System Hospitacji Wydziałowych
Architecture Notebook

1. Cel (Purpose)

Celem dokumentu jest opis decyzji, ograniczeń oraz uzasadnień decyzji dla istotnych elementów systemu, które mają istotny wpływ na projekt oraz implementację systemu.

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

W trakcie przeprowadzonej analizy biznesowej oraz systemowej zawartej w modelu ***model.vpp*** zdefiniowana została dziedzina oraz wymagania, które powinna spełniać architektura systemu hospitacji. Wybrane wymagania zaprezentowane są na diagramie ***[RD] Wybrane wymagania***. Sprowadzają się one do realizacji jednej dużej funkcjonalności jaką jest możliwość opracowania ramowego planu hospitacji przez pełnomocników dziekana do spraw kierunku oraz ich sekretarzy.

Szczegóły wymagań znajdują się w dokumencie, jednak ogólny zarys koniecznych do zrealizowania funkcjonalności prezentuje się następująco:

- Zarządzanie ramowym planem hospitacji
- Podgląd osób rekomendowanych do hospitacji
- Definicja nowej hospitacji dla prowadzącego zajęcia
    - Wybór kursu do hospitacji
    - Podgląd rekomendowanego zespołu hospitującego
    - Wybór zespołu hospitującego z listy potencjalnych hospitujących

Wymagania jakościowe

- Ochrona przed nieuprawnionym dostępem
  System zapewnia ochronę przed nieuprawnionym dostępem do wyników oraz planów hospitacji, oraz do danych pracowników zawartych w systemie.
- System dostępny w wielu wersjach językowych
  System dostępny jest w 2 wersjach językowych (angielski oraz polski) z możliwością rozszerzenia.
- Wydajność systemu
  Średnia odpowiedź systemu poniżej 5s przy obciążeniu 200 użytkowników oraz średniego czasu poniżej 7s przy podwojeniu liczby użytkowników do 400.

Ograniczenia

- System wymaga stałego dostępu do Internetu
- System jest dostępny w przeglądarkach Google Chrome oraz Opera

W celu realizacji funkcjonalności systemu konieczna jest również integracja z systemami zewnętrznymi. Punkty integracji prezentują się następująco:

- System kadrowy
    - Pobranie danych wszystkich pracowników. Możliwe wymuszenie synchronizacji na żądanie użytkownika
    - Wprowadzenie danych nowego semestru
- System zapisów
    - Pobranie wszystkich kursów w semestrze
    - Pobranie prowadzących bez konfliktu zajęć z kursem

- System uwierzytelniania - Firebase
    - Logowanie
    - Weryfikacja tokenu użytkownika

3. Decyzje i uzasadnienia (Decisions and justifications)

Cel	Sposób osiągnięcia
1. System zapewnia ochronę przed nieuprawnionym dostępem do wyników oraz planów hospitacji, oraz do danych pracowników zawartych w systemie	A. Uwierzytelnianie - użycie Firebase jako dostawcy tożsamości w oparciu o OAuth 2.0 B. Autoryzacja a. przydzielenie ról użytkownikom w oparciu o model RBAC b. korzystanie z tokenów Firebase w celu przesyłania informacji o rolach użytkownika c. kontrola nad dostępem uprawnionych osób do zasobów przy pomocy Spring Security C. Szyfrowanie połączenia z serwerem D. Ochrona przed atakami SQL Injection oraz XSS E. Logowanie transakcji prowadzonych przez użytkowników
2. Średnia odpowiedź systemu poniżej 5s przy obciążeniu 200 użytkowników oraz średniego czasu poniżej 7s przy podwojeniu liczby użytkowników do 400	A. Zrównoleglenie obciążenia pomiędzy niezależne skalowalne jednostki B. Obsługa żądań w innej puli wątków niż pula wątków przyjmująca żądania i zrównoleglenie obliczeń C. Optymalizacja komunikacji z bazą danych z wykorzystaniem puli połączeń (*Connection Pooling)* D. Optymalizacja zapytań do bazy danych a. Cache dla zapytań b. Ograniczenie żądanych danych E. Optymalizacja żądania kierowanych z aplikacji klienckiej do serwera a. Minimalizacja przesyłanych danych b. Stronicowanie wyników
3. System dostępny jest w 2 wersjach językowych (angielski oraz polski) z możliwością rozszerzenia	A. Tłumaczenie danych statycznych interfejsu użytkownika dzięki oddzielnym plikom z tłumaczeniami dla różnych języków B. Przechowywanie w bazie danych tłumaczeń dla danych ładowanych dynamicznie

4. Mechanizmy architektoniczne (Architectural Mechanisms)

4.1 Mechanizmy zapewniające ochronę przed nieuprawnionym dostępem do danych

A. Uwierzytelnianie - użycie Firebase jako dostawcy tożsamości w oparciu o **OAuth 2.0**

W celu uwierzytelnienia użytkowników w systemie zostanie zastosowany serwis Firebase. Jest to stworzona przez Google platforma udostępniająca wiele funkcjonalności takich jak system uwierzytelniający, baza danych czasu rzeczywistego, hosting aplikacji webowych, a także składowanie danych. W aplikacji zostanie wykorzystany jedynie system uwierzytelniający, który korzysta z protokołu **OAuth 2.0**. Firebase jest platformą, która skaluje się w zależności od zapotrzebowania. Gdy liczba aktywnych użytkowników wzrasta, serwis przeznacza dodatkowe zasoby, aby zapewnić jak najlepszą wydajność. Należy zaznaczyć, że uwierzytelnianie w Firebase jest darmowe niezależnie od liczby użytkowników systemu.

Integracja aplikacji z serwisem Firebase zostanie przeprowadzona przez **Firebase SDK**, które dostarczane jest na różne platformy przez Google. Zostanie wykorzystany jednak framework **AngularFire2**, który jest nakładkę na Firebase SDK wprowadzającą elementy reaktywnego programowania dzięki zastosowaniu **RxJs.** Aby umożliwić połączenie aplikacji z Firebase, należy dodać odpowiednio wygenerowany plik JSON, który zawiera konieczne do integracji dane min. klucz API, endpointy, czy inne właściwości jednoznacznie identyfikujące projekt. Uwierzytelnienie będzie odbywać się przez **email** oraz **hasło**.

B. Autoryzacja - przydzielenie ról użytkownikom w oparciu o model RBAC. Korzystanie z tokenów Firebase w celu przesyłania informacji o rolach użytkownika. Kontrola nad dostępem uprawnionych osób do zasobów przy pomocy Spring Security

Jako technika kontroli dostępu zostanie wykorzystany RBAC (role-based access control), który jest mechanizmem polegającym na rolach. Role, które będą istnieć w aplikacji to:

- Dziekan
- Pełnomocnik Kierunku
- Kierownik Katedry
- Pracownik
- Biuro Dziekana
- Biuro Pełnomocnika

Role oraz inne potrzebne do autoryzacji informacje (rodzaj prowadzącego zajęcia, klucze powiązań z klasami w systemie), zostaną przypisane do użytkownika w trakcie tworzenia konta. Dane te zostaną ustawione za pomocą mechanizmu **claims**, który jest dostarczany przez Firebase. Wprowadzone do **claims** dane mogą zostać uzyskane poprzez zdekodowanie tokenu dostarczanego przez Firebase bo uwierzytelnieniu użytkownika. Token ten to JSON Web Token (**JWT**), który będzie w swoim polu **payload** zawierał zdefiniowane **claims**. Przykładowe claims dla prowadzącego zajęcia bez dodatkowych obowiązków, prezentują się następująco:

```
{
   "claims":{
      "role":0,
      "employeeId":"4567890sdfghjkiuytrwer45678998765rfghj"
   }
}
```

Do kontroli nad przepływem danych z serwera zostanie wykorzystany moduł platformy **Spring** dostarczający metody autentykacji i autoryzacji. **Spring Security** pozwala na ochronę przed atakami takimi jak session fixation, clickjacking, czy cross site request forgery. Aplikacja będzie operować na danych wrażliwych, co podnosi wagę ograniczenia dostępu do poufnych informacji. Każdy endpoint będzie zabezpieczony przed nieuprawnionym dostępem. Wszystka żądanie kierowane do serwera będą musiały zawierać nagłówek **Authorization** z tokenem Firebase. Następnie token ten będzie weryfikowany poprzez **Firebase Admin SDK.** Dane użytkownika będą dekodowane z tokenu i na ich podstawie będą udostępniane odpowiednie zasoby. W przypadku błędów weryfikacji żądanie zakończy się błędem.

C. Szyfrowanie połączenia z serwerem

Dane przesyłane z klienta do serwera mogą być danymi wrażliwymi jak np. dane prowadzących zajęcia. Te dane powinny być dodatkowo zabezpieczone przed wyciekiem. Z tego powodu komunikacja z serwerem będzie odbywać się z wykorzystaniem protokołu **HTTPS**. Aby zastosować to rozwiązanie, do konfiguracji Springa będzie potrzebny certyfikat. Na potrzeby developerskie może być on wygenerowany. Jednak w przypadku wersji produkcyjnej certyfikat musi być dostarczony z Certificate Authority.

D. Ochrona przed atakami SQL Injection oraz XSS

System będzie przechowywał dane wrażliwe, dlatego ważna jest ochrona przed popularnymi atakami wykradającymi dane z aplikacji. Do obrony przed atakami SQL Injection zostanie wykorzystany moduł **Spring Data** umożliwiający bezpieczne pisanie zapytań do bazy danych**.** W razie wystąpienia błędów będą one logowane oraz obsługiwane po stronie backendu. Jeśli backend nie będzie w stanie obsłużyć błędu, do klienta zwróci tylko krótką informację o kodzie błędu. Ze względów bezpieczeństwa warto trzymać w tajemnicy informacje o dokładnej strukturze projektu.

XSS jest to najbardziej znany sposób ataku obok SQL Injection. Nazwa "cross site" pochodzi od sposobu, w jaki

kod jest przekazywany pomiędzy stronami, od niebezpiecznego źródła danych do atakowanej przeglądarki. Zabezpieczenie przed tego rodzaju atakami odbędzie się w głównej mierze dzięki wbudowanym mechanizmom ochronnym w Angulara. Dodatkowo zastosowany zostanie **Offline Template Compiler**, aby uniknąć ataku template injection.

E. Logowanie transakcji prowadzonych przez użytkowników

System będzie zawierał logowanie akcji wykonanych przez użytkownika. Do tworzenia logów zostanie użyty logger dostarczony przez bibliotekę **SLF4J**. Aby rozróżnić powód logowania informacji, zostaną użyte poziomy logów takie jak: *debug*, *error*, czy *info*. Dzięki logom stanie się możliwe śledzenie akcji użytkowników oraz identyfikacja podejrzanych zachowań. Logowanie błędów, które wystąpiły w aplikacji jest również ważne, aby szybko reagować na problemy i mieć więcej informacji na ich temat. W *Spring Boot* zachowanie logów będzie konfigurowalne w następujący sposób:

- nazwa pliku z logami - *applicationLog.log*
- maksymalna liczba historycznych plików z logami - *10*
- maksymalny rozmiar pliku z logami - *50 MB*
- wygląd pojedynczego logu - *2012-04-26 14:54:38,461 DEBUG [main] com.foo.App - Hello*
- ścieżka do katalogu z logami: */logs/evaluation-system*

4.2 Mechanizmy wspomagające wydajność aplikacji

A. Zrównoleglenie obciążenia pomiędzy niezależne skalowalne jednostki

Zastosowana zostanie architektura 4-poziomowa typu klient-serwer, aby zapewnić skalowanie systemu, wysokie bezpieczeństwo i łatwe utrzymanie. Komponenty warstwy prezentacji i logiki biznesowej mogą być rozmieszczone i skalowalny niezależnie, a przydzielaniem zadań kierował będzie load balancer. Wydzielone komponenty architektury prezentują się następująco:

- **Poziom klienta - Przeglądarka internetowa**
  Przeglądarka internetowa, która wyświetla HTML oraz wykonuje JavaScript serwowany przez aplikację webową systemu. Zapewnione jest wsparcie dla przeglądarki Google Chrome w wersji desktopowej.
- **Poziom prezentacji - Aplikacja webowa**
  Aplikacja serwująca kontent klientowi i zaimplementowana we frameworku **Angular**. Odpowiedzialnością tej warstwy jest prezentacja graficznego interfejsu użytkownikowi przy komunikacji z aplikacją serwerową w celu pozyskania informacji do wyświetlenia.
- **Poziom logiki biznesowej - Aplikacja serwerowa**
  Aplikacja udostępniająca REST API dla aplikacji webowej i zaimplementowana w frameworku **Spring**. Odpowiedzialnością tej warstwy jest realizacja logiki biznesowej aplikacji, dbanie o ochronę przed nieuprawnionym dostępem, udostępnienie danych w formie REST API, komunikowanie z bazą danych (**Hibernate**) oraz z zewnętrznymi systemami.
- **Poziom składowania danych - Baza danych**
  Odpowiedzialnością tej warstwy jest trwałe składowanie danych. Wykorzystana zostanie baza danych **PostgreSQL**.

Skalowalność na poziomie prezentacji oraz logiki biznesowej będzie realizowana przy pomocy mechanizmu horyzontalnej skalowalności instancjami *(Horizontal Pod Autoscaler)* w klastrze Kubernetes. Cały system zostanie uruchomiony w klastrze Kubernetes, który dynamicznie przydzielając zasoby będzie dbał o odpowiednią dostępność, skalowalność oraz rozłożenie pracy pomiędzy niezależne jednostki przetwarzania bazując na obciążeniu procesorów poszczególnych jednostek.

B. Obsługa żądań w innej puli wątków niż pula wątków przyjmująca żądania i zrównoleglenie obliczeń

Zastosowane zostanie asynchroniczne przetwarzanie żądań, aby odblokować pulę wątków przyjmującą żądania. W tym celu zostanie wykorzystany ***DeferredResult***, który pozwala zwrócić rezultat, który jeszcze nie został obliczony. Pozwala to odblokować wątki przyjmujące żądania, aby nowe żądania od klientów mogły zostać przyjęte, a przetwarzanie odbywało się w tle. Pozwala to na dobre skalowanie aplikacji w przypadku dużych obciążeń.

W celu zrównoleglania obliczeń wykonywanych na serwerze oraz odciążenia wątków przyjmujących żądania, wykorzystana zostanie biblioteka ***RxJava 2*.** Rozwiązanie to zapewnia w bardzo prosty sposób realizację zmian wątków oraz wprowadzania możliwości programowania reaktywnego, pozwalając uniknąć powszechnie znanego problemu jakim jest ***callback hell.*** Będzie ona spajać warstwy architektury. Aby nie tworzyć nowego wątku dla każdego żądania, zostanie wykorzystana pula wątków dostępna w ***Schedulers#computation*** (z pakietu ***io.reactivex***), której wielkość jest sterowana możliwościami maszyny.

C. Optymalizacja komunikacji z bazą danych z wykorzystaniem puli połączeń (Connection Pooling)

Większość żądań do serwera będzie wiązało się z komunikacją z bazą danych. Może stać się to wąskim gardłem, dlatego zostanie zastosowana technika ***connection pooling***. Jako, że nawiązywanie połączenia z bazą danych zazwyczaj zajmuje bardzo dużą część czasu komunikacji, połączenia będą nawiązywane i utrzymywane w puli. Aplikacja pyta o połączenie, wykorzystuje je, a następnie zwraca do puli. Pomoże to zdecydowanie przyspieszyć zapytania kierowane do bazy.

Zostanie wykorzystana implementacja connection poola przy wykorzystaniu **Apache Commons DBCP**. Parametry konfiguracji prezentują się następująco:

- początkowy rozmiar *(initialSize) - 8*
- maksymalna liczba aktywnych połączeń *(maxTotal) - 20*
- maksymalna liczba w stanie idle *(maxIdle) - 20*
- minimalna liczba w stanie idle *(minIdle) - 0*

D. Optymalizacja zapytań do bazy danych

Przeważającą część żądań w aplikacji stanowią odczyty. Dlatego szczególną uwagę należy poświęcić optymalizacji zapytań kierowanych do bazy, które będą generowały największe opóźnienia. Mechanizmy temu służące prezentują się następująco:

a. Cache dla zapytań. Jako że dane w aplikacji charakteryzują się małą zmiennością, istnieje możliwość cachowania zapytań kierowanych do bazy danych. Dzięki temu można zdecydowanie przyspieszyć pobieranie tych samych danych. W tym celu wykorzystane zostaną mechanizmy wbudowane w Spring w postaci adnotacji JSR 107 (JCache) w połączeniu z implementacją API dostarczaną przez bibliotekę **EHcache 3.0**. Adnotacje powinny być użyte do cachowania wyników zapytań Hibernate.

   Zapytania oraz encje zalecane do cachowania to: ***prowadzący zajęcia, rekomendowany zespół hospitujący, polecani hospitujący, prowadzący z kursami bez konfliktu zajęć***. Każdy przypadek cachowania powinien zostać potraktowany niezależnie i może wymagać zmiany parametrów, jednak zalecane jest korzystanie z *cache-template* o następujących parametrach składowania
   - heap size - *1000 wpisów*
   - off heap size - *10 MB*
   - expiry - time-to-idle - *5 minut*

b. Ograniczenie żądanych danych. Należy kierować zapytania zwracające jedynie potrzebne w danej chwili informacje, aby ograniczyć wymianę danych z bazą danych, co przełoży się na szybszą komunikację. Odpytując bazę danych nie należy zwracać całych encji JPA, a tworzyć nowe obiekty, tak aby pobierać tylko informacje, które w danej chwili są potrzebne.

E. Optymalizacja żądania kierowanych z aplikacji klienckiej do serwera

Jako, że cały ruch do serwera generowany jest przez aplikację kliencką, istnieje możliwość optymalizacji tych zapytań. Mechanizmy ku temu służące to:

a. Minimalizacja przesyłanych danych. Podstawową techniką przyspieszającą komunikację z serwerem jest ograniczenie liczby przesyłanych danych, do jedynie tych, które są w danym momencie potrzebne. Będzie ku temu służyć odpowiednia definicja kontraktu REST API.

b. Stronicowanie wyników. Dużą część kontentu aplikacji stanowią bardzo duże listy. Aby

ograniczyć liczbę wyświetlanych elementów zostanie wykorzystane stronicowanie wyników. Wyświetlanie ograniczy się do konkretnej liczby wyników na stronie i pobierania ich leniwie, kiedy są potrzebne.

Aby zaimplementować ten wzorzec zostanie wykorzystana biblioteka **RxJs**. Należy zasubskrybować się do żądania zmiany strony przez użytkownikach, następnie użyć operatora *switchMap* do wykonania żądania do serwera i ostatecznie wyświetlić pobrane wyniki. Paginacja po stronie serwerowej zostanie zaimplementowana z wykorzystaniem mechanizmu wbudowanego w Spring do pobierania ograniczonej liczby rezultatów z bazy danych - *PagingAndSortingRepository*.

4.3 Dostęp do dwóch wersji językowych

A. Tłumaczenie danych statycznych interfejsu użytkownika dzięki oddzielnym plikom z tłumaczeniami dla różnych języków

Aplikacja będzie posiadała statyczne części strony takie jak menu główne, nazwy pól w formularzu do utworzenia nowej hospitacji, nagłówki tabel i inne. Użytkownik w każdej chwili korzystania z aplikacji będzie miał swobodny dostęp do zmiany języka aplikacji z poziomu interfejsu użytkownika. Translacje będą przeładowywane dynamicznie bez potrzeby odświeżenia strony.

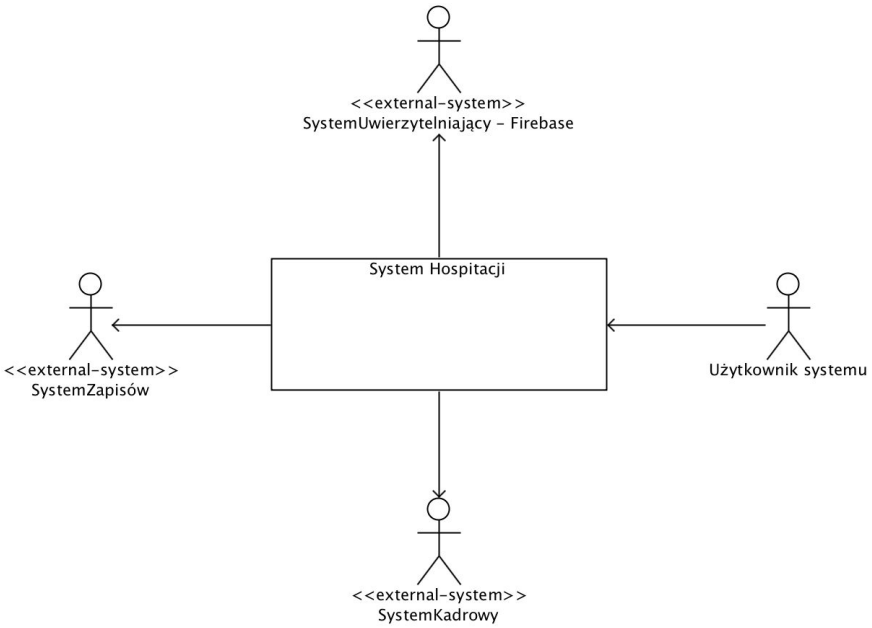
Do wydajnego obsłużenia translacji statycznych danych strony zostanie wykorzystany moduł Angulara o nazwie **ngx-translate.** Definicja tłumaczeń polega na zdefiniowaniu dla każdego języka, pliku w formacie **JSON** zawierającego klucze i odpowiadające im wartości dla konkretnego języka. Rozwiązanie zapewnia skalowalność, ponieważ dodanie nowego języka do aplikacji to stworzenie nowego pliku z odpowiednimi wartościami dla wszystkich zdefiniowanych kluczy. Do tłumaczenia dat na lokalny format używany będzie wbudowany moduł Angulara o nazwie **common**. W aplikacji będą wspierane dwie wersje językowe - **polska** i **angielska.**

B. Przechowywanie w bazie danych tłumaczeń dla danych ładowanych dynamicznie

Aplikacja w bazie danych będzie posiadała tłumaczenia danych, które są możliwe w kilku językach, jak np. nazwy kursów. Kursy w danym języku powinny wyświetlać się w zależności od wybranego przez użytkownika języka. Baza danych musi posiadać odpowiedni schemat zawierający tłumaczenia danych ładowanych dynamicznie. Aby uzyskać tłumaczenia do części interfejsu użytkownika, aplikacja kliencka będzie musiała przekazywać, w każdym żądaniu do serwera, parametr mówiący o aktualnie wybranym języku.

5. Perspektywa kontekstowa (Context view)

5.1 Diagram

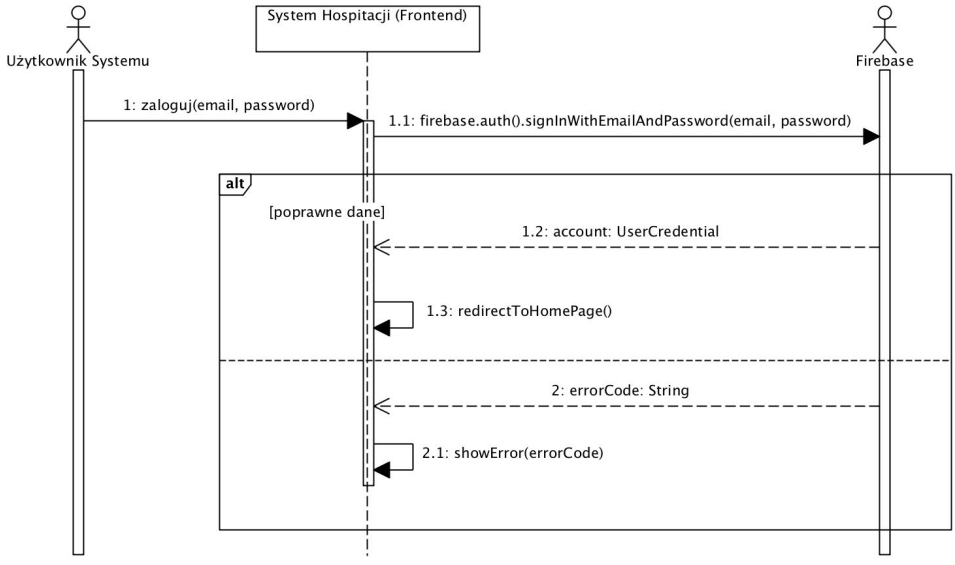


PSM/Perspektywy/Kontekstowa/Diagram/Perspektywa kontekstowa

5.2 Scenariusze interakcji (Interaction scenarios)

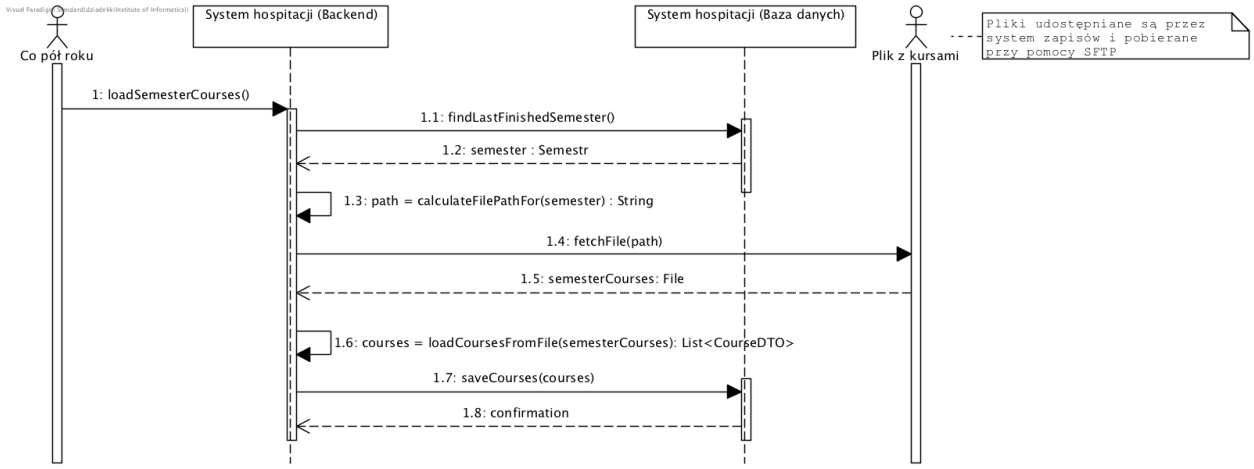
Dla każdego rodzaju zastosowanej integracji został przedstawiony przykładowy schemat integracji w postaci diagramu sekwencji.

5.2.1 Uwierzytelnianie



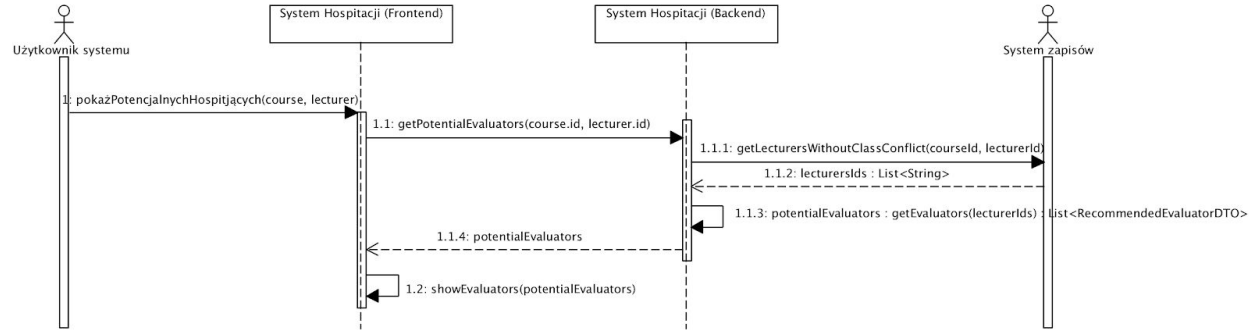
PSM/Perspektywy/Kontekstowa/Scenariusze/Uwierzytelnianie

5.2.2 Wczytanie kursów z semestru



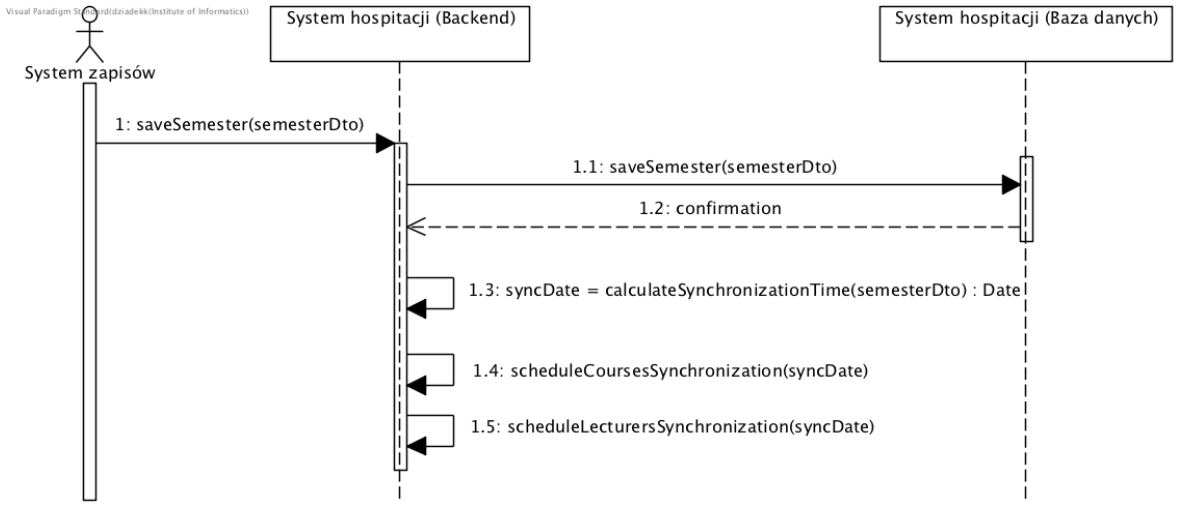
PSM/Perspektywy/Kontekstowa/Scenariusze/Wczytanie kursów z semestru

5.2.3 Pobranie listy wszystkich prowadzących bez konfliktu zajęć z kursem



PSM/Perspektywy/Kontekstowa/Scenariusze/Udostępnienie prowadzących bez konfliktu zajęć z kursem

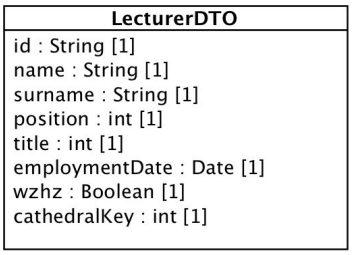
5.2.4 Wprowadzenie danych nowego semestru



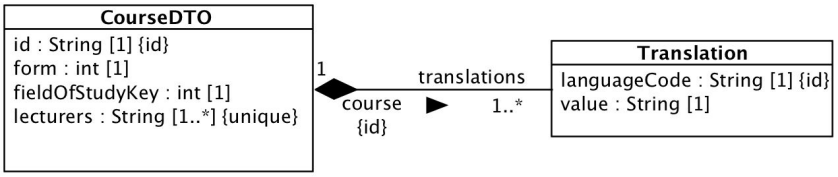
PSM/Perspektywy/Kontekstowa/Scenariusze/Wprowadzenie danych nowego semestru

5.3 Interfejsy integracyjne (Integration interfaces – logical level)

5.3.1 Interface 1 - Udostępnienie danych wszystkich prowadzących zajęcia

Description	Interfejs udostępniający dane pracowników wydziału, operujący na wrażliwych danych, które będą ładowane przed rozpoczęciem semestru oraz na żądanie osób przygotowujących harmonogram hospitacji	
Status	Planowany	
	Source application	Target application
Application name	System hospitacji - Backend	System kadrowy
Integration technology		SFTP
Authentication mechanism		SSH
Data contract	**Wejście:** brak **Wyjście:** lista prowadzących zajęcia **Klasa:**  **Format:** plik JSON **Przykład:** `[ { "id":"unique_id", "name":"Kamil", "surname":"Dziadek", "position":0, "title":0, "employmentDate":"2017-01-01", "wzhz":true, "cathedralKey":"K3" }, ... ]`	
Does the interface manipulate on the sensitive data (RODO)?	Tak	
Middleware used	Brak	
Initiating side	System hospitacji	
Communication model	Synchroniczny na żądanie użytkownika oraz Asynchroniczny wyzwalany zdarzeniem czasowym	
Performance	Automatycznie 1 / pół roku Użytkownik - 10 / dzień	
Volumetry	Liczba wywołań: 600 / rok (10 użytkowników, codziennie przez miesiąc, 2 miesiące w roku - w trakcie definiowania planów) Rozmiar danych: każde wywołanie to około 400 obiektów ProwadzącyZajęcia	
Required accessibility	LOW	

5.3.2 Interface 2 - Udostępnienie wszystkich kursów z danego semestru

Description	Interfejs udostępniający wszystkie kursy z danego semestru, które będą ładowane przed rozpoczęciem semestru. Pobieranie kursów odbywa się w celach archiwizacji (aby ustalić rekomendacje) oraz na potrzeby definiowania hospitacji w obecnym semestrze	
Status	Planowany	
	Source application	Target application
Application name	System hospitacji - Backend	System zapisów
Integration technology		SFTP
Authentication mechanism		SSH
Data contract	**Wejście:** ścieżka odnosząca się do semestru według identyfikatora **Wyjście:** lista kursów z identyfikatorami prowadzących zajęcia **Klasa:**  **Format:** plik JSON **Przykład:** `[ { "id":"unique_id", "form":0, "fieldOfStudyKey":1, "lecturers":[ "lecturer_1", "lecturer_2" ], "translations":{ "pl":"Projektowanie oprogramowania", "en":"Software design" } }, ... ]`	
Does the interface manipulate on the sensitive data (RODO)?	Nie	
Middleware used	Brak	
Initiating side	System hospitacji	
Communication model	Asynchroniczny wyzwalany zdarzeniem czasowym	
Performance	1/ pół roku	
Volumetry	Liczba wywołań: 2 / rok Rozmiar danych: każde wywołanie to około 2000 kursów (~ 100 kursów na semestr, 4 kierunki, jednocześnie 5 roczników)	
Required accessibility	LOW	

5.3.3 Interface 3 - Pobranie listy wszystkich prowadzących bez konfliktu zajęć z zajęciami konkretnego kursu podanego prowadzącego zajęcia

Status	Planowany	
	Source application	Target application
Application name	System hospitacji - Backend	System zapisów
Integration technology		HTTPS
Authentication mechanism		Klucz API

Data contract	**Wejście:** identyfikator kursu, identyfikator prowadzącego zajęcia
	Wyjście: lista identyfikatorów prowadzących zajęcia
	Format: JSON
	Przykład:
	`[`
	`  "lecturer_1",`
	`  "lecturer_2"`
	`]`	

Does the interface manipulate on the sensitive data (RODO)?	Nie	
Middleware used	Brak	
Initiating side	System hospitacji	
Communication model	Synchroniczny na żądanie użytkownika	
Performance	1200 / godzinę (10 użytkowników, każdy 2 x na minutę)	
Volumetry	Liczba wywołań: 19 200 / dzień (1200 / godzinę, 16 godzin w ciągu dnia) Rozmiar danych: każde wywołanie to maksymalnie 200 identyfikatorów (liczba potencjalnych hospitujących)	
Required accessibility	HIGH	

5.3.4 Interface 4 - Uwierzytelnienie użytkownika

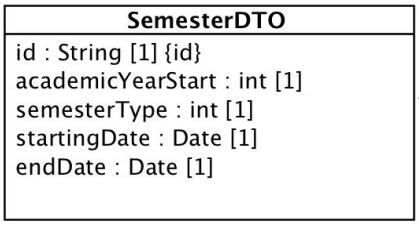
Status	Istniejący	
	Source application	Target application
Application name	System hospitacji - Frontend	Firebase Authentication
Integration technology	Firebase Authentication SDK	HTTPS
Authentication mechanism		API Key
Data contract	**Wejście:** nazwa użytkownika oraz hasło **Wyjście:** konto użytkownika Firebase oraz JWT **Format:** obiekt	
Does the interface manipulate on the sensitive data (RODO)?	Tak	
Middleware used	Brak	
Initiating side	System hospitacji	
Communication model	Synchroniczny na żądanie użytkownika	
Performance	1200 / godzinę (400 użytkowników, 3 logowania na godzinę, 16 godzin w ciągu dnia)	

Volumetry	Liczba wywołań: 19 200 / dzień Rozmiar danych: każde wywołanie to 1 konto użytkownika Firebase
Required accessibility	HIGH

5.3.5 Interface 5 - Weryfikacja tokenu użytkownika

Status	Istniejący	
	Source application	Target application
Application name	System hospitacji - Backend	Firebase Authentication
Integration technology	Firebase Admin SDK	HTTPS
Authentication mechanism		API Key
Data contract	**Wejście:** JWT użytkownika **Wyjście:** obiekt FirebaseToken lub wyjątek **Format:** obiekt	
Does the interface manipulate on the sensitive data (RODO)?	Tak	
Middleware used	Brak	
Initiating side	System hospitacji	
Communication model	Synchroniczny na żądanie użytkownika	
Performance	2000 / minutę (400 użytkowników, 5 żądań na minutę)	
Volumetry	Liczba wywołań 2 000 000 / dzień Rozmiar danych: każde wywołanie to 1 obiekt FirebaseToken	
Required accessibility	HIGH	

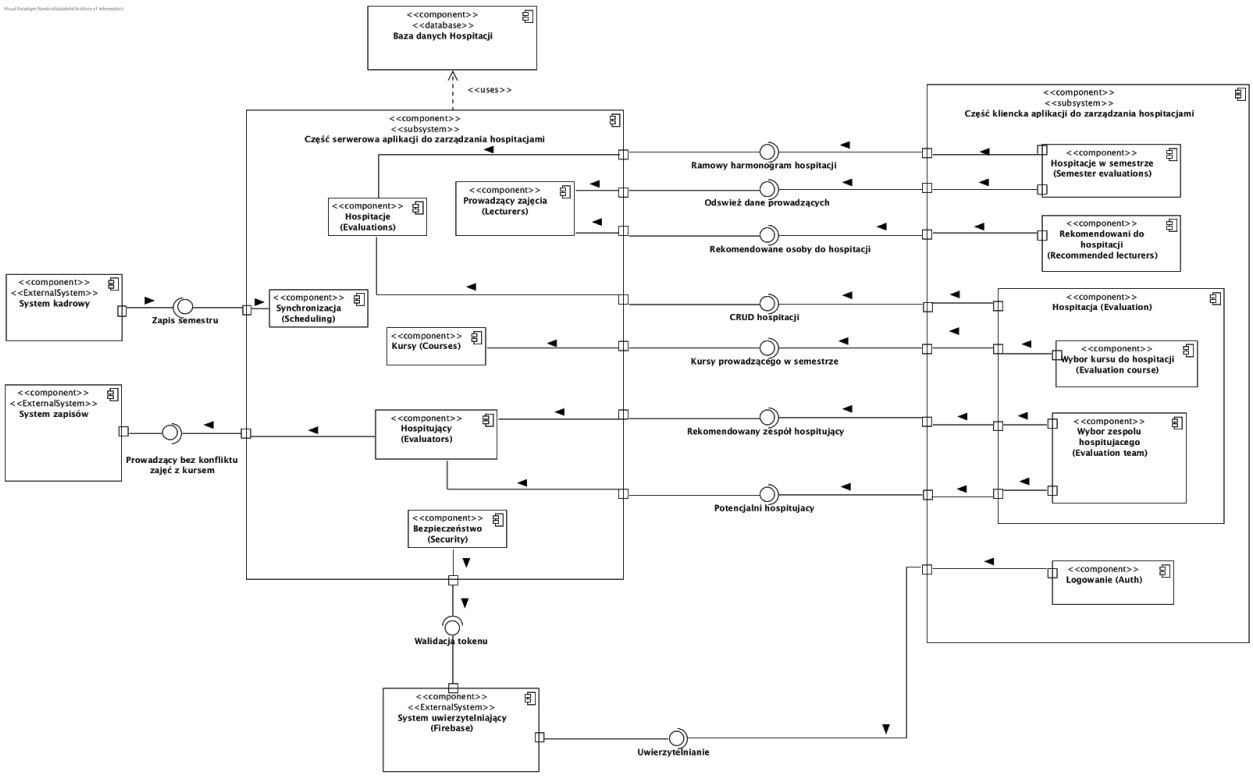
5.3.6 Interface 6 - Wprowadzenie danych nowego semestru

Status	Planowany	
	Source application	Target application
Application name	System zapisów	System hospitacji - Backend
Integration technology		HTTPS
Authentication mechanism		Klucz API
Data contract	**Wejście:** dane nowego semestru **Wyjście:** brak **Klasa:**  **Format:** JSON	

	Przykład:
	{
	"id":"semster_id",
	"academicYearStart":2018,
	"semesterType":0,
	"startingDate":"2018-10-01",
	"endDate":"2019-02-18"
	}

Does the interface manipulate on the sensitive data (RODO)?	Nie
Middleware used	Brak
Initiating side	System zapisów
Communication model	Synchroniczny na żądanie użytkownika
Performance	1/ pół roku
Volumetry	Liczba wywołań: 2 / rok Rozmiar danych: każde wywołanie to jeden prosty obiekt semestru z datami rozpoczęcia i zakończenia
Required accessibility	LOW

6. Perspektywa funkcjonalna (Functional view)



PSM/Perspektywy/Funkcjonalna/[CD] Perspektywa funkcjonalna

7. Perspektywa wdrożeniowa (Deployment view)

7.1 Wprowadzenie

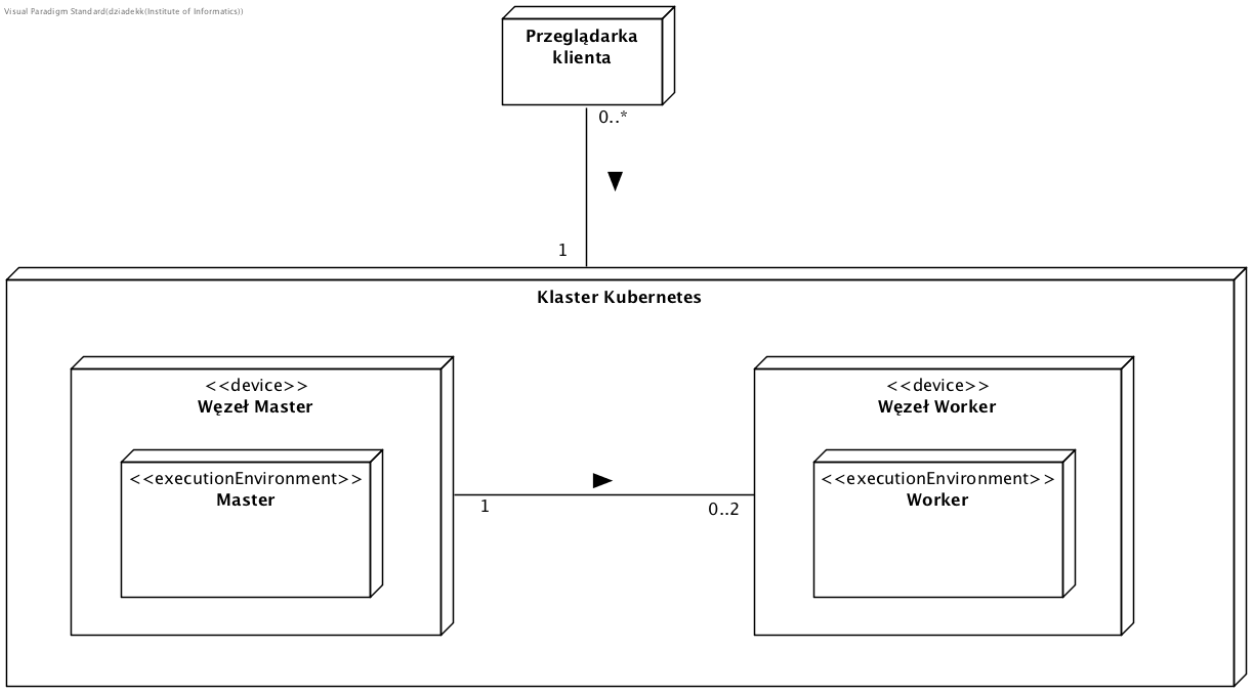
Do wdrożenia systemu, aby zapewnić dynamiczną skalowalność poszczególnych jednostek, zostanie wykorzystany klaster zbudowany przy wykorzystaniu systemu Kubernetes. Na klaster składają się 2 węzły robocze oraz 1 węzeł główny, które reprezentowane są przez niezależne jednostki logiczne.

W ramach klastra, wykorzystując platformę Doker, zostaną uruchomione przygotowane kontenery zawierające bazę danych, aplikację webową lub aplikację serwerową. Każdy kontener zostanie uruchomiony w oddzielnym Podzie Kubernetes.

W celu przetestowania środowiska zostanie wykorzystane rozwiązanie uruchomienia całego klastra Kubernetes lokalnie na maszynie wirtualnej. Aby to osiągnąć zostanie wykorzystana odpowiednia konfiguracja Vagrant dostarczana przez IBM (https://github.com/IBM/deploy-ibm-cloud-private/blob/master/docs/deploy-vagrant.md). Do przygotowania odpowiedniej konfiguracji poszczególnych środowisk zostanie wykorzystany Ansible, aby zapewnić automatyzację oraz powtarzalność uruchamiania nowych środowisk.

7.2 Diagram fizyczny

Struktura fizyczna uruchomienia systemu, który jest klastrem Kubernetes. Są to fizyczne elementy, na których uruchamiane będą Pody Kubernetes zdefiniowanie według schematu diagramu logicznego.



PSM/Perspektywy/Wdrozeniowa/[PSM] Struktura fizyczna

7.2.1 Kubernetes Master

General information	
Name	Kubernetes Master
Virtual	Nie
Data center?	Nie

OS	Ubuntu 16.04
Description	Master node klastra Kubernetes, który zarządza Workerami, zawiera bazę danych zadań (etcd) oraz nadzoruje pracę klastra

Hardware configuration	
Vendor	DELL
Processor	2x 3.3 GHz
RAM	8 GB
HDD	256 GB
RAID i HDD Netto	RAID 0
RAID connected?	Brak
Net cards bonding	Nie

Software configuration	
Users and groups	Jeden użytkownik (root-evaluations), który służy jedynie do utrzymania maszyny, ponieważ konfiguracja klastra będzie odbywać się przez konsolę *kubectl*
Poziom pracy systemu, czy jest wymagane środowisko graficzne	Środowisko graficzne nie jest wymagane, ponieważ wymagane jest jedynie konfigurowanie maszyny, które wykonywane jest zdecydowanie sprawniej w trybie konsolowym
Dodatkowe pakiety z dystrybucji systemu	Brak
Dodatkowe pakiety spoza dystrybucji systemu	● kubeadm ● docker ● python 3.6

7.2.2 Kubernetes Worker

General information	
Name	Kubernetes Worker
Virtual	Nie
Data center?	Nie
OS	Ubuntu 16.04
Description	Worker node klastra Kubernetes, który wykonuje zadanie dla niego przeznaczone przez master, tzn. uruchamia odpowiednie pody i przekierowuje do nich żądania

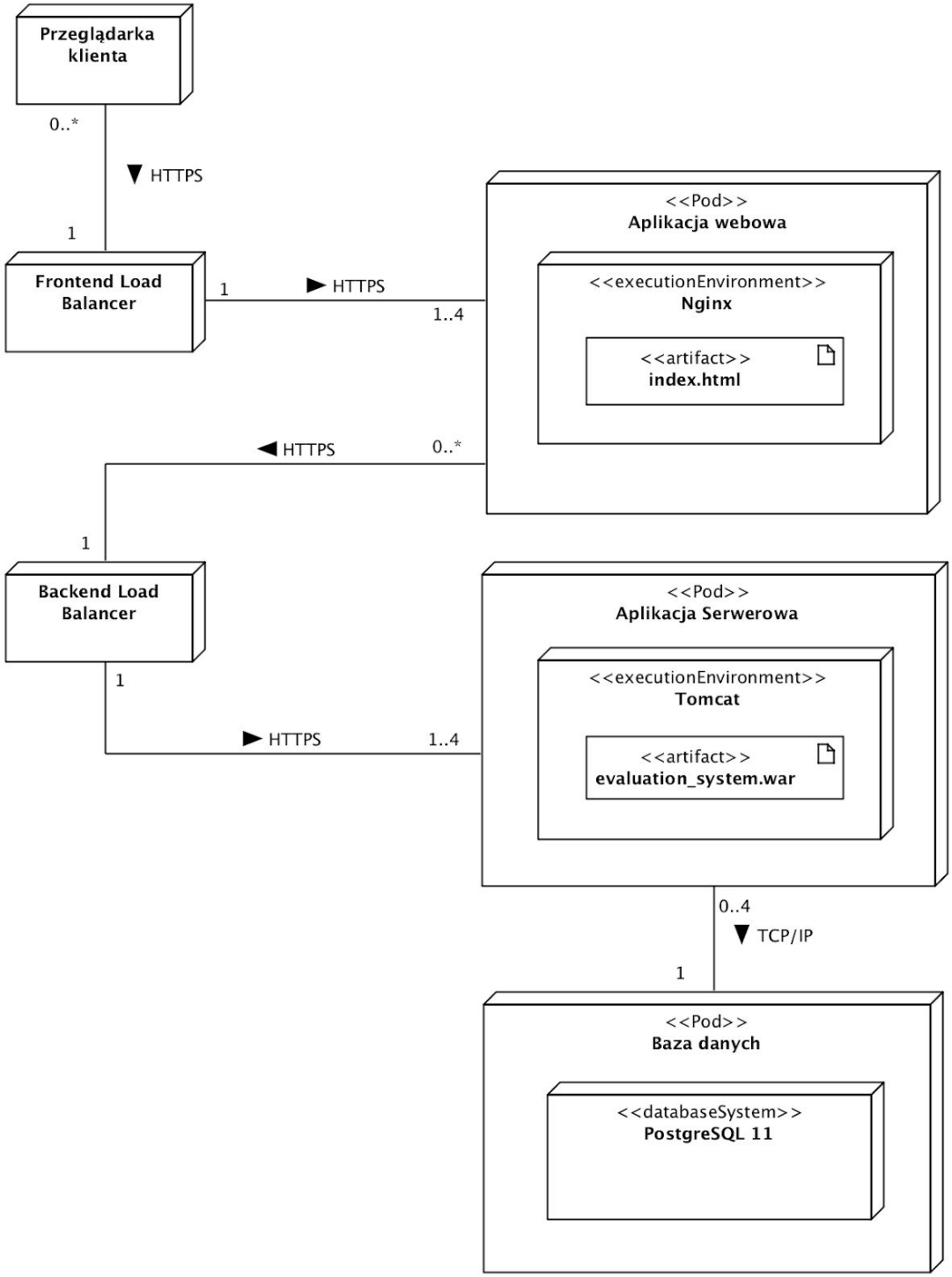
Hardware configuration	
Vendor	DELL
Processor	4x 3.3 GHz
RAM	32 GB
HDD	1TB
RAID i HDD Netto	RAID 1
RAID connected?	Brak
Net cards bonding	Nie

Software configuration	
Users and groups	Jeden użytkownik (root-evaluations), który służy jedynie do utrzymania maszyny, ponieważ konfiguracja klastra będzie odbywać się przez node

	Master
Poziom pracy systemu, czy jest wymagane środowisko graficzne	Środowisko graficzne nie jest wymagane, ponieważ wymagane jest jedynie konfigurowanie maszyny, które wykonywane jest zdecydowanie sprawniej w trybie konsolowym
Dodatkowe pakiety z dystrybucji systemu	Brak
Dodatkowe pakiety spoza dystrybucji systemu	● python 3.6 ● docker ● kubelet ● kube-proxy

7.3 Diagram logiczny

Struktura logiczna uruchomienia systemu, na klastrze Kubernetes. Load balancery są wbudowany mechanizmem w Kubernetes i rozdzielają żądania do konkretnych Pod'ów. Pody są jednostkami rozmieszczonymi na fizycznych węzłach klastra i ich zarządzaniem zajmuje się Kubernetes Master.



PSM/Perspektywy/Wdrozeniowa/[PSM] Struktura logiczna

7.3.1 Aplikacja webowa

General information	
Name	Aplikacja webowa
Virtual	Tak
Obraz docker	nginx 1.15.7
Description	Kontener na którym uruchomiona zostanie część frontendowa systemu jako zbudowana aplikacja Angulara. Może być uruchomiona w wielu instancjach.

Resources limits	
Processor	2 x 1.6 GHz
RAM	8 GB
HDD	50 GB

7.3.2 Aplikacja serwerowa

General information	
Name	Aplikacja serwerowa
Virtual	Tak
Obraz docker	tomcat 9.0.13-jre8
Description	Kontener na którym uruchomiona zostanie część serwerowa systemu w postaci artefaktu Spring, która komunikuje się z zewnętrznymi systemami i dostarcza dane. Może być uruchomiona w wielu instancjach.

Resources limits	
Processor	2 x 3.3GHz
RAM	8 GB
HDD	100 GB

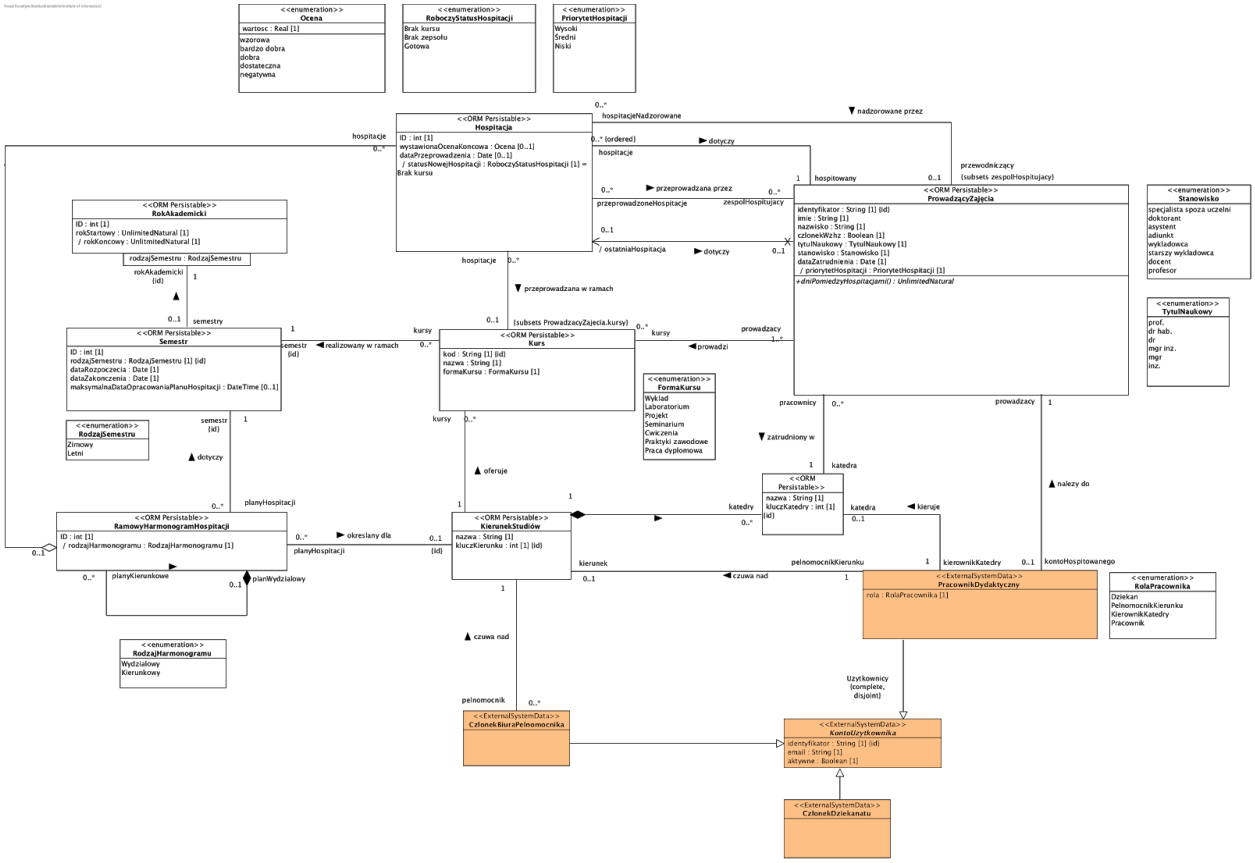
7.3.3 Baza danych

General information	
Name	Baza danych
Virtual	Tak
Obraz docker	postgres 11.1
Description	Kontener na którym uruchomiona zostanie baza danych systemu hospitacji w jednej instancji

Resources limits	
Processor	2 x 3.3GHz
RAM	8 GB
HDD	1 TB

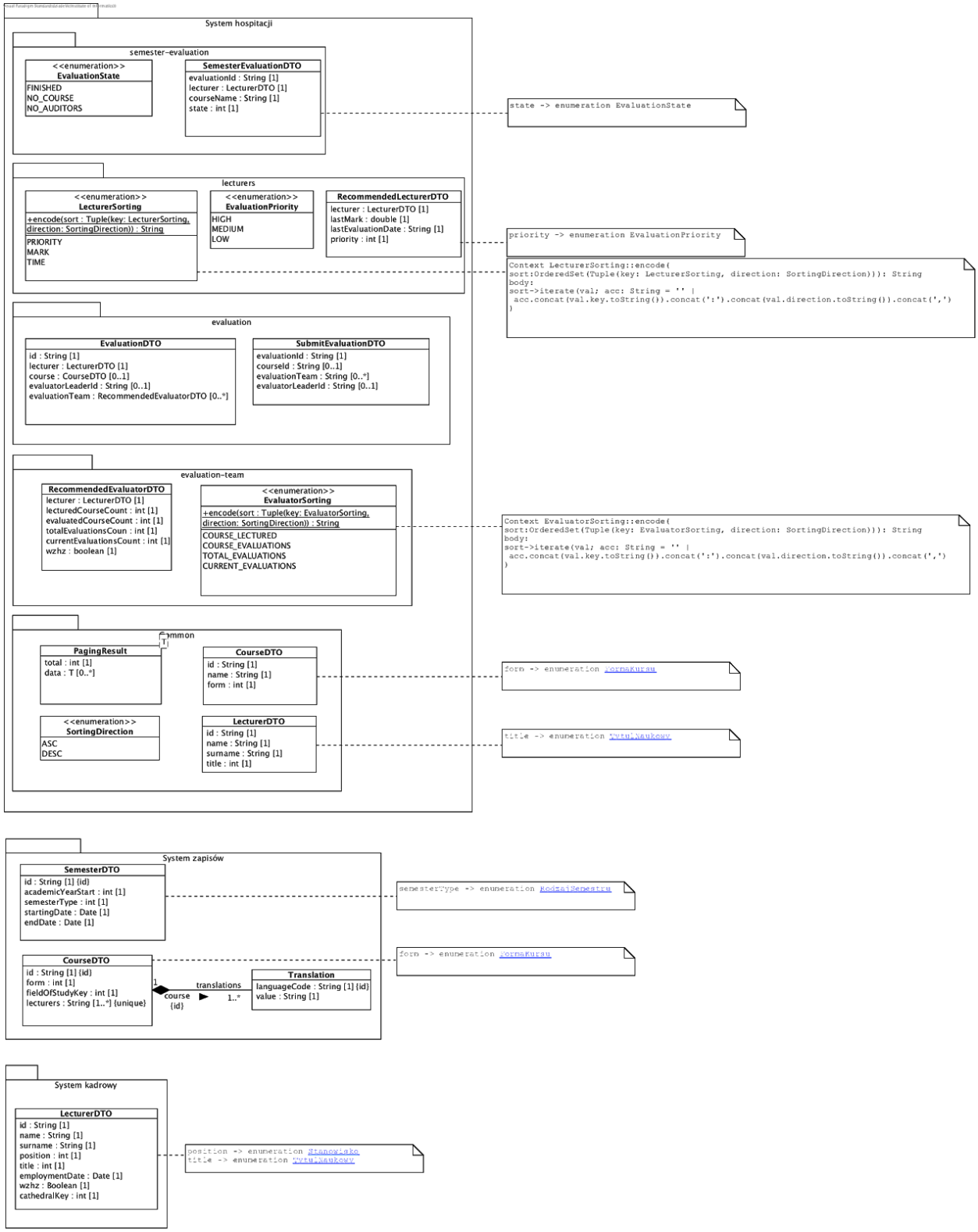
8. Perspektywa informacyjna (Information view)

8.1 Model informacyjny



PSM/Perspektywy/Informacyjna/[PSM] Model informacyjny

8.2 Model DTO

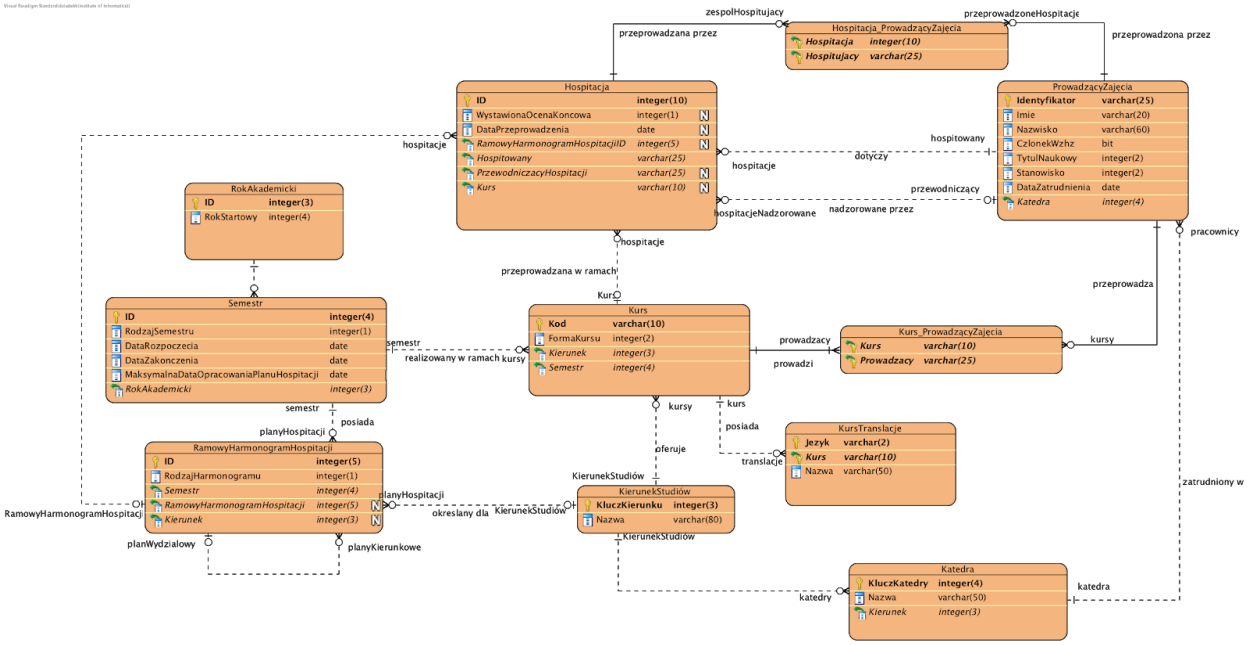


PSM/Perspektywy/Informacyjna/[PSM] DTO

8.3 Projekt bazy danych

Database General information (one table per server)	
SID/Service Name	Baza danych
Server name	db-evaluations
Port	5432
Type	Postgres 11.1
Character codding	Standardowe
Description	Relacyjna baza danych systemu hospitacji, przechowująca wszystkie informacje na temat historii hospitacji, kursów oraz prowadzących zajęcia i wynikających z tego procesu encji
Technologies	Domyślne, brak potrzeby stosowania dodatkowych mechanizmów

8.3.1 Diagram ERD



PSM/Perspektywy/Informacyjna/[PSM] Schemat bazy danych

8.3.2 Schematy danych

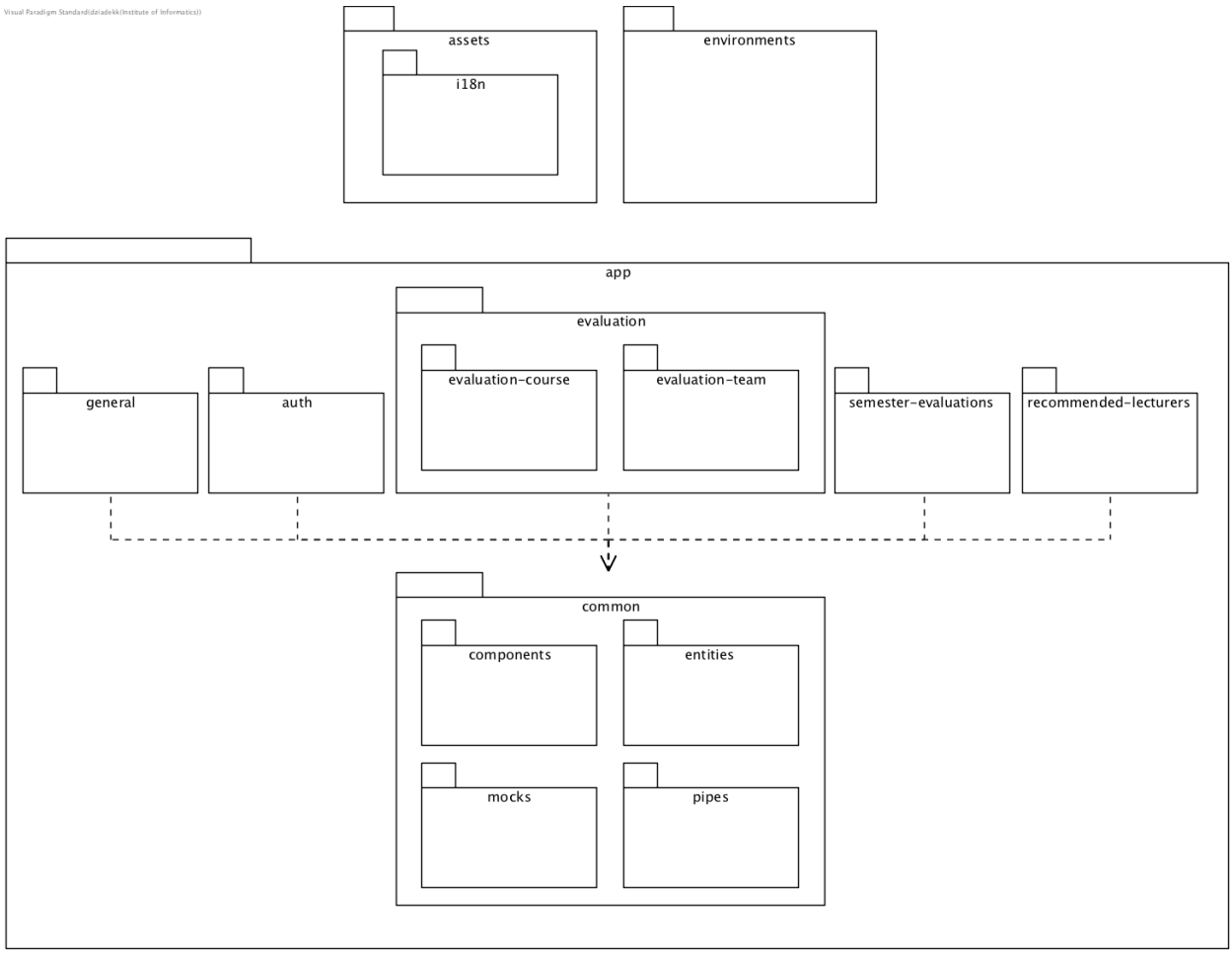
Schema information	
Schema name	EvaluationsScheme
Initial capacity	50 MB (konieczne wczytanie historii kursów w celach rekomendacyjnych)
Capacity increment (year)	10 MB
Others	Brak

9. Perspektywa wytwarzania (Development view)

9.1 Aplikacja webowa (Frontend)

Aplikacja napisana w Angularze 7. Zastosowana została zalecana struktura pakietów dla aplikacji Angular w oparciu o MVVM

- **environment** - ustawienia środowisk (osobne dla testowego oraz produkcyjnego)
- **assets** - wszystkie zasoby zewnętrzne aplikacji np. tłumaczenia interfejsu na różne języki
- **app** - pliki źródłowe aplikacji podzielone na pakiety według komponentów Angularowych, które odzwierciedlają części widoku. Dodatkowym pakietem jest **commons**, gdzie znajdują się wspólne dla wielu widoków pakiety jak np. wspólne encje, komponenty lub pipe

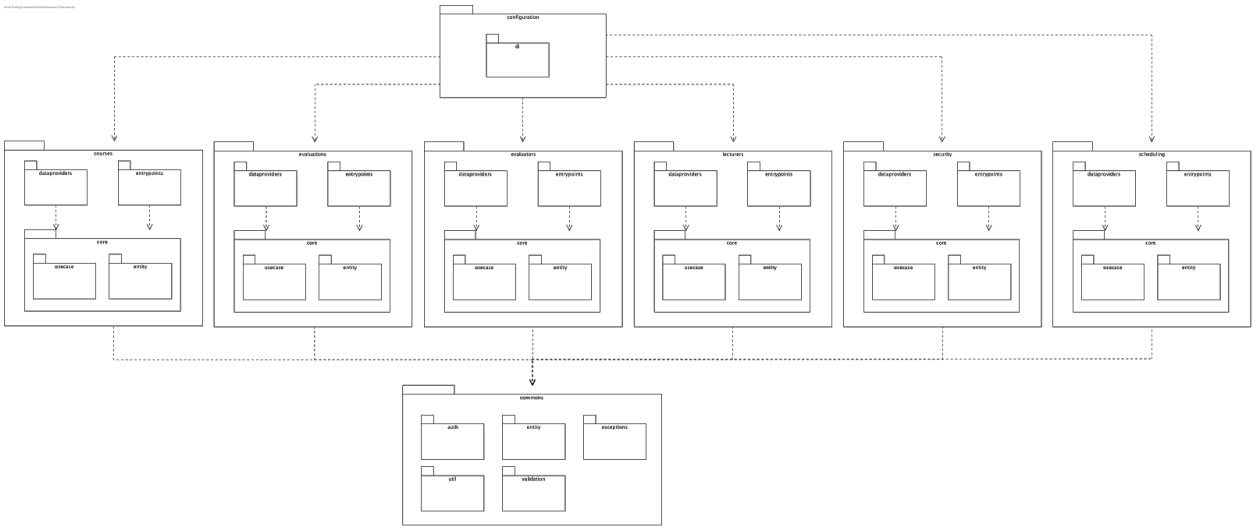


PSM/Perspektywy/Funkcjonalna/[PSM] Frontend - Perspektywa wytwarzania

9.2 Aplikacja serwerowa (Backend)

Aplikacja napisana w języku Java oraz Kotlin przy wykorzystaniu frameworka Spring. Zastosowana została architektura **Clean Architecture** oraz podział na pakiety według funkcjonalności. Kolejne poziomy prezentują się następująco:

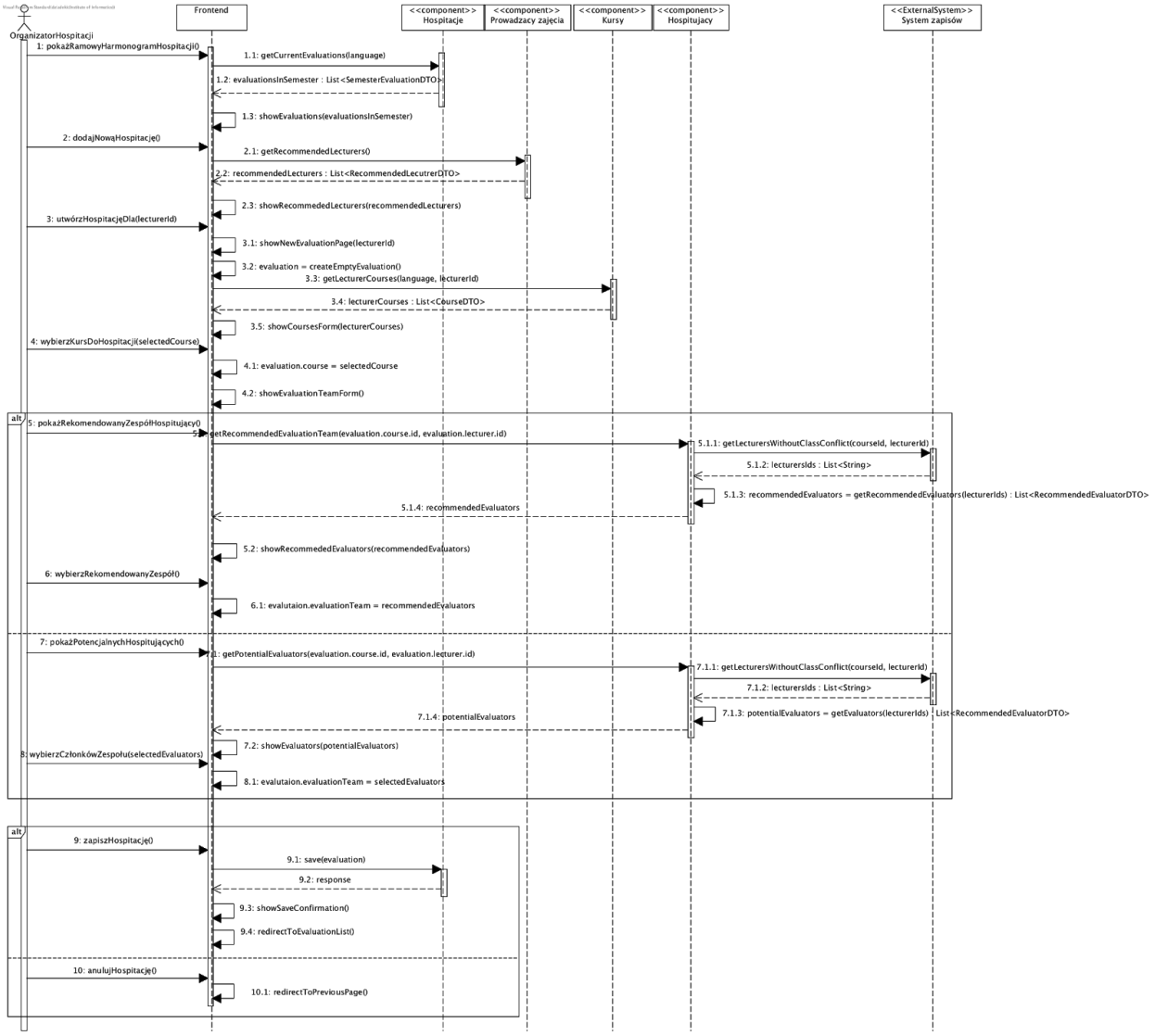
- **configuration** - konfiguracja aplikacji, której zadaniem jest spięcie poszczególnych funkcjonalności, dependency injection oraz konfiguracja aplikacji
- **pakiety funkcjonalności** - każda funkcjonalność istnieje w swoim pakiecie gdzie występuje następujący logiczny podział
    - **dataproviders** - dostarczają dane, implementacja repozytorium
    - **entrypoints** - punkty wejściowe do aplikacji, czyli Controller czy Job
    - **core** - logika biznesowa aplikacji
        - **entity** - obiekty domenowe na potrzeby danej funkcjonalności
        - **usecase** - akcje biznesowe, czyli to co aplikacja potrafi
- **commons** - wspólne dla funkcjonalności elementy takie jak np. encje, walidacja, czy klasy typu utils.



PSM/Perspektywy/Funkcjonalna/[PSM] Backend - Perspektywa wytwarzania

10. Realizacja przypadków użycia (Use-case realizations)

Przebieg całego procesu podlegającego implementacji, przedstawiono w formie diagramu sekwencji. Przedstawiono interakcje między różnymi komponentami systemu. Aby nie zaciemniać obrazu całego procesu, pominięto szczegóły implementacyjne takie jak komponenty po stronie Frontendu, komunikację z bazą danych, czy interakcję pomiędzy poszczególnymi klasami. Wszystkie żądania do backendu spełniają interfejs przedstawiony w perspektywie funkcjonalnej.



PSM/Przebieg/[SD] Opracowanie ramowego planu hospitacji