Aplikacja do nauki języków

Prezentacja UML

Marcin Kondrat, Jowita Ochrymiuk, Marcin Roszkowski

Założenia Projektu

Aplikacja umożliwia użytkownikowi **przyswajanie słownictwa** oraz **sprawdzanie swojej wiedzy w różnych trybach nauki**, takich jak fiszki (wyświetlanie słowa lub tłumaczenia z oczekiwaniem na odpowiedź i prezentacja poprawnej odpowiedzi), wybór poprawnego tłumaczenia spośród kilku opcji oraz wpisywanie słów w zdania. **Zachęca do regularnej nauki** poprzez **codzienne wyzwania** i **umożliwia powtórki słów**, które sprawiały trudność.

Użytkownik ma **możliwość logowania się** do aplikacji, co pozwala na **personalizację zestawów słówek** i **śledzenie postępów**. Aplikacja umożliwia także **wybór języka**, co pozwala dostosować doświadczenie do preferencji użytkownika.

Dodatkowo aplikacja oferuje **funkcje grupowania słów w kategorie** (np. „Podróże”, „Zakupy”), a także **eksportu i importu danych**. Dzięki temu użytkownik może **generować raporty** i efektywnie zarządzać swoim procesem nauki.

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**

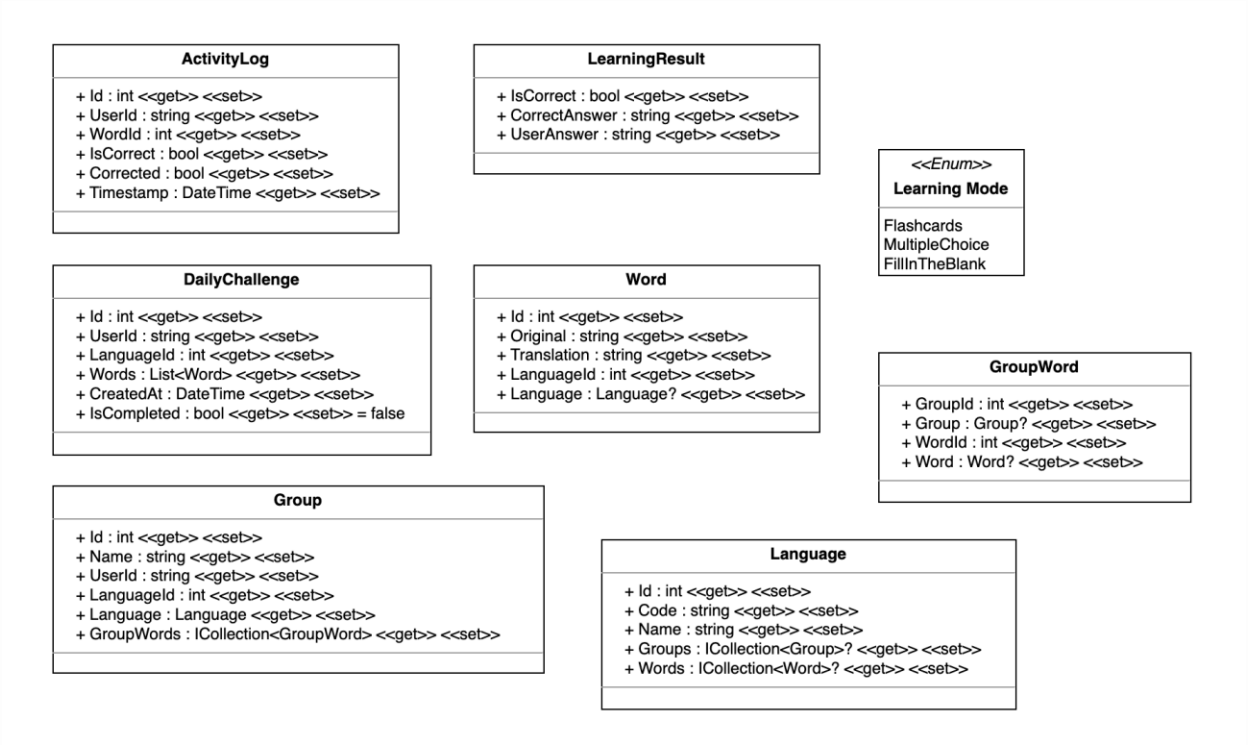
W projekcie wzorzec **MVC** (Model-View-Controller) został użyty do rozdzielenia logiki aplikacji, danych i interfejsu.

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

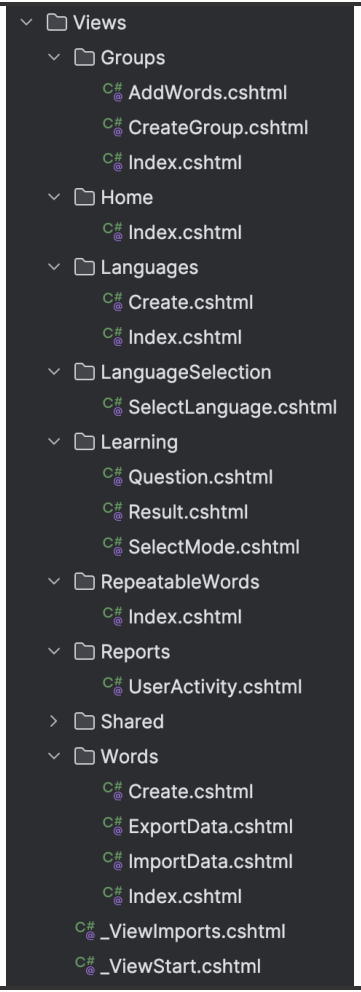
Kontrolery, np. *LearningController*, obsługują żądania i kierują je np. do fasady *LearningFacade*.

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

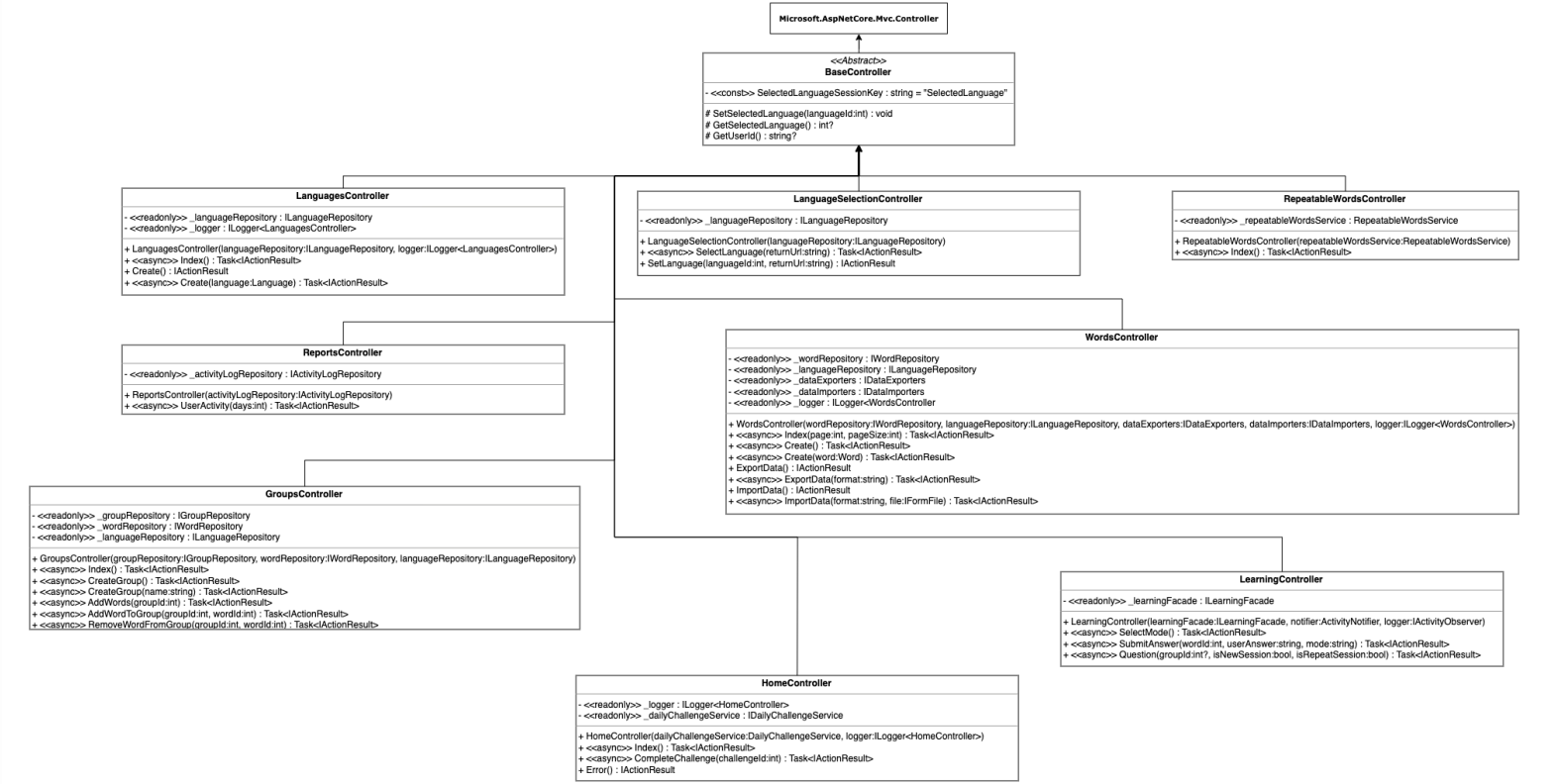
Models


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
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

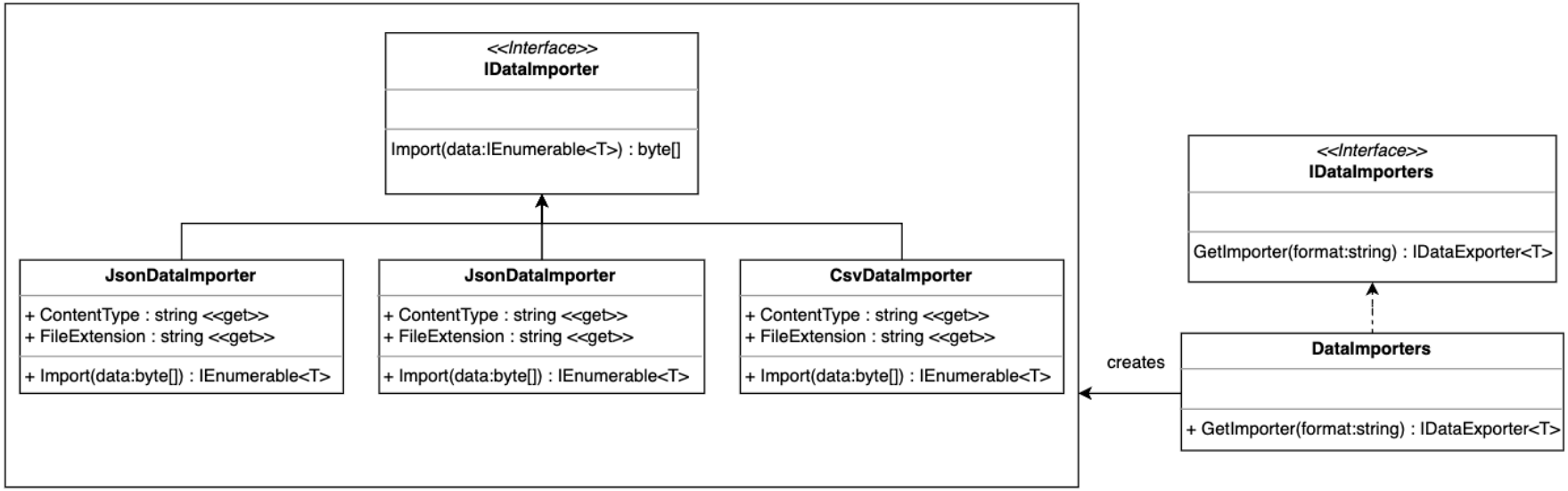
Factory

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>

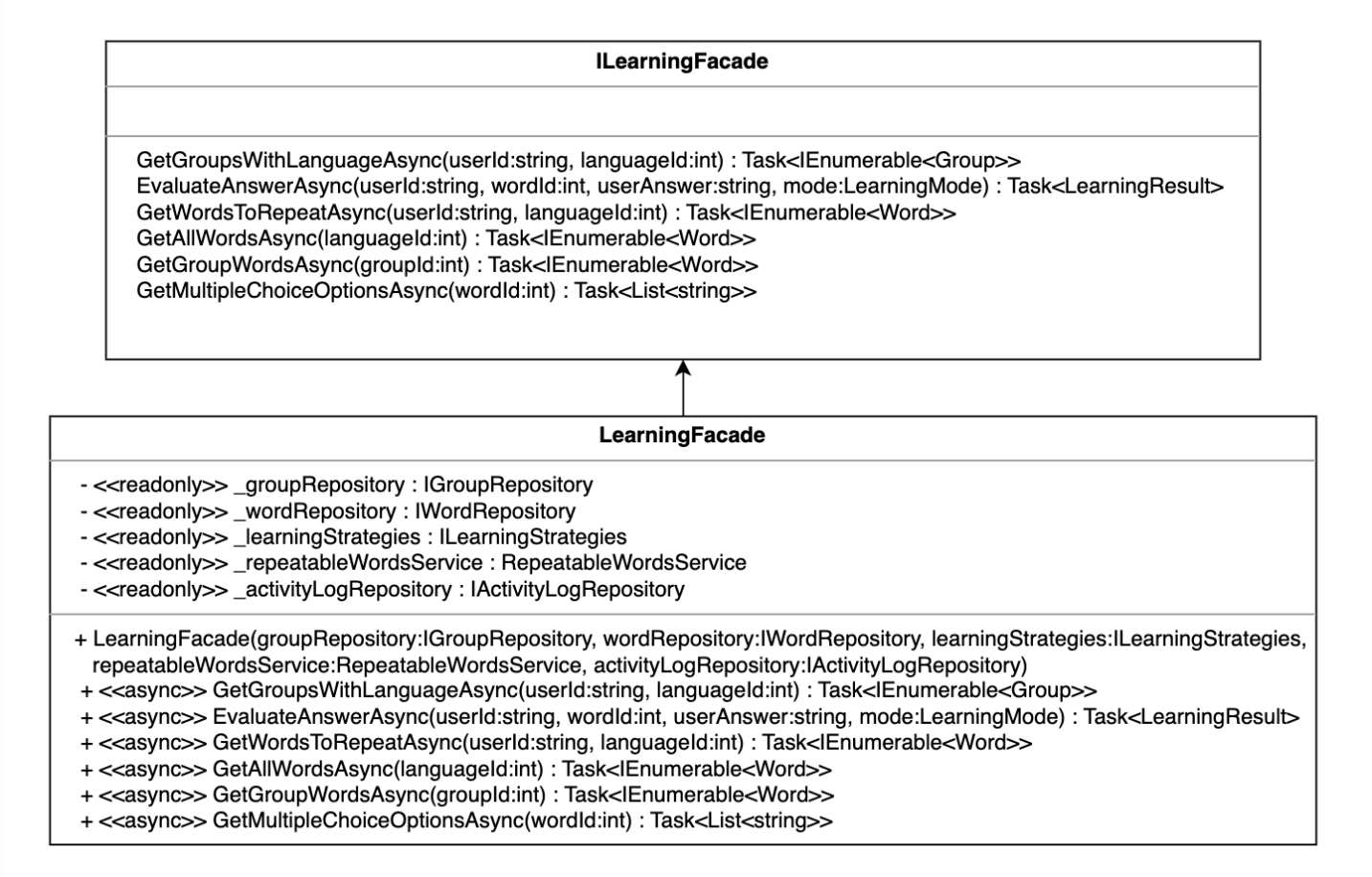
03 Facade

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.

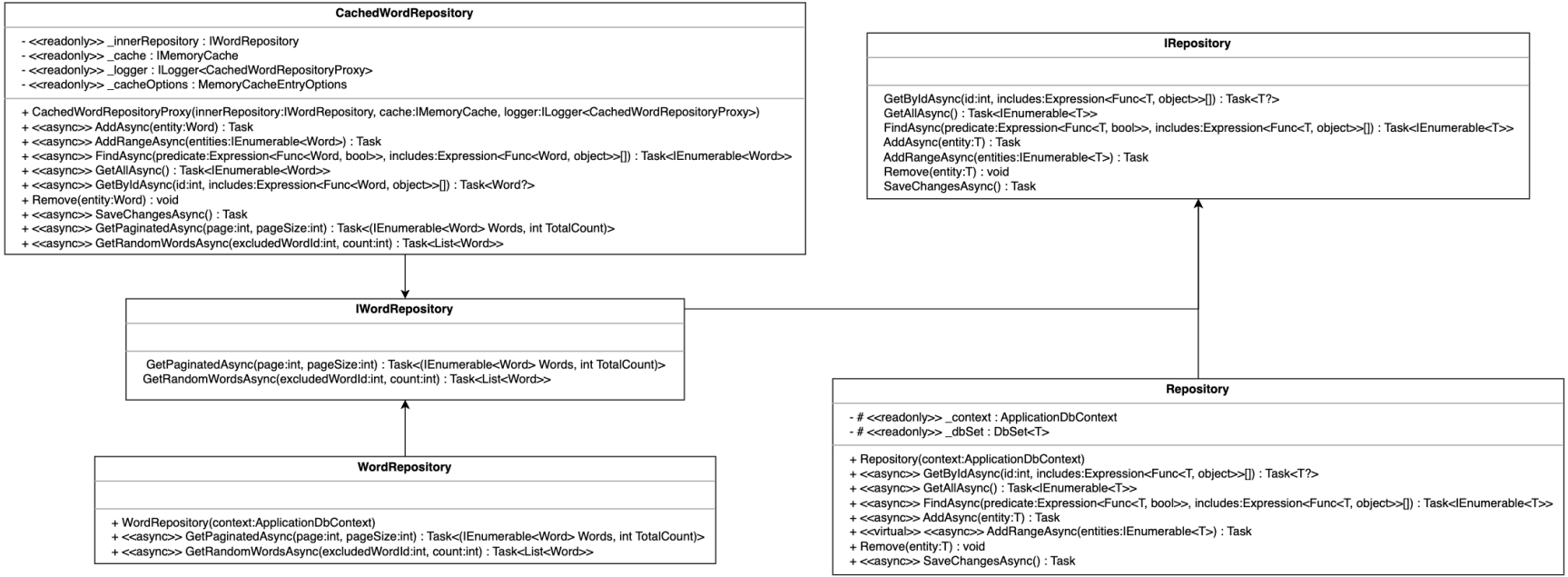

ILearningFacade
GetGroupsWithLanguageAsync(userId:string, languageId:int) : Task<IEnumerable<Group>>
EvaluateAnswerAsync(userId:string, wordId:int, userAnswer:string, mode:LearningMode) : Task<LearningResult>
GetWordsToRepeatAsync(userId:string, languageId:int) : Task<IEnumerable<Word>>
GetAllWordsAsync(languageId:int) : Task<IEnumerable<Word>>
GetGroupWordsAsync(groupId:int) : Task<IEnumerable<Word>>
GetMultipleChoiceOptionsAsync(wordId:int) : Task<List<string>>
LearningFacade
- <<readonly>> _groupRepository : IGroupRepository
- <<readonly>> _wordRepository : IWordRepository
- <<readonly>> _learningStrategies : ILearningStrategies
- <<readonly>> _repeatableWordsService : RepeatableWordsService
- <<readonly>> _activityLogRepository : IActivityLogRepository
+ LearningFacade(groupRepository:IGroupRepository, wordRepository:IWordRepository, learningStrategies:ILearningStrategies, repeatableWordsService:RepeatableWordsService, activityLogRepository:IActivityLogRepository)
+ <<async>> GetGroupsWithLanguageAsync(userId:string, languageId:int) : Task<IEnumerable<Group>>
+ <<async>> EvaluateAnswerAsync(userId:string, wordId:int, userAnswer:string, mode:LearningMode) : Task<LearningResult>
+ <<async>> GetWordsToRepeatAsync(userId:string, languageId:int) : Task<IEnumerable<Word>>
+ <<async>> GetAllWordsAsync(languageId:int) : Task<IEnumerable<Word>>
+ <<async>> GetGroupWordsAsync(groupId:int) : Task<IEnumerable<Word>>
+ <<async>> GetMultipleChoiceOptionsAsync(wordId:int) : Task<List<string>>

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.


CachedWordRepository
- <<readonly>> _innerRepository : IWordRepository
- <<readonly>> _cache : IMemoryCache
- <<readonly>> _logger : ILogger<CachedWordRepositoryProxy>
- <<readonly>> _cacheOptions : MemoryCacheEntryOptions
+ CachedWordRepositoryProxy(innerRepository:IWordRepository, cache:IMemoryCache, logger:ILogger<CachedWordRepositoryProxy>)
+ <<async>> AddAsync(entity:Word) : Task
+ <<async>> AddRangeAsync(entities:IEnumerable<Word>) : Task
+ <<async>> FindAsync(predicate:Expression<Func<Word, bool>>, includes:Expression<Func<Word, object>>[]) : Task<IEnumerable<Word>>
+ <<async>> GetAllAsync() : Task<IEnumerable<Word>>
+ <<async>> GetByIdAsync(id:int, includes:Expression<Func<Word, object>>[]) : Task<Word?>
+ Remove(entity:Word) : void
+ <<async>> SaveChangesAsync() : Task
+ <<async>> GetPaginatedAsync(page:int, pageSize:int) : Task<(IEnumerable<Word> Words, int TotalCount)>
+ <<async>> GetRandomWordsAsync(excludedWordId:int, count:int) : Task<List<Word>>
IRepository
GetByIdAsync(id:int, includes:Expression<Func<T, object>>[]) : Task<T?>
GetAllAsync() : Task<IEnumerable<T>>
FindAsync(predicate:Expression<Func<T, bool>>, includes:Expression<Func<T, object>>[]) : Task<IEnumerable<T>>
AddAsync(entity:T) : Task
AddRangeAsync(entities:IEnumerable<T>) : Task
Remove(entity:T) : void
SaveChangesAsync() : Task
IWordRepository
GetPaginatedAsync(page:int, pageSize:int) : Task<(IEnumerable<Word> Words, int TotalCount)>
GetRandomWordsAsync(excludedWordId:int, count:int) : Task<List<Word>>
WordRepository
+ WordRepository(context:ApplicationDbContext)
+ <<async>> GetPaginatedAsync(page:int, pageSize:int) : Task<(IEnumerable<Word> Words, int TotalCount)>
+ <<async>> GetRandomWordsAsync(excludedWordId:int, count:int) : Task<List<Word>>
Repository
- # <<readonly>> _context : ApplicationDbContext
- # <<readonly>> _dbSet : DbSet<T>
+ Repository(context:ApplicationDbContext)
+ <<async>> GetByIdAsync(id:int, includes:Expression<Func<T, object>>[]) : Task<T?>
+ <<async>> GetAllAsync() : Task<IEnumerable<T>>
+ <<async>> FindAsync(predicate:Expression<Func<T, bool>>, includes:Expression<Func<T, object>>[]) : Task<IEnumerable<T>>
+ <<async>> AddAsync(entity:T) : Task
+ <<virtual>> <<async>> AddRangeAsync(entities:IEnumerable<T>) : Task
+ Remove(entity:T) : void
+ <<async>> SaveChangesAsync() : Task

05 Repository

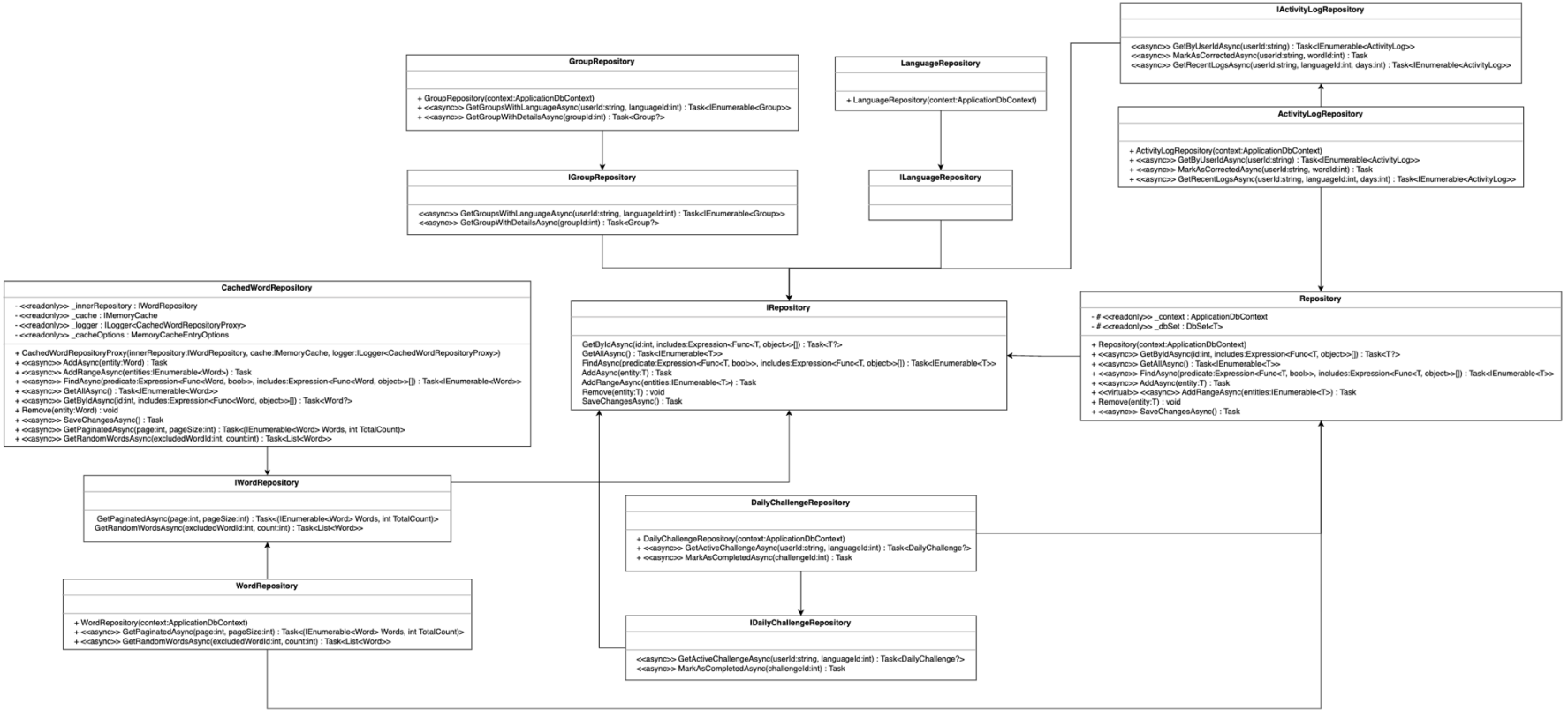
Wzorzec **strukturalny**

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.


IActivityLogRepository
<<async>> GetByUserIdAsync(userId:string) : Task<IEnumerable<ActivityLog>>
<<async>> MarkAsCorrectedAsync(userId:string, wordId:int) : Task
<<async>> GetRecentLogsAsync(userId:string, languageId:int, days:int) : Task<IEnumerable<ActivityLog>>
GroupRepository
+ GroupRepository(context:ApplicationDbContext)
+ <<async>> GetGroupsWithLanguageAsync(userId:string, languageId:int) : Task<IEnumerable<Group>>
+ <<async>> GetGroupWithDetailsAsync(groupId:int) : Task<Group?>
LanguageRepository
+ LanguageRepository(context:ApplicationDbContext)
ActivityLogRepository
+ ActivityLogRepository(context:ApplicationDbContext)
+ <<async>> GetByUserIdAsync(userId:string) : Task<IEnumerable<ActivityLog>>
+ <<async>> MarkAsCorrectedAsync(userId:string, wordId:int) : Task
+ <<async>> GetRecentLogsAsync(userId:string, languageId:int, days:int) : Task<IEnumerable<ActivityLog>>
IGroupRepository
<<async>> GetGroupsWithLanguageAsync(userId:string, languageId:int) : Task<IEnumerable<Group>>
<<async>> GetGroupWithDetailsAsync(groupId:int) : Task<Group?>
ILanguageRepository
CachedWordRepository
- <<readonly>> _innerRepository : IWordRepository
- <<readonly>> _cache : IMemoryCache
- <<readonly>> _logger : ILogger<CachedWordRepositoryProxy>
- <<readonly>> _cacheOptions : MemoryCacheEntryOptions
+ CachedWordRepositoryProxy(innerRepository:IWordRepository, cache:IMemoryCache, logger:ILogger<CachedWordRepositoryProxy>)
+ <<async>> AddAsync(entity:Word) : Task
+ <<async>> AddRangeAsync(entities:IEnumerable<Word>) : Task
+ <<async>> FindAsync(predicate:Expression<Func<Word, bool>>, includes:Expression<Func<Word, object>>[]) : Task<IEnumerable<Word>>
+ <<async>> GetAllAsync() : Task<IEnumerable<Word>>
+ <<async>> GetByIdAsync(id:int, includes:Expression<Func<Word, object>>[]) : Task<Word?>
+ Remove(entity:Word) : void
+ <<async>> SaveChangesAsync() : Task
+ <<async>> GetPaginatedAsync(page:int, pageSize:int) : Task<(IEnumerable<Word> Words, int TotalCount)>
+ <<async>> GetRandomWordsAsync(excludedWordId:int, count:int) : Task<List<Word>>
IRepository
GetByIdAsync(id:int, includes:Expression<Func<T, object>>[]) : Task<T?>
GetAllAsync() : Task<IEnumerable<T>>
FindAsync(predicate:Expression<Func<T, bool>>, includes:Expression<Func<T, object>>[]) : Task<IEnumerable<T>>
AddAsync(entity:T) : Task
AddRangeAsync(entities:IEnumerable<T>) : Task
Remove(entity:T) : void
SaveChangesAsync() : Task
Repository
- # <<readonly>> _context : ApplicationDbContext
- # <<readonly>> _dbSet : DbSet<T>
+ Repository(context:ApplicationDbContext)
+ <<async>> GetByIdAsync(id:int, includes:Expression<Func<T, object>>[]) : Task<T?>
+ <<async>> GetAllAsync() : Task<IEnumerable<T>>
+ <<async>> FindAsync(predicate:Expression<Func<T, bool>>, includes:Expression<Func<T, object>>[]) : Task<IEnumerable<T>>
+ <<async>> AddAsync(entity:T) : Task
+ <<virtual>> <<async>> AddRangeAsync(entities:IEnumerable<T>) : Task
+ Remove(entity:T) : void
+ <<async>> SaveChangesAsync() : Task
IWordRepository
GetPaginatedAsync(page:int, pageSize:int) : Task<(IEnumerable<Word> Words, int TotalCount)>
GetRandomWordsAsync(excludedWordId:int, count:int) : Task<List<Word>>
DailyChallengeRepository
+ DailyChallengeRepository(context:ApplicationDbContext)
+ <<async>> GetActiveChallengeAsync(userId:string, languageId:int) : Task<DailyChallenge?>
+ <<async>> MarkAsCompletedAsync(challengeId:int) : Task
WordRepository
+ WordRepository(context:ApplicationDbContext)
+ <<async>> GetPaginatedAsync(page:int, pageSize:int) : Task<(IEnumerable<Word> Words, int TotalCount)>
+ <<async>> GetRandomWordsAsync(excludedWordId:int, count:int) : Task<List<Word>>
IDailyChallengeRepository
<<async>> GetActiveChallengeAsync(userId:string, languageId:int) : Task<DailyChallenge?>
<<async>> MarkAsCompletedAsync(challengeId:int) : Task

06 Observer

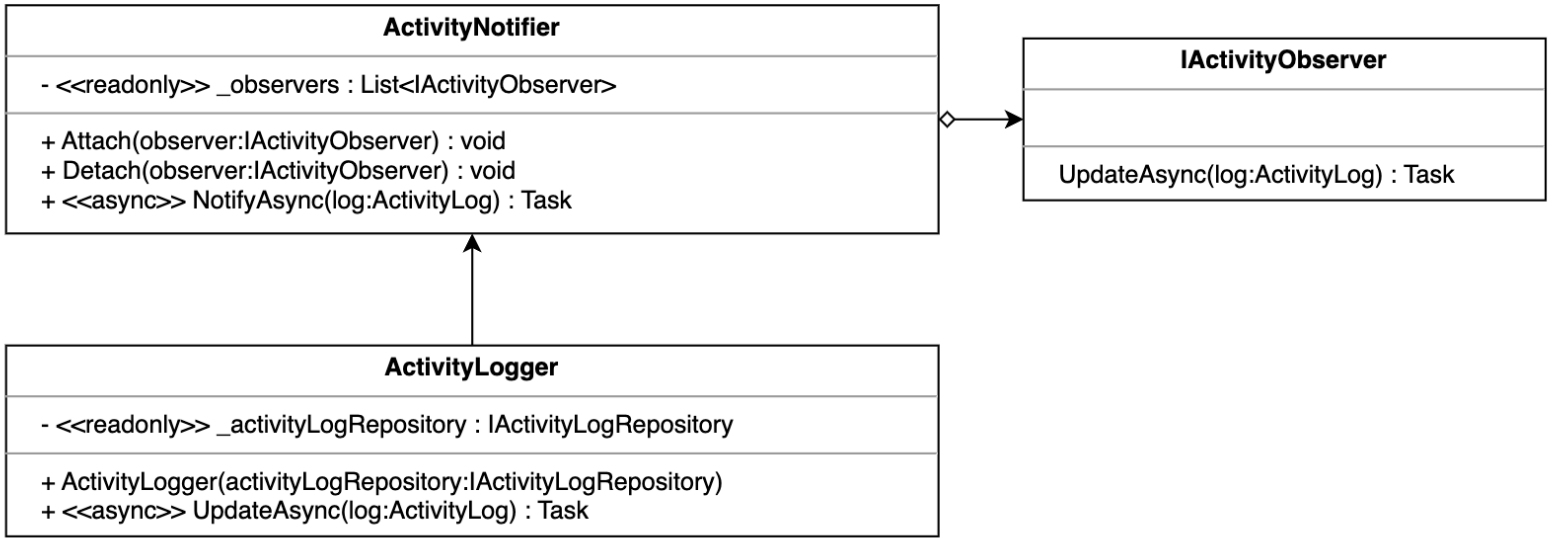
Wzorzec **czynnościowy**

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.


ActivityNotifier
- <<readonly>> _observers : List<IActivityObserver>
+ Attach(observer:IActivityObserver) : void
+ Detach(observer:IActivityObserver) : void
+ <<async>> NotifyAsync(log:ActivityLog) : Task
IActivityObserver
UpdateAsync(log:ActivityLog) : Task
ActivityLogger
- <<readonly>> _activityLogRepository : IActivityLogRepository
+ ActivityLogger(activityLogRepository:IActivityLogRepository)
+ <<async>> UpdateAsync(log:ActivityLog) : Task

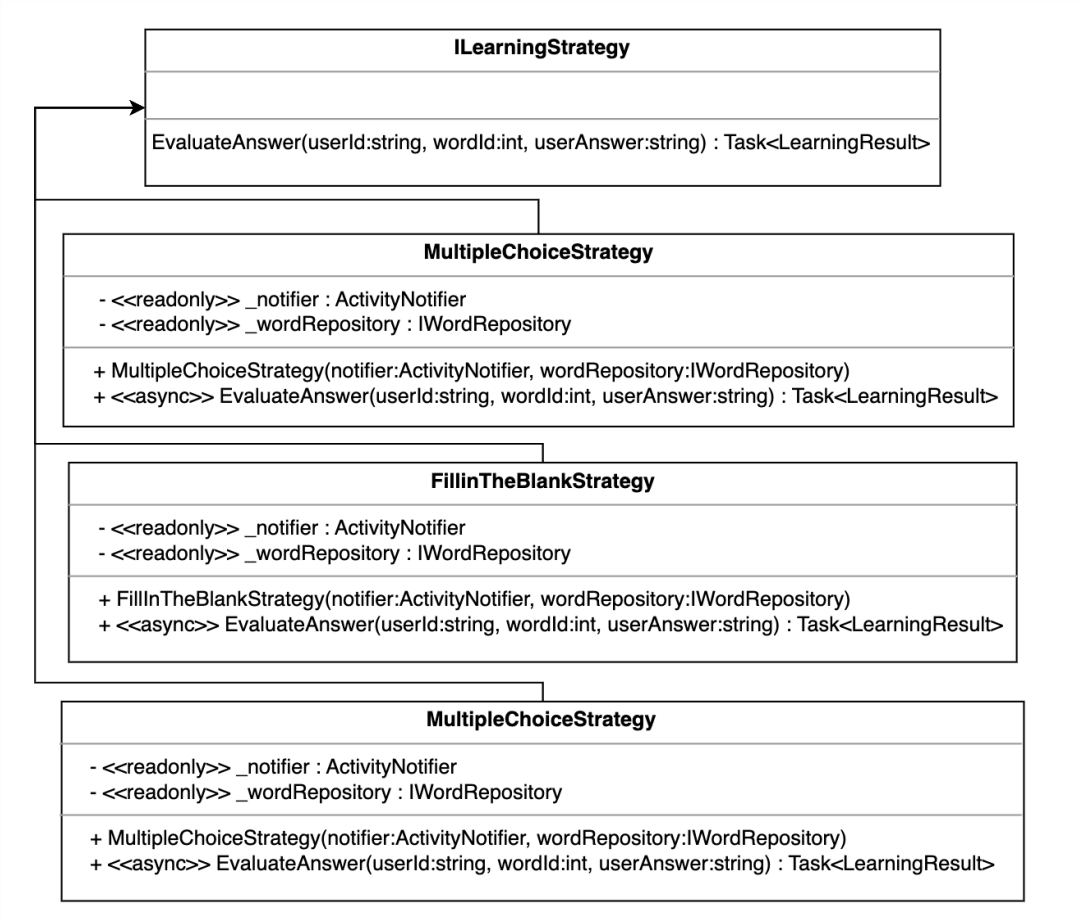
07 Strategy

Wzorzec **czynnościowy**

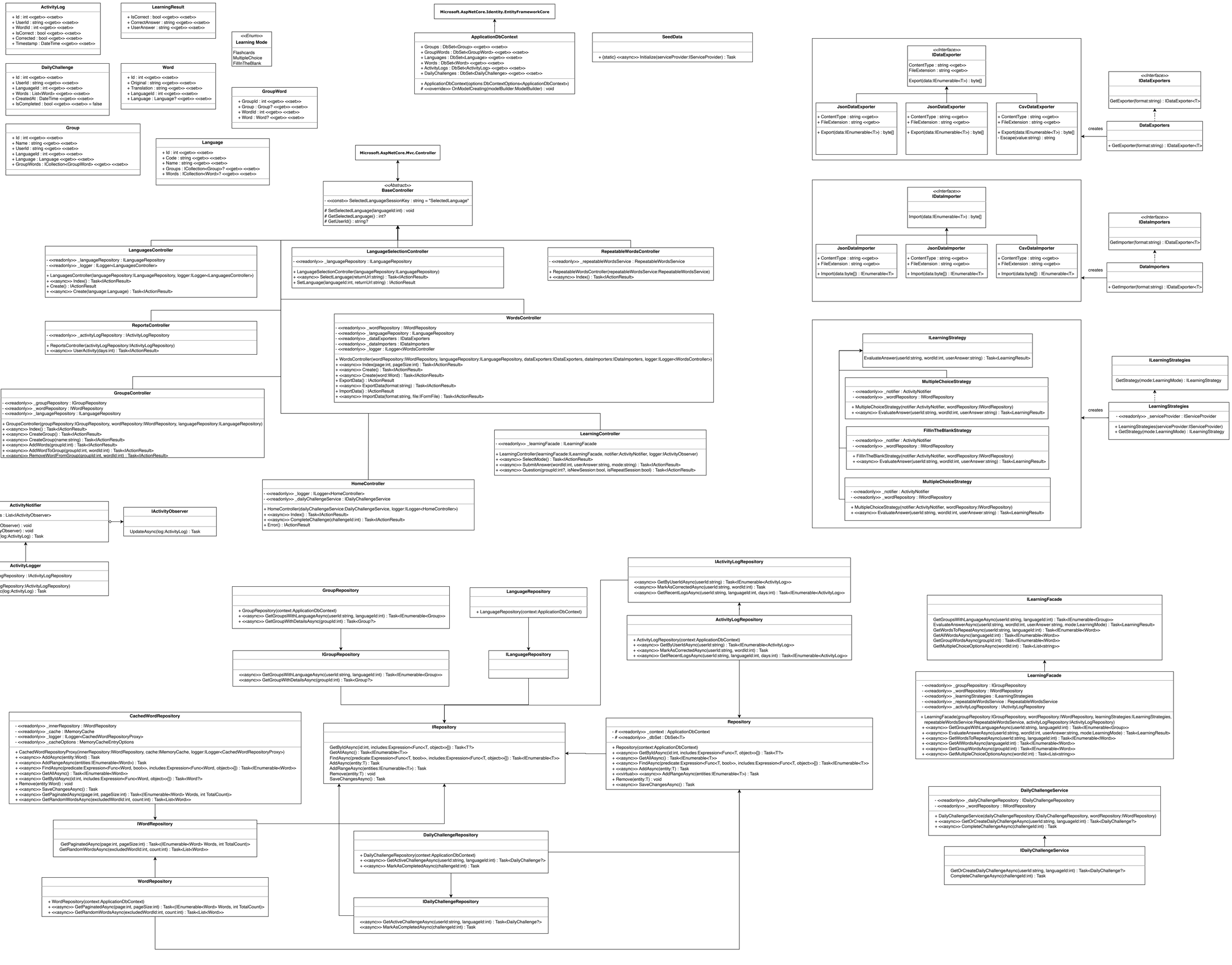
W projekcie wzorzec **Strategy** obsługuje różne tryby nauki dzięki interfejsowi *ILearningStrategy*, zaimplementowanemu przez strategie, takie jak *FlashcardsStrategy*, *MultipleChoiceStrategy*, czy *FillInTheBlankStrategy.*

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

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


ILearningStrategy
EvaluateAnswer(userId:string, wordId:int, userAnswer:string) : Task<LearningResult>
MultipleChoiceStrategy
- <<readonly>> _notifier : ActivityNotifier
- <<readonly>> _wordRepository : IWordRepository
+ MultipleChoiceStrategy(notifier:ActivityNotifier, wordRepository:IWordRepository)
+ <<async>> EvaluateAnswer(userId:string, wordId:int, userAnswer:string) : Task<LearningResult>
FillinTheBlankStrategy
- <<readonly>> _notifier : ActivityNotifier
- <<readonly>> _wordRepository : IWordRepository
+ FillInTheBlankStrategy(notifier:ActivityNotifier, wordRepository:IWordRepository)
+ <<async>> EvaluateAnswer(userId:string, wordId:int, userAnswer:string) : Task<LearningResult>
MultipleChoiceStrategy
- <<readonly>> _notifier : ActivityNotifier
- <<readonly>> _wordRepository : IWordRepository
+ MultipleChoiceStrategy(notifier:ActivityNotifier, wordRepository:IWordRepository)
+ <<async>> EvaluateAnswer(userId:string, wordId:int, userAnswer:string) : Task<LearningResult>

Schemat UML Projektu


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
Microsoft.AspNetCore.Identity.EntityFrameworkCore
ApplicationDbContext
+ Groups : DbSet<Group> <<get>> <<set>>
+ GroupWords : DbSet<GroupWord> <<get>> <<set>>
+ Languages : DbSet<Language> <<get>> <<set>>
+ Words : DbSet<Word> <<get>> <<set>>
+ ActivityLogs : DbSet<ActivityLog> <<get>> <<set>>
+ DailyChallenges : DbSet<DailyChallenge> <<get>> <<set>>
+ ApplicationDbContext(options:DbContextOptions<ApplicationDbContext>)
# <<override>> OnModelCreating(modelBuilder:ModelBuilder) : void
SeedData
+ {static} <<async>> Initialize(serviceProvider:IServiceProvider) : Task
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>>
Microsoft.AspNetCore.Mvc.Controller
<<Abstract>>
BaseController
- <<const>> SelectedLanguageSessionKey : string = "SelectedLanguage"
SetSelectedLanguage(languageId:int) : void
GetSelectedLanguage() : int?
GetUserId() : string?
LanguagesController
- <<readonly>> _languageRepository : ILanguageRepository
- <<readonly>> _logger : ILogger<LanguagesController>
+ LanguagesController(languageRepository:ILanguageRepository, logger:ILogger<LanguagesController>)
+ <<async>> Index() : Task<IActionResult>
+ Create() : IActionResult
+ <<async>> Create(language:Language) : Task<IActionResult>
LanguageSelectionController
- <<readonly>> _languageRepository : ILanguageRepository
+ LanguageSelectionController(languageRepository:ILanguageRepository)
+ <<async>> SelectLanguage(returnUrl:string) : Task<IActionResult>
+ SetLanguage(languageId:int, returnUrl:string) : IActionResult
RepeatableWordsController
- <<readonly>> _repeatableWordsService : RepeatableWordsService
+ RepeatableWordsController(repeatableWordsService:RepeatableWordsService)
+ <<async>> Index() : Task<IActionResult>
ReportsController
- <<readonly>> _activityLogRepository : IActivityLogRepository
+ ReportsController(activityLogRepository:IActivityLogRepository)
+ <<async>> UserActivity(days:int) : Task<IActionResult>
WordsController
- <<readonly>> _wordRepository : IWordRepository
- <<readonly>> _languageRepository : ILanguageRepository
- <<readonly>> _dataExporters : IDataExporters
- <<readonly>> _dataImporters : IDataImporters
- <<readonly>> _logger : ILogger<WordsController
+ WordsController(wordRepository:IWordRepository, languageRepository:ILanguageRepository, dataExporters:IDataExporters, dataImporters:IDataImporters, logger:ILogger<WordsController>)
+ <<async>> Index(page:int, pageSize:int) : Task<IActionResult>
+ <<async>> Create() : Task<IActionResult>
+ <<async>> Create(word:Word) : Task<IActionResult>
+ ExportData() : IActionResult
+ <<async>> ExportData(format:string) : Task<IActionResult>
+ ImportData() : IActionResult
+ <<async>> ImportData(format:string, file:IFormFile) : Task<IActionResult>
GroupsController
- <<readonly>> _groupRepository : IGroupRepository
- <<readonly>> _wordRepository : IWordRepository
- <<readonly>> _languageRepository : ILanguageRepository
+ GroupsController(groupRepository:IGroupRepository, wordRepository:IWordRepository, languageRepository:ILanguageRepository)
+ <<async>> Index() : Task<IActionResult>
+ <<async>> CreateGroup() : Task<IActionResult>
+ <<async>> CreateGroup(name:string) : Task<IActionResult>
+ <<async>> AddWords(groupId:int) : Task<IActionResult>
+ <<async>> AddWordToGroup(groupId:int, wordId:int) : Task<IActionResult>
+ <<async>> RemoveWordFromGroup(groupId:int, wordId:int) : Task<IActionResult>
LearningController
- <<readonly>> _learningFacade : ILearningFacade
+ LearningController(learningFacade:ILearningFacade, notifier:ActivityNotifier, logger:IActivityObserver)
+ <<async>> SelectMode() : Task<IActionResult>
+ <<async>> SubmitAnswer(wordId:int, userAnswer:string, mode:string) : Task<IActionResult>
+ <<async>> Question(groupId:int?, isNewSession:bool, isRepeatSession:bool) : Task<IActionResult>
HomeController
- <<readonly>> _logger : ILogger<HomeController>
- <<readonly>> _dailyChallengeService : IDailyChallengeService
+ HomeController(dailyChallengeService:DailyChallengeService, logger:ILogger<HomeController>)
+ <<async>> Index() : Task<IActionResult>
+ <<async>> CompleteChallenge(challengeId:int) : Task<IActionResult>
+ Error() : IActionResult
ActivityNotifier
- <<readonly>> _observers : List<IActivityObserver>
+ Attach(observer:IActivityObserver) : void
+ Detach(observer:IActivityObserver) : void
+ <<async>> NotifyAsync(log:ActivityLog) : Task
IActivityObserver
UpdateAsync(log:ActivityLog) : Task
ActivityLogger
- <<readonly>> _activityLogRepository : IActivityLogRepository
+ ActivityLogger(activityLogRepository:IActivityLogRepository)
+ <<async>> UpdateAsync(log:ActivityLog) : Task
<<cInterface>>
IDataExporter
ContentType : string <<get>>
FileExtension : string <<get>>
Export(data:IEnumerable<T>) : byte[]
JsonDataExporter
+ ContentType : string <<get>>
+ FileExtension : string <<get>>
+ Export(data:IEnumerable<T>) : byte[]
JsonDataExporter
+ ContentType : string <<get>>
+ FileExtension : string <<get>>
+ Export(data:IEnumerable<T>) : byte[]
CsvDataExporter
+ ContentType : string <<get>>
+ FileExtension : string <<get>>
+ Export(data:IEnumerable<T>) : byte[]
- Escape(value:string) : string
creates
<<cInterface>>
IDataExporters
GetExporter(format:string) : IDataExporter<T>
DataExporters
+ GetExporter(format:string) : IDataExporter<T>
<<cInterface>>
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
<<cInterface>>
IDataImporters
GetImporter(format:string) : IDataExporter<T>
DataImporters
+ GetImporter(format:string) : IDataExporter<T>
ILearningStrategy
EvaluateAnswer(userId:string, wordId:int, userAnswer:string) : Task<LearningResult>
MultipleChoiceStrategy
- <<readonly>> _notifier : ActivityNotifier
- <<readonly>> _wordRepository : IWordRepository
+ MultipleChoiceStrategy(notifier:ActivityNotifier, wordRepository:IWordRepository)
+ <<async>> EvaluateAnswer(userId:string, wordId:int, userAnswer:string) : Task<LearningResult>
FillInTheBlankStrategy
- <<readonly>> _notifier : ActivityNotifier
- <<readonly>> _wordRepository : IWordRepository
+ FillInTheBlankStrategy(notifier:ActivityNotifier, wordRepository:IWordRepository)
+ <<async>> EvaluateAnswer(userId:string, wordId:int, userAnswer:string) : Task<LearningResult>
MultipleChoiceStrategy
- <<readonly>> _notifier : ActivityNotifier
- <<readonly>> _wordRepository : IWordRepository
+ MultipleChoiceStrategy(notifier:ActivityNotifier, wordRepository:IWordRepository)
+ <<async>> EvaluateAnswer(userId:string, wordId:int, userAnswer:string) : Task<LearningResult>
ILearningStrategies
GetStrategy(mode:LearningMode) : ILearningStrategy
creates
LearningStrategies
- <<readonly>> _serviceProvider : IServiceProvider
+ LearningStrategies(serviceProvider:IServiceProvider)
+ GetStrategy(mode:LearningMode) : ILearningStrategy
GroupRepository
+ GroupRepository(context:ApplicationDbContext)
+ <<async>> GetGroupsWithLanguageAsync(userId:string, languageId:int) : Task<IEnumerable<Group>>
+ <<async>> GetGroupWithDetailsAsync(groupId:int) : Task<Group?>
LanguageRepository
+ LanguageRepository(context:ApplicationDbContext)
IGroupRepository
<<async>> GetGroupsWithLanguageAsync(userId:string, languageId:int) : Task<IEnumerable<Group>>
<<async>> GetGroupWithDetailsAsync(groupId:int) : Task<Group?>
ILanguageRepository
IActivityLogRepository
<<async>> GetByUserIdAsync(userId:string) : Task<IEnumerable<ActivityLog>>
<<async>> MarkAsCorrectedAsync(userId:string, wordId:int) : Task
<<async>> GetRecentLogsAsync(userId:string, languageId:int, days:int) : Task<IEnumerable<ActivityLog>>
ActivityLogRepository
+ ActivityLogRepository(context:ApplicationDbContext)
+ <<async>> GetByUserIdAsync(userId:string) : Task<IEnumerable<ActivityLog>>
+ <<async>> MarkAsCorrectedAsync(userId:string, wordId:int) : Task
+ <<async>> GetRecentLogsAsync(userId:string, languageId:int, days:int) : Task<IEnumerable<ActivityLog>>
ILearningFacade
GetGroupsWithLanguageAsync(userId:string, languageId:int) : Task<IEnumerable<Group>>
EvaluateAnswerAsync(userId:string, wordId:int, userAnswer:string, mode:LearningMode) : Task<LearningResult>
GetWordsToRepeatAsync(userId:string, languageId:int) : Task<IEnumerable<Word>>
GetAllWordsAsync(languageId:int) : Task<IEnumerable<Word>>
GetGroupWordsAsync(groupId:int) : Task<IEnumerable<Word>>
GetMultipleChoiceOptionsAsync(wordId:int) : Task<List<string>>
LearningFacade
- <<readonly>> _groupRepository : IGroupRepository
- <<readonly>> _wordRepository : IWordRepository
- <<readonly>> _learningStrategies : ILearningStrategies
- <<readonly>> _repeatableWordsService : RepeatableWordsService
- <<readonly>> _activityLogRepository : IActivityLogRepository
+ LearningFacade(groupRepository:IGroupRepository, wordRepository:IWordRepository, learningStrategies:ILearningStrategies, repeatableWordsService:RepeatableWordsService, activityLogRepository:IActivityLogRepository)
+ <<async>> GetGroupsWithLanguageAsync(userId:string, languageId:int) : Task<IEnumerable<Group>>
+ <<async>> EvaluateAnswerAsync(userId:string, wordId:int, userAnswer:string, mode:LearningMode) : Task<LearningResult>
+ <<async>> GetWordsToRepeatAsync(userId:string, languageId:int) : Task<IEnumerable<Word>>
+ <<async>> GetAllWordsAsync(languageId:int) : Task<IEnumerable<Word>>
+ <<async>> GetGroupWordsAsync(groupId:int) : Task<IEnumerable<Word>>
+ <<async>> GetMultipleChoiceOptionsAsync(wordId:int) : Task<List<string>>
CachedWordRepository
- <<readonly>> _innerRepository : IWordRepository
- <<readonly>> _cache : IMemoryCache
- <<readonly>> _logger : ILogger<CachedWordRepositoryProxy>
- <<readonly>> _cacheOptions : MemoryCacheEntryOptions
+ CachedWordRepositoryProxy(innerRepository:IWordRepository, cache:IMemoryCache, logger:ILogger<CachedWordRepositoryProxy>)
+ <<async>> AddAsync(entity:IWord) : Task
+ <<async>> AddRangeAsync(entities:IEnumerable<Word>) : Task
+ <<async>> FindAsync(predicate:Expression<Func<Word, bool>>, includes:Expression<Func<Word, object>>[]) : Task<IEnumerable<Word>>
+ <<async>> GetAllAsync() : Task<IEnumerable<Word>>
+ <<async>> GetByIdAsync(id:int, includes:Expression<Func<Word, object>>[]) : Task<Word?>
+ Remove(entity:Word) : void
+ <<async>> SaveChangesAsync() : Task
+ <<async>> GetPaginatedAsync(page:int, pageSize:int) : Task<(IEnumerable<Word> Words, int TotalCount)>
+ <<async>> GetRandomWordsAsync(excludedWordId:int, count:int) : Task<List<Word>>
IRepository
GetByIdAsync(id:int, includes:Expression<Func<T, object>>[]) : Task<T?>
GetAllAsync() : Task<IEnumerable<T>>
FindAsync(predicate:Expression<Func<T, bool>>, includes:Expression<Func<T, object>>[]) : Task<IEnumerable<T>>
AddAsync(entity:T) : Task
AddRangeAsync(entities:IEnumerable<T>) : Task
Remove(entity:T) : void
SaveChangesAsync() : Task
Repository
- # <<readonly>> _context : ApplicationDbContext
- # <<readonly>> _dbSet : DbSet<T>
+ Repository(context:ApplicationDbContext)
+ <<async>> GetByIdAsync(id:int, includes:Expression<Func<T, object>>[]) : Task<T?>
+ <<async>> GetAllAsync() : Task<IEnumerable<T>>
+ <<async>> FindAsync(predicate:Expression<Func<T, bool>>, includes:Expression<Func<T, object>>[]) : Task<IEnumerable<T>>
+ <<async>> AddAsync(entity:T) : Task
+ <<virtual>> <<async>> AddRangeAsync(entities:IEnumerable<T>) : Task
+ Remove(entity:T) : void
+ <<async>> SaveChangesAsync() : Task
IWordRepository
GetPaginatedAsync(page:int, pageSize:int) : Task<(IEnumerable<Word> Words, int TotalCount)>
GetRandomWordsAsync(excludedWordId:int, count:int) : Task<List<Word>>
WordRepository
+ WordRepository(context:ApplicationDbContext)
+ <<async>> GetPaginatedAsync(page:int, pageSize:int) : Task<(IEnumerable<Word> Words, int TotalCount)>
+ <<async>> GetRandomWordsAsync(excludedWordId:int, count:int) : Task<List<Word>>
DailyChallengeRepository
+ DailyChallengeRepository(context:ApplicationDbContext)
+ <<async>> GetActiveChallengeAsync(userId:string, languageId:int) : Task<DailyChallenge?>
+ <<async>> MarkAsCompletedAsync(challengeId:int) : Task
IDailyChallengeRepository
<<async>> GetActiveChallengeAsync(userId:string, languageId:int) : Task<DailyChallenge?>
<<async>> MarkAsCompletedAsync(challengeId:int) : Task
DailyChallengeService
- <<readonly>> _dailyChallengeRepository : IDailyChallengeRepository
- <<readonly>> _wordRepository : IWordRepository
+ DailyChallengeService(dailyChallengeRepository:IDailyChallengeRepository, wordRepository:IWordRepository)
+ <<async>> GetOrCreateDailyChallengeAsync(userId:string, languageId:int) : Task<DailyChallenge?>
+ <<async>> CompleteChallengeAsync(challengeId:int) : Task
IDailyChallengeService
GetOrCreateDailyChallengeAsync(userId:string, languageId:int) : Task<DailyChallenge?>
CompleteChallengeAsync(challengeId:int) : Task