczna i gotowa na przyszłe wymagania biznesowe.

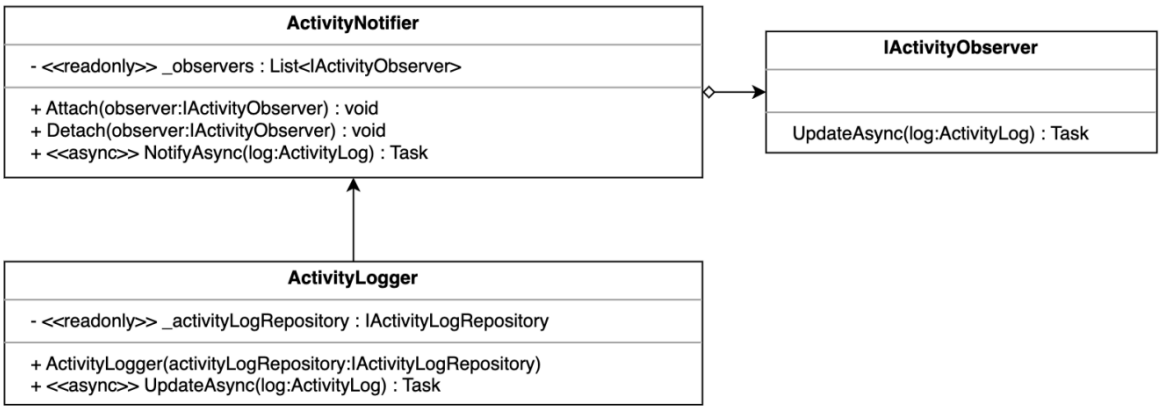


Observer

W projekcie wzorzec Observer został zastosowany w celu umożliwienia monitorowania i reagowania na zmiany w aktywności użytkownika. Klasa *ActivityNotifier* pełni rolę podmiotu (subject), zarządzając listą obserwatorów i powiadamiając ich o nowych zdarzeniach, takich jak poprawne odpowiedzi użytkownika. Klasa *ActivityLogger* działa jako obserwator (observer), zapisując zdarzenia w bazie danych za pomocą *IActivityLogRepository*.

Kod związany z implementacją wzorca znajduje się w pakiecie ***ZTP_Project.Learning.Activities***. Definicja *ActivityNotifier* i *ActivityLogger* znajduje się odpowiednio w plikach *ActivityNotifier.cs* oraz *ActivityLogger.cs*. Przykładowe użycie obejmuje wywołanie metody *Attach* do dodania obserwatora, a następnie *NotifyAsync,* która informuje wszystkie zarejestrowane obiekty o nowym zdarzeniu.

Dzięki zastosowaniu wzorca Observer aplikacja jest elastyczna i łatwa do rozbudowy. Dodanie nowych obserwatorów, które reagują na zmiany w aktywności użytkownika, wymaga jedynie implementacji interfejsu *IActivityObserver* i rejestracji w *ActivityNotifier,* bez modyfikacji istniejącego kodu. To pozwala na łatwe rozszerzenie funkcjonalności powiadamiania o dodatkowe działania, takie jak wysyłanie powiadomień czy analiza danych.

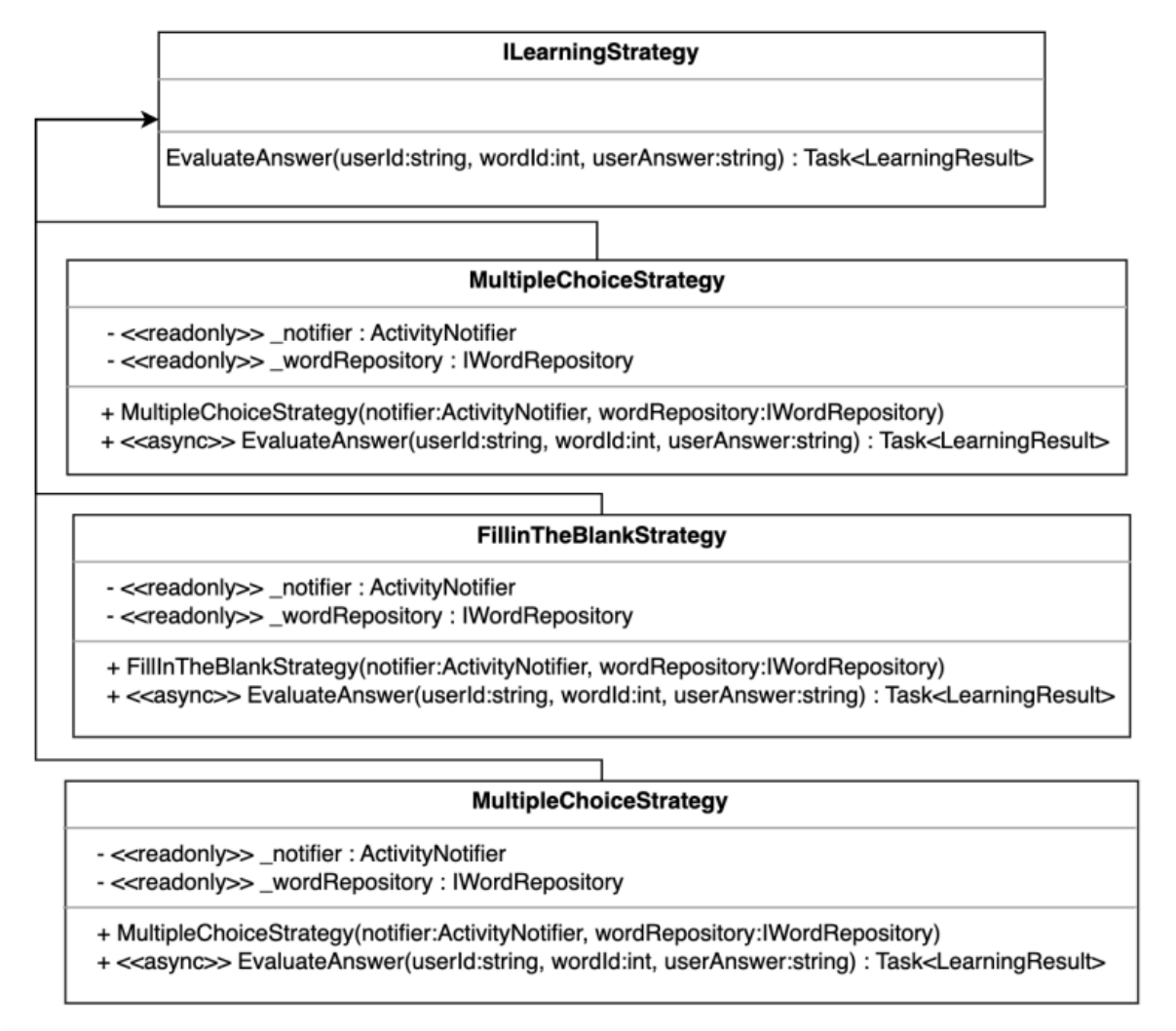


Strategy

W projekcie wzorzec Strategy został zastosowany do obsługi różnych trybów nauki, takich jak „Multiple Choice”, „Flashcards” i „Fill In The Blank”. Interfejs *ILearningStrategy* definiuje wspólny kontrakt dla wszystkich strategii, które są zaimplementowane w klasach *MultipleChoiceStrategy, FlashcardsStrategy i FillInTheBlankStrategy*. Klasa *LearningStrategies* odpowiada za dynamiczny wybór odpowiedniej strategii na podstawie trybu *LearningMode*.

Kod związany z implementacją wzorca znajduje się w pakiecie ***ZTP_Project.Learning.Strategies***. Definicje poszczególnych strategii znajdują się w plikach *MultipleChoiceStrategy.cs, FlashcardsStrategy.cs* i *FillInTheBlankStrategy.cs*. Metody strategii są wywoływane w kontekście oceny odpowiedzi użytkownika poprzez metodę *EvaluateAnswer*.

Zastosowanie wzorca Strategy pozwala na łatwe dodawanie nowych trybów nauki poprzez implementację nowej klasy strategii, która dziedziczy po *ILearningStrategy*. Dzięki temu logika specyficzna dla poszczególnych trybów jest odseparowana, co upraszcza rozwój i utrzymanie aplikacji. W przypadku wprowadzenia nowego trybu wystarczy zaimplementować nową strategię i uwzględnić ją w mechanizmie wyboru w *LearningStrategies,* bez ingerencji w istniejące klasy.



Podział pracy

Podział pracy w zespole był jasno określony, a każda osoba wniosła istotny wkład w realizację projektu. Regularne spotkania zdalne oraz intensywna komunikacja w czasie rzeczywistym pozwoliły na efektywne podejmowanie decyzji projektowych, omawianie problemów technicznych oraz wybór najlepszych wzorców projektowych. Oto szczegóły podziału pracy:

Marcin Kondrat

Odpowiedzialny za kluczowe elementy związane z architekturą projektu, takie jak wdrożenie wzorca MVC, w tym inicjalizację systemu oraz konfigurację bazy danych z wykorzystaniem Entity Framework Core. Zajął się implementacją interfejsów modeli i ich mapowaniem na tabele bazy danych. Opracował system autoryzacji użytkowników, zarządzanie routingiem aplikacji oraz wizualizację danych w dynamicznych widokach Razor. Zaimplementował również wzorce Proxy i Repository, optymalizując działanie aplikacji poprzez wprowadzenie mechanizmu pamięci podręcznej oraz oddzielenie

warstwy dostępu do danych od logiki biznesowej. Dodatkowo aktywnie wspierał pozostałych członków zespołu w implementacji wzorców, doradzając w zakresie ich projektowania oraz integracji z pozostałymi elementami systemu.

Jowita Ochrymiuk

Skoncentrowała się na implementacji wzorców Observer i Factory, które były kluczowe dla dynamicznego zarządzania procesami w aplikacji. W ramach wzorca Observer stworzyła mechanizm powiadamiania o aktywnościach użytkownika (np. *ActivityNotifier* i *ActivityLogger*), co umożliwiło monitorowanie postępów użytkownika i ich rejestrowanie w bazie danych. Zaimplementowała wzorzec Factory w klasie *DataImporters*, który odpowiada za obsługę importu danych w różnych formatach (JSON, XML, CSV), zapewniając spójność i łatwość rozszerzania aplikacji.

Marcin Roszkowski

Zajął się implementacją wzorców *Facade* i *Strategy*, które ułatwiają zarządzanie złożonymi procesami nauki w aplikacji. Wzorzec *Facade*, zaimplementowany w klasie *LearningFacade*, zapewnia uproszczony interfejs dla kontrolerów, integrując różne funkcjonalności, takie jak strategie nauki, zarządzanie grupami słów czy powtórki. Wzorzec *Strategy* pozwolił na elastyczną obsługę różnych trybów nauki (np. fiszki, wybór wielokrotny), co umożliwia łatwe dodawanie nowych strategii. Odpowiadał również za integrację testowania i ewaluacji odpowiedzi użytkownika w różnych trybach nauki oraz research lingwistyczny obejmujący m.in dobranie słów do bazy danych.

Współpraca zespołowa

Wszyscy członkowie zespołu uczestniczyli w tworzeniu schematu UML oraz dokumentacji projektu, dzieląc się swoimi uwagami i doświadczeniem z poszczególnych części implementacji. Spotkania zdalne pozwoliły na bieżąco monitorować postępy prac oraz rozwiązywać problemy techniczne.

Ten podział pracy zapewnił efektywną realizację projektu oraz umożliwił osiągnięcie rozwiązania, które spełnia wszystkie założone wymagania.