SOA

Wyszukiwarka kodów źródłowych w ogólnodostępnych repozytoriach

Jerzy Głowacki
Piotr Buchman

Cel systemu

Celem systemu jest zapewnienie możliwości wyszukiwania kodów źródłowych za pomocą wieloplatformowego klienta, który będzie korzystał z serwera będącego warstwą pośrednią, obsługującą żądania wysyłane przez klienta.
Docelowo, skalowalność systemu będzie ograniczona tylko przez możliwości API repozytoriów, które będziemy przeszukiwali aby zapewnić użytkownikom najdokładniejsze wyniki w możliwie najkrótszym czasie.

Udziałowcy

Użytkownicy zewnętrzni:

- Klient – osoba korzystająca z klienta SOAP, chcąca otrzymać połączone wyniki wyszukiwania w wielu systemach repozytoriów

Użytkownicy wewnętrzni:

- Administrator – osoba zajmująca się konfiguracją systemu na potrzeby klientów, mająca za zadanie maksymalną optymalizację algorytmów, pozwalających na indeksację, cache'owanie i przetwarzanie wyników wyszukiwania i dostarczania API, za pomocą którego klienci mogą uzyskać interesujące ich informacje.

Granice systemu

Granicą systemu dla wszystkich Użytkowników będzie interfejs klienta. Zapewniać będzie dostęp do wszystkich funkcjonalności zrealizowanych w systemie. Administrator będzie miał dostęp do panelu administracyjnego, za pomocą którego będzie miał dostęp do statystyk wykorzystania zasobów, szybkości działania oraz zbiorczych statystyk dotyczących historii wyszukiwań oraz zgromadzonych danych.

Wejścia w systemie:

- Formularz wyszukiwania w wieloplatformowym kliencie

Wyjścia w systemie:

- Lista wyników wyszukiwania w kliencie
- Statystyki serwera dostępne tylko dla administratorów w celach diagnostycznych
- Plik logu, zawierający informacje o wyjątkach występujących w trakcie działania systemu

Lista możliwości

Użytkownicy:

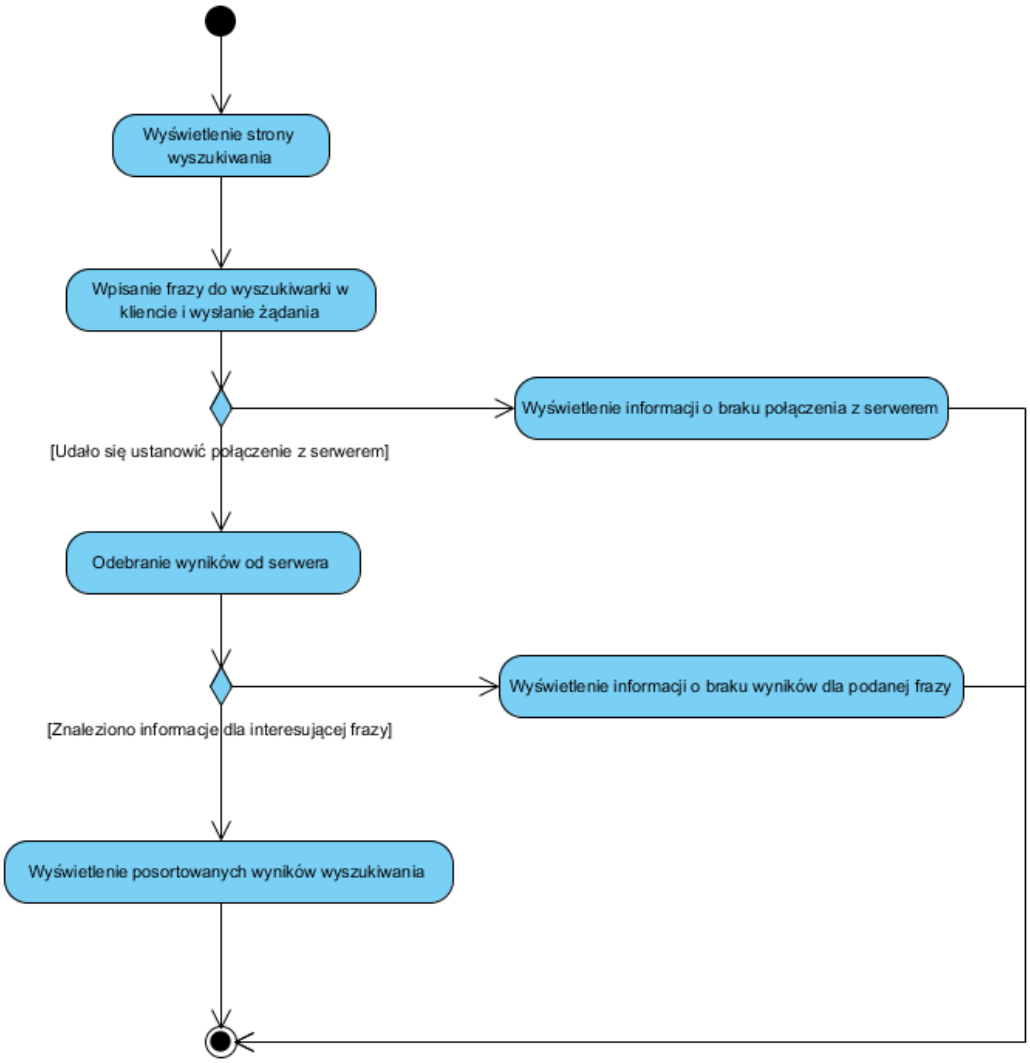
- Wyszukanie kodów źródłowych w systemach umożliwiających hostowanie repozytoriów

Administratorzy:

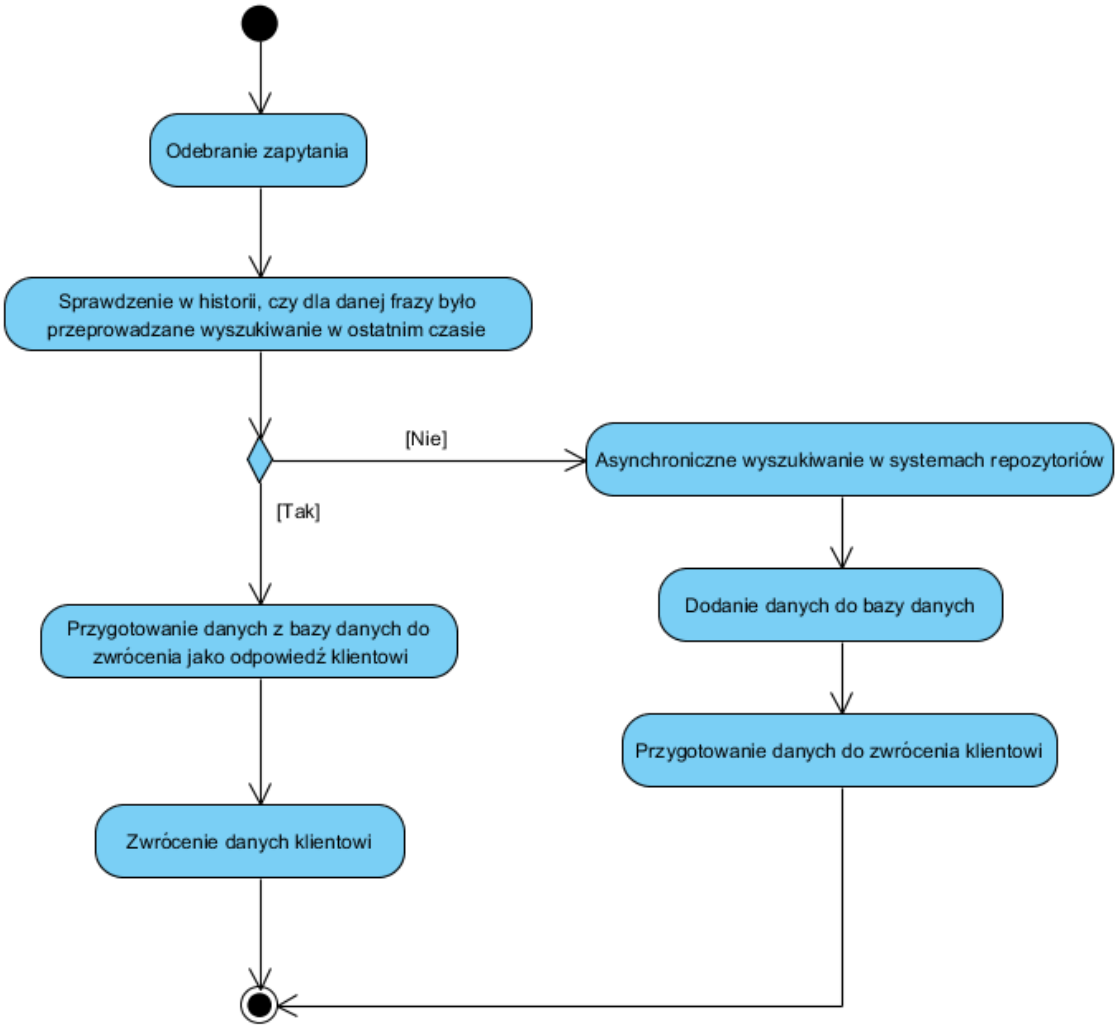
- Przeglądanie statystyk systemu
- Przeglądanie logów systemowych

Diagramy sekwencji wybranych czynności

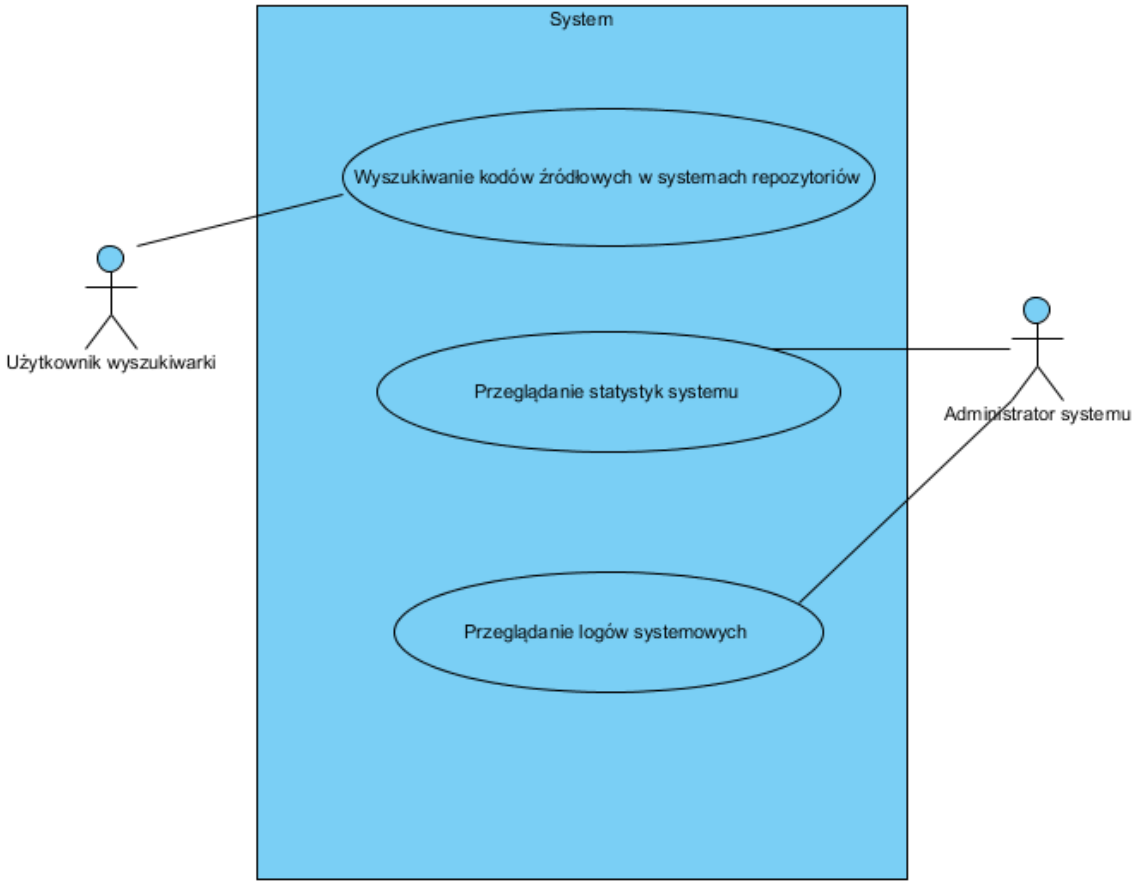
Klient - wyszukiwanie



Serwer – reakcja na zapytanie od klienta



Przypadki użycia systemu



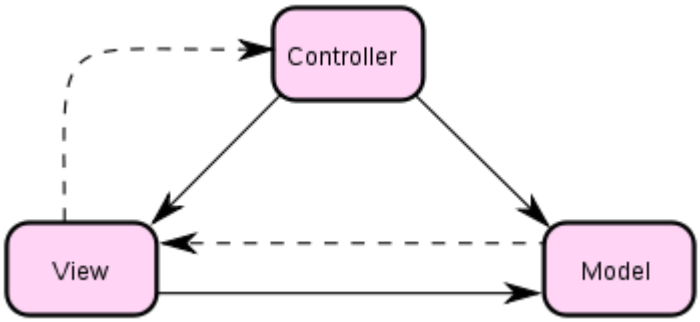
Architektura systemu

Aplikacja składać się będzie z dwóch głównych części – klienta, który będzie udostępniał użytkownikowi interfejs do wyszukiwania i przeglądania wyników zapytań oraz serwer, który będzie odpowiedzialny za dostarczanie danych w odpowiedzi na zapytania klienta.

Architektura serwera

Serwer utworzony będzie z zachowaniem zasad wzorca projektowego MVC, który wyodrębnia podstawowe trzy elementy aplikacji:

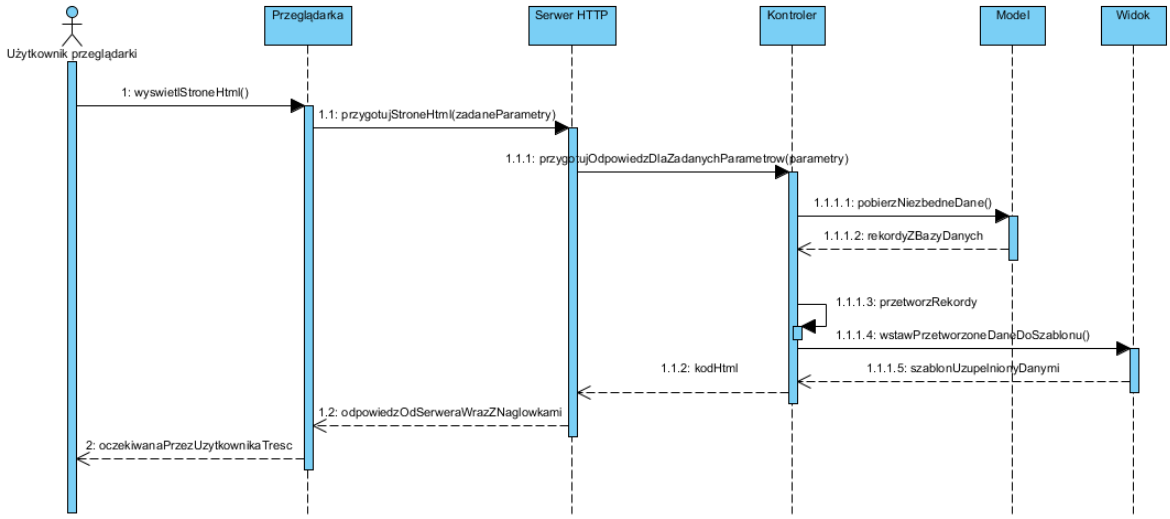
1. Warstwę logiki biznesowej (model)
2. Warstwę sterowania procesami w aplikacji (kontroler)
3. Warstwę prezentacji (widok)



Jest to obecnie najpopularniej stosowany sposób podziału aplikacji, który umożliwia jej łatwe rozwijanie i utrzymywanie. Przykładowo, gdy zapadnie decyzja o zmianie sposobu przechowywania danych, konieczne będzie tylko przepisanie kodu odpowiedzialnego za operacje na składowanych danych , bez żadnej ingerencji w logikę aplikacji czy też warstwę prezentacji wyników. Podobnie także w drugą stronę – zmiana systemu szablonów wykorzystywanego w systemie nie pociągnie za sobą konieczności modyfikacji warstwy sterowania a w szczególności modelu danych (pod warunkiem oczywiście, że wymieniane systemy szablonów będą posiadały interfejs kompatybilny z warstwą sterowania).

Działanie aplikacji, zapewnią serwer http, który także będzie warstwą umożliwiającą kontakt z wewnętrznym serwerem SOAP. W naszym przypadku, będziemy mieli dwa rodzaje widoków:

- Plik WSDL
- Interfejs dostępowy do statystyk dla administracji



1 - Ogólny schemat przepływu danych w aplikacji

Serwer, jak już zostało wspomniane wcześniej, będzie składała się z dwóch części, z których jedna będzie odpowiedzialna za prezentacje, przetwarzanie i składowanie danych w odpowiedzi na zapytania klienta, druga natomiast posłuży administratorom do kontroli działania aplikacji.

1. W relacyjnej bazie danych, przechowywać będziemy wszelkie informacje na temat wyników wyszukiwania, zapytań etc. Każdej klasie encji zawierającej dane, odpowiadać będzie klasa repozytorium, odpowiedzialna za operacje na większej ilości encji/rekordów bazodanowych. Zbiór tych modeli, uzupełniać będą fasady, zapewniające wygodny interfejs dostępowy do metod dla kontrolerów.
2. Wszelka walidacja, obróbka danych oraz kontrola dostępu do aplikacji, spoczywać będzie na kontrolerach, odpowiedzialnych także za przygotowywanie

informacji do wyświetlenia. Przykładowo, dane przychodzące będą walidowane i bindowane zostają właśnie przez kontrolery i to one wykorzystują do przygotowania odpowiedzi dla użytkownika.

3. W warstwie widoku prezentować będziemy dane z bazy (interfejs administratora) lub też informacje o dostępnych dla klientów możliwościach naszego serwera.

Przepływ oraz komunikacja pomiędzy przeglądarką użytkownika/klientem a aplikacją przebiegać będzie na bardzo standardowej drodze, którą zapewnia nam protokół HTTP 1.1. Jest to ogólnie przyjęty standard, którego specyfikacja nie należy do zakresu tego projektu. Wystarczy wspomnieć, że jest od obsługiwany na niskim poziomie abstrakcji przez wiodące Frameworki, umożliwiające nam dostęp do danych przesłanych zarówno metodami GET jak i POST na wyższym poziomie abstrakcji i po odpowiednim odfiltrowaniu, zabezpieczającym przed atakami ze strony użytkowników.

Wewnątrz aplikacji, wszelkie dane przekazywane będą jako parametry formalne do metod klas. Specyfikacja języka, w którym to oprogramowanie zostanie napisane (język PHP 5.3.9), zapewnia przekazywanie typów prostych oraz tablic przez kopię, obiekty natomiast przekazywane są przez referencję (bardziej szczegółowo: przekazywany jest identyfikator obiektu).

Serwer znajdować się będzie w **Amazon Elastic Compute Cloud (EC2),** który dostarczy nam skalowalną moc obliczeniową w chmurze obliczeniowej, zaprojektowany jest on w sposób umożliwiający jak najłatwiejsze korzystanie z tej mocy.

Na platformie Amazon można uruchamiać systemy operacyjne, jak i oprogramowanie użytkowe podlegające ochronie licencyjnej. Ich wykorzystanie jest możliwe w modelu opłaty za czas użycia i jest rozliczane razem z opłatą za użycie samej platformy. W ten sposób z obrazów AMI, bez kupowania licencji, uruchamia się odpowiednio prekonfigurowane oprogramowanie komercyjne, np. bazę MS SQL Server czy Oracle. Istnieje również możliwość dystrybucji i pobierania opłat za stworzone przez siebie oprogramowanie (usługa DevPay). Wymaga to jedynie przygotowania obrazu instancji z zainstalowanym oprogramowaniem oraz ustalenia wysokości opłat, jakie będą pobierane za czas wykorzystania naszego oprogramowania.

EC2 to prawdziwe wirtualne środowisko komputerowe pozwalające na uruchamianie instancji zawierających systemy operacyjne ładowane wraz z potrzebnymi aplikacjami. Umożliwia również pełne zarządzanie dostępem do naszych zasobów.

Wybrana przez nas Instancja:

- Micro Instance
- 613 MB memory
- Up to 2 EC2 Compute Units (for short periodic bursts)
- EBS storage only
- 64-bit platform
- I/O Performance: Low
- API name: t1.micro

W naszym wypadku, na maszynie wirtualnej zainstalowaliśmy Linuxa Ubuntu. Następnie skonfigurowaliśmy system operacyjny tak celem umożliwienia wykorzystania maszyny jako serwer HTTP.
Zainstalowana konfiguracja to:

- Apache 2.2
- PHP 5.3

Do zaimplementowania serwera, wykorzystamy jako główne narzędzie Framework **Symfony 2**, który ułatwia tworzenie aplikacji internetowych w zupełnie nowy sposób.

Głównym celem frameworka jest zapewnienie separacji zagadnień. Dzięki temu organizacja kodu pozwala aplikacji łatwo się rozrastać w czasie, unikając mieszania wywołań, zapytań bazy danych, HTML i logiki biznesowej w tym samym skrypcie. Framework jest bardzo przydatny, jest zadatkiem jakości, rozbudowy i utrzymania aplikacji przy niższych kosztach. Pozwala zaoszczędzić czas poprzez wykorzystanie modułów. Inwestujemy w zadanie a nie technologię.

Symfony 2 jest zintegrowany z Doctrine 2, biblioteką która daje potężne narzędzia aby ułatwić utrzymanie i czytanie informacji z i do bazy danych. Doctrine ORM pozwala zmapować obiekty na podstawie meta danych do relacyjnej bazy danych tj. MySQL, PostgreSQL, Microsoft SQL.

W skład Doctrine 2 wchodzą:

- **Doctrine DBAL** – warstwa abstrakcji bazy danych oparta na PHP Data Objects,
- **Doctrine ORM** – biblioteka ORM,
- **Doctrine MongoDB** – biblioteka do obsługi nierelacyjnej bazy danych MongoDB,
- **Doctrine Migrations** – narzędzie do migracji baz danych.

Wersja 2.x rozszerza PHP Data Objects o dodatkowe funkcjonalności oraz wprowadza tzw. mechanizm platform, które opisują możliwości i różnice między dialektami języka SQL używanymi przez różne systemy zarządzania bazą danych.

Na bazie DBAL zbudowana jest biblioteka ORM.

Jeśli chodzi o architekturę obiekty reprezentujące wiersze bazy danych nie są już nazywane modelami, lecz encjami.

Klasa encji może być dowolną klasą. Biblioteka nie narzuca tutaj żadnych klas bazowych koniecznych do rozszerzenia czy interfejsów – pola opisywane są poprzez mechanizm adnotacji.

Porzucono także wykorzystywanie elementów magicznych języka PHP na rzecz menedżera encji (entity manager), który zarządza obiektami encji w sposób jawny.

Do testowania aplikacji, będziemy korzystać z Framework **PHPUnit**. Został on stworzony do automatycznego testowania oprogramowania napisanego w języku programowania PHP przy pomocy testów jednostkowych. W celu sprawdzenia czy poszczególne komponenty aplikacji działają zgodnie z oczekiwaniami, należy stworzyć aplikację przygotowującą zestaw testów jednostkowych. PHPUnit wykonuje po kolei wszystkie testy i po zebraniu wyników, przedstawia raport o ilości zliczonych testów. Zawiera szczegółowe informacje o nieprawidłowych wynikach oraz błędach wykonania. Dzięki niemu jest łatwiejszy rozwój i zarządzanie aplikacji.

Klient

Aplikacja kliencka zostanie napisana z użyciem technologii JS/HTML/CSS. Do zapewnienia możliwości uruchamiania jej na wielu platformach, wykorzystamy **PhoneGap.**

Obsługa protokołu SOAP zostanie nam zagwarantowana przez **jQuery SOAP Client,** w zależności o stopnia skomplikowania integracji tych bibliotek z frameworkiem **jQuery Mobile**, który mamy zamiar wykorzystać jako główne narzędzie w implementacji klienta.

Obsługiwane systemy repozytoriów

CodeSearch - mimo zamknięcia wciąż działa pod adresem http://code.google.com/codesearch

URL:
http://code.google.com/codesearch/json?search_request=b&query={$query}+package:{$package}+lang:^{$lang}$&max_num_results=11&results_offset=0&exhaustive=false&return_snippets=true&return_all_snippets=false&return_all_duplicates=false&return_line_matches=false&sort_results=false&lines_context=1&return_directories=true&search_request=e

Zwraca: JSON

Github - jest API, ale nie pozwala na wyszukiwanie kodu, wiec trzeba parsować HTML.

URL:
https://github.com/search?utf8=%E2%9C%93&q={$query}&repo=&langOverride=&start_value=1&type={Code|Repositories}&language={$lang}

Zwraca: HTML

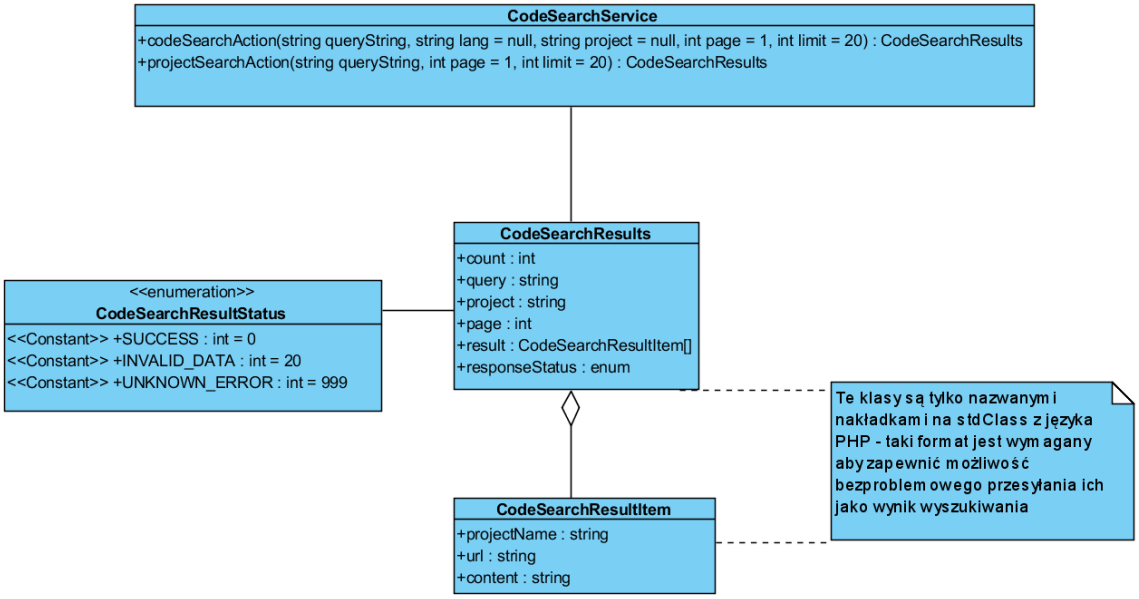
Koders

URL:
http://www.koders.com/default.aspx?s={$query}&submit=Search&la={$lang}&li=*

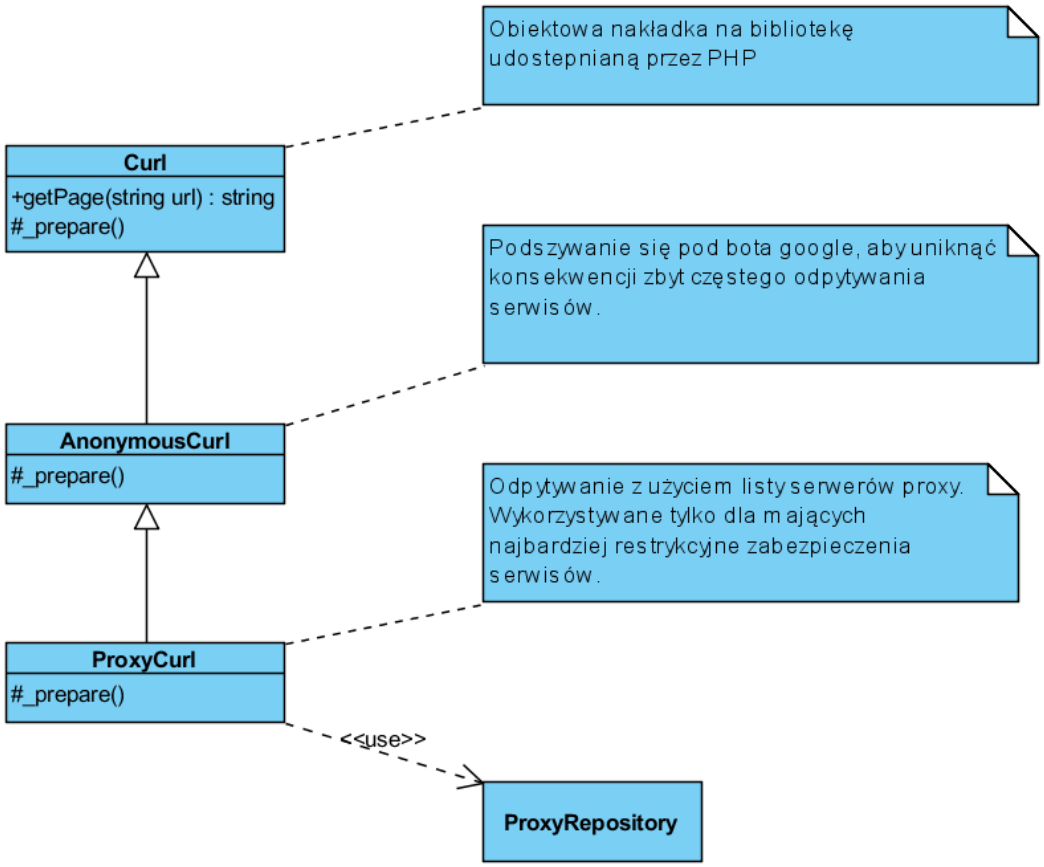
Zwraca: HTML

Główne założenia implementacyjne

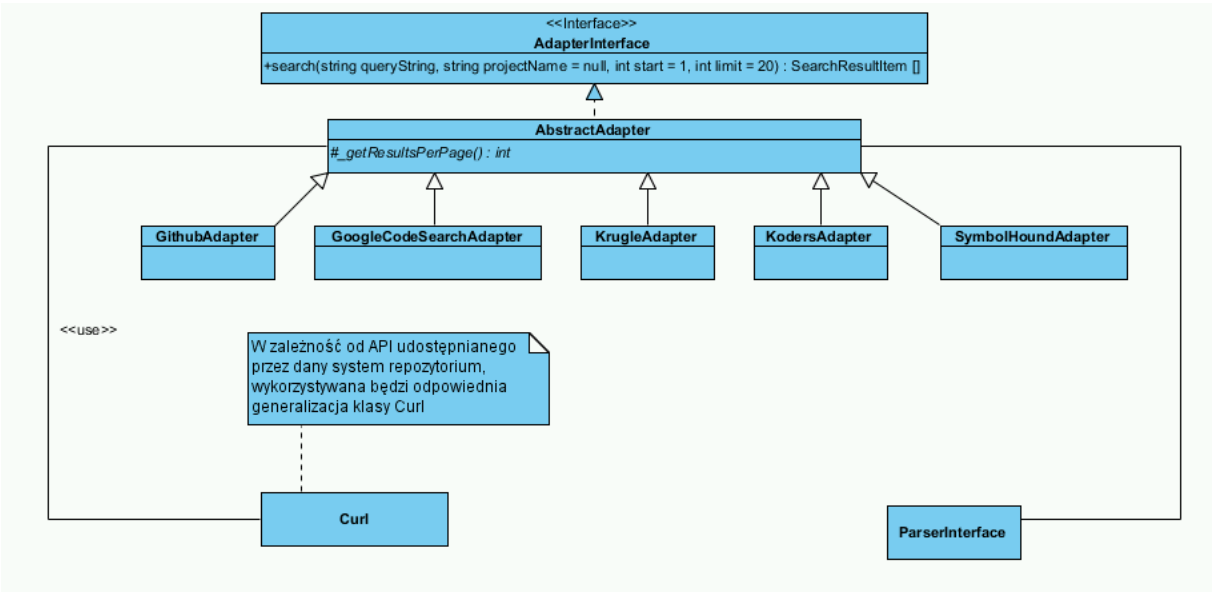
Główną klasą, udostępniającą wszystkie publiczne metody, będące operacjami WSDL, będzie *CodeSearchServiceFacade*.



Do pobierania wyników z systemów repozytoriów, gdzie nie ma dostępnego API, będziemy korzystać z następujących klas:



Proponowany przez nas system adapterów do wyszukiwarek



Funkcjonalności zapewniane przez fasadę do cache'owania wyników wyszukiwań