eKRK

Projekt architektury

1. Cel dokumentu

Zadaniem tego dokumentu jest opisanie architektury oraz sposobu realizacji funkcji systemu „eKRK". Precyzuje on cele jakie powinny zostać osiągnięte przy realizacji systemu i wynikające z nich ograniczenia oraz decyzje architektoniczne wraz z nich uzasadnieniem. Dokument przedstawia również użyte mechanizmy architektoniczne i korzyści osiągnięte dzięki ich zastosowaniu. Ostatnia sekcja jest dedykowana przedstawieniu wybranego przypadku użycia. Całość dokumentu pozwala na zrozumienie najważniejszych aspektów systemu „eKRK", które wpłyną na postać implementacji.

2. Cele i ograniczenia architektoniczne

Cele i ograniczenia wynikają z wymagań zawartych w następujących dokumentach:

1. Dokument wizji.
2. Wymagania systemowe.

Wymagania funkcjonalne:

1. CRUD przedmiotów kształcenia, programu studiów, modułów kształcenia oraz zajęć.
2. Tworzenie macierzy śladowania.
3. Wydruk programu studiów.

Wymagania niefunkcjonalne:

1. System jest dostarczony wraz ze skryptami tworzącymi relacyjną bazę danych
2. System pozwala na wydajne pobieranie danych (zapytania, przeglądanie) – średni czas odpowiedzi systemu poniżej 5 s przy obciążeniu do 200 użytkowników.
3. System zawiera mechanizmy chroniące przed nieuprawnionym dostępem.
4. System jest dostarczony w dwóch wersjach językowych: polskiej i angielskiej.

Ograniczenia systemu:

1. System wymaga dostępu do Internetu
2. System wymaga instalacji przez użytkownika przeglądarki wspierającej obsługę Java Scriptu.

3. Decyzje wraz z uzasadnieniem

Cel	Sposób osiągnięcia
1. Automatyzacja procesu tworzenia bazy danych	A. Dostarczenie pliku reprezentującego schemat bazy danych

2. Wydajne pobieranie danych	A. Rozłożenie obciążenia systemu na kilka fizycznych węzłów. Rozdzielenie odpowiedzialności pomiędzy fizyczne serwery odpowiedzialne za: dostarczanie treści statycznej, dostarczanie treści dynamicznej, utrzymanie bazy danych. Część logiki biznesowej oraz renderowanie widoku przesunięto na aplikację kliencką. B. Minimalizacja ruchu sieciowego pomiędzy aplikacją kliencką a serwerową Wykorzystanie komunikacji REST z formatem danych JSON. C. Wykorzystanie mechanizmów cache
3. Zabezpieczenie konta przed nieuprawionym dostępem	A. Autentykacja i autoryzacja z wykorzystaniem mechanizmu OAuth2 B. Szyfrowanie transmisji danych z użyciem TLS
4. System dostępny w dwóch wersjach językowych: polskiej, angielskiej	A. Wykorzystanie modułu angular-translate do przetłumaczenia interfejsu aplikacji.

4. Mechanizmy architektoniczne

1A. Dostarczenie pliku reprezentującego schemat bazy danych

Framework *ruby on rails* pozwala na automatyczną generacje modelu domenowego oraz schematu bazy przy pomocy komend.

Przykładowo komenda:

```
rails generate model User firstname:string lastname:string
```

Generuje model oraz kod modyfikujący (tzw. migracja) schemat bazy danych, ze względu na dodanie klasy User. Finalny schemat bazy danych jest przechowywany w pliku schema.rb

Natomiast komenda:

```
rake db:migrate
```

Wykonuje odpowiednie zmiany w strukturze bazy danych, doprowadzając bazę danych do stanu opisanego wygenerowanym plikiem schema.rb

Ponadto dzięki użyciu dodatkowych bibliotek możliwe jest wygenerowanie losowych, testowych danych. Daje to możliwość utworzenia schematu bazy danych i załadowania danych testowych za pomocą jednego polecania w środowisku ruby:

```
rake db:setup
```

Wraz z systemem zostanie dostarczony plik schema.rb reprezentujący finalny stan schematu bazy danych.

2A. Rozłożenie obciążenia systemu na kilka fizycznych węzłów.

Zdecydowano się rozdzielić system na cztery fizyczne węzły, by móc obsłużyć sumarycznie większy ruch sieciowy,

a także by być w stanie wykonać więcej operacji obliczeniowych.
Możemy wyróżnić w systemie następujące węzły:

- Serwer bazodanowy, obsługujący zapytania o dane, w tym realizujący pewne operacje na tych danych (np. złączenia)
- Serwer dostarczający danych statycznych do przeglądarki, odpowiada za wysłanie skryptów .js, a także plików .html, .css, .jpg, .png itp.
- Serwer generujący dynamiczne treści – odpowiada za zwracanie danych w formacie JSON aplikacji klienckiej, a także generowanie .pdf , znaczna części logiki biznesowej znajduje się tutaj.
- Urządzenie klienckie wykorzystujące przeglądarkę internetową w celu realizacji aplikacji Single Page Application – odpowiada za renderowanie strony, komunikacje z API, a także znajduje się tutaj pewna część logiki biznesowej

2B. Minimalizacja ruchu sieciowego pomiędzy aplikacją kliencką a serwerową

Aplikacja kliencka wymienia z aplikacją serwerową tylko dane, których użytkownik w danym momencie potrzebuje. Dane te są przesyłane z użyciem formatu JSON, a transportowane przy użyciu protokołu http. Format JSON jest w stanie reprezentować przesyłane struktury danych przy użyciu mniejszej ilości bajtów, w stosunku do np. formatu XML. Przesyłane informacji nie zawierają nic o sposobie prezentacji, jak w przypadku klasycznych aplikacji internetowych, co również przekłada się na minimalizację ruchu sieciowego. Dzięki komunikacji REST i użyciu frameworka AngularJS do budowy aplikacji SPA (Single Page Application) możliwe jest znaczne zredukowanie przesyłanej ilości danych.

2C. Wykorzystanie mechanizmów cache

W celu zwiększenia wydajności aplikacji zostaną zastosowane różne formy cache'owania danych:

- Cache dla danych statycznych – dla serwera udostępniającego statyczne zasoby takie jak pliki css, js, html czy pliki graficzne, możliwe jest wymuszenie ich cache'owania po stronie przeglądarki. Po ustawieniu czasu przedawnienia dla danego typu plików, serwer będzie zwracał odpowiedź *304 Not Modified,* przy próbie żądania zasobu, który nie został zmieniony. Jeśli plik nie został zmieniony, będzie on brany z pamięci przeglądarki a nie ponownie pobierany.
- Cache dla danych dynamicznych – dla REST API napisanego w framworku grape, dostępne są moduły rozszerzające funkcjonalność o mechanizm cache'owania. Dostępne są różne strategie obsługi cache'a na poziomie aplikacji. Są to m. in. ustawienie czasu przedawnienia dla żądanego zasobu, zapisywanie wykonanych żądań dla danych parametrów oraz przeładowywanie cache'a jedynie przy zmianie zapisanych danych.
- Dla bazy danych postgresql jesteśmy w stanie konfigurować ilość pamięci operacyjnej wykorzystywanej do cache'owania danych. Może mieć to istotny wpływ na czas przetwarzania zapytań po stronie serwera bazy danych.
  Parametr:
  - *shared_buffers* – określa maksymalny rozmiar cache'a w pamięci operacyjnej.
  - effective_cache_size – określa maksymalny rozmiar bazy danych trzymany w pamięci operacyjnej

3A. Autentykacja i autoryzacja użytkownika systemu

Autentykacja do systemu będzie się odbywać z wykorzystaniem loginu i hasła przydzielonego dla konkretnego użytkownika systemu. Po udanej autentykacji, autoryzacja będzie się odbywać z wykorzystaniem implementacji OAuth2 dla środowiska Ruby – doorkeeper.

Wygenerowany access token przez doorkeeper będzie dostarczony do aplikacji klienckiej. Aplikacja kliencka będzie przechowywać access token w zasięgu kodu javascript. Aplikacja kliencka będzie doklejać ww. token do nagłówka http podczas komunikacji z systemem, w szczególności REST API. Ryzyko, że access token zostanie ujawniony niepowołanej osobie/szkodliwej bibliotece JS w jakikolwiek sposób jest złagodzone, ze względu na krótki czas życia tokenu – w specyfikacji RFC 6749 pojawia się wartość jednej godziny, a sam doorkeeper domyślnie ustawia ten czas na dwie godziny.

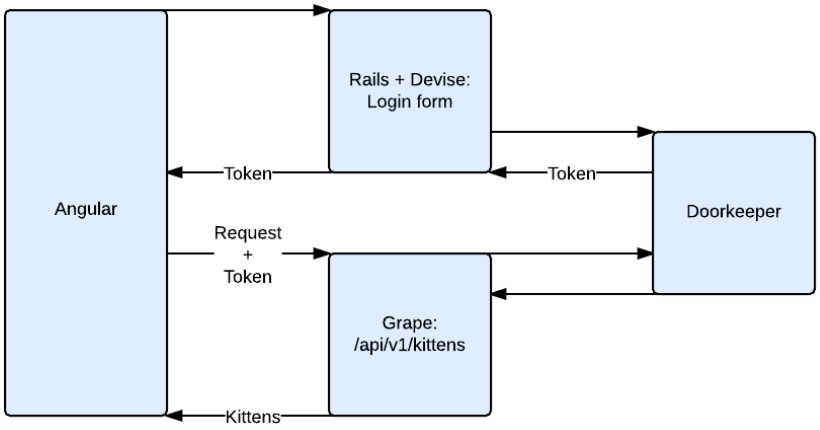
Doorkeeper wg dokumentacji ma możliwość generowania tzw. Refresh token podczas generowania access token. Taki token będzie w formie zaszyfrowanej trzymany w ciastku http. Gdy czas życia access token dobiegnie końca, serwer odszyfruje refresh token by odświeżyć tokeny, i w odpowiedzi zwróci do klienta nowy access token. Takie podejście zapobiegnie konieczności ponownego podania loginu i hasła przez użytkownika do systemu, w momencie gdy skończy się czas życia access token.

Warto zaznaczyć, że sama specyfikacja RFC nie mówi nic o czasie życia refresh token, a doorkeeper celowo nie umożliwia ustawienia czasu życia dla refresh token. Daje za to możliwość odwołania tokenów, co może być wykorzystane podczas wylogowywania użytkownika z systemu.

Poszczególne komponenty (w szczególności REST API) systemu będą weryfikować czy odebrany token pozwala na dostęp do danego zasobu, z użyciem doorkeeper.

Specyfikacja OAuth2 wymaga użycia TLS do przesyłania tokenów między elementami systemu.

Poniższy rysunek, zapożyczony z http://codetunes.com/2014/oauth-implicit-grant-with-grape-doorkeeper-and-angularjs , przedstawia w skrócie ten mechanizm:



3B. Szyfrowanie transmisji danych z użyciem TLS

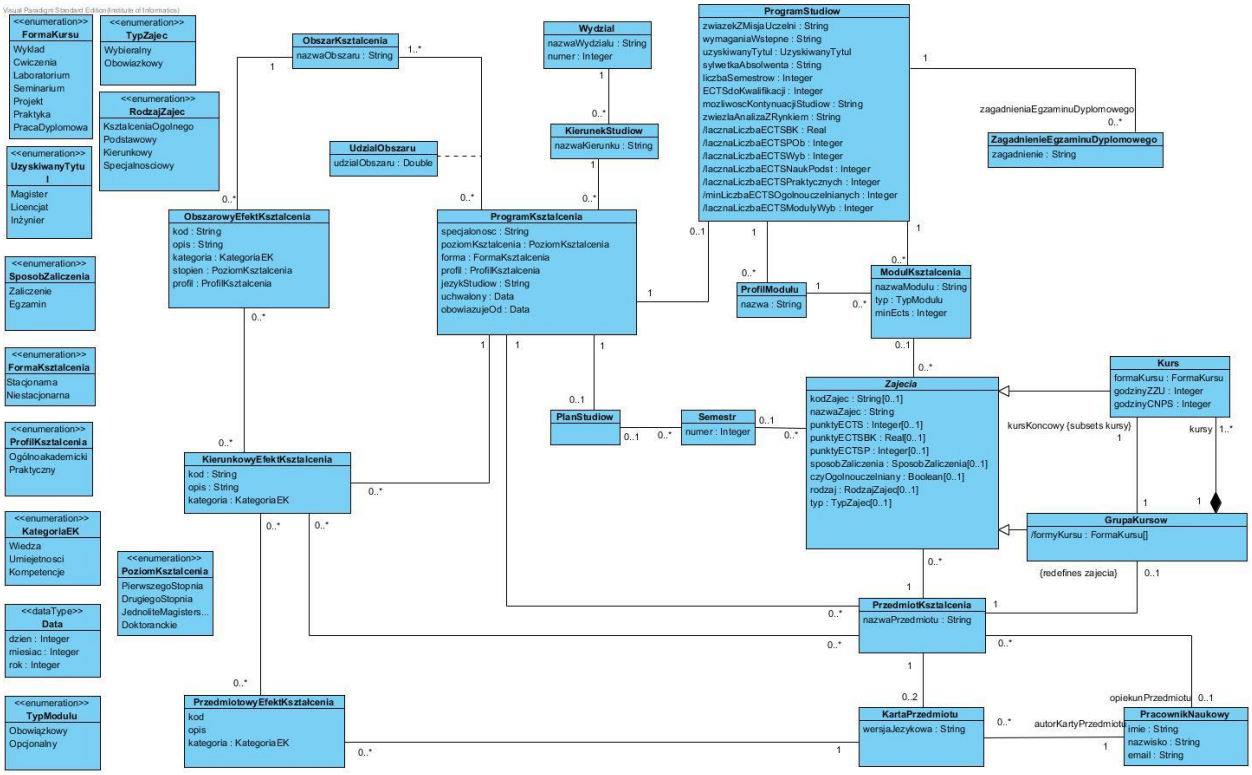
Dla uniemożliwienia podsłuchania transmisji danych poprzez sieć poszczególne komponenty systemu będą używały szyfrowania transmisji z użyciem TLS. Sam standard OAuth2 wymaga szyfrowanego kanału transmisji z użyciem standardu TLS.

4A. Wykorzystanie modułu angular-translate do przetłumaczenia interfejsu aplikacji

Dla każdej z wspieranych wersji językowych systemu, tj. polskiej, angielskiej zostaną utworzone pliki ze słownikami zawierającymi: klucze wraz z przypisaną do konkretnego klucza wartością. Każdy element interfejsu (np. etykieta) będzie opisany kluczem, który w trakcie działania aplikacji, w zależności od wybranej wersji językowej systemu zostanie zastąpiony wartością (tłumaczeniem) ze słownika (odpowiadającemu wybranej wersji językowej). Dynamiczna podmiana kluczy wartościami zostanie zrealizowana z użyciem modułu angular-translate.

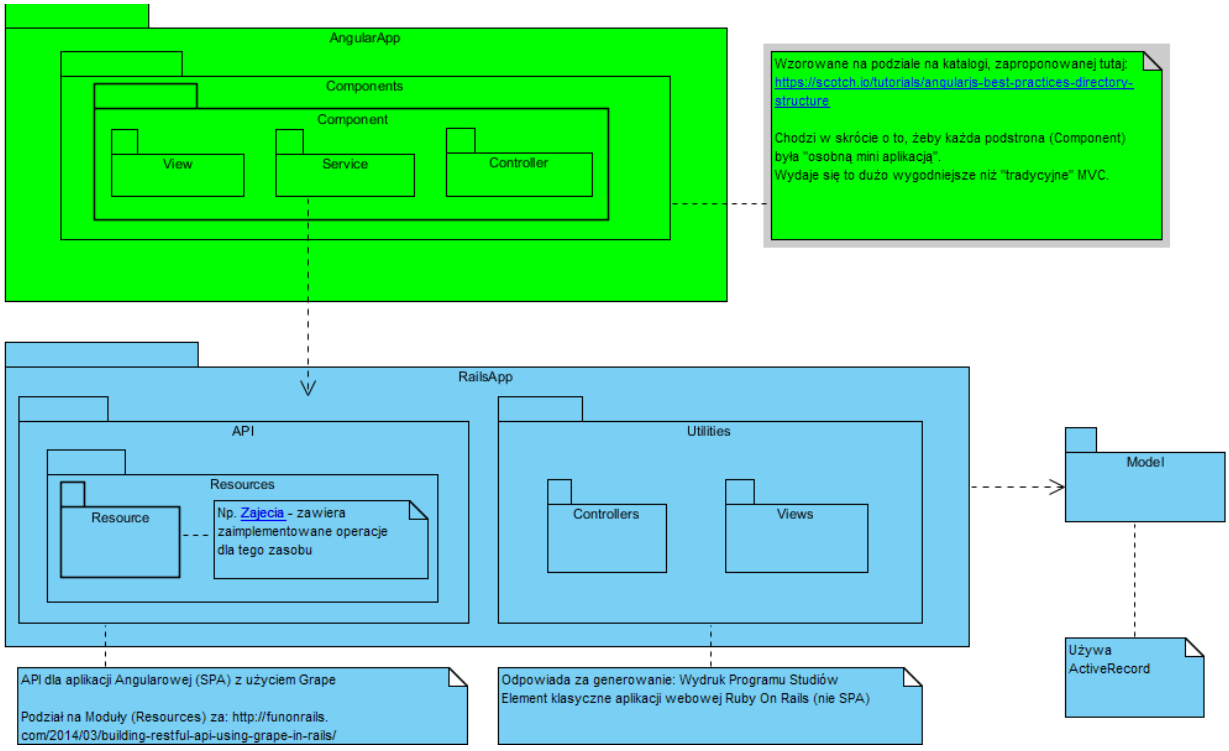
5. Kluczowe abstrakcje

Poniższy diagram przedstawia kluczowe abstrakcje, które będą dostępne w systemie, w modelu aplikacji Rails-owej.

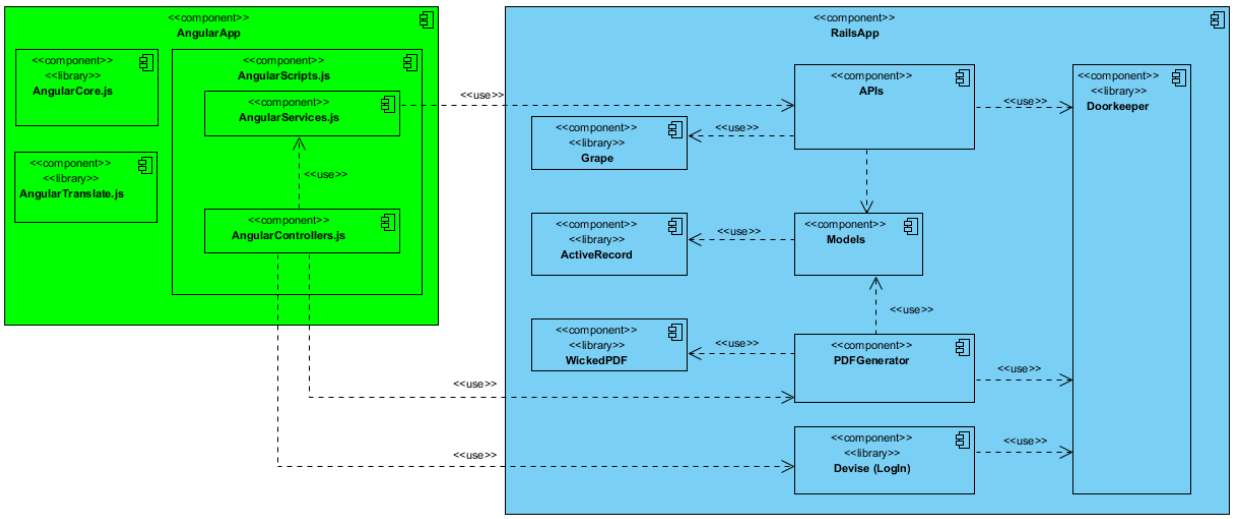


6. Widoki architektoniczne

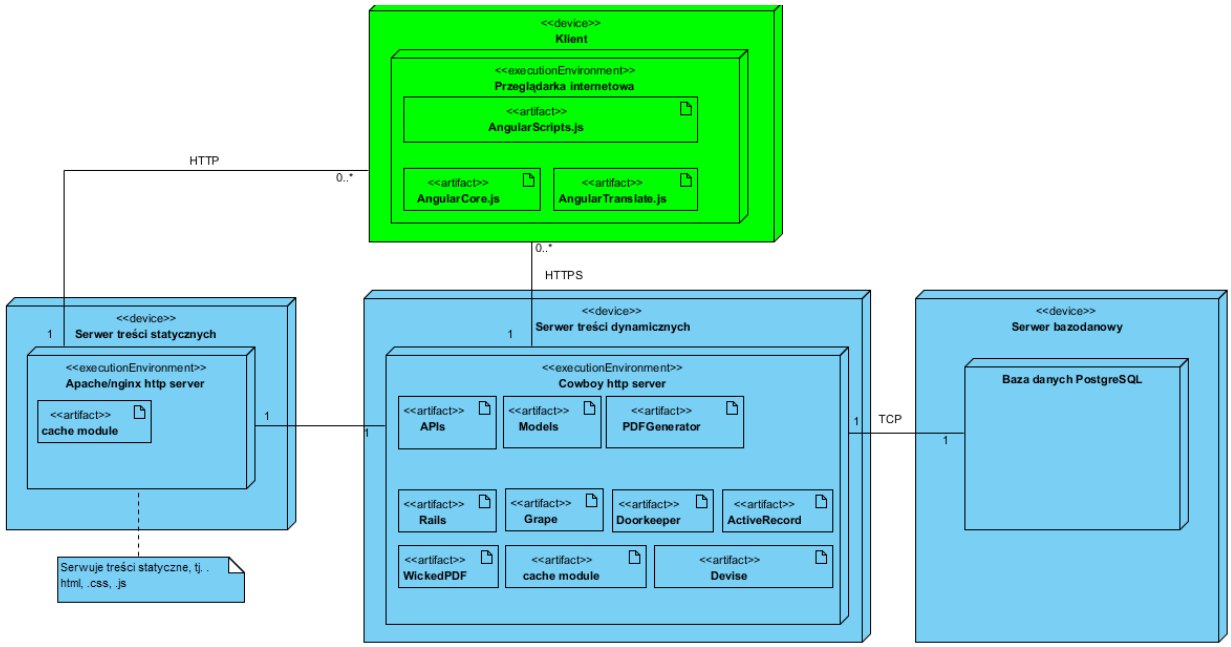
1. Diagram pakietów



2. Diagram komponentów



3. Diagram rozmieszczenia



Węzeł	Wymagania
Klient	Przeglądarka z włączoną obsługą Java Script
Serwer treści statycznych	Oprogramowanie: 1. Linux 2. Apache http server (wersja >2.4) lub Ngixn http server (wersja>1.9) Sprzęt: 1. Procesor serwerowy 2. Pamięć RAM > 4GB 3. UPS zapewniający zasilanie awaryjne na co najmniej 2 godziny 4. Dwie karty sieciowe 1Gbps
Serwer treści dynamicznych	Oprogramowanie: 1. Linux 2. Cowboy http server 3. Ruby 4. Rails Sprzęt: 1. Procesor serwerowy 2. Pamięć RAM > 4 GB 3. UPS zapewniający zasilanie awaryjne na co najmniej 2 godziny 4. Dwie karty sieciowe 1Gbps
Serwer bazodanowy	Oprogramowanie: 1. Postresql

	Sprzęt: 1. Procesor serwerowy 2. Macierz dyskowa RAID 0+1 z dysków twardych SATA 2 o pojemności co najmniej 200GB 3. UPS zapewniający zasilanie awaryjne na co najmniej 2 godziny 4. Dwie karty sieciowe 1Gbps

7. Realizacja przypadku użycia

Przypadek użycia opisujący stworzenie nowego kursu.