Uniwersytet Warszawski
Wydział Matematyki, Informatyki i Mechaniki

Marek Sapota
Nr albumu: 262969

Yummy — internetowy serwis kulinarny

Praca licencjacka
na kierunku JSIM

Praca wykonana pod kierunkiem
dra Jacka Sroki
Instytut Informatyki

18 czerwca 2010

Oświadczenie kierującego pracą

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Data Podpis kierującego pracą

Oświadczenie autora (autorów) pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Data Podpis autora (autorów) pracy

Streszczenie

Treścią pracy jest przedstawienie i omówienie serwisu internetowego Yummy, stworzonego jako projekt licencjacki. Serwis ten umożliwia wyszukiwanie i edytowanie informacji związanych z kulinariami i żywieniem. Ponadto dostosowuje on swoje działanie do indywidualnych gustów użytkowników, uwzględniając je na przykład przy polecaniu nowych przepisów. Praca zawiera opis procesu tworzenia i testowania serwisu, ze szczególnym uwzględnieniem niestandardowych rozwiązań umożliwiających współpracę Yummy z serwisami zewnętrznymi oraz algorytmu realizującego sztuczną inteligencję.

Słowa kluczowe

serwis internetowy, JavaScript, przepisy kulinarne, Ruby on Rails, sztuczna inteligencja, collaborative filtering, machine learning

Dziedzina pracy (kody wg programu Socrates-Erasmus)

11.3 Informatyka

Klasyfikacja tematyczna

H. Information Systems
H.3 INFORMATION STORAGE AND RETRIEVAL
H.3.5 Online Information Services
Web-based services

Tytuł pracy w języku angielskim

Yummy — an Internet culinary service

Spis treści

Wprowadzenie

Celem pracy jest stworzenie ogólnodostępnej bazy wiedzy o tematyce kulinarnej, między innymi o: daniach, składnikach, napojach, przepisach czy restauracjach, tworzonej przez jej użytkowników. Głównymi walorami serwisu powinny być: darmowe korzystanie z zasobów, mechanizm sugerowania przepisów kulinarnych oraz możliwość modyfikowania treści przez użytkowników, zrealizowana na zasadzie działania stron typu Wiki [6]. System, któremu nadano nazwę "Yummy", integruje i korzysta z wielu współczesnych sposobów udostępniania wiedzy. System napisano z użyciem nowoczesnych technologii oraz popularnych wzorców projektowych takich jak Ruby on Rails [29], JRuby [22], JSONP [18] czy jQuery [20].

Struktura dokumentu

W niniejszym dokumencie znajdują się następujące rozdziały i dodatki:

- *Podstawowe pojęcia* (patrz Rozdział 1): Rozdział wstępny, przedstawiający czytelnikowi serwis i problem, jaki ten serwis ma za zadanie rozwiązać. Znajduje się tutaj również objaśnienie nazewnictwa występującego w późniejszej części dokumentu.
- *Przypadki użycia i wymagania niefunkcjonalne* (patrz Rozdział 2): Rozdział opisujący założenia przyjęte przy projektowaniu serwisu Yummy, przedstawione tutaj jako scenariusze przypadków użycia.
- *Projekt* (patrz Rozdział 3): Rozdział, w którym zawarte są najważniejsze elementy projektu technicznego serwisu, między innymi podział na komponenty i moduły oraz model danych.
- *Implementacja (patrz Rozdział 4*: Opis najciekawszych elementów implementacji serwisu, nietrywialnych problemów napotkanych podczas realizowania niektórych komponentów oraz przyjętych rozwiązań.
- *Testy* (patrz Rozdział 5): Dokumentacja procedury testowania serwisu.
- *Podsumowanie* (patrz Rozdział 6): Wyjaśnienia, co zostało zrobione w pracy oraz wnioski podsumowujące roczną pracę nad serwisem.
- *Instalacja i konfiguracja* (patrz Dodatek A): Krótka instrukcja instalacji i administracji serwisu.
- *Podział i harmonogram pracy* (patrz Dodatek B): Kalendarium rocznej pracy nad serwisem i niniejszym dokumentem.
- *Opis zawartości płytki dołączonej do pracy* (patrz Dodatek C).

Rozdział 1

Podstawowe pojęcia

W niniejszym rozdziale przedstawione są definicje skrótów i specyficznych pojęć z zakresu tematyki niniejszej pracy, które są używane w późniejszej części dokumentu. Następnie zdefiniowany jest problem, który omawiana aplikacja ma za zadanie rozwiązać.

1.1. Definicje

Definicja 1.1.1 Yummy *(pot. przepyszny) — nazwa tworzonego serwisu.*

Definicja 1.1.2 Sieć 2.0 *(ang. Web 2.0) [28] — klasa dynamicznych serwisów internetowych, w których działaniu podstawową rolę odgrywa treść generowana przez ich użytkowników.*

Definicja 1.1.3 Aplikacja internetowa — *aplikacja komunikująca się z użytkownikiem za pośrednictwem przeglądarki internetowej.*

Definicja 1.1.4 Serwis społecznościowy — *rodzaj interaktywnych stron WWW, których zawartość jest współtworzona przez grupy osób posiadających wspólne zainteresowania lub chcących poznać zainteresowania innych.*

Definicja 1.1.5 Model-Widok-Kontroler *(ang. Model-View-Controller, MVC) [5] — architektoniczny wzorzec projektowy często używany w aplikacjach internetowych w celu zwiększenia ich czytelności oraz łatwiejszego wprowadzania zmian.*

Definicja 1.1.6 Sztuczna inteligencja — *dział informatyki, którego przedmiotem jest badanie reguł rządzących inteligentnymi zachowaniami człowieka, tworzenie modeli formalnych tych zachowań i w rezultacie programów komputerowych symulujących te zachowania, terminem tym są również określane same komponenty programu o takich cechach.*

Definicja 1.1.7 Interfejs programowania aplikacji *(ang. Application Programming Interface, API) [23] — specyfikacja procedur, funkcji lub interfejsów umożliwiających komunikację z biblioteką, systemem operacyjnym lub innym systemem zewnętrznym.*

Definicja 1.1.8 GNU Affero General Public License w wersji 3 lub wyższej *(GNU AGPLv3+) [10] — wersja szeroko stosowanej licencji GNU General Public License [11] przystosowana do aplikacji sieciowych.*

Definicja 1.1.9 Creative Commons Attribution-ShareAlike 3.0 Unported License *(CC-BY-SA) [8] — popularna licencja przystosowana do treści innych niż oprogramowanie, np. zdjęcia, filmy czy dokumenty tekstowe.*

Definicja 1.1.10 Mapowanie Obiektowo-Relacyjne *(ang. Object-Relational Mapping, ORM) [1] — technika programistyczna służąca do konwersji danych pomiędzy relacyjnymi bazami danych a obiektowymi językami programowania.*

Definicja 1.1.11 Wiki *[6] — klasa stron internetowych, które można nie tylko oglądać, ale też tworzyć, edytować i zmieniać. Nazwą tą określa się również oprogramowanie umożliwiające wspólną pracę wielu użytkowników przy tworzeniu zawartości takich stron.*

Definicja 1.1.12 MediaWiki *[46] — oprogramowanie do tworzenia stron typu Wiki wykorzystywane przede wszystkim przez serwisy Fundacji Wikimedia [14].*

Definicja 1.1.13 Strukturalny Język Zapytań *(ang. Structured Query Language, SQL) [49] — strukturalny język zapytań używany do tworzenia i modyfikowania relacyjnych baz danych.*

Definicja 1.1.14 Zastrzyk SQL *(ang. SQL injection) [51] — luka w zabezpieczeniach aplikacji internetowych polegająca na nieodpowiednim filtrowaniu lub niedostatecznym typowaniu i późniejszym wykonaniu poleceń przesyłanych w postaci zapytań SQL w bazie danych.*

Definicja 1.1.15 Obiektowa Notacja JavaScript *(ang. JavaScript Object Notation, JSON) [38] — format wymiany danych komputerowych będący podzbiorem języka JavaScript, dzięki czemu można w prosty sposób wykorzystać go w kodzie napisanym w tym języku. Pomimo powyższej właściwości format ten jest uznawany za uniwersalny i niezależny od języka programowania.*

Definicja 1.1.16 Objektowa Notacja JavaScript z Obiciem *(ang. JSON with Padding, JSONP) [18] — format wymiany danych komputerowych umożliwiający obejście restrykcyjnej kontroli "same origin" [43] wymuszanej przez przeglądarki internetowe.*

Definicja 1.1.17 Zwierciadło *(ang. Mirror) — dokładna kopia strony internetowej lub innych zasobów.*

1.2. Postawienie problemu

Istnieje wiele stron z przepisami kulinarnymi, jednakże bardziej przypominają one klasyczne książki kucharskie — nie ma możliwości podania swoich preferencji, np. co do składników czy rodzaju kuchni. Często potrzeba spędzić wiele czasu na szukaniu w Internecie żądanego przepisu i, na przykład, serwującej to danie restauracji. Za pomocą Yummy ma być to możliwe przy użyciu zaledwie jednego, dwóch zapytań. Yummy również poleca potrawy na podstawie wybranych kryteriów, wcześniejszych wyborów użytkownika i popularnych wyborów innych użytkowników. Przykładowo wielu użytkowników lubiących parówki lubi także kiełbasę, więc użytkownikowi szukającemu parówek polecone zostaną też przepisy z kiełbasą w liście składników.

1.3. Kontekst produktu

Głównym celem aplikacji jest umożliwienie wyszukiwania informacji jednocześnie w kilku serwisach zewnętrznych oraz wyświetlania wyszukanych wyników w kolejności potencjalnie najbardziej odpowiadającej gustom użytkownika (inteligentne wyszukiwanie). Serwis pozwala także na edycję danych zgromadzonych w serwisach zewnętrznych i dostarcza jednolity interfejs dostępu do informacji zgromadzonych w różnych źródłach.

Rozdział 2

Przypadki użycia i wymagania niefunkcjonalne

Niniejszy rozdział zawiera informacje o założeniach, które zostały poczynione przez twórców w trakcie projektowania aplikacji. Założenia te zostały zebrane w postaci nieformalnego opisu przypadków użycia i wymagań niefunkcjonalnych.

2.1. Przypadki użycia

Przypadki użycia są opisane poniżej i przedstawione na Rysunku 2.1.

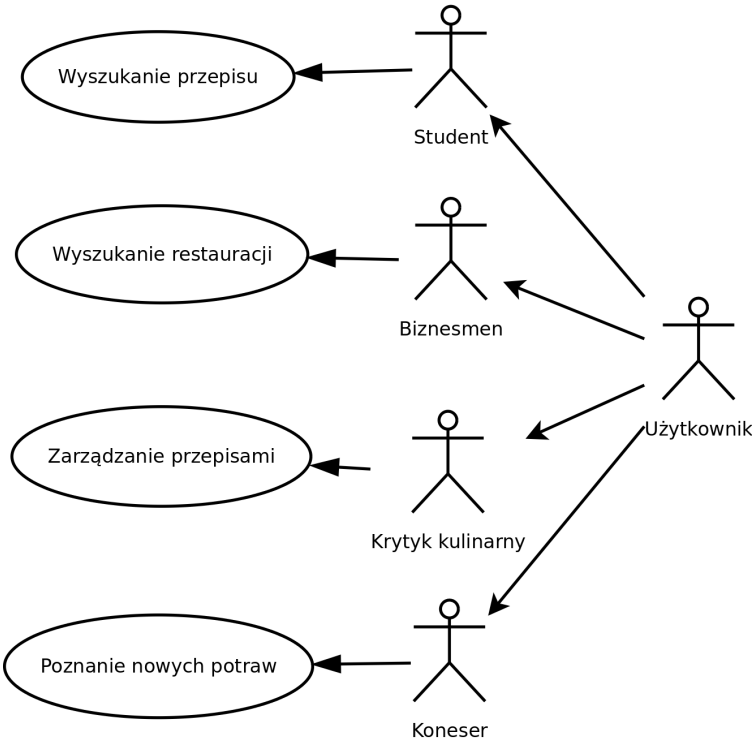
2.1.1. Aktorzy w systemie

Każdy użytkownik systemu zawsze występuje jako jeden z poniższych aktorów:

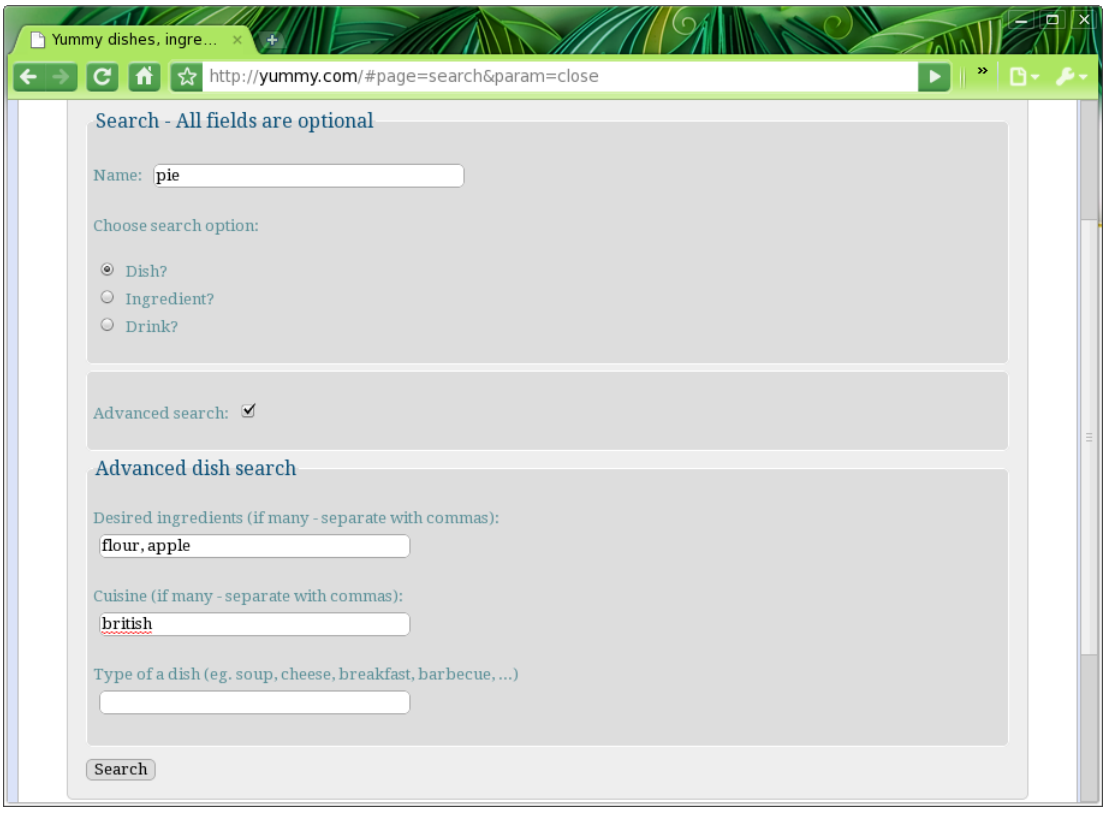
- **Student** - wyszukuje przepisy do interesujących go potraw,
- **Biznesmen** - wyszukuje restauracje, w których można zjeść dane potrawy,
- **Krytyk kulinarny** - uaktualnia i dodaje nowe informacje w serwisie,
- **Koneser** - poznaje nowe potrawy odpowiadające jego gustowi.

2.1.2. Wyszukanie przepisu

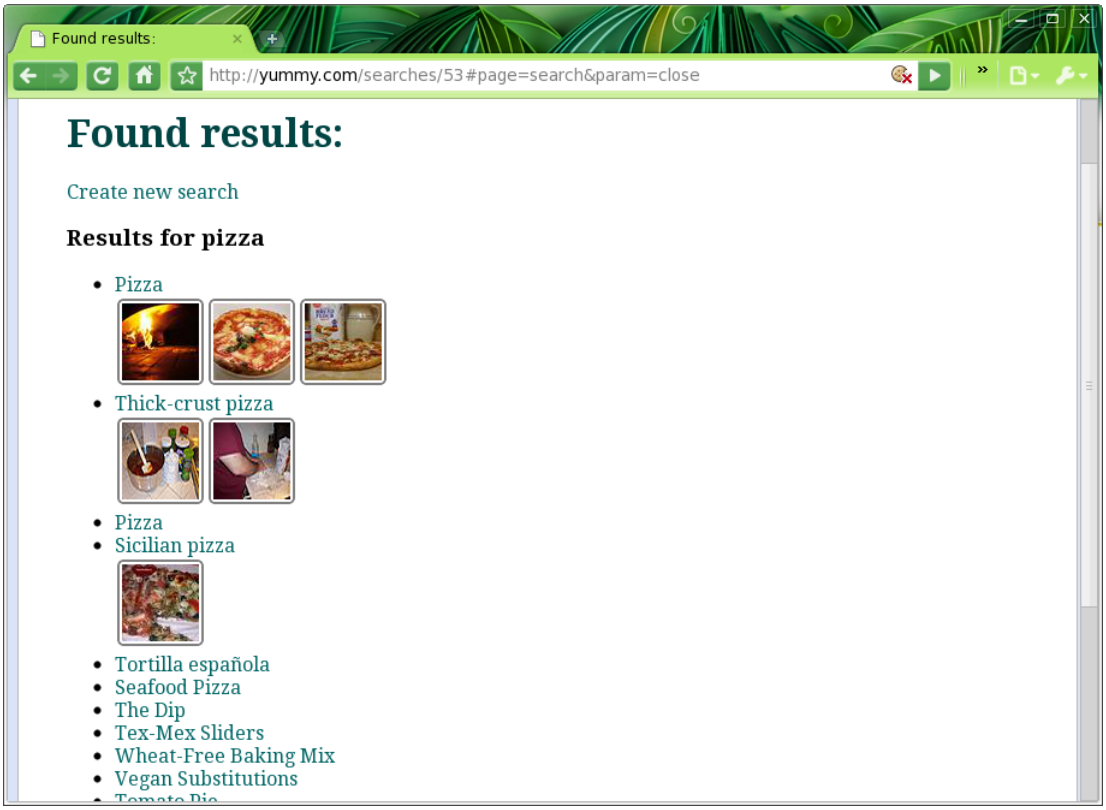
- **Główny aktor:** Student
- **Cel:** student chce znaleźć przepis na potrawę spełniającą jego kryteria
- **Scenariusz:** Student wyszukuje wszystkie potrawy spełniające jego wyrafinowane kryteria (patrz Rysunek 2.2). Następnie przegląda różne przepisy przygotowania tych potraw, w celu wybrania najbardziej mu odpowiadającego (patrz Rysunek 2.3). Może przy tym kierować się ocenami i opiniami pozostałych klientów systemu.



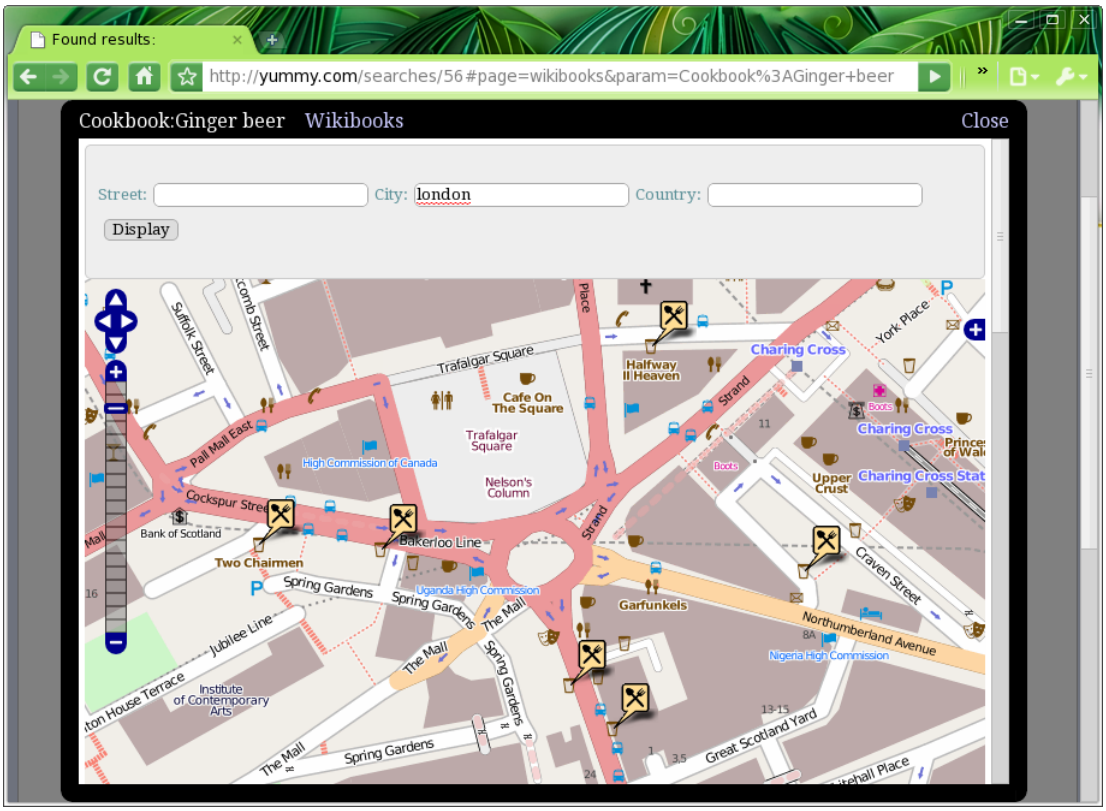
Rysunek 2.1: Diagram przypadków użycia



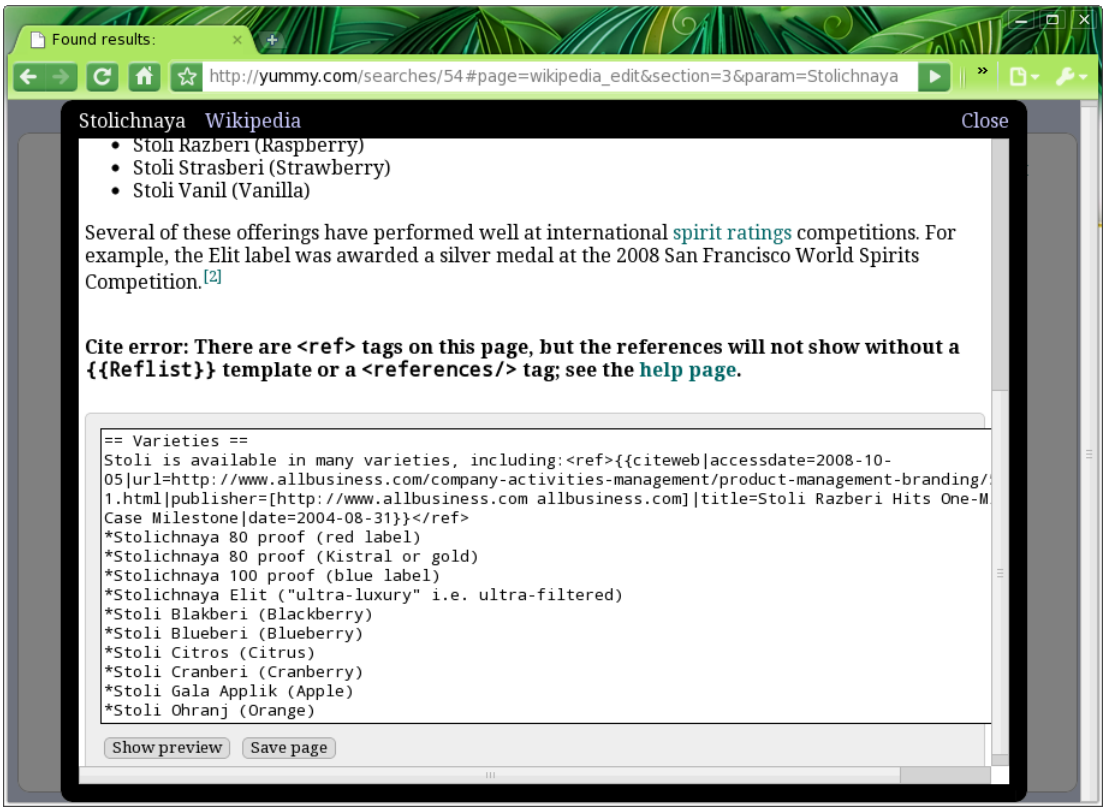
Rysunek 2.2: Formularz wyszukiwania



Rysunek 2.3: Wyniki wyszukiwania



Rysunek 2.4: Wyniki wyszukiwania restauracji serwujących wybraną potrawę



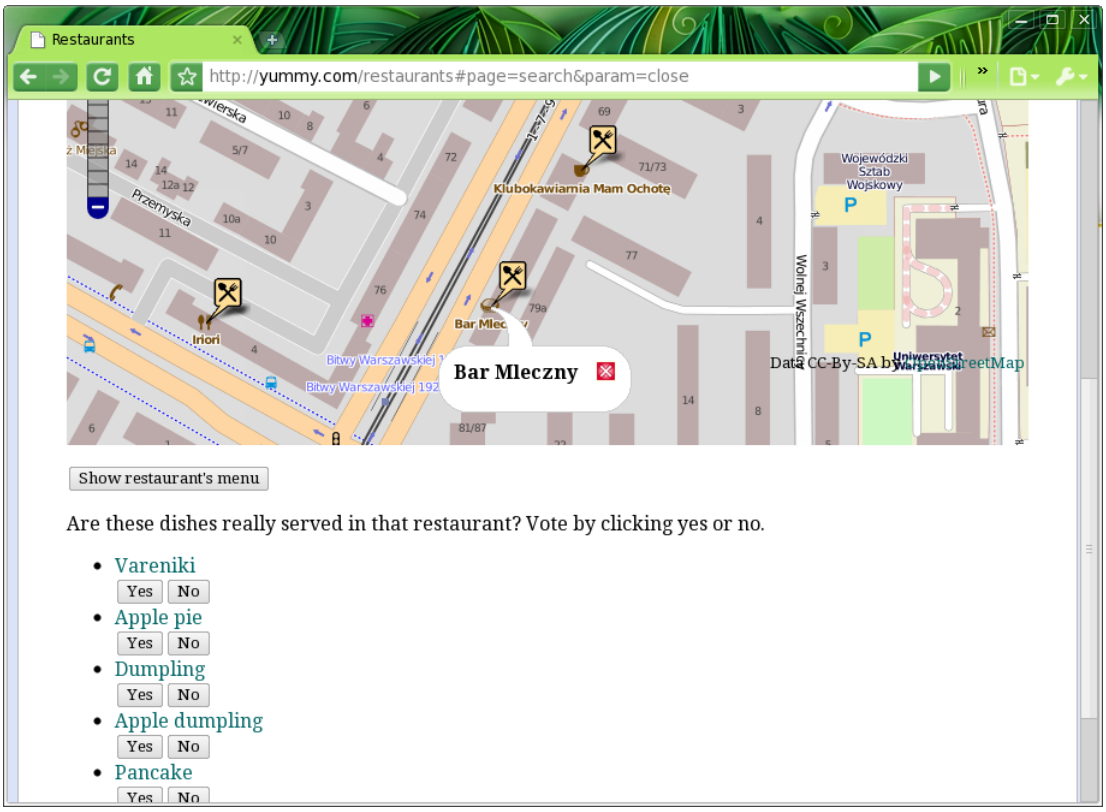
Rysunek 2.5: Edycja artykułu

2.1.3. Wyszukanie restauracji

- **Główny aktor:** Biznesmen
- **Cel:** Biznesmen chce zjeść w restauracji potrawę spełniającą jego kryteria
- **Scenariusz:** Jeżeli Biznesmen chciałby zjeść wybraną przez siebie potrawę poza domem, może on zobaczyć restauracje, w których jest ona serwowana. Mając do dyspozycji mapę znajduje w dowolnym miejscu na świecie, restauracje które zostały oznaczone jako serwujące wybrane danie (patrz Rysunek 2.4). Po wybraniu restauracji, dzięki głosom oddanym przez innych użytkowników systemu, łatwo ocenia czy dana potrawa rzeczywiście jest w tej restauracji serwowana.

2.1.4. Zarządzanie przepisami

- **Główny aktor:** Krytyk kulinarny
- **Cele:** Krytyk kulinarny chce uzupełnić informacje dotyczące danego przepisu
- **Scenariusz:** Krytyk kulinarny po wybraniu przepisu ocenia jakość artykułu oraz podaje czy ta potrawa jest zgodna z jego gustem. Ma też możliwość edycji dowolnego artykułu oraz, w razie gdyby przepis nie istniał w żadnym serwisie zewnętrznym, stworzenia nowego przepisu i samodzielnego opisania go (patrz Rysunek 2.5). Jeżeli Krytyk



Rysunek 2.6: Edycja menu restauracji

kulinarny wie, w których restauracjach jest serwowana dana potrawa, zapisuje te informacje w systemie. Oprócz tego, może zweryfikować czy wybrana potrawa rzeczywiście jest serwowana w podanych restauracjach (patrz Rysunek 2.6).

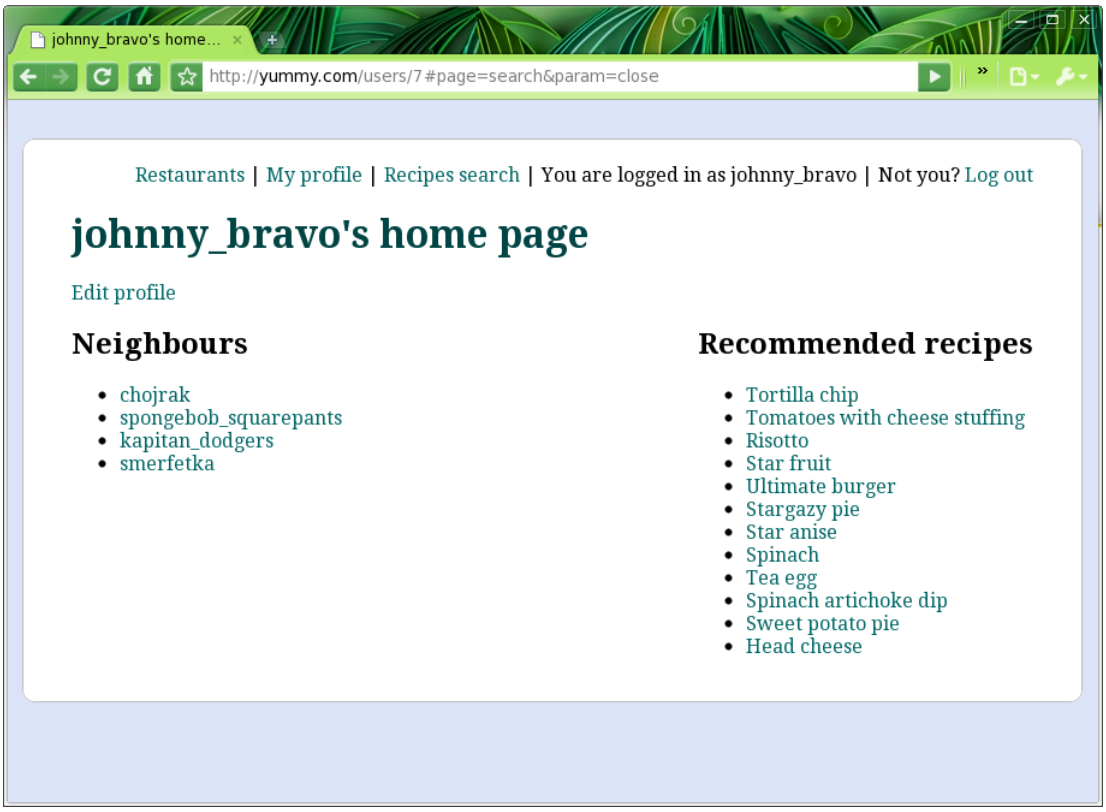
2.1.5. Poznanie nowych potraw

- **Główny aktor:** Koneser
- **Cele:** Koneser chce poznać nowe potrawy zgodne z jego gustem
- **Scenariusz:** Koneser na stronie swojego profilu widzi listę potraw polecanych mu przez system, które zostały uznane za najbardziej pasujące do jego gustów. Może też przeglądać profile innych użytkowników systemu, którzy mają najbardziej zbliżone preferencje żywieniowe (patrz Rysunek 2.7).

2.2. Wymagania niefunkcjonalne

- **Przenośność aplikacji klienta**

  Serwis powinien działać niezależnie od doboru przeglądarki internetowej, czy systemu operacyjnego, na którym ta przeglądarka jest uruchomiona. Aplikacja ma gwarantować pełną funkcjonalność w przeglądarkach korzystających z silnika Gecko [36] i WebKit [9]. Wymaganiem jest, aby aplikacja klienta składała się wyłącznie z HTML [47] + CSS [50] i JavaScriptu [48]. Nie jest wymagana wysoka przenośność aplikacji serwera.



Rysunek 2.7: Polecane potrawy

- **Wydajność serwisu**

  Czas reakcji systemu na zapytania użytkownika w 90% przypadków nie powinien przekraczać 5 sekund, nawet dla maksymalnie skomplikowanych zapytań. Przy dużej liczbie jednoczesnych zapytań, czas reakcji może się wydłużyć, jednak w 90% przypadków nie powinien przekroczyć 10 sekund nawet dla 1000 jednoczesnych zapytań. Duży nacisk należy położyć na skalowalność serwisu i możliwość jak największego zrównoleglenia wykonywanych operacji.

Rozdział 3

Projekt

W rozdziale tym przedstawione są wyniki kolejnego etapu prac nad systemem, czyli elementy projektu technicznego, zawartego w dokumencie "Dokumentacja architektury oprogramowania" (patrz Dodatek C). W pierwszej części opisane są główne założenia projektowe, w kolejnej — dekompozycja systemu i podział na moduły jak również przedstawienie struktury bazy danych aplikacji.

3.1. Główne założenia projektowe

- Uwzględnienie preferencji użytkownika przy wyszukiwaniu i sugerowaniu przepisów — branie pod uwagę gustów użytkownika, wnioskowanych na podstawie historii jego działań.

- Korzystanie z zewnętrznych serwisów typu Wiki do przechowywania właściwych danych, tzn. zawartości artykułów o potrawach. Na serwerze serwisu Yummy powinny być przechowywane tylko dane niedostępne na innych serwisach — o użytkownikach, oceny przepisów itp.

- Przerzucenie jak największej części pracy serwisu na aplikację klienta i odciążenie serwera poprzez minimalizację przesyłu informacji pomiędzy serwerem a klientem.

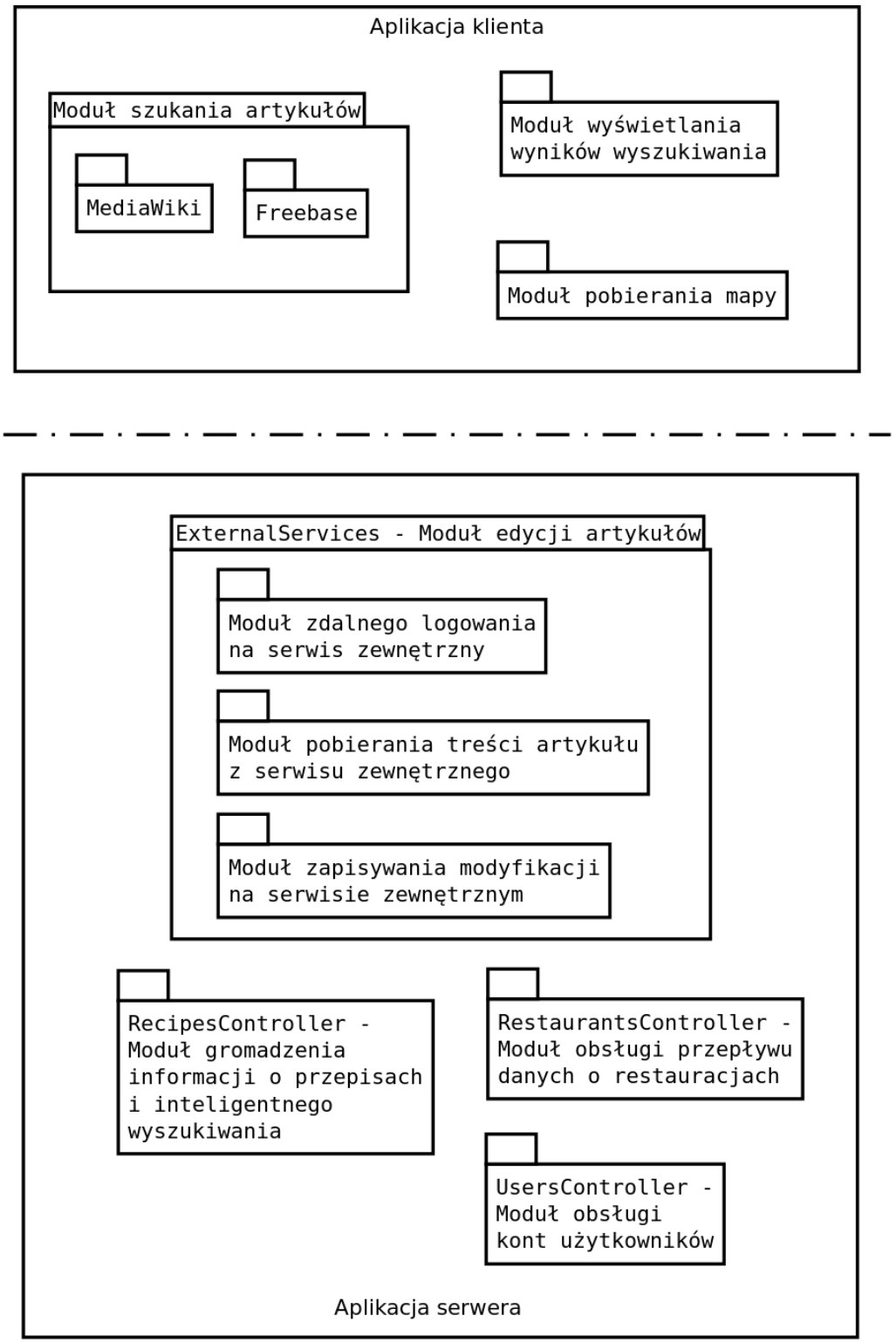
3.2. Podział na moduły

Serwis Yummy został napisany z myślą o możliwości łatwego rozszerzania. Z tego powodu podzielono aplikację na praktycznie niezależne od siebie moduły. Takie podejście pozwala na zmianę jednego z modułów bez potrzeby dokonywania zmian w innych, a także na proste dodawanie nowych funkcjonalności.

Najbardziej ogólnym i naturalnym podziałem systemu jest podział na aplikacje klienta i serwera. Struktura każdej z tych składowych przedstawiona jest na Rysunku 3.1 i opisana w poniższych podrozdziałach.

3.2.1. Aplikacja klienta

Aplikacja klienta sklada się z następujących modułów:



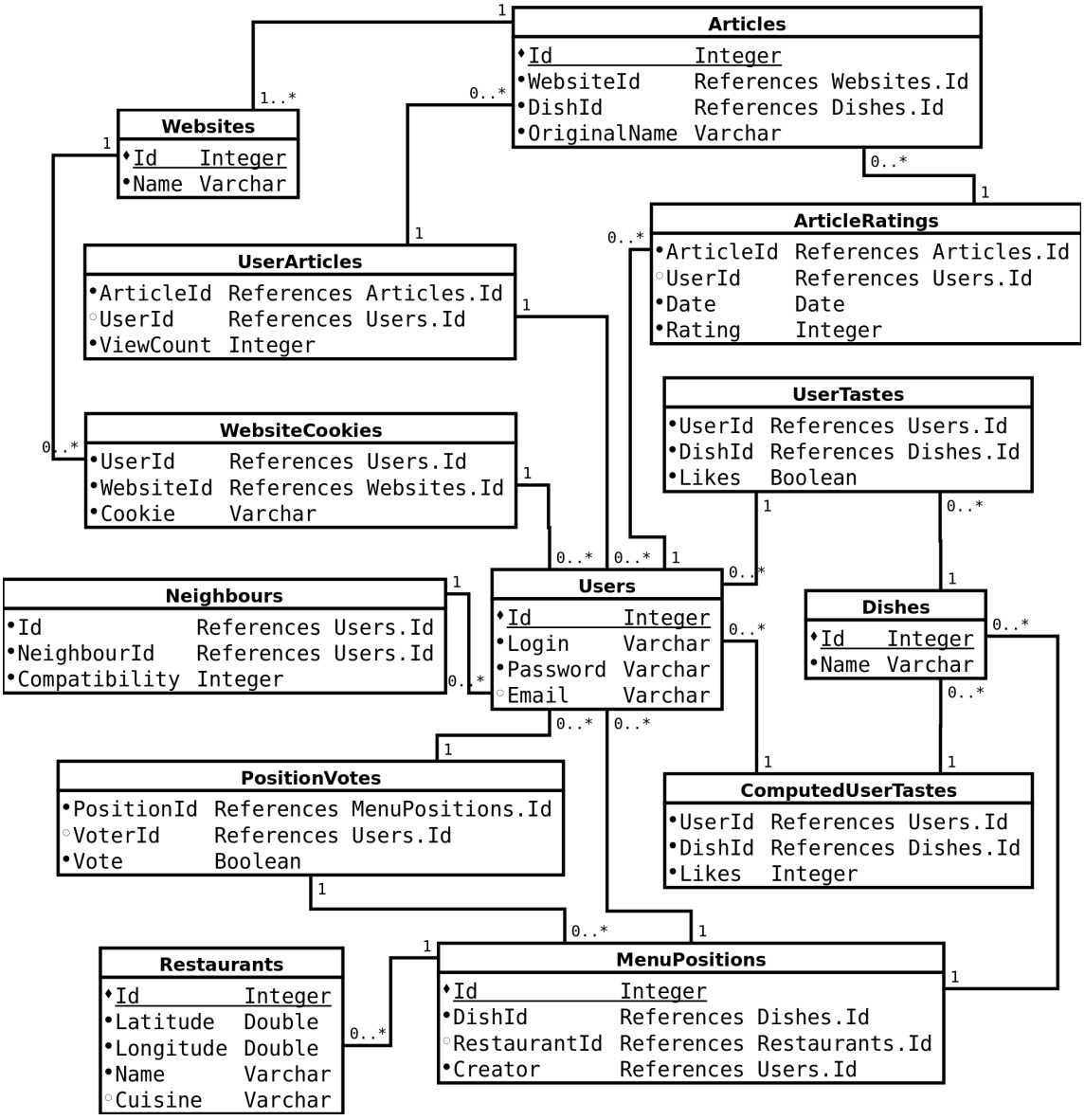
Rysunek 3.1: Podział systemu na moduły

- Moduł szukania artykułów składa się z podmodułów obsługujących wyszukiwanie danych w poszczególnych serwisach zewnętrznych. Implementacja umożliwia proste dodawanie nowych modułów wyszukiwania bez potrzeby wprowadzania zmian w innych częściach systemu. Zaimplementowane są następujące podmoduły:
  - Freebase — moduł korzystający z API strony FreeBase.com [16], umożliwiającej wymianę treści w formacie JSON [38] i JSONP [18], przypominającej zapytania i odpowiedzi SQL [49]. Strona udostępnia mechanizm MQL - Metaweb Query Language [15].
  - MediaWiki — moduł korzystający z API oprogramowania MediaWiki [27], umożliwiającego wyszukiwanie przy wykorzystaniu formularzy. Odpowiedzi są wysyłane w formacie JSONP.
- Moduł wyświetlający przefiltrowane wyniki wyszukiwania, uszeregowane po stronie serwera.
- Moduł pobierania mapy — używany przy wskazywaniu lokalizacji wyszukanych restauracji, korzystający z oprogramowania OpenLayers [31] i API serwisu OpenStreetMap.org [32].
- Moduł łączności ze sklepami internetowymi — docelowo używający API wspieranych przez Yummy sklepów internetowych z żywnością i artykułami kuchennymi, służący do wyszukania interesujących użytkownika artykułów i umożliwienia mu dokonania ich zakupu w sklepie — niezaimplementowany z powodu braku wsparcia ze strony sklepów.

3.2.2. Aplikacja serwera

Aplikacja serwera składa się z następujących komponentów:

- Klasa `UsersController` zarządzająca standardowymi akcjami związanymi z obsługą kont użytkowników, takimi jak logowanie, rejestracja, usunięcie konta, ustawienie preferencji.
- Klasa `ExternalServices`, odpowiedzialna za edycję artykułu, obsługuje:
  - zdalne logowanie użytkownika na podane przez niego konto w zewnętrznym serwisie opartym na oprogramowaniu MediaWiki,
  - pobieranie artykułu z serwisu zewnętrznego,
  - zapisywanie zmodyfikowanego artykułu na serwisie zewnętrznym.
- Klasa `RecipesController`, odpowiada za zapisywanie w bazie danych akcji użytkownika związanych z wyszukiwaniem i oceną jakości przepisów, oraz późniejszą interpretację tych informacji, służącą szeregowaniu wyników wyszukiwania oraz sugerowaniu artykułów.
- Klasa `RestaurantsController` wyszukiwania restauracji, odpowiada za przepływ informacji o restauracjach pomiędzy bazą danych a aplikacją klienta.



Rysunek 3.2: Model danych

3.3. Model danych

Baza danych zrealizowana jest za pomocą mapowania obiektowo-relacyjnego, dzięki czemu jej implementacja była możliwa z wykorzystaniem wyłącznie narzędzi oferowanych przez Ruby on Rails, bez potrzeby sięgania po narzędzie niższego poziomu czyli SQL.

Na Rysunku 3.2 znajduje się diagram klas zbliżony do diagramu związków encji, który ilustruje strukturę stworzonej bazy danych. Klasy te dzielą się na trzy zasadnicze kategorie ze względu na ich przeznaczenie:

- Niezbędne informacje o użytkownikach:
  - `Users` — dane o użytkownikach wymagane aby istniała możliwość np. zalogowania się.
  - `WebsiteCookies` — ciasteczko (ang. cookie) pamiętające aktualną sesję użytkownika dla danego serwisu zewnętrznego.
- Dane służące do inteligentnego podpowiadania artykułów:
  - `Dishes` — podstawowe dane o potrawach, składnikach, napojach itd.
  - `UserTastes` — informacje o potrawach lubianych bądź nie przez użytkownika; użytkownik może określić, czy lubi daną potrawę.
  - `UserArticles` — historia artykułów przeglądanych przez użytkownika.
  - `ArticleRatings` — oceny jakości artykułów; chodzi o jakość ich treści, nie zaś opisywanej w nich potrawy.
  - `ComputedUserTastes` — przewidywane gusta użytkowników (wyliczane).
  - `Neighbours` — informacje o użytkownikach posiadających prawdopodobnie zbliżone gusta kulinarne (wyliczane).
- Baza informacji o restauracjach:
  - `Restaurants` — podstawowe informacje o restauracjach; m.in. położenie geograficzne.
  - `MenuPositions` — dane o potrawach serwowanych w restauracjach.
  - `PositionVotes` — oceny trafności pozycji w menu wystawiane przez użytkowników; konieczne było umożliwienie użytkownikowi wyrażenia akceptacji — bądź jej braku — wystąpienia każdej pozycji w menu. W razie niskich ocen pozycja jest usuwana z menu. Dzięki temu wyświetlane menu restauracji może być jak najbardziej aktualne i zgodne z rzeczywistością.

Rozdział 4

Implementacja

W tym rozdziale opisano najciekawsze fragmenty implementacji systemu oraz nietrywialne problemy, z jakimi twórcy zetknęli się podczas realizacji projektu.

Zaprojektowano i wykonano następujące oprogramowanie

- Aplikacja serwera — Część systemu napisana głównie w Ruby on Rails realizująca następujące funkcjonalności:

  1. Wnioskowanie gustów użytkownika, potrzebne do realizacji inteligentnego wyszukiwania przepisów;
  2. Edycja przepisów;
  3. Manipulowanie danymi o restauracjach oraz funkcje realizujące obsługę kont użytkowników.

- Aplikacja klienta — Część systemu napisana w języku JavaScript, realizująca następujące funkcjonalności: wyszukiwanie przepisów, wyszukiwanie restauracji, oraz przepływ danych pomiędzy interfejsem użytkownika a serwerem.
- Interfejs użytkownika — Strony internetowe, napisane w języku HTML + CSS.

4.1. Aplikacja klienta

Aplikacja klienta stanowi istotnie większy fragment systemu niż aplikacja serwera. Spowodowane jest to realizacją głównych założeń projektowych, wśród których znajdowało się m.in obciążenie jak największą ilością pracy maszyny klienta oraz umożliwienie przepływu dużych danych z zewnętrznych serwisów do klienta bez pośrednictwa serwera. Wszystkie funkcjonalności zrealizowane są w języku JavaScript i jego bibliotekach. Poniżej opisane są nietrywialne komponenty aplikacji oraz rozwiązania wykorzystane w celu obejścia napotkanych ograniczeń.

4.1.1. Wyszukiwanie przepisów

Po przekazaniu przez użytkownika parametrów wyszukiwania, aplikacja klienta szuka przepisów spełniających podane kryteria. W tym celu generuje i wysyła ciąg zapytań do API poszczególnych serwisów zewnętrznych, które są następnie asynchronicznie przetwarzane w miarę otrzymywania odpowiedzi. Uzyskanie wyników interesujących użytkownika często nie jest jednak trywialne, co pokazuje poniższy przykład.

Załóżmy, że użytkownik szuka potraw z kategorii X oraz kuchni kraju V, zawierających składnik Y lub Z, które w nazwie zawierają ABC. Niech ABC_X oznacza zbiór potraw mających ABC w nazwie i należących do kategorii X, analogicznie zdefiniujmy ABC_V, ABC_Y i ABC_Z. Aby uzyskać żądane wyniki, do API muszą zostać wysłane osobne zapytania dla uzyskania wszystkich czterech zbiorów, a następnie policzone $ABC_X \cap ABC_V \cap (ABC_Y \cup ABC_Z)$, co wymaga dostosowania rezultatów wyszukiwania do łatwego wykonywania działań na zbiorach — zostało to zrealizowane przez uporządkowanie wyników. Dodatkowym utrudnieniem jest to, że niektóre ze zbiorów — w tym przypadku ABC_V, ABC_Y i ABC_Z — nie mogą z powodu ograniczeń MediaWiki API zostać obliczone w wyniku pojedynczego zapytania do API.

Nie było też proste zrealizowanie wyszukiwania napojów i składników potraw. MediaWiki API nie pozwala zawęzić obszaru wyszukiwania do samych napojów / składników potraw. Jedynym możliwym sposobem na zrealizowanie tej funkcjonalności okazało się ściągnięcie przez klienta listy wszystkich napojów / składników potraw i przefiltrowanie jej z uwzględnieniem pozostałych kryteriów wyszukiwania.

Dodatkowym wyzwaniem programistycznym było przeprowadzane na bieżąco łączenie ze sobą i wyświetlanie wyników pochodzących z różnych serwisów zewnętrznych, które przychodzą do klienta w sposób asynchroniczny.

4.1.2. Ograniczenia MediaWiki API

MediaWiki API udostępnia dość ubogi zbiór procedur. Niemożliwe za pomocą pojedynczego zapytania jest na przykład uzyskanie listy wszystkich przepisów z danym składnikiem, które jednocześnie zawierają w nazwie podany ciąg znaków. Dlatego przy wyszukiwaniu, nazwa potrawy jest ostatnim rozpatrywanym kryterium. To znaczy, dopiero po obliczeniu listy przepisów spełniających wszystkie pozostałe kryteria brana jest pod uwagę nazwa. Z tego powodu, w przypadku opisanym powyżej, zbiory ABC_Y i ABC_Z muszą być obliczane po stronie klienta. API daje jedynie możliwość uzyskania listy wszystkich przepisów zawierających odpowiednio składnik Y i Z.

Ponadto, MediaWiki API ma ustalony limit wielkości zwracanego zbioru (zazwyczaj 500 elementów, ale dla niektórych zapytań nawet 50), także aby faktycznie uzyskać żądany zbiór koniecznych może być wiele zapytań do API, w przypadku gdy wielkość tego zbioru przekracza limit.

Bardziej szczegółowy opis API MediaWiki jest dostępny w [26].

Pomimo wyżej wymienionych trudności i konieczności używania sztuczek programistycznych do zrealizowania opisanej funkcjonalności w MediaWiki API, udało się wyabstrahować samą ideę wyszukiwania i sparametryzować funkcję szukającą. Dzięki temu niezwykle łatwe jest umożliwienie wyszukiwania w innych serwisach działających zgodnie z MediaWiki API.

Jedyne co należy wykonać, to przekazanie do funkcji szukającej danych specyficznych dla danego serwisu, takich jak adres internetowy API danego serwisu, lista kategorii zawierających napoje i składniki potraw oraz kilka innych łatwych do podania parametrów. W praktyce oznacza to konieczność napisania maksymalnie 20, 30 linii kodu, co w porównaniu do rozmiaru samej funkcji szukającej jest ilością znikomą.

4.1.3. Wyświetlanie mapki z restauracjami

Inną interesującą funkcjonalnością zrealizowaną przy użyciu JavaScriptu jest funkcja wyświetlająca mapkę z restauracjami, opcjonalnie tylko tymi, które serwują wybraną potrawę. Niemożliwe okazało się pobranie przez klienta listy wszystkich restauracji na świecie naraz, gdyż

może być ona gigantyczna. Użytkownik powinien mieć jednak możliwość wyświetlenia mapki Poznania, a chwilę później sprawdzenia restauracji serwujących pizzę w Warszawie, czy nawet w Fairbanks po drugiej stronie oceanu.

Klient dynamicznie pobiera z serwera listę tych restauracji, które są mu potrzebne (mieszczą się na aktualnie widocznym kawałku mapy) i tylko one są wyświetlane. Gdy użytkownik przesunie lub oddali mapę, pobierana jest nowa lista restauracji. Zrealizowanie tego rozwiązania wymusiło zaimplementowanie rozszerzenia do biblioteki OpenLayers, używanej do wyświetlania mapy.

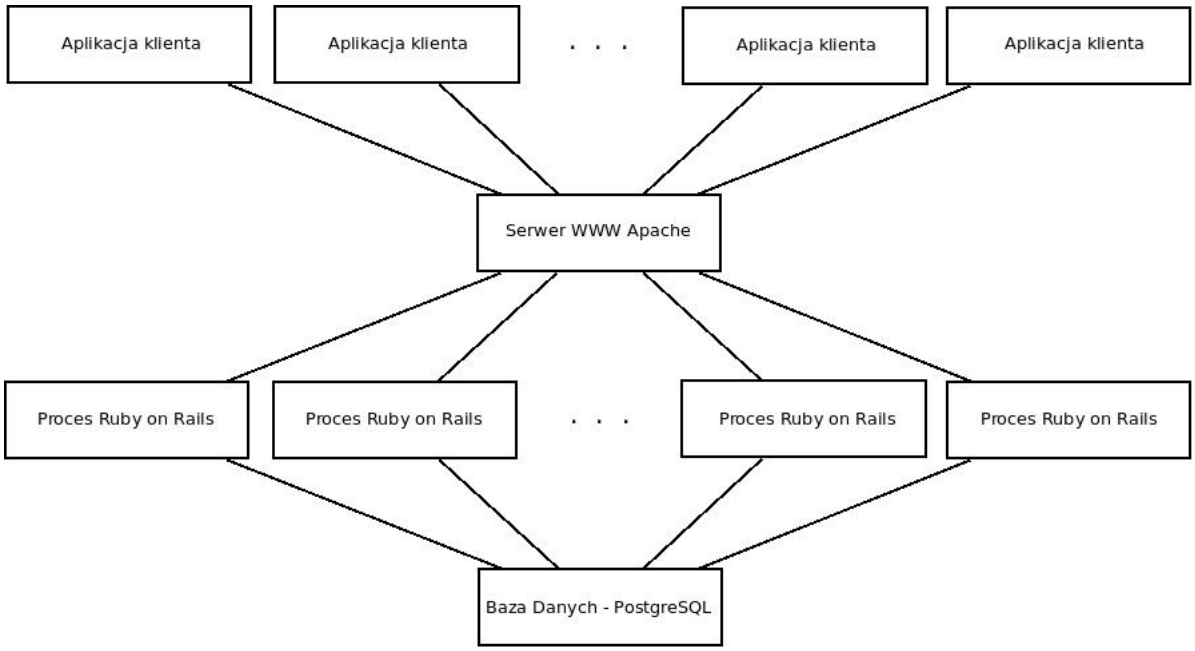
W pierwszym podejściu klient pobierał listę restauracji z bezpośrednio serwera OpenStreetMap za pośrednictwem udostępnionego API. Niestety z powodu wolnego działania serwera udostępniającego listę restauracji, pobranie kompletnej listy zajmowało często ponad 10 sekund, a w najgorszym wypadku około minuty. Z takim opóźnieniem mapka reagowała na przesunięcia czy oddalenia, co było nie do zaakceptowania.

Stwierdzono zatem, że najlepiej będzie mirrorować dane o restauracjach na serwerze, na którym działa Yummy. Został napisany skrypt, który kontaktuje się z serwerem OpenStreetMap i ściąga listę wszystkich restauracji na świecie — w czasie pisania tego dokumentu lista ta miała rozmiar około 70MB, a pobranie jej zajmowało czas od 2 do 3 godzin. Nie jest kluczowe do działania aplikacji, aby dane o restauracjach były idealnie aktualne, z tego powodu ustalono, że wystarczy pobierać nową bazę restauracji mniej więcej co tydzień. Napisano również drugi skrypt, pozwalający na wyekstrahowanie potrzebnych systemowi informacji (współrzędne geograficzne oraz nazwa restauracji) z pobranej listy oraz zapisanie tych informacji w bazie danych. Ostatnim fragmentem było napisanie kontrolera, dzięki któremu możliwe stało się używanie zapisanych danych. Aby umożliwić korzystanie z danych zapisanych na naszym serwerze zamiast danych z serwera OpenStreetMap, podmieniając jedynie adres, nasz kontroler przyjmuje parametry zapytania oraz generuje odpowiedź w tym samym formacie, co API OpenStreetMap. Dzięki temu nie były wymagane żadne zmiany w zaimplementowanym już rozszerzeniu biblioteki OpenLayers. Dodatkowo, w razie poprawy wydajności serwera OpenStreetMap, możliwe jest odciążenie serwera Yummy i ponowna migracja do wersji wykorzystującej bezpośrednie pobieranie listy restauracji z tego serwera.

Zapisywanie listy restauracji na serwerze Yummy nie jest zgodne z jednym z wymagań, które zakłada, że serwis powinien przechowywać jedynie metadane. Jednakże wyraźna poprawa wydajności aplikacji przesądziła o zastosowaniu tego rozwiązania mimo wszystko.

4.1.4. Użyta technologia

Wszystkie strony HTML są dynamicznie generowane przez Ruby on Rails, większość stron jest następnie dynamicznie modyfikowana przy użyciu JavaScriptu. Podjęto decyzję o wykorzystaniu języka JavaScript jako głównego narzędzia, aby umożliwić implementację większości funkcjonalności po stronie klienta. Zwiększa to responsywność aplikacji oraz znacznie zmniejsza obciążenie serwera. Pomimo generowania stron dynamicznie po stronie klienta położono duży nacisk na umożliwienie korzystania z serwisu w sposób jak najbardziej zbliżony do tradycyjnej strony HTML: przyciski “Wstecz” i “Dalej”, działające zgodnie z oczekiwaniami klienta, poprawne zakładki, linki odnoszące się do aktualnych stron. Efekt ten osiągnięto przy użyciu JavaScriptu nasłuchującego na nadejście “hashchange event” [44] generowanego przez przeglądarkę przy zmianie części adresu występującej po znaku “#”. W przypadku skopiowania URL z przeglądarki i wklejenia do nowej karty, Yummy wyświetli użytkownikowi, zgodnie z oczekiwaniami, dokładnie tą samą stronę.



Rysunek 4.1: Architektura sytemu

4.2. Aplikacja serwera

Głównymi celami istnienia rozbudowanej aplikacji serwera oraz nietrywialnej bazy danych są:

- Personalizacja serwisu — tworzenie kont użytkowników, system rekomendacji przepisów, system sąsiadów.
- Baza danych o restauracjach — przechowywanie i wykorzystywanie danych o położeniu i menu restauracji.

Poniżej znajduje się krótkie omówienie architektury aplikacji serwera, a następnie obszerne objaśnienia dotyczące najważniejszego jej komponentu, czyli funkcji wnioskowania o gustach użytkownika.

4.2.1. Architektura i technologia

Aplikacja serwera ma architekturę trójwarstwową, przedstawioną na Rysunku 4.1.

- Baza danych powstała przy użyciu PostgreSQL [34], przechowuje dane, które dostatecznie dobrze są omówione we wcześniejszej części dokumentu. Warto wspomnieć, że przy odwoływaniu się do właściwych tabel wykorzystywany jest mechanizm mapowania obiektowo-relacyjnego (ORM), który oszczędza pisania zapytań bezpośrednio w SQL, oferując wygodniejszy sposób realizacji zapytań w Ruby oraz umożliwia bezproblemową podmianę bazy danych na inną, jeśli zajdzie taka potrzeba. Całkowita rezygnacja z pisania SQL była możliwa dzięki modułowi SearchLogic [3] — jednemu z rozszerzeń standardowych funkcjonalności Ruby on Rails.
- Logika serwera czyli procesy Ruby on Rails służą jako łącznik pomiędzy bazą danych a aplikacją klienta, przetwarzają zapytania i modyfikują dane.

- Serwer WWW Apache zajmuje się przekazywaniem żądań od klientów do odpowiednich procesów Ruby on Rails. Z wielu możliwości wdrożenia wybrano użycie serwera Apache httpd [13] z modułem Phusion Passenger [33] z powodu prostoty instalacji oraz wysokiej wydajności takiego ustawienia.

4.3. Funkcja wnioskowania o gustach użytkownika

Jedną z kluczowych funkcjonalności serwisu jest wnioskowanie o gustach użytkownika na podstawie zapamiętanej przez system historii jego akcji. Na podstawie tych wniosków wyznaczani są sąsiedzi każdego uczestnika. Realizacja tego zagadnienia jest związana z tematyką systemów uczących się. W poniższych podpunktach przedstawiony jest opis rozwiązania użytego w serwisie.

4.3.1. Technologia i sposób użycia

Użyty algorytm sprowadza zagadnienie do rozwiązania problemu z algebry liniowej, używającego obliczeń numerycznych. Ze względu na niezbędną dużą wydajność implementacji algorytmu a jednocześnie wymaganą integrację z resztą serwisu, została podjęta decyzja o użyciu technologii JRuby [22]. Ta technologia pozwala używać jednocześnie aplikacji napisanych w języku Ruby / Ruby on Rails oraz bibliotek w języku Java. Z bibliotek wspierających operacje algebry liniowej została wybrana biblioteka `org.apache.commons.math.linear` Apache Commons Math [12], realizująca potrzebne operacje.

Użyta technologia JRuby pozwala na zrównoleglenie wykonywania algorytmu i wykorzystanie mocy obliczeniowej komputerów o większej ilości procesorów. W odróżnieniu od standardowej implementacji języka Ruby, JRuby pozwala na użycie natywnych wątków bez globalnej blokady interpretera [21] (GIL — global interpreter lock). W implementacji algorytmu położono duży nacisk na minimalizację ilości sekcji krytycznych umożliwiając większą równoległość wykonywania kodu.

W założeniu gusta użytkowników powinny być liczone raz na 1 – 2 dni. Powód jest taki, że niżej przedstawiony algorytm, dla dużej liczby użytkowników i przepisów w bazie danych, przykładowo: 5 milionów ocen, działa w czasie zbliżonym do jednej doby. Dlatego niemożliwe jest wykorzystanie go do przewidywania gustów w czasie rzeczywistym. Jednocześnie okres dwóch dni potrzebnych do uaktualnienia przewidywanego gustu użytkownika jest na tyle mały, że nie stanowi istotnego ograniczenia.

W przypadku sukcesu serwisu i znacznego wzrostu ilości użytkowników i przepisów, konieczne byłoby przeniesienie niżej opisanych obliczeń numerycznych na wyspecjalizowane maszyny, wspierające obliczenia wektorowe, w celu maksymalnego przyspieszenia wykonywania obliczeń.

4.3.2. Opis algorytmu

Tworzenie danych wejściowych

Serwis pamięta historię odwiedzin danego artykułu przez danego użytkownika oraz ewentualną ocenę użytkownika typu lubię/nie lubię, którą mógł wystawić dla artykułu. Na podstawie tych danych tworzona jest opinia użytkownika o przepisie będąca liczbą rzeczywistą. Niech $U = \{1 \dots n\}$ będzie zbiorem użytkowników, $P = \{1 \dots m\}$ - zbiorem przepisów. Zdefiniujmy funkcję częściową $F : U \times P \longrightarrow [-100; 120]$, określającą ocenę przepisu p wystawioną przez użytkownika u. Wartość funkcji $F(u, p)$ wyznaczona jest w następujący sposób:

- +100 jeśli użytkownik u określił że lubi przepis p, -100 jeśli określił że nie lubi.
- Do powyższej wartości dodaje się liczbę wyświetleń przepisu u przez użytkownika p, ale jeśli liczba ta jest większa niż 20, dodaje się tylko 20.
- Jeśli użytkownik nie wykonywał żadnych z powyższych działań związanych z przepisem p, to $F(u,p)$ pozostaje nieznana.

Przykładowo, jeśli wiadomo, że użytkownik u lubi artykuł p i dodatkowo odwiedził stronę tego artykułu 5 razy, to $F(u,p) = 105$.
Wartym odnotowania jest fakt, że funkcja oceny jest prosta do podmienienia.

Kluczową rzeczą, którą trzeba wyszczególnić, jest to, że wartość $F(u,p)$ nie jest znana dla przeważającej większości par (u,p). Założono, że stosunek liczby znanych ocen do liczby wszystkich możliwych ocen jest mniejszy niż 0.1. Zadaniem algorytmu jest przypisanie estymowanej oceny $F'(u,p)$ dla każdej pary (u,p) użytkownika i przepisu.

Funkcję F przyjmujemy za dane wejściowe dla algorytmu. Algorytm zwróci funkcję zupełną $F' : U \times P \longrightarrow \mathbb{R}$ będącą dość dobrym przybliżeniem F dla istniejących ocen, to znaczy — o ile znane było wcześniej $F(u,p)$, to $F'(u,p) \approx F(u,p)$. Ponadto F' opierając się na rozsądnych założeniach probabilistycznych, na przykład takich jak niezależność działań różnych użytkowników, będzie przewidywać pozostałe, nieznane oceny.

W praktyce przekazywanie funkcji F polega na przekazywaniu trójek $(u,p,F(u,p))$, zaś wynik — czyli funkcja F' — zwracany jest w formie macierzy M takiej, że $M[i,j] = F'(i,j)$.

Właściwy algorytm

Ustalmy z jako liczbę znanych ocen, czyli liczbę par (u,p) dla których istnieje $F(u,p)$.

Wybierzmy również liczbę naturalną f, którą nazwiemy wymiarem algorytmu. Niech $v \cdot w$ oznacza standardowy iloczyn skalarny wektorów v i w z $\mathbb{R}^f$.

Zdefiniujmy funkcję $p : \mathbb{R}^f \times \mathbb{R}^f \longrightarrow \mathbb{R}$, którą nazwiemy funkcją predykcji, jako:

$$p(v,w) = \begin{cases} 120 & \text{dla} \quad v \cdot w > 120 \\ v \cdot w & \text{dla} \quad -100 \leqslant v \cdot w \leqslant 120 \\ -100 & \text{dla} \quad v \cdot w < -100 \end{cases} \tag{4.1}$$

Innymi słowy: jest to obcięcie standardowego iloczynu skalarnego do odpowiedniego zakresu.

Algorytm znajduje dwie macierze — macierz użytkowników: $\mathfrak{U} \in \mathbb{R}^{f \times n}$ i macierz przepisów: $\mathfrak{P} \in \mathbb{R}^{f \times m}$. Wiersze tych macierzy można utożsamiać intuicyjnie z wektorami odpowiednio "cech użytkowników" i "cech przepisów".

Funkcja predykcji p jest używana, aby wyliczyć wartości funkcji F'.

Przyjmujemy:

$$F'(i,j) := p(\mathfrak{U}_i, \mathfrak{P}_j), \tag{4.2}$$

gdzie $\mathfrak{U}_i$ i $\mathfrak{P}_j$ są wektorami cech użytkownika i oraz przepisu j.

Należy skupić się na znalezieniu takich macierzy. Miernikiem "odpowiedniości" tych macierzy będzie błąd średniokwadratowy (ang. root mean square error, RMSE), intuicyjnie oznaczającą średnie odchylenie wyliczonych ocen funkcji F' od rzeczywistych ocen danych przez funkcję F w punktach, w których te oceny faktycznie są znane, to znaczy tam, gdzie F jest określone.

Formalnie:

$$RMSE = \sqrt{\frac{\sum_{i,j:\exists F(i,j)} \left(F(i,j) - F'(i,j)\right)^2}{z}} \tag{4.3}$$

Algorytm polega na iteracyjnych zmianach wartości macierzy $\mathfrak{U}$ i $\mathfrak{P}$ metodą gradientu prostego [25], które zmniejszają $RMSE$. Określmy funkcje, skorelowane z $RMSE$, których gradienty będziemy brać pod uwagę. O ile $F(i,j)$ istnieje, to

$$E_{ij} = \frac{1}{2}(F(i,j) - F'(i,j))^2 + \frac{k_u}{2}\|\mathfrak{U}_i\|^2 + \frac{k_p}{2}\|\mathfrak{P}_j\|^2 \tag{4.4}$$

Wartości k_u i k_p to współczynniki regulacji, które mają za zadanie niwelację potencjalnych anomalii w danych i w samym algorytmie. Ich rola objaśniona jest poniżej, po wzorach na gradienty E_{ij}.

Pomijając drugą część powyższego wzoru, w której występują wspomniane współczynniki, widzimy wyrażenie $(F(i,j) - F'(i,j))^2$, które jest jedną z sum występujących pod pierwiastkiem w $RMSE$. Zmniejszając to wyrażenie mamy więc szansę na zmniejszenie całego $RMSE$, do czego będziemy dążyć.

Odpowiednie gradienty E_{ij} wynoszą:

$$\nabla_{\mathfrak{U}_i} = -\frac{\partial E_{ij}}{\partial \mathfrak{U}_i} = (V_{ij} - F'(i,j))\mathfrak{P}_j - k_u\mathfrak{U}_i \tag{4.5}$$

$$\nabla_{\mathfrak{P}_j} = -\frac{\partial E_{ij}}{\partial \mathfrak{P}_j} = (V_{ij} - F'(i,j))\mathfrak{U}_i - k_p\mathfrak{P}_j \tag{4.6}$$

Gradient wyznacza kierunek, w którym różniczkowana funkcja najszybciej maleje. Będziemy "schodzić" po gradiencie E_{ij}, to znaczy przemieszczać się w kierunku minimum E_{ij} zmieniając $\mathfrak{U}_i$ i $\mathfrak{P}_j$.

Rola współczynników k_u i k_p polega na tym, aby wskutek anomalii danych nie doszło do nadmiernego wzrostu wektorów $\mathfrak{U}_i$ i $\mathfrak{P}_j$. Jeśli bowiem norma któregoś z nich, przykładowo $\mathfrak{U}_i$ staje się nadmiernie duża, to wówczas dodanie gradientu $\nabla_{\mathfrak{U}_i}$, a konkretnie odjęcie części $k_u\mathfrak{U}_i$ wymusi jej zmniejszenie.

Algorytm działa według następującego schematu:

1. Ustaw wartości początkowe macierzy $\mathfrak{U}$ i $\mathfrak{P}$ na losowe wartości oscylujące wokół $\sqrt{\frac{Avg}{f}}$, gdzie Avg jest średnią ze znanych ocen.

2. Iteruj poniższą pętlę, którą można utożsamić z pojedynczym zejściem po gradiencie:

   - Dla każdych i,j takich, że istnieje $F(i,j)$:

   $$\mathfrak{U}_i \leftarrow \mathfrak{U}_i - \mu\nabla_{\mathfrak{U}_i} \tag{4.7}$$

   $$\mathfrak{P}_j \leftarrow \mathfrak{P}_j - \mu\nabla_{\mathfrak{P}_j} \tag{4.8}$$

   dopóki błąd $RMSE$ nie zacznie wzrastać.

   Kolejność wykonywania kroków w powyższej pętli jest dowolna więc operacje te można wykonywać współbieżnie lub równolegle — tak zostało to zaimplementowane.

Wyznaczanie sąsiadów

Mając obliczoną funkcję F' możemy zdefiniować współczynnik sąsiedztwa dla dwóch użytkowników u_1 i u_2:

$$S_{u1,u2} := \sum_{p \in P} (F'(u_1,p) - F'(u_2,p))^2 \tag{4.9}$$

Funkcja ta jest tym większa, im większe są różnice między poszczególnymi estymowanymi ocenami użytkowników. (Im mniejsze $S_{u1,u2}$, tym bliżej znajdują się u_1 i u_2). Oczywiście $S_{u,u} = 0$.

4.3.3. Podsumowanie

W niniejszym rozdziale opisano użyty algorytm, który zaczerpnięto z pracy [7] (patrz Algorytm 3 w rozdziale 2.2).

Podstawy teoretyczne i wyprowadzenia matematyczne w języku rachunku prawdopodobieństwa, które sprowadzają model probabilistyczny do opisanego wyżej problemu optymalizacyjnego minimalizacji funkcji $RMSE$, znajdują się w książce [4] (patrz Rozdział 3 — *Linear Models for Regression*).

Empiryczne eksperymenty wykazały, że algorytm najlepiej się zachowuje dla następująco dobranych stałych: $k_u = k_p = 0.05$, $\mu = 0.0004$.

Rozdział 5

Testy

W tym rozdziale opisane są wykonane testy systemu oraz czynniki, które ograniczyły możliwości testowania.

5.1. Ograniczenia

Niemożliwe było utworzenie testów wydajnościowych przy użyciu dostępnych narzędzi. Większość z nich (JMeter [19], WebTest [45] i inne) nie wspiera JavaScript, a właśnie ta część aplikacji klienta jest wąskim gardłem wydajności całego systemu, gdyż większość logiki aplikacji wykonuje się po stronie klienta. Wspomniane narzędzia pozwoliłyby zaś tylko na zmierzenie czasu reakcji serwera.

5.2. Testy funkcjonalne

Do testów funkcjonalnych użyto narzędzia Selenium [42]. Przypadki testowe w Selenium to po prostu nagrany ciąg akcji takich jak wypełnienie formularza w podany sposób czy kliknięcie na podany link, przeplatanych asercjami które sprawdzają, czy wykonywane akcje osiągają zamierzony efekt. Selenium dosłownie wykonuje podaną sekwencję czynności w przeglądarce internetowej, pozwala więc również na testowanie stron dynamicznie modyfikowanych JavaScriptem.

Używając wtyczki Selenium IDE do przeglądarki Firefox [35], zostały zaimplementowane testy sprawdzające poprawny przebieg najważniejszych przypadków użycia, to znaczy:

- Logowanie i zarządzanie kontem
- Wyszukiwanie przepisów (przy podaniu możliwie skomplikowanych kryteriów) i ocena artykułu
- Dodawanie potrawy do restauracji oraz wyszukiwanie restauracji serwujących daną potrawę

Zautomatyzowanie tych testów pozwoliło na łatwe ich uruchamianie po wprowadzeniu każdej zmiany w systemie.

5.3. Testy funkcji wnioskowania o gustach użytkownika

Testowanie tego komponentu wiązało się z istotnym problemem znalezienia danych do testowania. Funkcja ta powinna w założeniu działać dobrze dla rzeczywistych danych będących

wynikiem akcji dużej grupy niezależnie działających użytkowników. Tym samym niemożliwe było stworzenie realnych testów samemu — dane trzeba było uzyskać z zewnętrznego źródła.

Istnieje kilka portali, które gromadzą i udostępniają tego typu dane. Jednym z nich jest strona `www.grouplens.org` [17], na której znajduje się zestaw ocen filmów, wystawionych przez grupę użytkowników. Tego zestawu użyto do przetestowania omawianej funkcji.

5.3.1. Użyte dane

Dane testowe składały się z 100000 ocen ze zbioru $\{1, 2, 3, 4, 5\}$, wystawionych przez 943 użytkowników dla 1682 filmów. Aby dostosować te oceny do formatu danych wejściowych przyjmowanych przez funkcję wnioskującą, dokonano ich liniowego przeskalowania:

$$\lambda x.55(x - 1) - 100 \tag{5.1}$$

Dane te zostały podzielone losowo na 5 rozłącznych sekcji po 20000 ocen. Każda z tych sekcji posłużyła jako dane treningowe dla funkcji wnioskowania, której trafność była następnie sprawdzana na pozostałych 80000 ocen. Jest to system testowania algorytmów uczących się zwany "walidacją krzyżową" [52], z podziałem na 5 części.

5.3.2. Wyniki

- Najlepsze wyniki osiągnięto dla wymiaru algorytmu $f = 2$.
- Błąd $RMSE$ dla danych testowych wyniósł w przybliżeniu 50.
- Średnia liczba przypadków, gdy ocena estymowana różniła się od oceny realnej o więcej niż 55, wyniosła w przybliżeniu 20000.

5.3.3. Wnioski

Na podstawie powyższych wyników możemy wywnioskować empirycznie, że prawdopodobieństwo zdarzenia, że ocena estymowana będzie się różnić od oceny realnej o więcej, niż $\frac{1}{4}$ zakresu ocen, jest równe w przybliżeniu $\frac{1}{4}$.

$$\mathbb{P}(|F'(u, p) - F(u, p)| \geqslant \frac{1}{4}r) \approx \frac{1}{4}, \tag{5.2}$$

gdzie r jest zakresem ocen, czyli 220.

Taki wynik świadczy o dość dobrym zachowaniu funkcji wnioskowania.
Przykładowo:
Załóżmy, że jeśli użytkownik u zapoznałby się z przepisem p, to polubiłby go, czyli wystawiłby mu ocenę nie mniejszą niż 100. Wówczas funkcja wnioskowania na 75% obliczy ocenę estymowaną dla tego użytkownika i przepisu na nie mniej niż 45 — czyli wciąż znacząco "na plusie".

Warto również wspomnieć, że powyższe wyniki zgadzają się z wynikami testów opisanych w rozdziale 4 pracy [7].

Rozdział 6

Podsumowanie

Udało się stworzyć wydajny, łatwo rozszerzalny system gotowy do działania, spełniający kluczowe założenia — jest inteligentną, precyzyjną wyszukiwarką przepisów, artykułów kulinarnych i restauracji umożliwiającą jej użytkownikom ingerencję w proces rozszerzania bazy wiedzy kulinarnej.

6.1. Możliwe usprawnienia i rozszerzenia

Zwiększyć szansę na popularność serwisu mogłoby zrealizowanie następujących usprawnień i rozszerzeń:

- zmiana wyglądu i rozszerzenie możliwości interfejsu użytkownika, ze współpracą profesjonalnego grafika, wraz z przeprowadzeniem testów przez użytkowników,
- dodanie funkcjonalności umożliwiającej współpracę i integrację serwisu z internetowymi sklepami spożywczymi,
- integracja z większą ilością zewnętrznych baz artykułów kulinarnych, być może w różnych językach,
- zwiększenie wydajności sztucznej inteligencji poprzez przeniesienie obliczeń na sprawniejsze maszyny dedykowane lub umożliwienie wykonywania obliczeń na klastrze komputerowym,
- poprawa trafności przewidywań funkcji wnioskującej o gustach użytkownikach, która mogłaby nastąpić po zebraniu dużej ilości historii działań użytkowników i — na podstawie analizy statystycznej tych danych — wyprowadzeniu modelu probabilistycznego jeszcze bardziej zgodnego z rzeczywistością.

Dodatek A

Instalacja i konfiguracja

Aplikacja klienta nie wymaga instalacji i korzystanie z niej ogranicza się do wejścia na odpowiednią stronę internetową, więc szczegółowo zostanie opisana tylko instalacja serwera. Opis instalacji serwera zakłada użycie uniksopodobnego systemu operacyjnego [37].

A.1. Wymagane oprogramowanie

Aplikacja serwera jest napisana w języku Ruby, zatem aby ją uruchomić należy najpierw zainstalować interpreter tego języka. Do pewnych funkcjonalności Yummy wymaga instalacji interpretera JRuby, możliwe jest hostowanie serwisu przy użyciu jedynie tego interpretera, lecz prostsze jest jednoczesne użycie JRuby oraz standardowego interpretera Ruby [39], to podejście zostanie bardziej szczegółowo opisane. Do prostej instalacji wielu interpreterów Ruby jednocześnie można wykorzystać oprogramowanie RVM [41]. Instalacja RVM i implementacji Ruby przy jego użyciu jest opisana na stronie domowej tego projektu i nie będzie tutaj przytaczana.

Do działania serwisu jest potrzebna instalacja bazy danych PostgreSQL oraz serwera Apache httpd, można tego dokonać przy użyciu standardowej techniki instalacji nowego oprogramowania dla danej platformy systemowej (np. apt-get dla Debiana).

Wymagane biblioteki języka Ruby można zainstalować przy użyciu RubyGems [40] dostarczanych z implementacjami języka Ruby. Należy zainstalować gemy `rails`, `authlogic`, `searchlogic`, `nokogiri` oraz — w przypadku standardowego interpretera Ruby — `pg` i `passenger`, a w przypadku JRuby — `activerecord-jdbcpostgresql-adapter`.

Do działania funkcji wnioskowania gustów użytkownika potrzebna jest biblioteka Apache Commons Math.

A.2. Instalacja i przygotowanie Yummy

A.2.1. Konfiguracja bazy danych

Po rozpakowaniu Yummy w pliku `config/database.yml`, w podpunktach `production` i `production_jruby` należy wpisać konfigurację połączenia z bazą danych (przykład konfiguracji można znaleźć w [30]), wpisując jako adapter odpowiednio `postgresql` i `jdbcpostgresql`.

A.2.2. Wypełnienie bazy danych

Kolejnym krokiem jest stworzenie tabel w bazie danych i wypełnienie bazy potrzebnymi danymi. Następującą komendą stworzymy tabele w bazie danych:

```
RAILS_ENV=production rake db:setup
```

Kolejne dwie komendy wypełnią informacje odpowiednio o wspieranych stronach i restauracjach:

```
RAILS_ENV=production rake db:seed
RAILS_ENV=production rake yummy:download_map && rake yummy:update_map
```

A.2.3. Konfiguracja Apache

Aby zakończyć instalację Yummy należy skonfigurować serwer Apache, konfiguracja jest opisana szczegółowo na stronie domowej modułu Phusion Passenger [33] oraz jest podawana po instalacji gemu `passenger`, więc nie będzie tutaj opisywana.

A.3. Wyliczanie gustów użytkowników

W celu jednorazowego wyliczenia gustów użytkownika należy wywołać komendę:

```
RAILS_ENV=production_jruby rake yummy:update_taste
```

Warto zauważyć, że ta komenda wymaga użycia interpretera JRuby oraz odpowiedniej konfiguracji Java Classpath, aby zawierała bibliotekę Apache Commons Math.

A.4. Aktualizacja informacji o restauracjach i gustach użytkowników

Aby automatycznie aktualizować informacje o restauracjach i gustach użytkowników można użyć demona Cron [24]. Wystarczy skonfigurować Crona tak, aby uruchamiał co jakiś czas wcześniej wymienione komendy. W przypadku posiadania więcej niż jednej maszyny serwerowej można wykonywać aktualizacje na innej maszynie niż tej, która serwuje Yummy, gdyż te operacje potrzebują dużo pamięci i zasobów procesora.

A.5. Możliwe problemy przy korzystaniu z Yummy

Funkcja wyliczania gustów użytkowników potrzebuje znacznej ilości pamięci, dla każdej pary użytkownik — artykuł jest zapisywana wartość funkcji predykcji (patrz 4.3.2), przypadku większej ilości użytkowników i artykułów może to przekroczyć ilość pamięci, którą realnie można zainstalować na maszynie serwerowej. W takim przypadku należy użyć pamięci wirtualnej i plików/partycji wymiany.

Dodatek B

Podział i harmonogram pracy

B.1. Harmonogram - postępy prac

Październik 2009: Wykrystalizowanie się pomysłu, otrzymanie zgody prowadzącego. Zarysowanie funkcjonalności i ram projektu. Prace nad dokumentem "Wizja".

Listopad 2009: Tworzenie dokumentu "Model przypadków użycia". Poszukiwanie technologii odpowiedniej do realizacji pomysłu. Częściowa realizacja głównego przypadku użycia "Wyszukiwanie przepisów".

Grudzień 2009: Dodanie obsługi kont użytkowników. Prace nad umożliwieniem edycji artykułów przez użytkowników. Kolejny ważny przypadek użycia — "Edycja przepisu kulinarnego".

Styczeń 2010: Znaczne rozbudowanie struktury relacji w bazie danych, przygotowanie bazy danych dla algorytmu sztucznej inteligencji. Powstaje kolejny dokument — "Dokumentacja architektury oprogramowania".

Luty 2010: Rozpoczęcie prac nad niniejszą pracą licencjacką. Początek zajęć z przedmiotu "Systemy uczące się" — rozmowy z prowadzącymi, znawcami tematyki Systemów Uczących Się, o doborze algorytmu sztucznej inteligencji.

Marzec 2010: Implementacja algorytmu sztucznej inteligencji, wykonana część prac nad obsługą restauracji. Sąsiedzi i polecane przepisy. Równocześnie tworzony niniejszy dokument.

Kwiecień 2010: Finalizowanie prac implementacyjnych, kończenie implementacji obsługi restauracji, poprawa interfejsu użytkownika. Pełna wersja wstępna niniejszej pracy.

Maj 2010: Testowanie komponentów systemu. Uzupełnianie i dopracowywanie niniejszej pracy, aż do wersji ostatecznej.

B.2. Podział pracy

Michał Kijewski:

- implementacja modułu aplikacji klienta wyszukującego dane w serwisie Freebase [16]

- implementacja formularza wyszukiwania w interfejsie klienta
- implementacja i testowanie funkcji wnioskowania o gustach użytkownika
- projekt bazy danych

Marek Kiszkis:

- implementacja komponentu aplikacji klienta wyświetlającego restauracje
- implementacja modułu aplikacji klienta wyszukującego dane w serwisach korzystających z MediaWiki przy użyciu MediaWiki API
- implementacja modelu bazy danych wraz z klasami ORM Ruby on Rails

Marek Sapota:

- implementacja komponentu wyświetlającego treść artykułów i listę wyników wyszukiwania (wraz z pobraniem i wyświetleniem miniatur zdjęć)
- implementacja standardowych funkcjonalności przeglądarki ("Wstecz", zakładki itp.) w interfejsie JavaScriptu
- implementacja serwerowej części modułu restauracji
- implementacja serwerowej części funkcjonalności oceny artykułu
- optymalizacja i zrównoleglenie funkcji wnioskowania o gustach użytkownika

Oskar Wantoła:

- implementacja edycji i dodawania artykułów przy użyciu MediaWiki API
- integracja modułu wyświetlającego restauracje z modułem obsługi przepisów (dodawanie potrawy do menu restauracji, wyświetlenie na mapie restauracji serwujących daną potrawę, wyświetlenie menu restauracji, ocenianie trafności pozycji menu)
- implementacja klienckiej części funkcjonalności oceny artykułu

Dodatek C

Opis zawartości płyty dołączonej do pracy

Na dołączonej płycie CD znajdują się:

- praca licencjacka w formacie PDF,
- pliki źródłowe systemu,
- scenariusze testowe wykonane przy użyciu Selenium,
- filmy pokazujące korzystanie z systemu,
- dokumenty:
  - Wizja,
  - Model przypadków użycia,
  - Dokumentacja architektury oprogramowania,
- prezentacje:
  - prezentacja architektury,
  - prezentacja postępów i planów na przyszłość,
  - prezentacja projektu.

Bibliografia

[1] Scott W. Ambler. Artykuł opisujący technikę programistyczną ORM. http://www.agiledata.org/essays/mappingObjects.html.

[2] Binarylogic. Strona domowa modułu AuthLogic, rozszerzającego możliwości Ruby on Rails. http://github.com/binarylogic/authlogic.

[3] Binarylogic. Strona domowa modułu SearchLogic, rozszerzającego możliwości Ruby on Rails. http://github.com/binarylogic/searchlogic.

[4] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2007.

[5] Java blueprints. Opis architektury Model, Widok, Kontroler. http://java.sun.com/blueprints/patterns/MVC-detailed.html.

[6] Encyclopaedia Britannica. Opis stron typu Wiki. http://www.britannica.com/EBchecked/topic/1192819/wiki.

[7] Ma Chih-Chao. Large-scale collaborative filtering algorithms. Master's thesis, Department of Computer Science and Information Engineering, College of Electrical Engineering & Computer Science, National Taiwan University, June 2008. http://www.csie.ntu.edu.tw/~r95007/thesis/svdnetflix/thesis.pdf.

[8] Creative Commons. Tekst licencji Creative Commons BY-SA w wersji 3.0. http://creativecommons.org/licenses/by-sa/3.0/legalcode.

[9] WebKit developers. Strona domowa projektu WebKit. http://webkit.org/.

[10] Free Software Foundation. Tekst licencji GNU AGPL. http://www.gnu.org/licenses/agpl.html.

[11] Free Software Foundation. Tekst licencji GNU GPL. http://www.gnu.org/licenses/gpl.html.

[12] The Apache Software Foundation. Strona domowa projektu Apache Commons Math. http://commons.apache.org/math/.

[13] The Apache Software Foundation. Strona domowa projektu Apache httpd. http://httpd.apache.org/.

[14] Wikimedia Foundation. Strona domowa Fundacji Wikimedia. http://wikimediafoundation.org/wiki/Home.

[15] Freebase. Specyfikacja języka zapytań MQL w serwisie Freebase. http://www.freebase.com/docs/mql.

[16] Freebase. Strona WWW projektu Freebase. http://www.freebase.com/.

[17] GroupLens. Zestaw danych do testowania algorytmu sztucznej inteligencji. http://www.grouplens.org/node/73.

[18] Bob Ippolito. Artykuł opisujący format JSONP. http://bob.pythonmac.org/archives/2005/12/05/remote-json-jsonp/.

[19] JMeter. Strona domowa aplikacji JMeter. http://jakarta.apache.org/jmeter/.

[20] jQuery. Strona domowa biblioteki JavaScriptu jQuery. http://jquery.com/.

[21] JRuby. Skrótowy opis różnic pomiędzy JRuby a standardową implementacją języka Ruby. http://jruby.org/git?p=jruby.git;a=blob_plain;f=docs/Differences.txt;hb=0c9363863a8b59b042a2c79c191d73cdb6ddae1b.

[22] JRuby. Strona domowa implementacji języka Ruby w Javie. http://www.jruby.org/.

[23] PC Magazine. Definicja API w PC Magazine. http://www.pcmag.com/encyclopedia_term/0,2542,t=application+programming+interface&i=37856,00.asp.

[24] Linux man pages. Opis demona Cron w podręczniku Linuxa. http://linux.die.net/man/8/cron.

[25] MathWorld. Opis metody gradientu prostego na stronie MathWorld. http://mathworld.wolfram.com/MethodofSteepestDescent.html.

[26] MediaWiki. Opis języka zapytań obsługiwanego przez MediaWiki API. http://www.mediawiki.org/wiki/API:Query_-_Lists.

[27] MediaWiki. Pełny opis możliwości MediaWiki API. http://www.mediawiki.org/wiki/API.

[28] O'Reilly Network. Opis terminu Web 2.0. http://oreilly.com/web2/archive/what-is-web-20.html.

[29] Ruby on Rails. Strona domowa frameworku Ruby on Rails. http://rubyonrails.org/.

[30] Ruby on Rails wiki. Przykład konfiguracji Ruby on Rails i bazy danych PostgreSQL. http://wiki.rubyonrails.org/database-support/postgres#databaseyml_example.

[31] OpenLayers. Strona domowa biblioteki OpenLayers. http://openlayers.org/.

[32] OpenStreetMap. Strona domowa projektu OpenStreetMap. http://www.openstreetmap.org/.

[33] Phusion. Strona domowa modułu Apache, Phusion Passenger. http://www.modrails.com/.

[34] PostgreSQL. Strona domowa bazy danych PostgreSQL. http://www.postgresql.org/.

[35] Mozilla Project. Strona domowa aplikacji Firefox. http://www.firefox.com.

[36] Mozilla Project. Strona domowa projektu Gecko. https://developer.mozilla.org/en/Gecko.

[37] The Linux Information Project. Definicja sytemów Uniksopodobnych. http://www.linfo.org/unix-like.html.

[38] RFC. Formalna specyfikacja formatu JSON. http://tools.ietf.org/html/rfc4627.

[39] Ruby. Strona domowa języka Ruby oraz jego standardowej implementacji. http://www.ruby-lang.org.

[40] RubyGems. Strona domowa oprogramowania RubyGems. http://docs.rubygems.org/.

[41] RVM. Strona domowa oprogramowania RVM. http://rvm.beginrescueend.com/.

[42] Selenium. Strona domowa aplikacji Selenium. http://seleniumhq.org/.

[43] W3C. Opis zagadnienia same origin policy. http://www.w3.org/Security/wiki/Same_Origin_Policy.

[44] W3C. Specyfikacja "hashchange event". http://www.whatwg.org/specs/web-apps/current-work/#event-hashchange.

[45] WebTest. Strona domowa aplikacji WebTest. http://webtest.canoo.com/webtest/manual/WebTestHome.html.

[46] Wikimedia. Strona domowa oprogramowania MediaWiki. http://www.mediawiki.org.

[47] Wikipedia. Opis języka HTML na Wikipedii. http://en.wikipedia.org/wiki/HTML.

[48] Wikipedia. Opis języka JavaScript na Wikipedii. http://en.wikipedia.org/wiki/JavaScript.

[49] Wikipedia. Opis języka zapytań SQL na Wikipedii. http://en.wikipedia.org/wiki/SQL.

[50] Wikipedia. Opis kaskadowych arkuszy stylów na Wikipedii. http://en.wikipedia.org/wiki/CSS.

[51] Wikipedia. Opis luki w zabezpieczeniach SQL injection na Wikipedii. http://en.wikipedia.org/wiki/Sql_injection.

[52] Wikipedia. Opis walidacji krzyżowej na Wikipedii. http://en.wikipedia.org/wiki/Cross-validation_(statistics).