System wspomagający tworzenie i modyfikowanie programów kształcenia

Architecture Notebook

1. Cel (Purpose)

Celem dokumentu jest opis decyzji, ograniczeń oraz uzasadnień decyzji dla istotnych elementów systemu, które mają wpływ na projekt oraz implementację systemu. Dokument ten przedstawia różne punkty spojrzenia na system wspomagający tworzenie i modyfikowanie programów kształcenia na Politechnice Wrocławskiej. Stanowi on odpowiedź na przyszłe pytania deweloperów w kwestiach związanych z architektonicznym spojrzeniem na implementowany system oraz podsumowanie decyzji podjętych przez architektów systemu.

2. Cele architektoniczne i ograniczenia (Architectural goals and constraints)

W trakcie przeprowadzonej analizy biznesowej oraz systemowej zdefiniowana została dziedzina oraz wymagania, które powinna spełniać architektura systemu. Wybrane wymagania zaprezentowane zostały na diagramach**.** Sprowadzają się one do realizacji jednej dużej funkcjonalności jaką jest możliwość opracowania programów kształcenia (studiów) dla Politechniki Wrocławskiej oraz ich modyfikacja.

Szczegóły dotyczące wymagań znajdują się w dalszej części dokumentu, ogólny zarys koniecznych do zrealizowania funkcjonalności prezentuje się następująco:

- CRUD efektów ministerialnych z podziałem na dziedziny, profile i poziomy
- CRUD programu studiów w tym planu studiów oraz macierzy śladowania
- Ustalanie opiekunów przedmiotów
- Wyszukiwanie i przeglądanie obowiązujących efektów ministerialnych, planów studiów, programów studiów, kart przedmiotów
- Możliwość pobrania pliku z kartą przedmiotu
- CRU oraz wersjonowanie kart przedmiotu dla danego programu kształcenia

Wymagania jakościowe:

- Dwie wersje językowe interfejsu użytkownika: polski i angielski
- Dane o efektach/planach/programach/kartach nie mogą zostać utracone

- Wydajne pobieranie danych (zapytania, przeglądanie) – średni czas odpowiedzi systemu poniżej 5 s przy obciążeniu do 200 użytkowników
- Migracja systemu na inny system operacyjny (np. do Linux z Windows) czy innego dostawcę serwera bazy danych (np. do Oracle z SQL Serwer) powinna zająć mniej niż 1 osobodzień prac

Ograniczenia:

- System wymaga stałego dostępu do Internetu
- System jest dostępny w najnowszych wersjach przeglądarek Google Chrome, Firefox

W ramach realizowanego system przewidziana jest integracja z poniżej wymienionymi systemami zewnętrznymi:

- System weryfikujący osiągnięcia kierunkowych efektów kształcenia
  - Wysyłanie programów studiów w celu weryfikacji
  - Pobranie syntetycznych raportów weryfikacyjnych
- System wyznaczający zamienniki kursów
  - Wysyłanie programu studiów w celu wyznaczenia zamienników
- System planowania powierzeń dla kierunków
  - Wysyłanie programu studiów dla poszczególnych kierunków

3. Decyzje i uzasadnienia (Decisions and justifications)

Cel	Sposób osiągnięcia
1. Dane o efektach/ planach/ programach/ kartach nie mogą zostać utracone.	A. Dane o efektach/planach/programach/kartach zapisywane będą w relacyjnej bazie danych. B. Tworzenie kopii zapasowych bazy danych
2. Wydajne pobieranie danych (zapytania, przeglądanie) – średni czas odpowiedzi systemu poniżej 5 s przy obciążeniu do 200 użytkowników.	A. Wykorzystanie modułu równoważenia obciążenia pomiędzy niezależne jednostki B. Optymalizacja zapytań do bazy danych a. Cache dla zapytań b. Pobieranie tylko wymaganych danych c. Wykorzystanie mechanizmu stronicowania C. Optymalizacja zapytań wysyłanych przez aplikację kliencką do serwera a. Przesyłanie tylko wymaganych danych b. Ograniczenie liczby zapytań do API c. Wykorzystanie mechanizmu stronicowania d. Wykorzystywanie danych zapisanych w pamięci podręcznej, pamięci przeglądarki D. Implementacja asynchronicznego API

3. Dwie wersje językowe interfejsu użytkownika: polski i angielski.	A. Wyświetlanie interfejsu użytkownika w jednej z dwóch wersji językowych poprzez stworzenie plików ze wspólnymi etykietami i osobnymi ich wartościami dla języka polskiego i angielskiego
4. Migracja systemu na inny system operacyjny (np. do Linux z Windows) czy innego dostawcę serwera bazy danych (np. do Oracle z SQL Serwer) powinna zająć mniej niż 1 osobodzień prac.	A. Wykorzystanie narzędzi wirtualizacji, w szczególności umożliwiających konteneryzacje aplikacji. B. Wykorzystanie technologii, działających zarówno na systemach Windows jak i Linux. C. Wykorzystanie narzędzi mapowania obiektowo-relacyjnego współpracującego z różnymi systemami zarządzania relacyjnymi bazami danych.

4. Mechanizmy architektoniczne (Architectural Mechanisms)

Mechanizmy zapewniające trwałość danych

A. Dane o efektach/planach/programach/kartach zapisywane będą w relacyjnej bazie danych.

W systemie zaimplementowana zostanie warstwa składowania danych oparta o relacyjną bazę danych Microsoft SQL Server 2017. Z poziomu serwera dostęp do niej zostanie zapewniony poprzez system mapowania obiektowo-relacyjnego - Entity Framework Core. Relacyjny model danych MSSQL zapewnia implementację zbioru właściwości gwarantujących przetwarzanie transakcyjne - ACID. W szczególności zapewniona zostanie trwałość danych rozumiana jako możliwość uruchomienia się, udostępnienia spójnych, nienaruszonych i aktualnych danych zapisanych w ramach zatwierdzonych transakcji po nagłych awariach zasilania.

B. Tworzenie kopii zapasowych bazy danych

Dodatkowym zabezpieczeniem przed utratą danych składowanych w bazie okresowo utworzone zostaną jej aktualne kopie. Proces tworzenia kopii bazy danych będzie uruchamiany raz w tygodniu lub na żądanie administratora. Kopie bazy danych będą przetrzymywane na serwerze o innej lokalizacji fizycznej niż ta na której umieszczona zostanie baza danych. Dzięki temu w przypadku utraty fizycznego nośnika składowania danych możliwe będzie ich odzyskanie. Również w przypadku zniszczenia zawartości bazy danych lub jej niepożądanego wyczyszczenia możliwy będzie powrót do działającej wersji.

Mechanizmy zapewniające wydajność

A. Wykorzystanie modułu równoważenia obciążenia.

W systemie zostanie wykorzystany moduł równoważenia obciążenia (load balancer). Do uruchomienia systemu zostanie użyty Kubernetes, który będzie odpowiadał za odpowiednie przydzielanie zasobów poszczególnym węzłom. Aplikacja zostanie podzielona na trzy warstwy:

- Prezentacji - odpowiada za przesyłanie danych takich jak kod HTML i skrypty, które są uruchamiane i wyświetlane w przeglądarce użytkownika. Warstwa ta komunikuje się z warstwą logiki biznesowej i odbiera bądź wysyła do niej dane. Do prezentacji zostanie wykorzystany framework Vue.js.

- Logiki biznesowej - zadaniem tej warstwy jest realizowanie logiki biznesowej. Komunikuje się z warstwą składowania danych oraz z warstwą prezentacji. Realizuje walidację danych otrzymywanych z warstwy prezentacji. Udostępnia REST API, które pozwala na pobieranie oraz tworzenie zasobów. API zostanie stworzone z wykorzystaniem ASP.NET Core Web API 2.

- Składowania danych - jej celem jest przechowywanie danych otrzymywanych z warstwy logiki biznesowej. Do składowania danych użyty zostanie Microsoft SQL Server 2017.

B. Optymalizacja zapytań do bazy danych

W implementowanym systemie w zdecydowanej większości przypadków przeważać będą próby odczytu informacji z bazy danych, dlatego szczególną uwagę zwrócono na optymalizację tych zapytań. Zostaną wykorzystane następujące mechanizmy optymalizacyjne:

- Cache dla zapytań - dane zapisane w bazie rzadko będą edytowane, a wykonywane zapytania często będą parametryzowane tymi samymi wartościami. Sprawia to, że opłacalne jest wykorzystanie pamięci podręcznej. Do cache'owania użyta zostanie paczka Microsoft.Extensions.Caching.Memory.
- Pobieranie tylko wymaganych danych - z bazy będą pobierane tylko te informacje, które są w danej chwili potrzebne. Pola obiektów nie będą wypełniane jeżeli nie będą wykorzystane na dalszym etapie przetwarzania. Zostanie wykorzystany mechanizm lazy loading, który pobierze dodatkowe informacje w chwili, gdy będzie wykonana próba ich użycia.
- Wykorzystanie mechanizmu stronicowania - rekordy będą pobierane partiami o określonej wielkości. Dzięki niepobieraniu wszystkich rekordów jednocześnie odpowiedź serwera będzie szybsza.

C. Optymalizacja zapytań aplikacji frontendowej do REST API aplikacji backendowej

- **Przesyłanie tylko wymaganych danych**

Niektóre dane pojawiać się będą w interfejsie aplikacji wielokrotnie, jednak w innej formie. Każdy z widoków potrzebować może jedynie fragmentu danych danego obiektu, a co za tym idzie nie ma potrzeby przesyłania z serwera wszystkich informacji dla danej encji. Przykładem takiej sytuacji jest wyświetlanie listy obiektów z możliwością przejścia do szczegółów danego obiektu - lista potrzebuje bardzo ‘okrojonej’ wersji i nie ma potrzeby przesyłania wszystkich danych wszystkich obiektów. Osiągnąć to można za pomocą wyspecyfikowania obiektów służących tylko do przesyłania ich pomiędzy procesami/warstwami aplikacji przy pomocy interfejsów: DTO(*ang. Data Transfer Object).* W odpowiedzi na konkretne zapytanie frontendu należy przesłać jedynie dane, które będą wyświetlane w danym widoku.

- **Ograniczenie liczby zapytań do API**
  Należy ograniczyć do minimum liczbę zapytań wysyłanych z aplikacji frontendowej do aplikacji backendowej. Przykładem zastosowania jest użycie mechanizmu czasu odbicia (debounce), który sprawi, że zapytanie dla podpowiedzi wyszukiwania nie będzie wysyłane, dopóki użytkownik nie skończy wpisywać zapytania lub nie kontynuuje wpisywania przez określony czas. We frameworku Vue.js można w tym celu użyć biblioteki *vue-debounce.*

- **Wykorzystanie mechanizmu stronicowania**
  W przypadku dużej liczby zwróconych wyników wyszukiwania należy użyć mechanizmu stronicowania. Oznacza to zwracanie jedynie określonej liczby wyników i przesyłanie kolejnych dopiero wtedy, gdy użytkownik ich potrzebuje. Dzięki temu nie będzie potrzeby wysyłania dużej liczby obiektów, gdy jest duże prawdopodobieństwo, że użytkownik potrzebuje jedynie jednego z pierwszych kilku wyników.

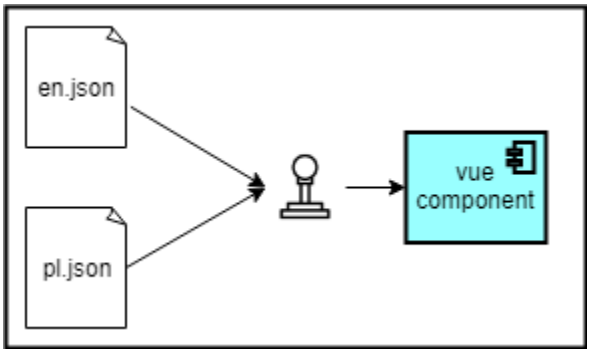
D. Implementacja asynchronicznego API

Asynchroniczne API znacząco poprawia wydajność oraz czas odpowiedzi serwera. ASP.NET Core udostępnia konstrukcję, która umożliwia asynchroniczne wywoływanie metod, w tym także akcji kontrolera. Dzięki temu mechanizmowi wątki odpowiadające za operacje I/O nie są blokowane w przypadku wykonywania długich operacji, takich jak np. pobieranie danych z bazy i mogą one obsłużyć inne przychodzące żądania.

Mechanizmy zapewniające wielojęzyczność aplikacji

Aplikacja ma wspierać rozwiązania zapewniające użytkownikowi możliwość wyboru używanej wersji językowej. Wybór języka z puli wspieranych języków dostępny będzie dla każdego użytkownika aplikacji z poziomu warstwy prezentacji. Mimo że aplikacja wspiera również różne języki dla zapisanych danych nie wpływa to znacząco na rozwiązania w warstwie persystencji, gdyż zapisywane dane są spójne co do używanego języka. Przykładowo karta przedmiotu w języku angielskim oraz karta w języku polskim będą oddzielnymi encjami, a co za tym idzie wszystkie ich pola uzupełnione będą w odpowiednim języku. W związku z tym tłumaczone będą jedynie statyczne elementy interfejsu użytkownika.

Framework frontendowy Vue.js, który zostanie użyty do stworzenia warstwy prezentacji aplikacji wspiera tłumaczenie przy użyciu dodatkowych bibliotek, przykładowo *vue-i18n.* Przy pomocy zaproponowanej biblioteki wystarczy zdefiniować odpowiednie etykiety z wartościami dla użytych statycznych pól tekstowych, a następnie odnosić się do nich w kodzie. Dla każdej wersji językowej stworzony zostanie osobny plik, z którego będą pobierane wartości do wyświetlenia. Dodanie nowej wersji językowej sprowadzać się będzie zatem do przetłumaczenia wartości dla konkretnych etykiet i umieszczenia takiego tłumaczenia z tymi samymi etykietami w osobnym pliku. Zmiana wersji językowej dokonywać będzie się w runtime, zatem nie będzie wymagała ponownego kompilowania aplikacji.



Mechanizmy zapewniające możliwość migracji aplikacji na inny system operacyjny oraz bazę danych.

A. Wykorzystanie narzędzi wirtualizacji i konteneryzacji.

System zostanie podzielony na 3 moduły: aplikację serwerową, aplikację kliencką oraz bazę danych. Każdy z nich zostanie umieszczony w osobnym kontenerze Dockera. Wykorzystaną cechą kontenerów, w tym przypadku, jest ich odizolowanie. Stanowią one kompletne i niezależne środowisko uruchomieniowe, które może zostać uruchomione zarówno na Windowsie jak i Linuxie. Docker jest wspierany natywnie na systemach Linuxowych oraz przy pomocy dodatkowych narzędzi *„Docker for Windows”* lub *„Docker Toolbox”* na Windowsie. W przypadku migracji na inny system operacyjny kontenery powinny zachowywać się dokładnie tak samo.

B. Wykorzystanie wieloplatformowych technologii.

Webowe API aplikacji zostanie zaimplementowane z wykorzystaniem technologii .NET Core 3.0. Technologia ta, w porównaniu do .NET Framework jest wieloplatformowa, co oznacza, że aplikacje stworzone przy jej użyciu mogą być uruchamiane na różnych systemach operacyjnych (np. Windows i Linux). Do stworzenia aplikacji frontendowej zostanie użyty Node.js – Javascriptowe środowisko uruchomieniowe. Podobnie jak .NET Core zapewnia ono wieloplatformowość.

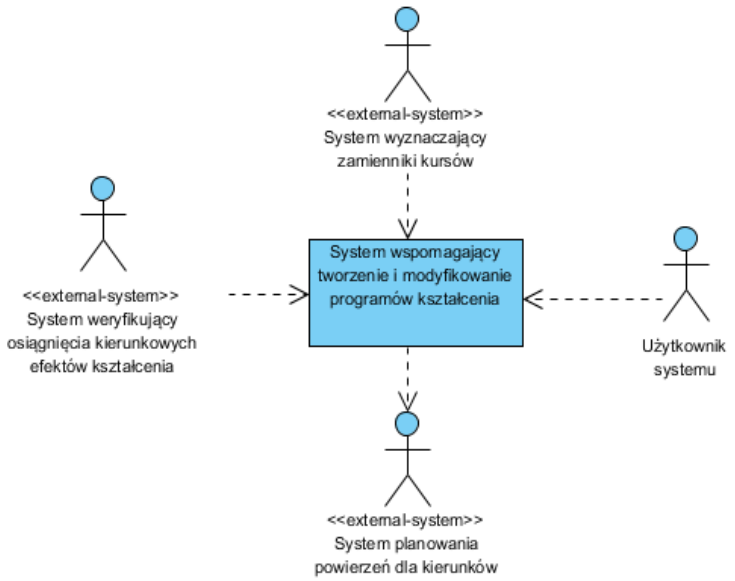
C. Wykorzystanie narzędzi ORM współpracującego z różnymi systemami RDBMS.

Entity Framework, który obsługuje mapowanie relacyjno-obiektowe, przy użyciu map providera tłumaczy zapytania SQLowe na odpowiednią wersję SQLa. Połączenie z bazą danych również wyekstraktowane zostało do odpowiedniej abstrakcji, dzięki czemu wystarczy zmienić dostawcę źródła danych, aby zmienić system bazodanowy używany przez projekt. Duże zaangażowanie społeczności programistycznej w rozwój frameworka doprowadziło do powstania dostawców dla wszystkich popularnych systemów baz danych, m.in. SQL Server, PostgreSQL, SQLite oraz MySql. Zmiana systemu bazy danych sprowadza się zatem do przekonfigurowania dostawcy w konfiguracji projektu.

5. Widoki architektoniczne

5.1 Perspektywa kontekstowa (Context view)

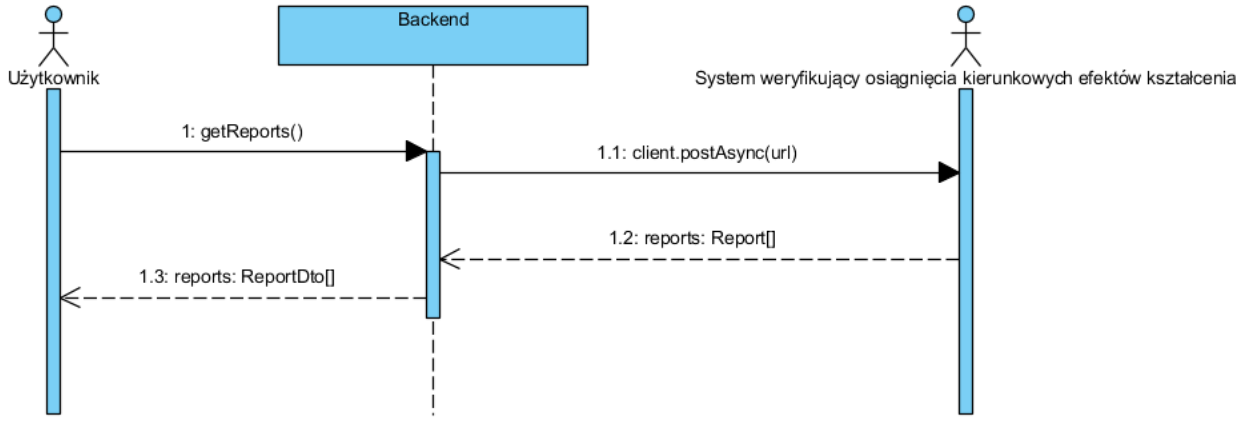
5.1.1 Diagram kontekstowy



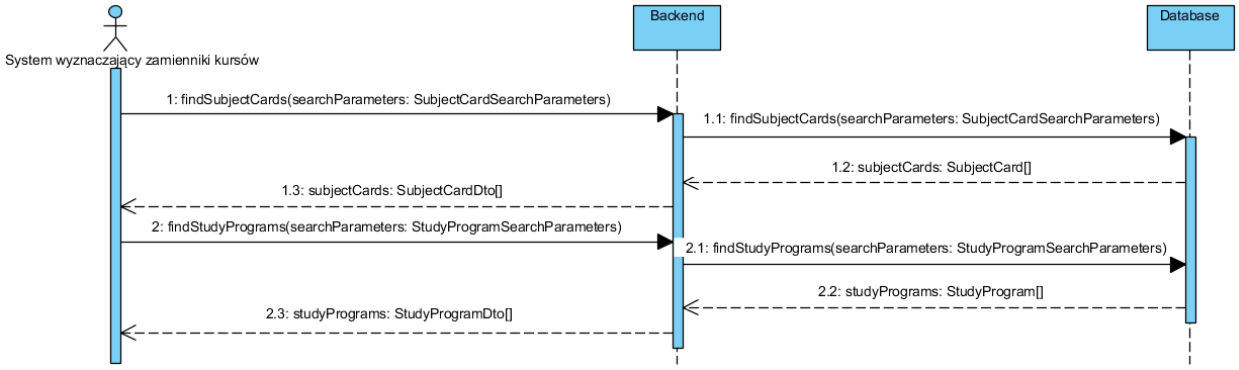
5.1.2 Interaction scenarios

Dla każdego zintegrowanego systemu stworzony został diagram sekwencji pokazujący kolejne kroki wykonywane podczas interakcji między systemami.

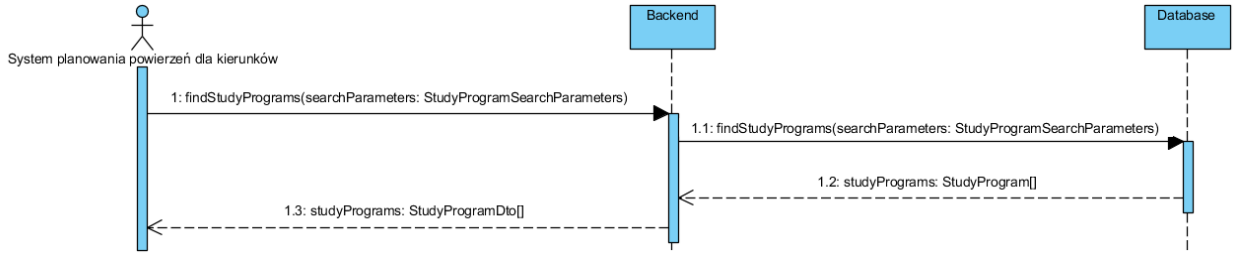
1. **Pobieranie syntetycznych raportów**



2. **Wyszukiwanie kart przedmiotów oraz programów studiów przez system wyznaczający zamienniki kursów**



3. **Wyszukiwanie programów studiów przez system planowania powierzeń**



5.1.3 Integration interfaces – logical level

Interface 1- Pobieranie syntetycznych raportów

Pracownik uczelni musi w niektórych przypadkach podsumowywać wyniki kursu, w tym efekty kształcenia osiągnięte oraz nie osiągnięte przez uczestniczących w kursie studentów. W tym celu na koniec realizacji kursu tworzy syntetyczny raport, w którym specyfikuje, które efekty kształcenia udało się osiągnąć, a które nie. W przypadku nieosiągniętych przez jakąś liczbę studentów efektów przedstawia powód takiego stanu rzeczy oraz może podać sugerowane zmiany, przykładowo konieczność realizacji kursu w innym semestrze.

Description	Interfejs odpowiadający za pobieranie raportów zawierających informacje o stopniu osiągnięcia efektów kształcenia.	
Status	Planowany	
	Source application	Target application
Application name	System wspomagający weryfikację osiągnięcia kierunkowych efektów kształcenia	System do tworzenia i modyfikowania efektów kształcenia
Integration technology	REST/HTTPS	REST/HTTPS
Authentication mechanism	Brak	Brak
Data contract	Wejście: pliki raportu Format: plik PDF Wyjście: brak	
Does the interface manipulate on the sensitive data (RODO)?	Nie	
Middleware used	Brak	
Initiating side	System do tworzenia i modyfikowania efektów kształcenia	
Communication model	Synchroniczny na żądanie użytkownika	

Performance	5 / h
Volumetry	Liczba wywołań: 40 / rok (2 użytkowników na koniec każdego semestru po około 10 razy)
Required accessibility	LOW

Interface 2 - Wyszukiwanie kart przedmiotów

<table>
<tr><td>Description</td><td colspan="2">Interfejs służący do wyszukiwania kart przedmiotów na podstawie takich parametrów jak nazwa oraz kod karty. Wyniki są stronicowane. Interfejs będzie używany również w aplikacji frontendowej.</td></tr>
<tr><td>Status</td><td colspan="2">Planowany</td></tr>
<tr><td></td><td>Source application</td><td>Target application</td></tr>
<tr><td>Application name</td><td>System do tworzenia i modyfikowania efektów kształcenia</td><td>System wyznaczający zamienniki kursów</td></tr>
<tr><td>Integration technology</td><td>REST/HTTPS</td><td>REST/HTTPS</td></tr>
<tr><td>Authentication mechanism</td><td>Brak</td><td>Brak</td></tr>
<tr><td>Data contract</td><td colspan="2">Wejście: obiekt klasy SubjectCardSearchParameters
Wyjście:
Dto będące odwzorowaniem poniższych klas.
SubjectCard, FieldOfStudy, Faculty, Course, Class, SubjectEducationalEfect, Discipline oraz Employee
oraz klasa:<table><tr><td>SubjectCardSearchParameters</td></tr><tr><td>searchText: string
pageSize: integer
pageNumber: integer</td></tr></table></td></tr>
</table>

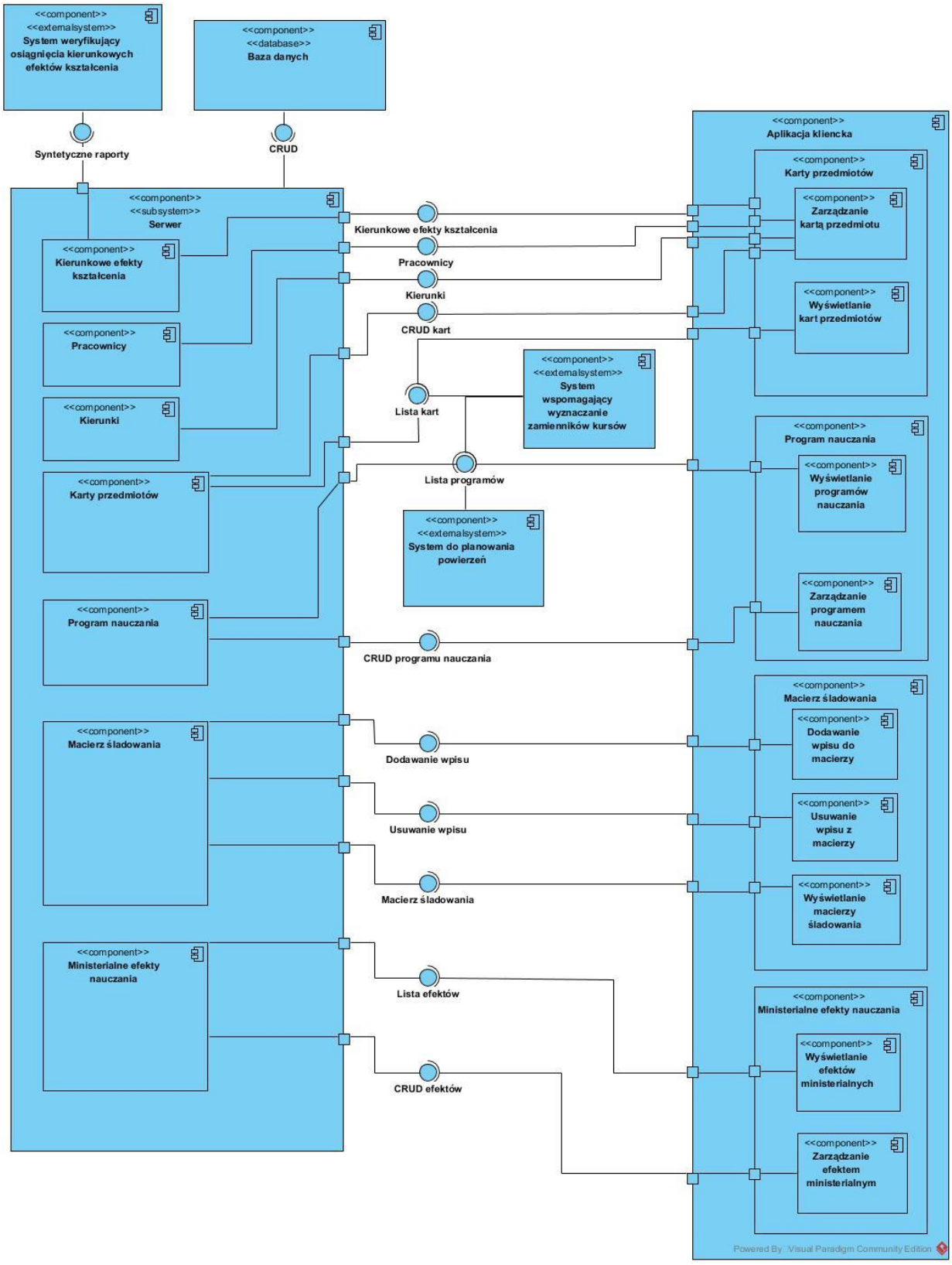
Does the interface manipulate on the sensitive data (RODO)?	Tak/Nie + jakie dane objęte GIODO/RODO
Middleware used	Brak
Initiating side	System wyznaczający zamienniki kursów
Communication model	Synchroniczny na żądanie użytkownika
Performance	100/h
Volumetry	6000/miesięcznie (około 200 dziennie)
Required accessibility	HIGH

Interface 3 - Wyszukiwanie programów nauczania

Description	Interfejs służący do wyszukiwania programów nauczania na podstawie zawieranego tekstu. Wyniki są stronnicowane.	
Status	Planowany	
	Source application	Target application
Application name	System do tworzenia i modyfikowania efektów kształcenia	System wyznaczający zamienniki kursów, System planowania powierzeń dla kierunków
Integration technology	REST/HTTPS	REST/HTTPS
Authentication mechanism	Brak	Brak

Data contract	Wejście: obiekt klasy StudyProgramSearchParameters Wyjście: Dto będące odwzorowaniem poniższych klas: StudyProgram, StudyPlan, Semester, FieldOfStudy, Faculty oraz klasa: **StudyProgramSearchParameters** searchText: string subjectCardId: string pageSize: integer pageNumber: integer
Does the interface manipulate on the sensitive data (RODO)?	Nie
Middleware used	Brak
Initiating side	System wyznaczający zamienniki kursów, System planowania powierzeń dla kierunków
Communication model	Synchroniczny na żądanie użytkownika
Performance	100/h
Volumetry	6000/miesięcznie (około 200 dziennie)
Required accessibility	HIGH

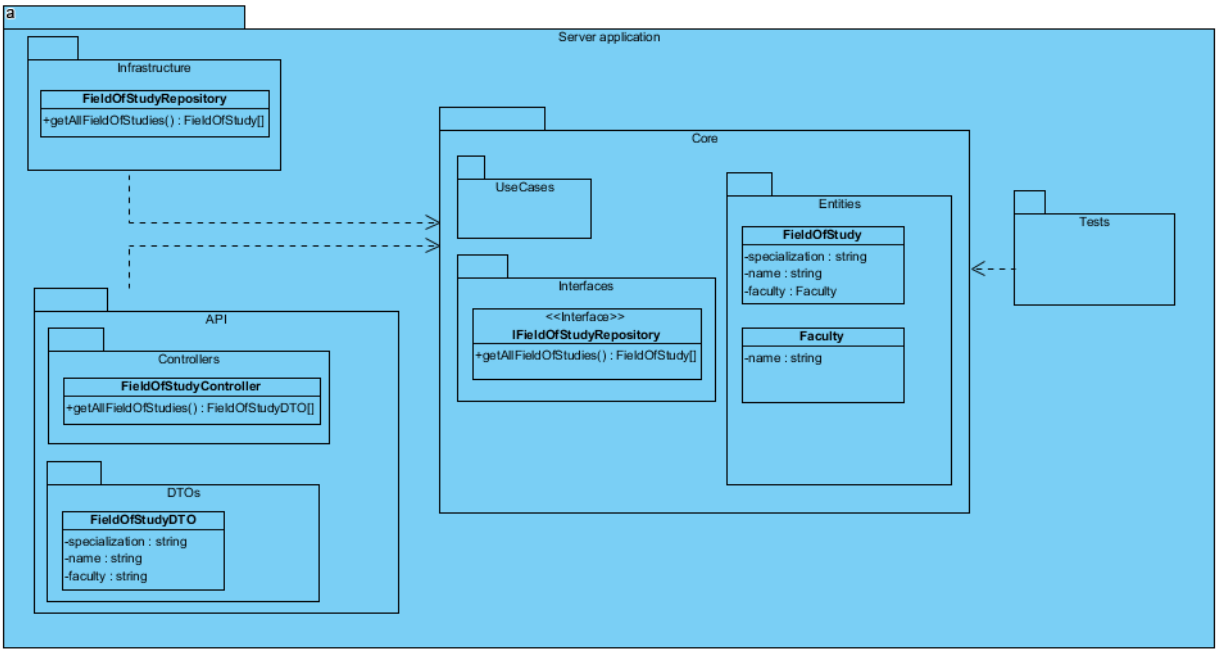
6. Perspektywa funkcjonalna (Functional view)



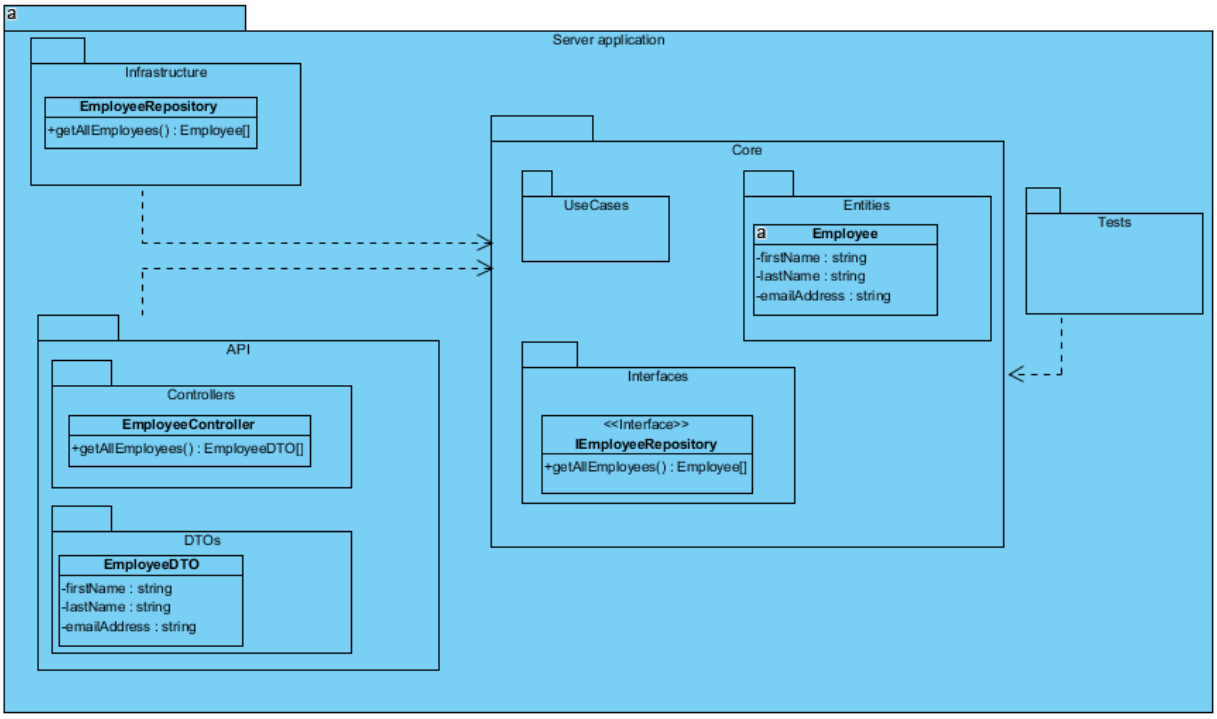
6.1 Diagramy pakietów dla wybranych komponentów

Poniżej zaprezentowano diagramy pakietów dla wybranych komponentów biorących udział w operacjach przedstawionych na diagramie przypadku użycia przedstawionym w punkcie 11.

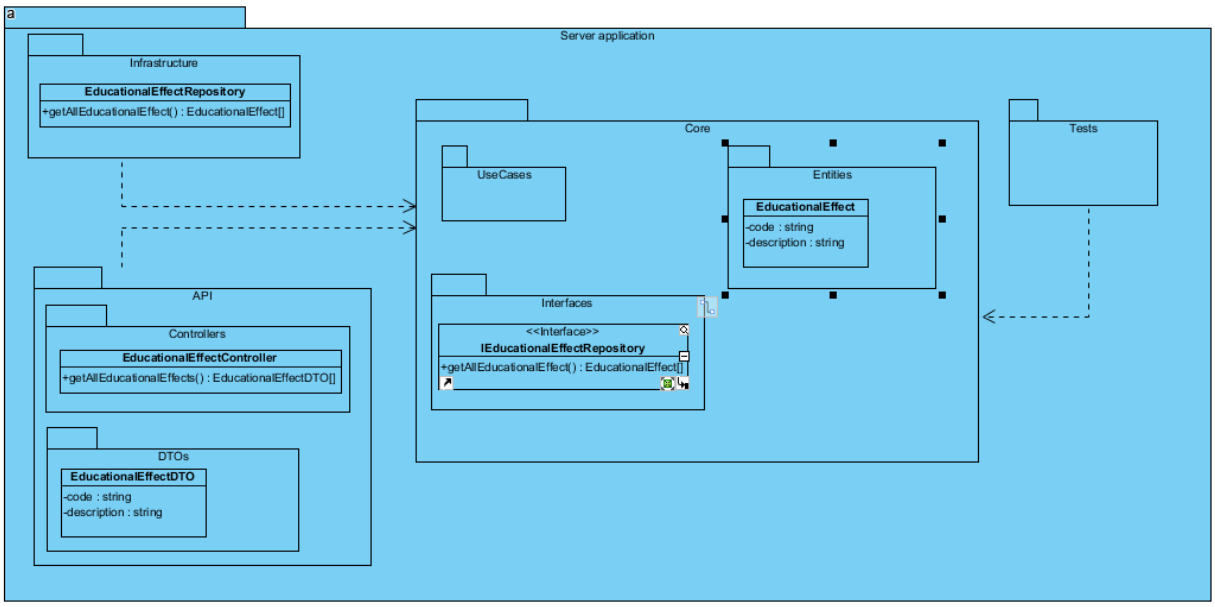
6.1.1 Komponent *„Kierunki”*



6.1.2 Komponent *„Pracownicy”*

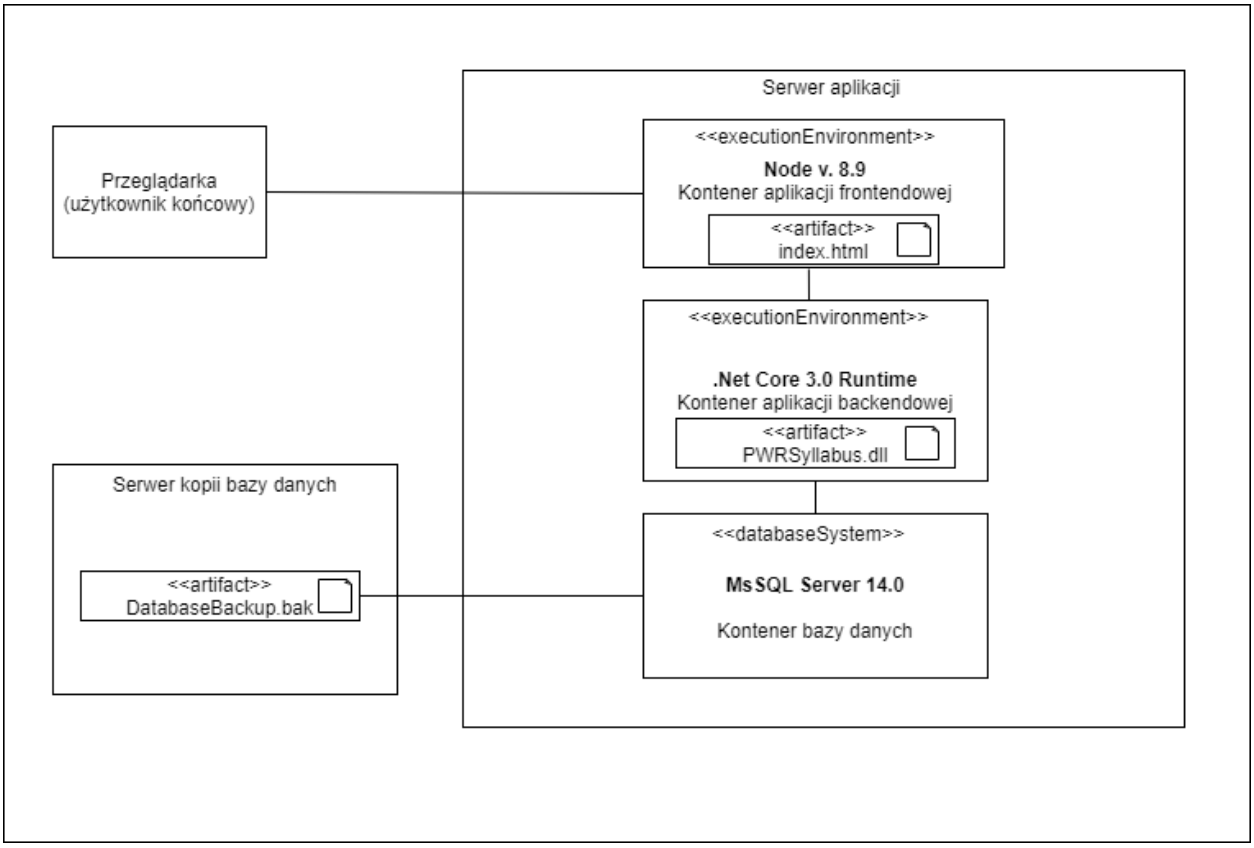


6.1.3 Komponent *„Pracownicy”*



7. Perspektywa wdrożeniowa (Deployment view)

System zostanie wdrożony przy użyciu systemu Kubernetes, służącego do zarządzania skonteneryzowanymi aplikacjami. Kontenery zostaną stworzone przy użyciu narzędzia Docker. Dzięki temu każda część aplikacji posiadać będzie własne, wyizolowane środowisko, w którym będzie działać. Pozwoli to na łatwiejsze zarządzanie nimi, zapewni skalowalność oraz przenośność systemu. Istnieje wiele możliwości konfiguracyjnych pozwalających na wdrożenie sprawnie działającego systemu. Poniższy diagram przedstawia logiczne rozmieszczenia wdrażanych części aplikacji.



Ze względu na ograniczenia finansowe projektu wszystkie części aplikacji uruchomione zostaną na jednej fizycznej maszynie, zostaną one jednak osadzone w osobnych kontenerach komunikujących się między sobą. Niewielkie wymagania aplikacji pod kątem liczby użytkowników sprawiają, że jedna fizyczna maszyna pozwoli spełnić wymagania projektowe.

7.1. Główny węzeł

Informacje ogólne	
Nazwa	Mario
Wirtualny	Nie
Data center	Nie
System operacyjny	Ubuntu 16.04 Server
Opis	Maszyna służąco zarówno do uruchomienia systemu Kubernetes oraz wdrożenia kontenerów z aplikacją backendową, frontendową oraz bazą danych.

Konfiguracja sprzętu	
Procesor	2x3.6Ghz
RAM	16GB
HDD	1x60GB, preferowany SSD

Konfiguracja oprogramowania	
Użytkownicy	Admin
Poziom pracy systemu, czy jest wymagane środowisko graficzne	Środowisko graficzne nie jest wymagane, wszystkie niezbędne operacje można wykonać z linii poleceń, a dzięki jego braku można ograniczyć zapotrzebowanie na pamięć operacyjną.

Dodatkowe pakiety spoza dystrybucji systemu	Kubernetes Docker

7.2. Kontenery

Konteneryzacja umożliwia odpowiedni poziom wyizolowania warstwy prezentacji, warstwy przetwarzania oraz bazy danych. Dzięki wydzieleniu części odpowiedzialnych za różne warstwy do osobnych kontenerów zmniejszy się ryzyko związane z zależnością pomiędzy komponentami oraz pomiędzy komponentami a maszyną fizyczną, na którą kontenery zostaną wdrożone. Poniższe tabele opisują kontenery, które wspólnie tworzyć będą aplikację dostarczoną w części implementacyjnej przedsięwzięcia.

7.2.1. Warstwa prezentacji

Informacje ogólne	
Nazwa	PrincessPeach
Wirtualny	Tak
Opis	Kontener zawierający aplikację z warstwą prezentacji aplikacji. Odpowiedzialny za udostępnienie warstwy prezentacji użytkownikom końcowym.
Ograniczenia sprzętowe	
Procesor	2x1.6GHz
RAM	4GB
HDD	16GB

7.2.2. Warstwa przetwarzania

Informacje ogólne	
Nazwa	Luigi

Wirtualny	Tak
Opis	Kontener obsługujący warstwę przetwarzania aplikacji, odpowiedzialną za udostępnianie API dla aplikacji frontendowej, przetwarzanie oraz komunikację z warstwą persystencji.
Ograniczenia sprzętowe	
Procesor	2x1.6GHz
RAM	6GB
HDD	16GB

7.2.3. Baza danych

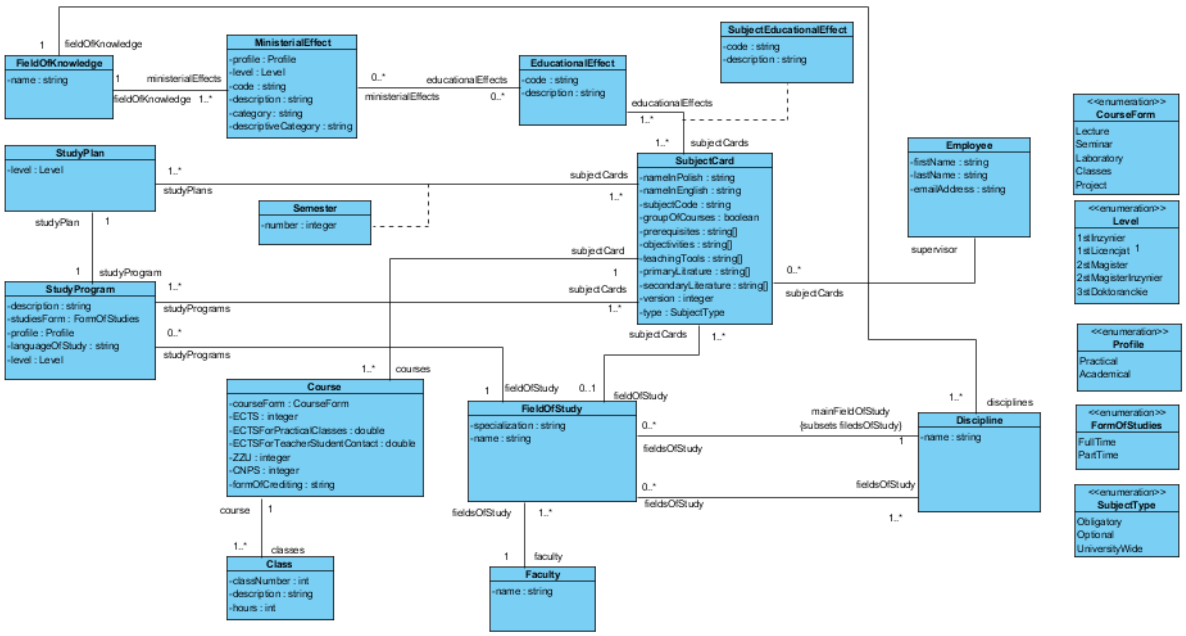
Informacje ogólne	
Nazwa	Yoshi
Wirtualny	Tak
Opis	Kontener zawierający serwer bazy danych oraz instancję bazy.
Ograniczenia sprzętowe	
Procesor	2x1.6GHz
RAM	4GB
HDD	16GB

7.3. Użytkownik końcowy

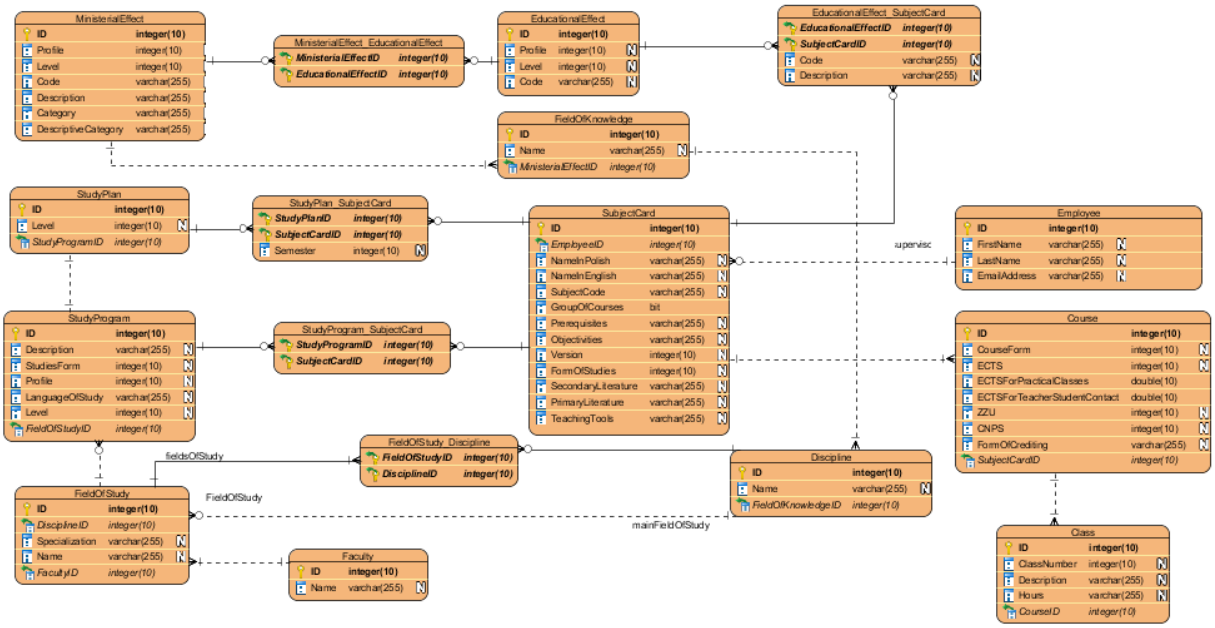
Użytkownik końcowy będzie musiał posiadać komputer z przeglądarką internetową oraz dostępem do Internetu. Ze względu na brak dużego zapotrzebowania na zasoby obliczeniowe nie zostanie wyspecyfikowana wymagana konfiguracja maszyny użytkownika. Mimo, że planowana aplikacja jest aplikacją webową, nie jest planowane wspieranie wersji mobilnych przeglądarek. Spowodowane jest to skomplikowanym interfejsem, który na małym ekranie byłby trudny lub wręcz niemożliwy do przedstawienia.

8. Perspektywa informacyjna (Information view)

8.1. Model informacyjny



8.2. Projekt bazy danych (model ERD)



Database General information (one table per server)	
SID/Service Name	Baza danych
Server name	db-study-programs
Port	1433
Type	Microsoft SQL Server 2017 (14.00)
Character codding	Standardowe
Description	Relacyjna baza danych systemu wspomagającego tworzenie i modyfikowanie programów kształcenia, przechowująca wszystkie informacje na temat programów studiów, kursów oraz prowadzących zajęcia i wynikających z tego procesu encji.
Technologies	Domyślne, brak potrzeby stosowania dodatkowych mechanizmów.

Backup
Proces tworzenia kopii bazy danych będzie uruchamiany raz w tygodniu lub na żądanie administratora. Kopie bazy danych będą przetrzymywane na serwerze o innej lokalizacji fizycznej niż ta na której umieszczona zostanie baza danych. Backupy będą przechowywane przez 3 lata, po upływie tego czasu możliwe jest przeniesienie ich na inny fizyczny nośnik lub usunięcie.

8.2.1. Schematy danych

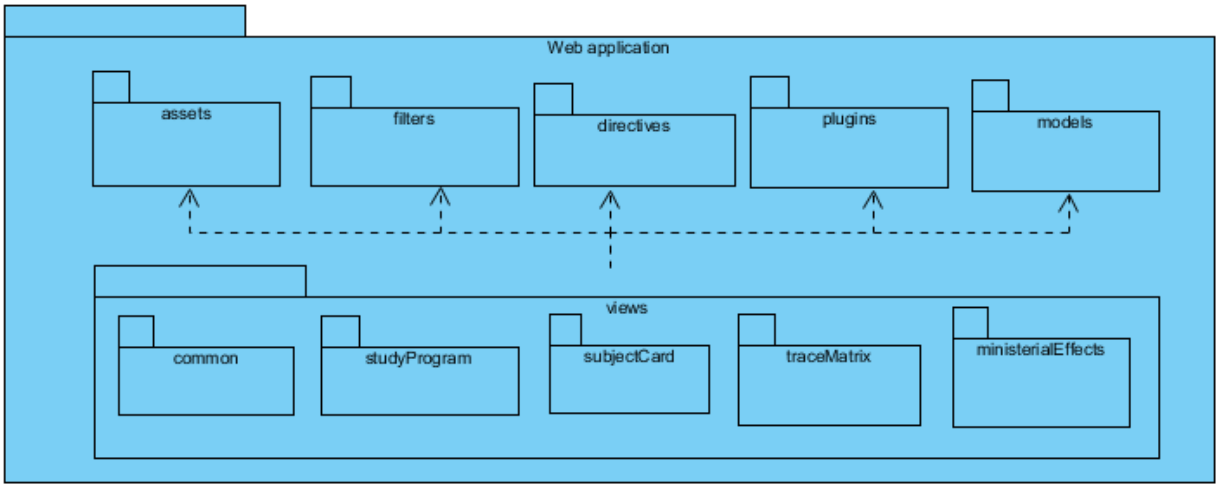
Schema information	
Schema name	Studies
Initial capacity	100MB
Capacity increment (year)	20MB
Others	Brak

9. Perspektywa wytwarzania (Development view)

9.1 Aplikacja webowa

Aplikacja kliencka typu Single Page Application stworzona przy użyciu frameworka Vue w wersji 2.6. Struktura pakietów jest następująca:

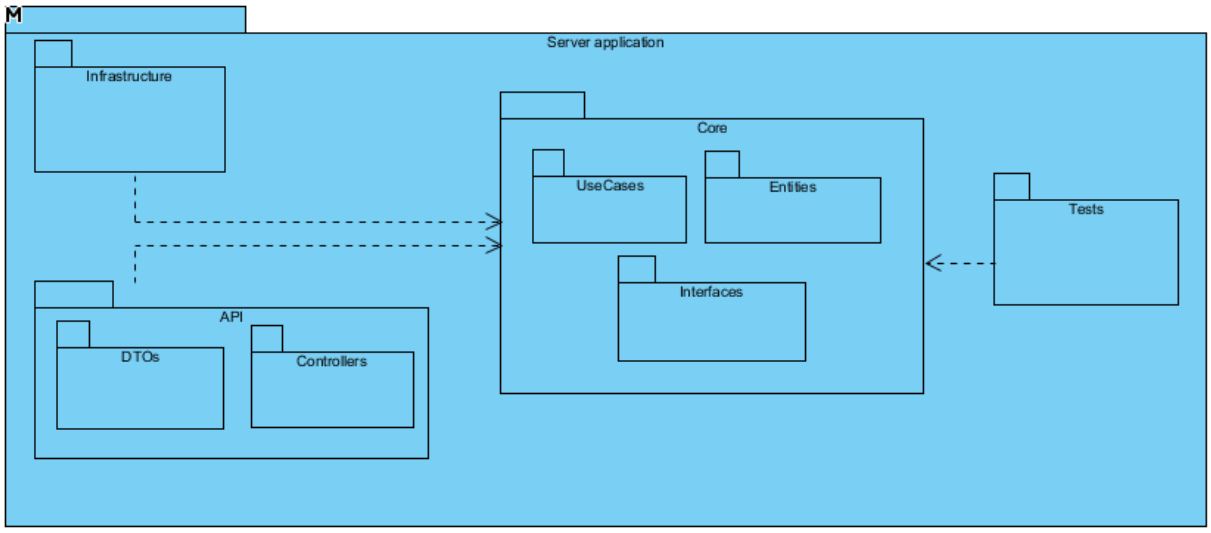
- assets - zasoby statyczne aplikacji takie jak grafiki, czcionki, tłumaczenia
- filters – implementacja filtrów umożliwiających scentralizowanie funkcji transformujących wyświetlane dane (np. wspólny format dat)
- directives - część kodu implementująca niestandardowe dyrektywy umożliwiające kontrolowanie zachowania komponentów poprzez użycie atrybutów w szablonie HTML.
- plugins – implementacja oraz konfiguracja wtyczek, czyli zestawów różnego rodzaju elementów udostępnianych przez API Vue oraz przeglądarki internetowe (np. klient http, biblioteka komponentów graficznych, biblioteka walidacji)
- views - pakiet zawierający implementację poszczególnych widoków na które składają się komponenty, wyróżniono pakiet common zawierający komponenty używane w wielu widokach



9.2 Aplikacja serwerowa

Web API napisane w języku C# przy wykorzystaniu frameworka .NET Core. Zastosowano architekturę opartą o rozwiązania opisane przez Roberta Martina w książce “Czysta architektura” . Aplikacja składa się z 4 projektów:

- Core - projekt zawierający logikę biznesową aplikacji, definiujący interfejsy implementowane przez projekt Infrastructure, które pozwalają na porozumiewanie się z zewnętrznymi systemami oraz warstwą persystencji. Nie posiada zależności do żadnego z projektów.
- Infrastructure - projekt implementujący interfejsy używane w celu komunikacji z zewnętrznymi zasobami oraz bazą danych zdefiniowanymi w projekcie Core.
- API - projekt zawierający implementację punktów końcowych, wywołujący odpowiednie metody zawarte w projekcie Core w celu obsłużenia zapytań.
- Tests – projekt zawierający testy jednostkowe. Testowana będzie część aplikacji zawierającą logikę biznesową zawartą w projekcie Core.



11. Realizacja przypadków użycia (Use-case realizations)

Przykładowy przypadek użycia do dodawania karty przedmiotu został przedstawiony na poniższym diagramie. Pominięto szczegóły implementacyjne w celu zwiększenia czytelności diagramu.