

NIIKT
Architektura systemu

Zespół Gr0No3

w składzie:

Michał Ejdys

Urszula Herman-Iżycka

Elżbieta Krępska

Piotr Witusowski

12 stycznia 2004

Spis treści

1	Wprowadzenie	7
1.1	Cel	7
1.2	Zakres	7
1.3	Definicje	7
1.4	Załączniki	7
1.5	Omówienie reszty dokumentu	8
1.5.1	Omówienie rozdziałów	8
1.5.2	Omówienie perspektyw	8
2	Prezentacja architektury systemu	10
3	Założenia i zależności	11
4	Przegląd przypadków użycia	12
4.1	Omówienie	12
4.2	Diagramy przypadków użycia	12
4.3	Przegląd przypadków użycia	15
4.3.1	Rozpoczęcie pracy z Modułem Użytkownika	15
4.3.2	Ustalanie parametrów wyszukiwania	15
4.3.3	Wyświetlanie wyników wyszukiwania	15
4.3.4	Lokalizacja firmy	15
4.3.5	Dodanie wpisu do książki adresowej konta lokalnego	15
4.3.6	Zamieszczanie ogłoszenia	15
4.3.7	Logowanie firmy	16
4.3.8	Aktualizacja/zmiana ogłoszenia	16
4.3.9	Logowanie administratora	16
4.3.10	Dodanie/zmiana pozycji w katalogu branż	16
4.3.11	Dodanie/zmiana branży w najpopularniejszych branżach	16
4.3.12	Wyświetlanie najpopularniejszych branż	16
4.3.13	Zatwierdzenie nowego ogłoszenia	16
4.3.14	Wprowadzanie mapy	17
4.3.15	Dodanie/zmiana ulicy na mapie	17
4.4	Realizacja przypadków użycia	17

5	Dekompozycja logiczna systemu	18
5.1	Omówienie	18
5.2	Diagram	19
5.3	Najważniejsze komponenty - Warstwa interfejsu	20
5.3.1	Pakiet KlientNIIKT	20
5.4	Najważniejsze komponenty - Warstwa sieci	21
5.4.1	Pakiet ObslugaKlienta	21
5.5	Najważniejsze komponenty - Warstwa logiki biznesowej	22
5.5.1	Pakiet TransportowaneObiekty	22
5.5.2	Pakiet OpcjeUzytkownika	23
5.5.3	Pakiet Ogloszenia	24
5.5.4	Pakiet Konto	24
5.5.5	Pakiet Mapa	25
5.5.6	Pakiet Branze	26
5.5.7	Pakiet Szablony	27
5.6	Najważniejsze komponenty - Warstwa danych	27
5.6.1	Pakiet Baza	27
6	Dekompozycja na procesy	28
6.1	Omówienie	28
6.2	Diagram	29
7	Instalacja systemu	30
7.1	Moduł Użytkownika	30
7.2	Moduł Firmy	30
7.3	Moduł Administratora	30
7.4	Serwlet NIIKT	31
8	Implementacja systemu	32
8.1	Omówienie	32
8.2	Warstwy	32
8.2.1	Warstwa interfejsu	32
8.2.2	Warstwa sieci	45
8.2.3	Warstwa logiki biznesowej	52
8.2.4	Warstwa bazy danych	59
8.3	Realizacja przypadków użycia	61
8.3.1	Rozpoczęcie pracy z Modułem Użytkownika	61
8.3.2	Ustalanie parametrów wyszukiwania	62
8.3.3	Wyświetlanie wyników wyszukiwania	63
8.3.4	Lokalizacja firmy	64
8.3.5	Dodanie wpisu do książki adresowej konta lokalnego	65
8.3.6	Zamieszczanie ogłoszenia	66
8.3.7	Logowanie firmy	67

8.3.8	Aktualizacja/zmiana ogłoszenia	68
8.3.9	Logowanie administratora	69
8.3.10	Dodanie/zmiana pozycji w katalogu branż	70
8.3.11	Dodanie/zmiana branży w najpopularniejszych branżach	71
8.3.12	Wyświetlanie najpopularniejszych branż	72
8.3.13	Zatwierdzenie nowego ogłoszenia	73
8.3.14	Wprowadzanie mapy	73
8.3.15	Dodanie/zmiana ulicy na mapie	73
9	Przechowywane dane	77
9.1	Omówienie	77
9.2	Dane nieprzechowywane w bazie	77
9.3	Ogłoszenie	78
9.4	Ogłoszenie płatne	79
9.5	Telefony	79
9.6	Katalog branż	80
9.7	Najpopularniejsze branże	81
9.8	Mapa	81
9.8.1	Mapa	81
9.8.2	Ulice	82
10	Wydajność systemu	83
10.1	Omówienie	83
10.2	„Wąskie gardła” systemu	83
10.2.1	Obciążenie serwera	83
10.2.2	Optymalizacja zapytań na bazie	84
10.2.3	Dostrajanie bazy danych	84
10.2.4	Monitorowanie przepustowości łącz	84
11	Jakość	85
11.1	Omówienie	85
11.2	Aspekty jakości systemu NIIKT zależne tylko od implementacji	85
11.2.1	Rozszerzalność	85
11.2.2	Przenośność	85
11.2.3	Łatwość obsługi	85
11.3	Aspekty jakości systemu NIIKT zależne od użytkowania systemu	86
11.3.1	Niezawodność	86
11.3.2	Bezpieczeństwo	86
12	Historia zmian	87

Spis rysunków

Diagramy przypadków użycia	12
4.1 Moduł Użytkownika	12
4.2 Moduł Firmy	13
4.3 Moduł Administratora	14
 Diagram dekompozycji logicznej systemu	 19
5.1 Dekompozycja logiczna	19
 Diagram dekompozycji systemu na procesy	 29
6.1 Dekompozycja na procesy	29
 Diagramy klas	 32
8.1 Warstwa interfejsu – pakiet InterfejsUzytkownika	33
8.2 Warstwa interfejsu – pakiet InterfejsFirmy	36
8.3 Warstwa interfejsu – pakiet InterfejsAdministratora	39
8.4 Warstwa sieci	46
8.5 Warstwa logiki biznesowej	52
 Diagramy sekwencji	 61
8.6 Rozpoczęcie pracy z Modułem Użytkownika	61
8.7 Ustalanie parametrów wyszukiwania	62
8.8 Wyświetlanie wyników wyszukiwania	63
8.9 Lokalizacja firmy	64
8.10 Dodanie wpisu do książki adresowej konta lokalnego	65
8.11 Zamieszczanie ogłoszenia	66
8.12 Logowanie firmy	67
8.13 Aktualizacja/zmiana ogłoszenia	68
8.14 Logowanie administratora	69
8.15 Dodanie/zmiana pozycji w katalogu branż	70
8.16 Dodanie/zmiana branży w najpopularniejszych branżach	71
8.17 Wyświetlanie najpopularniejszych branż	72

8.18	Zatwierdzenie nowego ogłoszenia	74
8.19	Wprowadzanie mapy	75
8.20	Dodanie/zmiana ulicy na mapie	76

Rozdział 1

Wprowadzenie

1.1 Cel

Celem niniejszego dokumentu jest opisanie budowy systemu NIIKT oraz omówienie różnorodnych aspektów realizacji projektu i podjętych decyzji architektonicznych z różnych perspektyw. Dokument powstał na potrzeby projektu NIIKT zespołu *Gr0No3*.

1.2 Zakres

Niniejszy dokument jest podstawą implementacji systemu NIIKT, opisuje jego architekturę po stronie:

1. serwera;
2. klienta, którym może być:
 - (a) *Firma* – ogłoszeniodawca,
 - (b) *Użytkownik* Książki NIIKT,
 - (c) *Administrator* systemu.

1.3 Definicje

Patrz załącznik: Słownik NIIKT.

1.4 Załączniki

1. Słownik NIIKT.

1.5 Omówienie reszty dokumentu

1.5.1 Omówienie rozdziałów

Dokument jest podzielony na 12 rozdziałów. Poszczególne rozdziały zajmują się kolejno:

Rozdział 1	Wstęp.
Rozdział 2	Prezentacja architektury.
Rozdziały 3-5	Wizja architektury z punktu widzenia logicznego oraz funkcjonalnego.
Rozdział 6	Wizja architektury jako puli wątków.
Rozdziały 7-9	Koncepcja implementacji i wdrożenia systemu.
Rozdziały 10-11	Prognozy dotyczące funkcjonowania systemu.
Rozdział 12	Historia zmian.

1.5.2 Omówienie perspektyw

W niniejszym dokumencie architektura została opisana w następujących perspektywach:

1. Dekompozycja na przypadki użycia.
Polega na zdekomponowaniu systemu na poszczególne funkcjonalności. Jest opisana w rozdziałach [4](#) i [8](#).
2. Dekompozycja logiczna.
Zawiera podział systemu na następujące logiczne warstwy:
 - (a) Interfejsy.
 - (b) Sieć.
 - (c) Logika biznesowa.
 - (d) Baza danych.

Dla każdej warstwy wyszczególniono w dokumencie zbiór logicznych, wysokopoziomowych pakietów. Pakiety zostały dalej podzielone na klasy analityczne, które dają „zgrubny” opis dekompozycji systemu na klasy implementacyjne. Dekompozycja logiczna jest opisana w rozdziale [5](#).
3. Dekompozycja na procesy.
Zawiera podział systemu na działające w nim procesy. Dekompozycja na procesy obrazuje również komunikację międzyprocesową. Dekompozycja ta jest opisana w rozdziale [6](#).
4. Implementacja.
Podaje szczegółowy podział systemu na klasy implementacyjne wraz z ich odpowiedzialnościami i realizującymi je metodami. Implementacja jest opisana w rozdziale [8](#).
5. Organizacja.
Zawiera opis technikaliów związanych z:

(a) instalacją systemu

Pokazuje rozmieszczenie poszczególnych komponentów systemu w różnych węzłach. Zawiera także opis przeprowadzania czynności związanych z instalacją systemu na serwerze. Jest opisana w rozdziale 7.

(b) danymi przechowywanymi w bazie

Omawia rodzaj bazy danych i jej kluczowe, niezbędne do działania systemu właściwości. Pokazuje rodzaj i organizację przechowywanych danych. Jest opisana w rozdziale 9.

(c) przewidywanymi własnościami systemu

W rozdziale 3 pokazuje czynniki mogące mieć wpływ na zmianę parametrów systemu. W rozdziale 10 oraz 11 szacuje wydajność i jakość systemu w przeciętnych warunkach, a także analizuje możliwe zachowanie systemu przy jego przeciążeniu.

Rozdział 2

Prezentacja architektury systemu

Do budowy systemu NIIKT zostaną zastosowane następujące technologie:

- Architektura klient-serwer.
System NIIKT będzie zbiorem komunikujących się ze sobą aplikacji:
 - serwlet na serwerze, dostarczający interfejs do bazy danych NIIKT,
 - wiele aplikacji klienckich zdalnie zlecających wyszukiwania,
 - wiele aplikacji dla firm zarządzających ich ogłoszeniami,
 - aplikacja dla *Administratora*.
- Obiektowe techniki modelowania systemu (ang. *Object-oriented design*).
System zostanie zaimplementowany w Javie – nowoczesnym języku obiektowym. Pociąga to za sobą wykorzystanie wszystkich technik projektowania zorientowanych obiektowo, m.in. dekompozycja systemu na klasy i obiekty, hybrydyzacja, polimorfizm.
- Technologia budowy interfejsu użytkownika (GUI) – MVP (ang. *Model-View-Presenter*).
W podejściu MVP, zapożyczonym ze *Smalltalka*, aplikacje klienckie (*Firmy* oraz *Użytkownika*) zajmują się graficzną interpretacją komunikatów dochodzących do nich z serwera Książki Telefonicznej NIIKT oraz przekazywaniem zleceń od użytkownika aplikacji. Natomiast cała logika systemu jest zawarta w aplikacji na serwerze, która zarządza danymi przechowywanymi w bazie oraz komunikacją z klientem.
W opisie architektury systemu NIIKT (rozdz. 5), okaże się, iż pakiet dostarczający opcje dla *Użytkownika* odchodzi od tej zasady, gdyż funkcjonalnie jest częścią logiki biznesowej, lecz komunikuje się z *Użytkownikiem* bez pośrednictwa sieci.
- Technologia JDBC (ang. *Java DataBase Connectivity*).
JDBC 2.0 API jest interfejsem Javy do bazy danych zintegrowanym ze standardem języka SQL3 (rozszerzenie SQL-99). Korzystanie ze zunifikowanego interfejsu bazy danych zapewnia możliwość postawienia systemu na różnych platformach.

Rozdział 3

Założenia i zależności

1. Projekt NIIKT zostanie zrealizowany w obiektowym języku programowania Java.
2. Projekt jest niekomercyjny, zostanie wykonany przy pomocy bezpłatnych lub dostępnych bezpłatnie dla studentów WMIMUW narzędzi. Po zakończeniu nie ma przynosić żadnych korzyści materialnych zespołowi nad nim pracującemu.
3. *Firma* może posiadać jedno ogłoszenie (związane z jej numerem NIP).
4. W NIIKT jest jedno konto *Administradora*.

Rozdział 4

Przegląd przypadków użycia

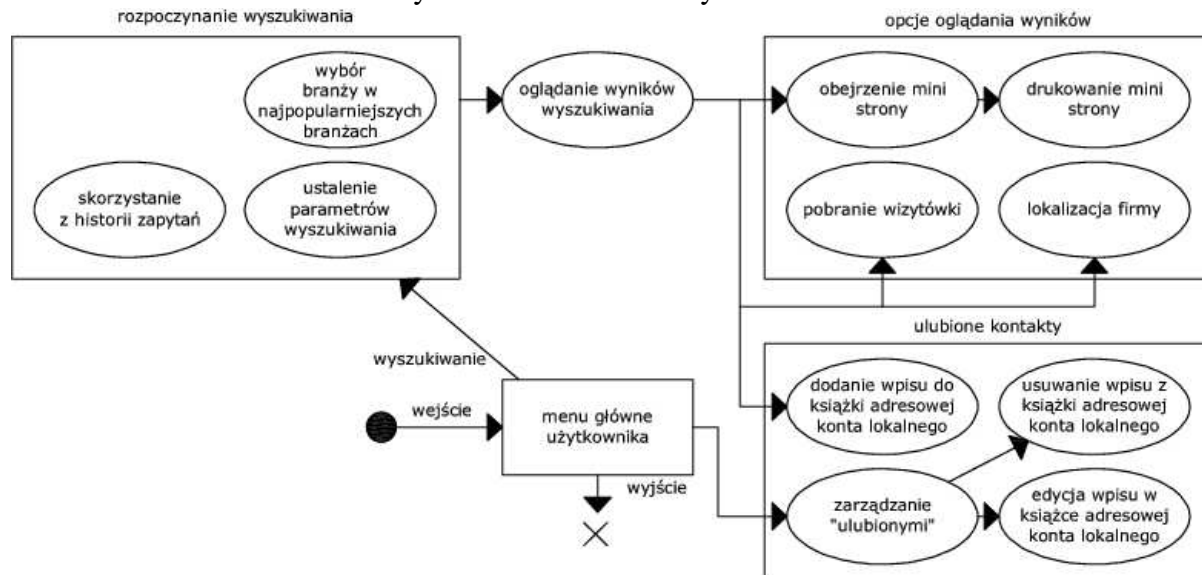
4.1 Omówienie

W tym rozdziale zostaną przedstawione najważniejsze przypadki użycia – te, których funkcjonalność jest niezwykle istotna dla działania systemu oraz te, które ilustrują bądź podkreślają delikatne aspekty architektury systemu.

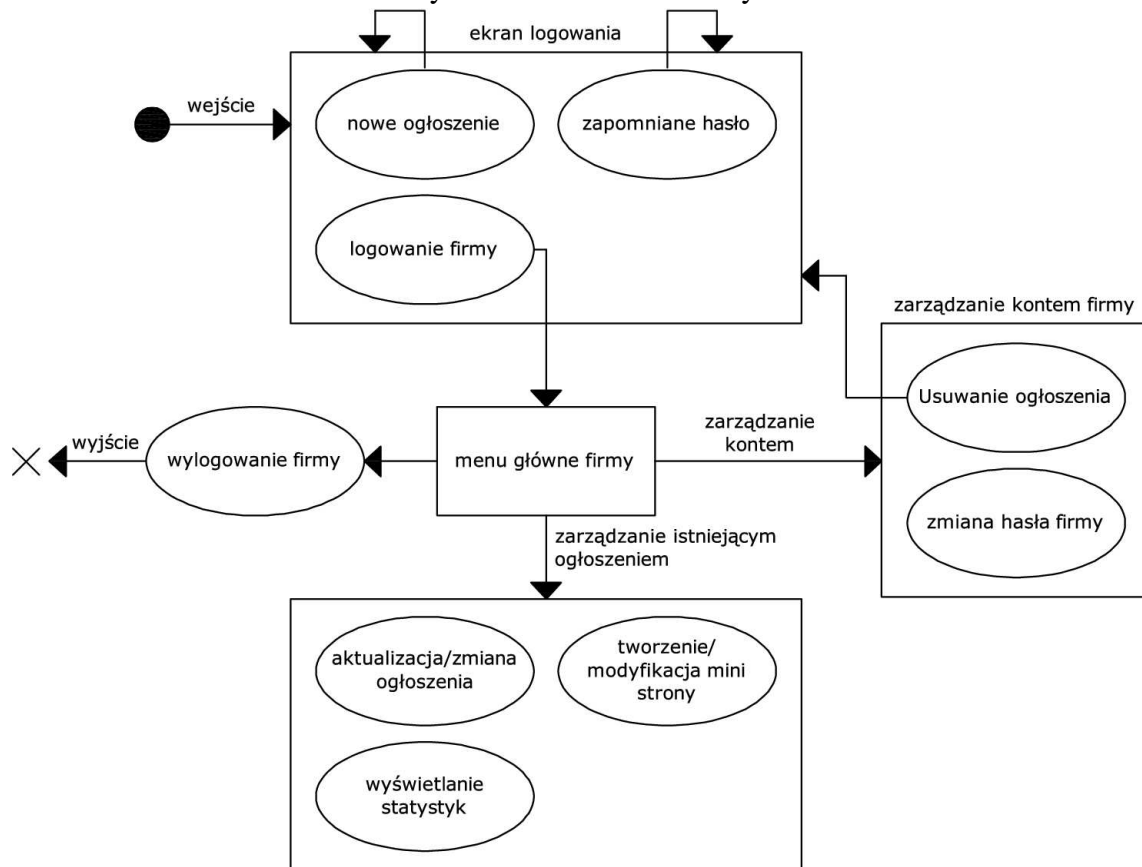
4.2 Diagramy przypadków użycia

Na rysunkach 4.1, 4.2 oraz 4.3 znajdują się diagramy przypadków użycia dla Modułu Firmy, Modułu Użytkownika oraz Modułu Administratora.

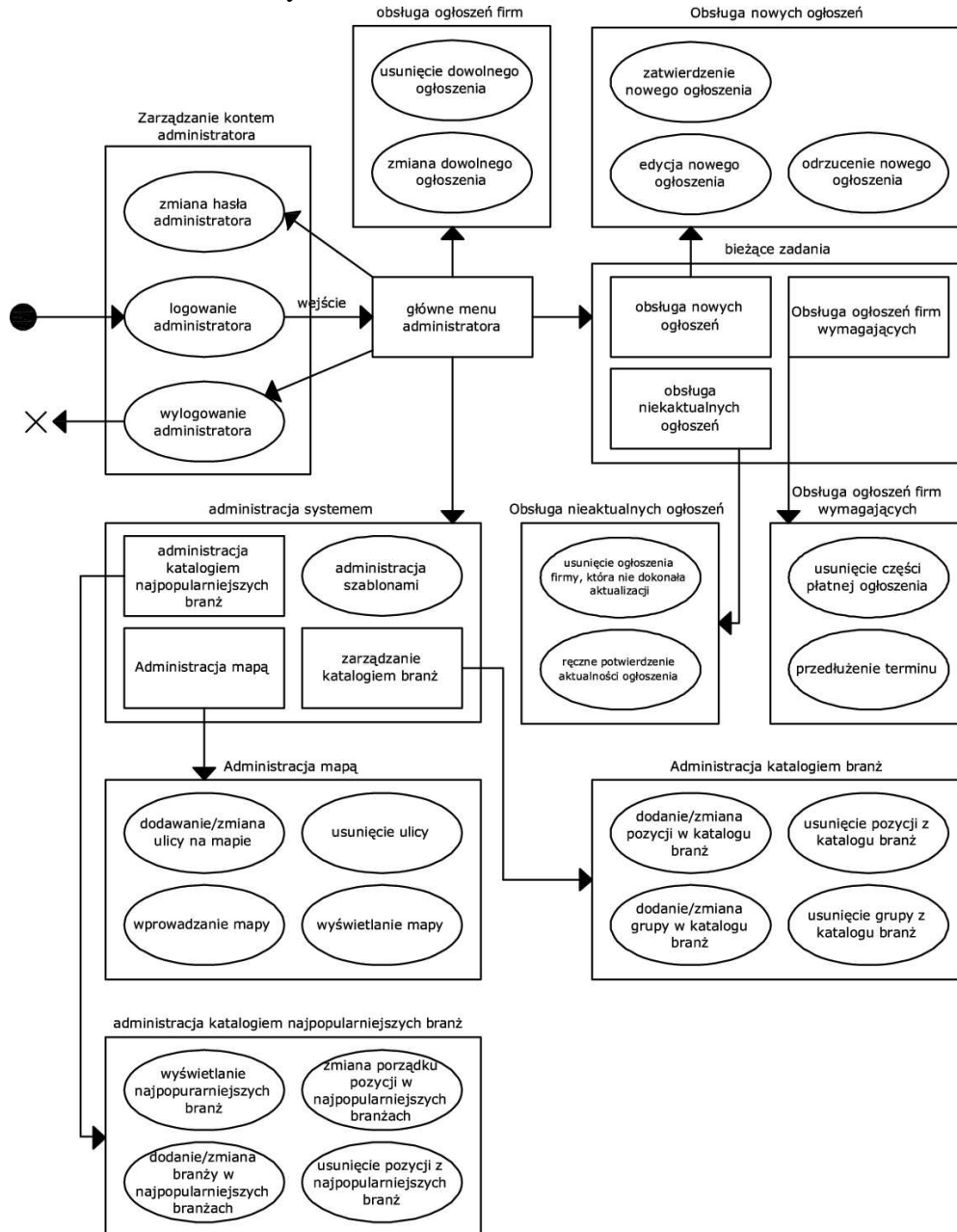
Rysunek 4.1: Moduł Użytkownika



Rysunek 4.2: Moduł Firmy



Rysunek 4.3: Moduł Administratora



4.3 Przegląd przypadków użycia

4.3.1 Rozpoczęcie pracy z Modułem Użytkownika

Ten przypadek użycia opisuje czynności, które musi wykonać *Użytkownik*, aby rozpocząć pracę z Książką NIIKT. Zapoczątkowuje wszystkie przebiegi akcji używania Książki przez *Użytkownika*.

4.3.2 Ustalanie parametrów wyszukiwania

Ten przypadek użycia opisuje przebieg czynności wykonywanych przy ustalaniu parametrów wyszukiwania. Przypadek ten rozpoczyna proces wyszukiwania, który jest kontynuowany w przypadku użycia *Wyświetlanie wyników wyszukiwania*.

4.3.3 Wyświetlanie wyników wyszukiwania

Ten przypadek użycia opisuje przebieg czynności od zatwierdzenia parametrów wyszukiwania przez *Użytkownika* do chwili, gdy wyniki zostają przez system wyświetlone u *Użytkownika*. Wyświetlone wyniki mogą być przetwarzane przez *Użytkownika*, który może np. wykonać przypadek użycia *Lokalizacja firmy* lub dodać interesujący go wpis do „Ulubionych”.

4.3.4 Lokalizacja firmy

Ten przypadek użycia opisuje przebieg czynności, gdy *Użytkownik* chce zobaczyć na mapie lokalizację danej firmy. Zaczyna się od zadania żądania podczas wyświetlania wyników zapytania.

4.3.5 Dodanie wpisu do książki adresowej konta lokalnego

Ten przypadek użycia opisuje przebieg czynności, jakie musi wykonać *Użytkownik*, aby zapamiętać interesujący go wpis do „Ulubionych”. Przypadek rozpoczyna się, gdy *Użytkownik* zażąda zapamiętania danych konkretnej firmy podczas wyświetlania wyników wyszukiwania.

4.3.6 Zamieszczanie ogłoszenia

Ten przypadek użycia opisuje przebieg czynności, jakie musi wykonać *Firma*, aby zamieścić nowe ogłoszenie na stronach Książki NIIKT. Dany przypadek rozpoczyna proces, jaki musi przejść ogłoszenie nim zostanie opublikowane. Po zatwierdzeniu przez *Firmę* danych z ogłoszenia, zostaje ono dołączone do kolejki nowych ogłoszeń, którymi zajmie się *Administrator* w przypadku *Zatwierdzanie nowego ogłoszenia*.

4.3.7 Logowanie firmy

Ten przypadek użycia opisuje przebieg czynności, jakie musi wykonać *Firma*, aby rozpocząć pracę ze swoimi ogłoszeniami w Księżce NIIKT. Po zalogowaniu się będzie ona mogła zarządzać zarówno swoim kontem (zmiana hasła, usunięcie ogłoszenia), jak też istniejącym ogłoszeniem (aktualizacja ogłoszenia, tworzenie mini strony, wyświetlenie statystyk).

4.3.8 Aktualizacja/zmiana ogłoszenia

Ten przypadek użycia opisuje przebieg czynności, jakie musi wykonać *Firma*, aby zmienić już istniejące swoje ogłoszenie (robi to po zalogowaniu się). Zmienione ogłoszenie nie jest już, jak w przypadku nowego ogłoszenia, weryfikowane przez *Administradora*, lecz od razu pojawia się w Księżce w wersji określonej przez *Firmę*.

4.3.9 Logowanie administratora

Ten przypadek użycia zapoczątkowuje pracę *Administradora* z systemem. Po zalogowaniu się *Administrator* ma dostęp do głównego menu.

4.3.10 Dodanie/zmiana pozycji w katalogu branż

Ten przypadek użycia umożliwia *Administratorowi* modyfikację katalogu branż w celu usprawnienia wyszukiwania w drzewie katalogu (pomocne zarówno dla *Administradora*, jak i dla *Użytkowników*).

4.3.11 Dodanie/zmiana branży w najpopularniejszych branżach

Ten przypadek użycia umożliwia *Administratorowi* dokonanie modyfikacji na liście najpopularniejszych branż w celu jej aktualizacji.

4.3.12 Wyświetlanie najpopularniejszych branż

Ten przypadek użycia umożliwia *Administratorowi* przeglądanie statystyk najpopularniejszych branż. Po stwierdzeniu, które branże były w ostatnim czasie (np. miesiąca) najpopularniejsze, może on dokonać właściwych modyfikacji w katalogu branż, co powinno usprawnić korzystanie z Książki.

4.3.13 Zatwierdzenie nowego ogłoszenia

Ten przypadek użycia opisuje czynności, jakie musi wykonać *Administrator*, aby zatwierdzić nowe ogłoszenie i aby pojawiło się ono na stronach Książki NIIKT. Jeśli dane w ogłoszeniu są nieprawidłowe lub nieodpowiednie (zły NIP, niecenzuralne sformułowania), to *Administrator* je odrzuca.

4.3.14 Wprowadzanie mapy

Ten przypadek użycia umożliwia *Administratorowi* wprowadzenie do systemu planu miasta. Dany przypadek będzie wywoływany bardzo rzadko (instalacja systemu, zmiany w planie miasta).

4.3.15 Dodanie/zmiana ulicy na mapie

Ten przypadek użycia umożliwia *Administratorowi* wprowadzanie nowych ulic do mapy bądź modyfikowanie już istniejących, ma to służyć aktualizacji mapy.

4.4 Realizacja przypadków użycia

Opis realizacji przypadków użycia znajduje się w rozdziale [8.3](#).

Rozdział 5

Dekompozycja logiczna systemu

5.1 Omówienie

Dekompozycja logiczna systemu jest podziałem systemu na logiczne komponenty – pakiety. Wyróżnione zostały cztery warstwy o następującym znaczeniu:

Warstwa interfejsu. Opisuje część systemu widoczną z zewnątrz. Odpowiada części *View* z MVP.

Warstwa sieci. Opisuje komunikację klient - serwer Książki Telefonicznej. Odpowiada części *Presenter* z MVP.

Warstwa logiki biznesowej. Jest to warstwa kluczowa dla funkcjonowania systemu, odpowiada części *Model* z MVP. Zawiera logikę serwera, odpowiedzialną za:

1. odbieranie zleceń klienta,
2. przetwarzanie ich,
3. wykonywanie aktualizacji i zapytań na bazie danych,
4. przetwarzanie wyników zapytań,
5. przekazywanie wyników do klienta,

gdzie klientem może być *Użytkownik*, *Firma* lub *Administrator*.

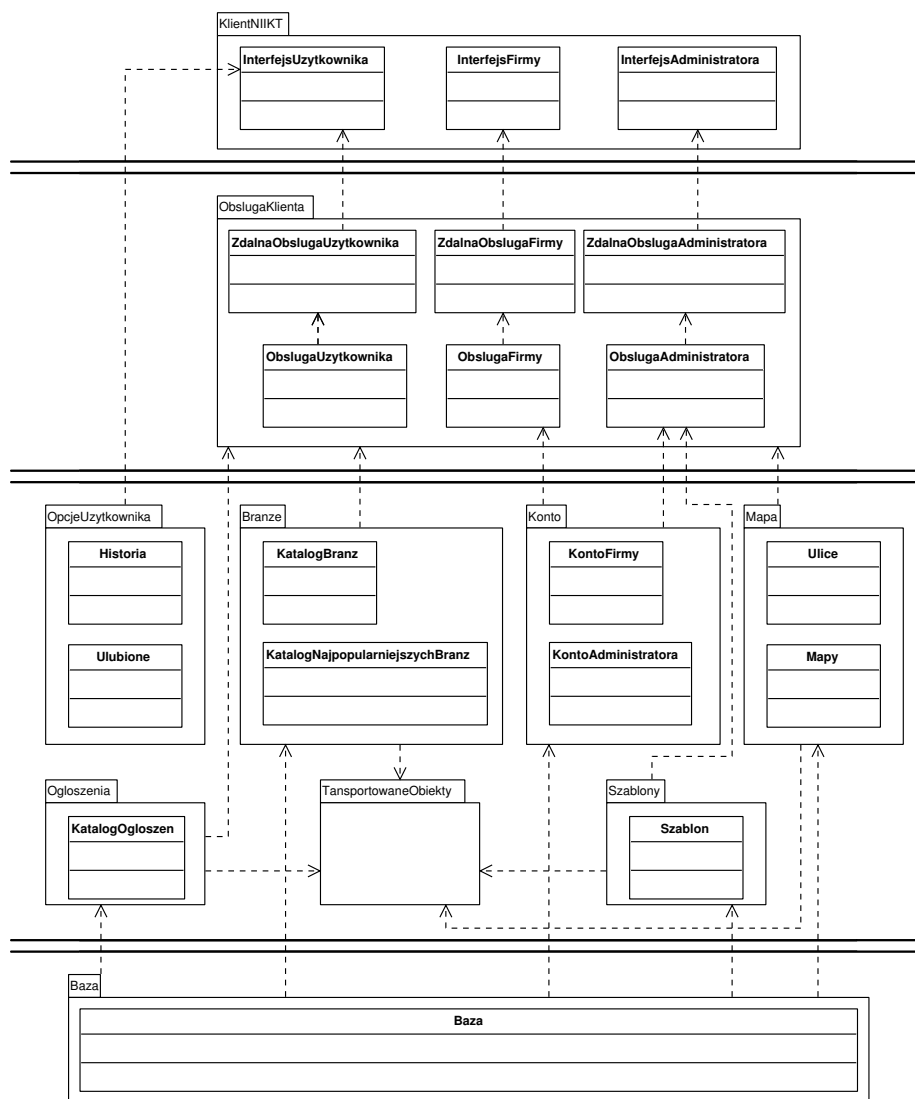
Warstwa bazy danych. Opisuje część systemu odpowiedzialną za utrzymywanie informacji o firmach i operacje, które można na nich wykonywać.

Na szczególną uwagę zasługuje pakiet *OpcjeUzytkownika*, opisany szerzej w rozdziale 5.5.2. Znajduje się on po stronie logiki biznesowej, jednak komunikacja pomiędzy *InterfejsemUzytkownikiem* a *OpcjamiUzytkownika* *nie* odbywa się za pomocą sieci. Instancje klas tego pakietu znajdują się lokalnie u klienta, jednak funkcjonalnie jest on częścią logiki biznesowej.

5.2 Diagram

Rysunek 5.1 (str. 19) prezentuje dekompozycję logiczną systemu z uwzględnieniem podziału na warstwy. Pakiet na diagramie jest logiczną jednostką dostarczającą pewną funkcjonalność w systemie. Dla każdego pakietu wyróżniono najważniejsze jego klasy. Strzałki oznaczają obustronną komunikację pomiędzy pakietami.

Rysunek 5.1: Dekompozycja logiczna



5.3 Najważniejsze komponenty - Warstwa interfejsu

5.3.1 Pakiet KlientNIIKT

Zakres odpowiedzialności

Pakiet KlientNIIKT jest odpowiedzialny za „kontakty” z klientem systemu NIIKT. Jest częścią systemu widoczną przez *Użytkownika*, *Firmę* lub *Administradora* i ukrywającą szczegóły implementacji Książki. Zgodnie z ideą MVP, pakiet zajmuje się interpretowaniem informacji otrzymanych z pakietów warstwy sieciowej oraz przekazywaniem zleceń *Użytkownika* do systemu.

W systemie zostały wyróżnione trzy rodzaje klientów:

1. *Użytkownik* – klient korzystający z Książki NIIKT, aby wyszukiwać w niej informacje.
2. *Firma* – firma zwykła lub wymagająca – klient, który chce zamieścić / zamieścił ogłoszenie w Książce.
3. *Administrator* – administrator systemu.

Każdy z klientów posiada własny interfejs. Może powstać wiele instancji klasy InterfejsuUzytkownika oraz InterfejsuFirmy, ale nie powstanie więcej niż jedna instancja InterfejsuAdministradora.

Każda klasa wchodząca w skład pakietu KlientNIIKT dostarcza interfejs dla jednego rodzaju klienta. Interfejs ten:

1. potrafi zlecać operacje systemowi NIIKT, kontaktując się z pakietem ZdalnaObslugaKlienta,
2. potrafi oczekiwać na wyniki tych operacji i odbierać je,
3. wie, jak przetworzyć otrzymane informacje.

Klasy

1. Klasa InterfejsUzytkownika

Jest odpowiedzialna za dostarczenie interfejsów:

- (a) ustalania parametrów wyszukiwania,
- (b) prezentacji wyników wyszukiwania,
- (c) oglądania wyników wyszukiwania,
- (d) zarządzania „Ulubionymi” kontaktami,
- (e) oglądania historii wyszukiwań,
- (f) pomocy *Użytkownika*.

2. Klasa `InterfejsFirmy`

Jest odpowiedzialna za dostarczenie interfejsów:

- (a) logowania do istniejącego ogłoszenia,
- (b) zarządzania ogłoszeniami: tworzenie, usuwanie,
- (c) modyfikacji / aktualizacji ogłoszenia,
- (d) pomocy *Firmy*.

3. Klasa `InterfejsAdministradora`

Jest odpowiedzialna za dostarczenie interfejsów:

- (a) logowania się *Administradora*,
- (b) zadań administracyjnych: nowych ogłoszeń, nieaktualnych ogłoszeń, ogłoszeń firm wymagających,
- (c) zarządzania katalogiem branż,
- (d) zarządzania **Katalogiem najpopularniejszych branż**,
- (e) administracji mapą,
- (f) administracji szablonami,
- (g) pomocy *Administradora*.

5.4 Najważniejsze komponenty - Warstwa sieci

5.4.1 Pakiet `ObslugaKlienta`

Zakres odpowiedzialności

Pakiet `ObslugaKlienta` zajmuje się:

1. przesyłaniem informacji z logiki biznesowej do `KlientaNIIKT`,
2. przekazywaniem zleceń `KlientaNIIKT` systemowi NIIKT.

Pakiet został zdekomponowany na trzy główne „kanały komunikacji”, odpowiadające trzem rodzajom klientów w systemie. Każdy kanał składa się z dwóch komunikujących się po sieci klas:

1. Klasa `ZdalnaObslugaKlienta`

Jest klasą, której instancja nie znajduje się na serwerze, a zdalnie, u klienta. Jej zadaniem jest komunikacja z `ObslugaKlienta`. Jest odpowiedzialna za:

- (a) odbieranie zleceń od pakietu `KlientNIIKT`,
- (b) przekazywanie ich poprzez sieć do `ObslugiKlienta`,

- (c) odbieranie wyników od ObslugiKlienta,
- (d) przekazywanie wyników do pakietu KlientNIIKT.

2. Klasa ObslugaKlienta

Jest klasą, której instancja znajduje się na serwerze. Jest odpowiedzialna za:

- (a) odbieranie zleceń od ZdalnejObslugiKlienta,
- (b) przekazywanie ich do logiki biznesowej,
- (c) odbieranie wyników,
- (d) przekazywanie ich z powrotem do ZdalnejObslugiKlienta.

Klasy

Powstaną, odpowiednio, następujące klasy:

1. ZdalnaObslugaUzytkownika i ObslugaUzytkownika
2. ZdalnaObslugaFirmy i ObslugaFirmy
3. ZdalnaObslugaAdministradora i ObslugaAdministradora

5.5 Najważniejsze komponenty - Warstwa logiki biznesowej

5.5.1 Pakiet TransportowaneObiekty

Zakres odpowiedzialności

Jest odpowiedzialna za opisanie wszystkich rodzajów obiektów, które będą transportowane za pośrednictwem warstwy sieci i/lub będą używane poza logiką biznesową.

Klasy

1. Klasa Kryteria
Odpowiedzialna za umożliwienie określenia kryteriów wyszukiwania.
2. Klasa Ogloszenie
Odpowiedzialna za udostępnienie interfejsu do ogłoszenia, które jest częścią wyniku wyszukiwania.
3. Klasa OgloszeniePlatne
4. Klasa MiniStrona
5. Klasa Wizytowka

6. Klasa Szblon
7. Klasa WarstwaMapy
8. Klasa FragmentMapy
9. Klasa Ulica
10. Klasa FragmentUlicy
11. Klasa Branza
12. Klasa Grupa
13. Klasa NajpopularniejszaBranza
14. Klasa PlikGraficzny
15. Klasa ZawKataloguBranz
Zawiera zawartość całego katalogu branż.
16. Klasa ZawKataloguNajpop
Zawiera zawartość **Katalogu najpopularniejszych branż**.
17. Klasa Statystyka
Zawiera obliczone statystyki najpopularniejszych branż i firm.
18. Klasa KrytStat
Zawiera kryteria obliczenia statystyk.

5.5.2 Pakiet OpcjeUzytkownika

Jak już wspomniano w punkcie 5.1, pakiet OpcjeUzytkownika funkcjonalnie przynależy do warstwy logiki biznesowej. Jednakże komunikuje się on z warstwą interfejsu bezpośrednio, bez udziału sieci.

Zakres odpowiedzialności

Jest odpowiedzialny za:

1. zarządzanie „Ulubionymi” wpisami:
 - (a) dodawanie wpisu,
 - (b) modyfikacja wpisu,
 - (c) usuwanie wpisu;
2. przechowywanie historii zapytań i korzystanie z niej.

Pakiet OpcjeUzytkownika kontaktuje się z pakietem InterfejsUzytkownika.

Klasy

1. Klasa `Ulubione`
Jest odpowiedzialna za zarządzanie „Ulubionymi”.
2. Klasa `Historia`
Jest odpowiedzialna za umożliwienie *Użytkownikowi* korzystania z historii zapytań, tj. z kilku ostatnich wyników jego wyszukiwań.

5.5.3 Pakiet Ogłoszenia

Zakres odpowiedzialności

Pakiet `Ogłoszenia`:

1. jest interfejsem do informacji zawartej w ogłoszeniach,
2. umożliwia zarządzanie ogłoszeniami.

Klasy

1. Klasa `KatalogOgloszen`
Umożliwia:
 - (a) zarządzanie nowymi ogłoszeniami,
 - (b) zarządzanie nieaktualnymi ogłoszeniami,
 - (c) zarządzanie nieopłaconymi ogłoszeniami,
 - (d) wyszukiwanie ogłoszeń spełniających zadane kryterium.
2. Klasa `ZarzOglosz`
Klasa `ZarzOglosz` jest odpowiedzialna za wykonywanie działań administracyjnych na ogłoszeniu: potwierdzanie aktualności, zatwierdzanie, modyfikacja, przedłużanie terminu, konwersja płatnego ogłoszenia na bezpłatne, etc.

5.5.4 Pakiet Konto

Zakres odpowiedzialności

Jest odpowiedzialny za operacje dotyczące zarządzania kontem *Firmy* i *Administradora*:

1. tworzenie / usuwanie konta,
2. autoryzacja,
3. zmiana hasła,
4. zapomniane hasło.

Klasy

1. Klasa KontoFirmy

Jest odpowiedzialna za zarządzanie kontem *Firmy*.

2. Klasa KontoAdministradora

Jest odpowiedzialna za zarządzanie kontem *Administradora*. Nie można utworzyć więcej niż jednego konta *Administradora*.

5.5.5 Pakiet Mapa

Zakres odpowiedzialności

Jest odpowiedzialny za:

1. zarządzanie fragmentami mapy i ulic,
2. zarządzanie przetwarzaniem mapy i ulic.

Klasy

1. Klasa Mapy

Jest odpowiedzialna za administrowanie całą mapą. Potrafi podać dostępne fragmenty mapy i ich własności. Udostępnia najważniejsze operacje:

- (a) wprowadzenie nowej warstwy mapy,
- (b) pobranie wszystkich warstw mapy,
- (c) udostępnienie obiektu służącego do zarządzania warstwą mapy.

2. Klasa ZarzWarstwaMapy

Jest odpowiedzialna za umożliwienie operowania na warstwie mapy. Dostarcza metody:

- (a) pobierając fragment mapy,
- (b) dodając / usuwając fragment mapy,
- (c) dając obiekt do zarządzania fragmentem mapy.

3. Klasa ZarzFragmentMapy

Jest odpowiedzialna za zarządzanie fragmentem mapy. Umie pobrać ulice, które przebiegają przez fragment mapy.

4. Klasa Ulice

Jest odpowiedzialna za zarządzanie zbiorem ulic. Dostarcza operacje:

- (a) aktualizacja zbioru ulic,

- (b) wyszukiwanie ulicy,
- (c) podanie obiektu umożliwiającego zarządzanie ulicą.

5. Klasa ZarzUlica

Jest odpowiedzialna za zarządzanie ulicą i zdekomponowaniem jej na fragmenty. Umożliwia:

- (a) dodawanie / usuwanie fragmentów ulicy,
- (b) wyszukiwanie fragmentów ulicy,
- (c) obliczenie współrzędnych domu przy ulicy.

5.5.6 Pakiet Branze

Zakres odpowiedzialności

Jest odpowiedzialny za:

1. zarządzanie katalogiem (drzewem) branż,
2. zarządzanie **Katalogiem najpopularniejszych branż**.

Klasy

1. Klasa ZarzGrupa

Umożliwia zarządzanie grupą branż (czyli branżami, które posiadają przynajmniej jedną podbranż).

Klasa udostępnia operacje:

- (a) pobieranie podbranż lub podgrup dla grupy,
- (b) utworzenie, likwidacja, modyfikacja grupy branż.

2. Klasa KatalogBranz

Klasa umożliwia zarządzanie wszystkimi branżami:

- (a) dodawanie / usuwanie / modyfikacja branży,
- (b) wyszukiwanie branży,
- (c) generowanie statycznego katalogu branż,
- (d) generowanie statycznego **Katalogu najpopularniejszych branż**.

3. Klasa KatalogNajpopularniejszychBranz

Umożliwia pobranie **Katalogu najpopularniejszych branż** oraz jego zamianę na nowy.

5.5.7 Pakiet Szablony

Zakres odpowiedzialności

Pakiet Szablony jest odpowiedzialny za zarządzanie szablonami do korespondencji automatycznej. Szablony są wykorzystywane w trzech sytuacjach:

- do przypomnienia o zbliżającym się terminie płatności,
- do przypomnienia o zbliżającym się terminie aktualizacji ogłoszenia,
- do poinformowania o zmianie ogłoszenia z płatnego na bezpłatne.

Klasy

1. Klasa Szablon

Jest odpowiedzialna za dostarczenie interfejsu do pobierania i modyfikowania szablonów.

5.6 Najważniejsze komponenty - Warstwa danych

5.6.1 Pakiet Baza

Zakres odpowiedzialności

Jest odpowiedzialny za zarządzanie danymi przechowywanymi w bazie danych Książki Telefonicznej NIKT oraz w plikach danych na serwerze. Pakiet ten zarządza spójnością oraz dostępem do:

1. danych ogłoszeń firm,
2. drzewa dostępnych branż,
3. listy najpopularniejszych branż,
4. danych dotyczącymi mapy i ulic,
5. danych dotyczącymi identyfikacji / logowania się.

Umożliwia:

1. wprowadzanie, modyfikowanie i usuwanie danych,
2. przeszukiwanie danych,
3. potwierdzanie autoryzacji przy logowaniu.

Rozdział 6

Dekompozycja na procesy

6.1 Omówienie

Główny proces serwera: Proces uruchamiany jako pierwszy na serwerze, inicjuje i kończy obsługę zdalnego wywołania metod (RMC) dla *Użytkownika*, *Firmy* i *Administradora*.

Proces *Użytkownika*: Proces obsługujący interfejs *Użytkownika* i jego komunikację z serwerem. Działa po stronie *Użytkownika*.

Proces obsługi zdalnego wywoływania metod *Użytkownika*: Proces działa po stronie serwera, odbiera wywołane zdalnie przez interfejs *Użytkownika* metody, tworzy procesy mające je wykonać i przydziela im zadania.

Grupa procesów obsługująca *Użytkowników*: Procesy obsługujące po stronie serwera zapytania pochodzące od interfejsów *Użytkowników*: wydobywają dane z bazy, przetwarzają je i wysyłają *Użytkownikom*.

Proces *Firmy*: Proces obsługujący interfejs *Firmy* i jego komunikację z serwerem. Działa po stronie *Firmy*.

Proces obsługi zdalnego wywoływania metod *Firmy*: Proces działa po stronie serwera, odbiera wywołane zdalnie przez interfejs *Firmy* metody, tworzy procesy mające je wykonać i przydziela im zadania.

Grupa procesów obsługująca *Firmy*: Procesy obsługujące po stronie serwera zapytania pochodzące od interfejsów *Firm*: dokonują zleconych przez *Firmy* zmian w danych.

Proces *Administradora*: Proces obsługujący interfejs *Administradora* i komunikację *Administradora* z serwerem. Działa po stronie *Administradora*.

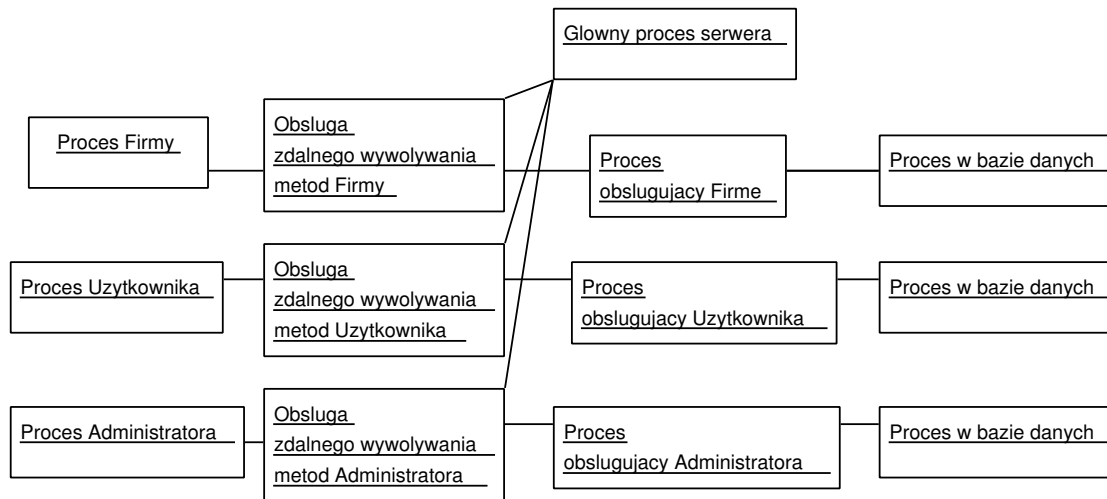
Proces obsługi zdalnego wywoływania metod *Administradora*: Proces działa po stronie serwera, odbiera wywołane zdalnie przez interfejs *Administradora* metody, tworzy procesy mające je wykonać i przydziela im zadania.

Proces obsługujący *Administradora*: Proces działający po stronie serwera, dokonujący zleconych przez interfejs *Administradora* zmian w bazie danych NIIKT.

Procesy w bazie danych: Procesy działające po stronie serwera, wykonują zlecone operacje na bazie danych sytemu.

6.2 Diagram

Rysunek 6.1: Dekompozycja na procesy



Rozdział 7

Instalacja systemu

Szczegółowy opis procesu instalacji całego systemu zostanie zawarty w Podręczniku instalacji, będącym częścią tworzonego projektu.

7.1 Moduł Użytkownika

W skład modułu wchodzi skompilowany aplet oraz kody źródłowe strony internetowej. Po stronie serwera uruchamiany jest skrypt instalacyjny, który umieszcza aplet *Użytkownika* i stronę zawierającą aplet w katalogu wskazanym przez *Administradora*, następnie z pomocą *Administradora* konfiguruje aplet (uzupełnia: adres serwera, z którym będzie się kontaktować, lokalizację plików pobieranych przez aplet). Po stronie *Użytkownika* moduł nie wymaga żadnej instalacji.

7.2 Moduł Firmy

W skład modułu wchodzi skompilowany aplet oraz kody źródłowe strony internetowej. Po stronie serwera uruchamiany jest skrypt instalacyjny, który umieszcza aplet *Firmy* i stronę zawierającą aplet w katalogu wskazanym przez *Administradora*, następnie z pomocą *Administradora* skonfiguruje aplet (uzupełnia: adres serwera, z którym będzie się kontaktować, lokalizację plików pobieranych przez aplet). Po stronie *Firmy* moduł nie wymaga żadnej instalacji.

7.3 Moduł Administratora

Moduł dostarczany jest w postaci binariów. Uruchamiany jest skrypt instalacyjny, który umieszcza aplikację we wskazanym przez *Administradora* katalogu, następnie z pomocą *Administradora* konfiguruje aplikację (uzupełnia: adres serwera, z którym będzie się aplikacja kontaktować, lokalizację plików pobieranych przez aplikację).

7.4 Serwlet NIIKT

Serwlet dostarczany jest w postaci binariów. Uruchamiany jest skrypt instalacyjny, który umieszcza serwlet we wskazanym przez *Administratora* katalogu, kopiuje pliki potrzebne apletom w podanych przez *Administratora* katalogach (cenniki, pliki z pomocą), zakłada we wskazanym miejscu strukturę katalogów dla logo, mapy, katalogu branż, **Katalogu najpopularniejszych branż**, szablonów. Następnie z pomocą *Administratora* konfiguruje:

1. Połączenie z serwerem bazy danych.
2. Bazę danych, zakładając odpowiednie tabele i nakładając na nie więzy.
3. Połączenie z serwerem WWW.
4. Połączenie z serwerem poczty elektronicznej.

Rozdział 8

Implementacja systemu

8.1 Omówienie

Niniejszy rozdział zawiera opis implementacji systemu NIIKT. Logiczna dekompozycja systemu została opisana w rozdziale 5 oraz zilustrowana na diagramie 5.1.

Na opis implementacji systemu składają się:

1. opisy warstw: interfejsu (8.2.1), sieci (8.2.2), logiki (8.2.3) oraz bazy danych (8.2.4). Każda warstwa została zdekomponowana na zbiór klas wraz z zależnościami między nimi oraz zilustrowana na odpowiednim diagramie.
2. opisy realizacji przypadków użycia w punkcie 8.3, uwzględniające przypadki wyróżnione w rozdziale 4. Każdy przypadek użycia został zilustrowany diagramem sekwencji.

8.2 Warstwy

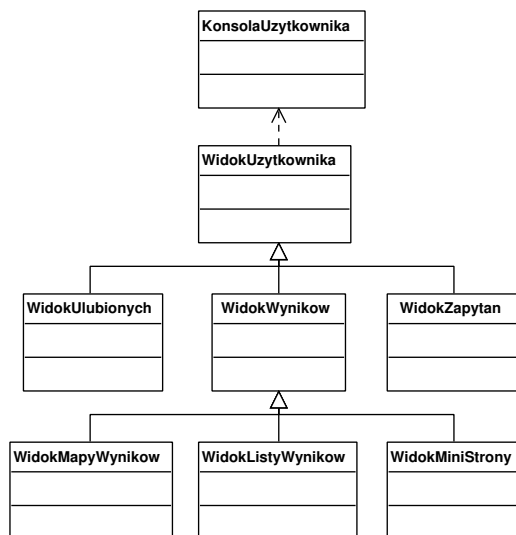
8.2.1 Warstwa interfejsu

W klasach `KonsolaUzytkownika`, `KonsolaFirmy` i `KonsolaAdministratora` zostaną zaimplementowane metody, które będzie mogła wywołać warstwa sieciowa w wypadku zakończenia operacji zleconych przez *Użytkownika*, *Firmę* lub *Administratora* po stronie serwera. Będzie to zbiór metod, wywoływanych zarówno w wypadku powodzenia jak i niepowodzenia akcji, których zadaniem będzie powiadomienie korzystającego z interfejsu o przebiegu i wyniku zleconych przez niego akcji. Ze względu na i tak znaczną objętość warstwy interfejsu oraz ścisły związek z metodami warstwy sieciowej (wyspecyfikowanymi w rozdz. 8.2.2) zdecydowaliśmy się nie zamieszczać tych metod w opisie, aby pozostał on bardziej czytelny.

1. Pakiet `InterfejsUzytkownika`

Diagram klas dla pakietu `InterfejsUzytkownika` przedstawiony został na rysunku 8.1 na stronie 33.

Rysunek 8.1: Warstwa interfejsu – pakiet InterfejsUzytkownika



(a) Klasa KonsolaUzytkownika

- `void pokazUlubione()`
Przełącza widok na „Ulubione” – adresy zapamiętane przez *Użytkownika*.
- `void pokazWyniki(Collection of Ogloszenie) wynik`
Przełącza widok na wizualizację wyników wyszukiwania. Jeśli *Użytkownik* zlecił wyszukiwanie z wizualizacją na mapie, to operacja jest zlecana obiektowi klasy `WidokMapyWynikow`, w przeciwnym wypadku – obiektowi klasy `WidokListyWynikow`.
- `void pokazZapytania()`
Przełącza widok na ekran umożliwiający skonstruowanie zapytania.
- `void usunUlubioneOgloszenie(UlubioneOgloszenie ogloszenie)`
Przekazuje obiektowi odpowiedzialnemu za zarządzanie „Ulubionymi” polecenie usunięcia ogłoszenia `ogloszenie`.
- `void zmienUlubioneOgloszenie(UlubioneOgloszenie stare, UlubioneOgloszenie nowe)`
Przekazuje obiektowi odpowiedzialnemu za zarządzanie „Ulubionymi” polecenie zmiany ogłoszenia `stare` na `nowe`.
- `void pokazLokalizacje(Ogloszenie ogloszenie)`
Przekazanie obiektowi klasy `WidokMapyWynikow` polecenia wyświetlenia fragmentu mapy z wyróżnioną lokalizacją *Firmy* związanej z ogłoszeniem.
- `void pobierzFragmentMapy(Punkt p)`
Zleca obiektowi klasy `ZdalnaObslugaUzytkownika` pobranie odpowied-

niego fragmentu mapy.

(b) Klasa `WidokUzytkownika`

Klasa abstrakcyjna zawierająca wizualizację menu *Użytkownika*, **Historii zapytań** i zapewniająca komunikację systemu z *Użytkownikiem*.

- `void wyswietl()`
Metoda wyświetlająca segmenty ekranu, w których będzie znajdował się dostęp do **Historii zapytań**, menu dostępnych opcji i komunikacja z *Użytkownikiem*.
- `void pokazUlubione()`
Przekazuje obiektowi klasy `KonsolaUzytkownika` polecenie wyświetlenia „Ulubionych”.
- `void pokazZapytania()`
Przekazuje obiektowi klasy `KonsolaUzytkownika` polecenie wyświetlenia ekranu umożliwiającego skonstruowanie zapytania.

Podklasy:

i. Klasa `WidokUlubionych`

- `void pokazUlubione()`
Ponieważ wyświetlone są już „Ulubione” metoda nie wykonuje żadnych działań.
- `void usunUlubioneOgloszenie()`
Przekazuje obiektowi klasy `KonsolaUzytkownika` polecenie usunięcia wskazanego przez *Użytkownika* ogłoszenia z „Ulubionych”.
- `void edytujUlubioneOgloszenie()`
Wyświetla ekran edycji ogłoszenia z „Ulubionych” wskazanego przez *Użytkownika*.
- `void zmienUlubioneOgloszenie()`
Przekazuje obiektowi odpowiedzialnemu za zarządzanie „Ulubionymi” polecenie zmiany treści ogłoszenia wskazanego przez *Użytkownika*.
- `void wyswietlUlubione()`
Wyświetlenie adresów zapamiętanych w „Ulubionych” na ekranie.

ii. Klasa `WidokWynikow`

Klasa abstrakcyjna; jej podklasy są odpowiedzialne za wyświetlanie wyników na ekranie.

- `void pokazMiniStrone()`
Wyświetlenie ekranu z zawartością mini strony związanej z ogłoszeniem wskazanym przez *Użytkownika*.
- `void dodajDoUlubionych()`
Zlecenie dodania do „Ulubionych” nowego ogłoszenia wskazanego przez *Użytkownika*.
- `void zapytanie()`
Przekazuje obiektowi klasy `ZdalnaObslugaUzytkownika` (poprzez o-

biekt klasy *ZdalnaObslugaUzytkownika*) zapytania złożone przez *Użytkownika*.

- `void generujWizytowke()`
Wygenerowanie wizytówki firmy.

Podklasy:

A. Klasa *WidokListyWynikow*

- `void wyswietlListeWynikow(Collection of Ogloszenie) wynik, Kryteria kryteria)`
Wyświetlenie wyników wyszukiwania w postaci listy.
- `void pokazNaMapie()`
Zlecenie obiektowi klasy *ZdalnaObslugaUzytkownika* ściągnięcia i przekazania fragmentu mapy, na którym zostanie zaznaczona lokalizacja firmy z ogłoszenia wskazanego przez *Użytkownika*.

B. Klasa *WidokMapyWyników*

- `void wyswietlMapeWynikow(Collection of Ogloszenie) wynik, Kryteria kryteria)`
Wyświetlenie wyników wyszukiwania w postaci mapy.
- `void zmianaPrzyblizenia()`
Zmiana przybliżenia widocznej mapy.
- `void zmienWidocznyFragment()`
Zmiana wyświetlanego fragmentu na wskazany przez *Użytkownika*.
- `void wyswietlLokalizajce(Ogloszenie ogloszenie)`
Wyróżnienie na mapie lokalizacji *Firmy* związanej z podanym ogłoszeniem.

C. Klasa *WidokMiniStrony*

- `void wyswietlMinistrone(Ogloszenie ogloszenie)`
Wyświetla na ekranie mini stronę związaną z ogłoszeniem.
- `void drukuj()`
Przekazuje specjalną, przeznaczoną do wydruku wersję wyświetlanej mini strony.

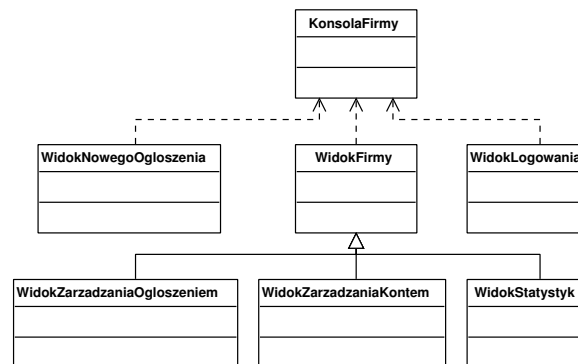
iii. Klasa *WidokZapytan*

- `void pokazZapytania()`
Przedefiniowanie metody z nadklasy tak, aby nie wykonywała żadnych działań.
- `void zapytanie()`
Przekazuje obiektowi klasy *ZdalnaObslugaKlienta* polecenie wykonania zapytania złożonego przez *Użytkownika*.

2. Pakiet InterfejsFirmy

Diagram klas dla pakietu `InterfejsFirmy` przedstawiony został na rysunku 8.2 na stronie 36.

Rysunek 8.2: Warstwa interfejsu – pakiet `InterfejsFirmy`



(a) Klasa `KonsolaFirmy`

- `void pokazEkranLogowania()`
Wyświetlenie ekranu logowania *Firmy*.
- `void pokazEkranNowegoOgloszenia()`
Wyświetlenie ekranu, służącego do dodawania nowego konta i ogłoszenia.
- `void pokazEkranZarzadzaniaOgloszeniem(Ogloszenie ogloszenie)`
Wyświetlenie ekranu, w którym można zmieniać zawartość ogłoszenia.
- `void pokazEkranZarzadzaniaKontem(Ogloszenie ogloszenie)`
Wyświetlenie ekranu, w którym można zmieniać szczegóły dotyczące konta *Firmy*.
- `void pokazStatystyki(Statystyka statystyka)`
Wyświetlenie ekranu, w którym można obejrzeć statystyki związane z ogłoszeniem
- `void pobierzKatNajpopularniejszychBranz()`
Pobiera z serwera i interpretuje plik z katalogiem branż.
- `void pobierzKatNajpopularniejszychBranz()`
Pobiera z serwera i interpretuje plik z **Katalogiem najpopularniejszych branż**.
- `void zmianaOgloszeniaOk(Ogloszenie ogloszenie)`
Metoda zmienia ogłoszenie, na którym aktualnie pracuje interfejs.

- `void noweOgloszenieOk()`
Potwierdzenie przyjęcia nowego ogłoszenia do kolejki ogłoszeń oczekujących na zatwierdzenie przez *Administratora*.

(b) Klasa `WidokFirmy`

- `void wyloguj()`
Kończy sesję pracy.
- `void pokazEkranZarzadzaniaOgloszeniem()`
Przekazanie obiektowi klasy `KonsolaFirmy` polecenia wyświetlenia ekranu, w którym można zmieniać zawartość ogłoszenia.
- `void pokazEkranZarzadzaniaKontem()`
Przekazanie obiektowi klasy `KonsolaFirmy` polecenia wyświetlenia ekranu, w którym można zmieniać szczegóły dotyczące konta *Firmy*.
- `void wyswietlWidokFirmy()`
Wyświetlenie menu dostępnych *Firmie* opcji.

Podklasy:

i. Klasa `WidokZarzadzaniaOgloszeniem`

- `void pokazEkranZarzadzaniaOgloszeniem()`
Przeddefiniowanie metody tak, aby nie podejmowała żadnych działań.
- `void wyswietlWidokZarzadzaniaOgloszeniem()`
Wyświetlenie ekranu umożliwiającego edycję i aktualizację ogłoszenia.
- `void aktualizujOgloszenie()`
Metoda zleca obiektowi klasy `ZdalnaObslugaFirm` aktualizację ogłoszenia.
- `void edytujOpcjeFirmyWymagajacej()`
Wyświetlenie ekranu umożliwiającego edycję danych związanych z ogłoszeniem *Firmy wymagającej* (mini strona, logo i dodatkowe dane).
- `void zatwierdzOpcjeFirmyWymagajacej()`
Zlecenie obiektowi klasy `ZdalnaObslugaUzytkownika` zatwierdzenia zmian w ogłoszeniu dokonanych przez *Firmę wymagającą*.
- `void zmienOgloszenie()`
Metoda zleca obiektowi klasy `ZdalnaObslugaFirmy` zmianę treści ogłoszenia.

ii. Klasa `WidokZarzadzaniaKontem`

- `void pokazEkranZarzadzaniaKontem()`
Przeddefiniowanie metody tak, aby nie podejmowała żadnych działań.
- `void wyswietlWidokZarzadzaniaKontem(Ogloszenie ogloszenie)`
Wyświetlenie ekranu, w którym można dokonywać zmian w danych związanych z kontem *Firmy*.

- `void zmienHaslo()`
Metoda zleca obiektowi klasy `ZdalnaObslugaFirmy` zmianę hasła *Firmy* do konta.
- `void usunOgloszenie()`
Metoda zleca obiektowi klasy `ZdalnaObslugaFirmy` usunięcie ogłoszenia i konta z nim związanego.

iii. Klasa `WidokStatystyk`

- `void pokazStatystyki()`
Przedefiniowanie metody tak, aby nie podejmowała żadnych działań.
- `void wyswietlStatystyki(Statystyka statystyka)`
Wyświetlenie na ekranie statystyk dotyczących ogłoszenia *Firmy*.

(c) Klasa `WidokLogowania`

- `void wyswietlWidokLogowania()`
Wyświetla ekran umożliwiający zalogowanie się.
- `void pokazEkranNowegoOgloszenia()`
Przekazanie obiektowi klasy `KonsolaFirmy` polecenia wyświetlenia ekranu umożliwiającego zgłoszenie nowego ogłoszenia.
- `Ogloszenie loguj()`
Przekazanie chęci zalogowania wraz z hasłem i NIPem wpisanym przez *Firmę* obiektowi klasy `ZdalnaObslugaFirmy`.

(d) Klasa `WidokNowegoOgloszenia`

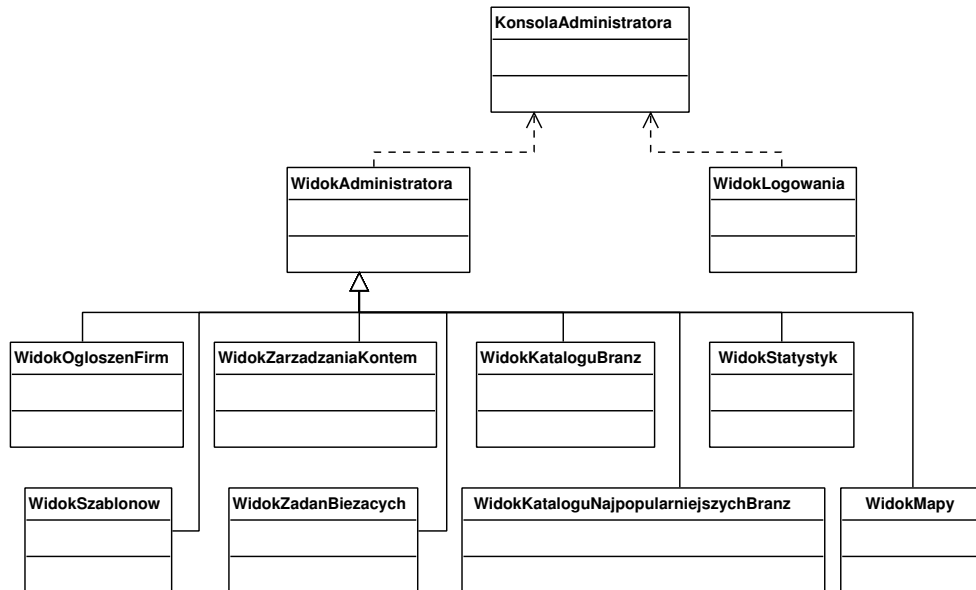
- `void wyswietlWidokNowegoOgloszenia()`
Wyświetla ekran umożliwiający zgłoszenie nowego ogłoszenia.
- `void pokazEkranLogowania()`
Przekazanie obiektowi klasy `KonsolaFirmy` polecenia wyświetlenia ekranu logowania.
- `void noweOgloszenie()`
Przekazanie obiektowi klasy `ZdalnaObslugaFirmy` nowego ogłoszenia *Firmy*.
- `void edytujOpcjeFirmyWymagajacej()`
Wyświetlenie ekranu umożliwiającego edycję danych związanych z ogłoszeniem *Firmy wymagającej* (mini strona, logo i dodatkowe dane).
- `void zatwierdzOpcjeFirmyWymagajacej()`
Zapamiętanie dodatkowej części ogłoszenia *Firmy wymagającej* (mini strona, logo i dodatkowe dane).

3. Pakiet `InterfejsAdministratora`

Diagram klas dla pakietu `InterfejsAdministratora` przedstawiony został na rysunku 8.3 na stronie 39.

(a) Klasa `KonsolaAdministatora`

Rysunek 8.3: Warstwa interfejsu – pakiet InterfejsAdministratora



- `void pokazWidokOgloszenFirm()`
Przełączenie się na ekran, w którym można dokonywać zmian w ogłoszeniach *Firm*.
- `void pokazWidokZadanBiezacych()`
Przełączenie się na ekran, w którym *Administrator* obsługuje nowe, nieopłacone i nieaktualne ogłoszenia *Firm*.
- `void pokazWidokLogowania()`
Przełączenie się na ekran, w którym *Administrator* może się zalogować.
- `void pokazWidokKataloguBranz()`
Przełączenie się na ekran, w którym *Administrator* wykonuje czynności związane z zarządzaniem katalogiem branż.
- `void pokazWidokKataloguNajpopularniejszychBranz()`
Przełączenie się na ekran, w którym *Administrator* wykonuje czynności związane z zarządzaniem **Katalogiem najpopularniejszych branż**.
- `void pokazWidokSzablonow()`
Przełączenie się na ekran, w którym *Administrator* modyfikuje treść e-maili wysyłanych automatycznie przez system.
- `void pokazWidokMapy()`
Przełączenie się na ekran, w którym *Administrator* wykonuje czynności związane z zarządzaniem mapą.
- `void odbierzWarstwy()`

Zapamiętuje odebrane warstwy mapy.

- void odbierzFragmentXY()
Zapamiętuje odebrany fragment mapy.

(b) Klasa WidokLogowania

- void wyswietl()
Wyświetlenie ekranu logowania *Administratora*.
- void loguj()
Zlecenie obiektowi klasy ZdalnaObslugaAdministradora zalogowania *Administratora*.

(c) Klasa WidokAdministradora

- void wyloguj()
Zakończenie pracy z **Modułem Administoratora**
- void pokazWidokOgloszenFirm()
Przekazanie obiektowi klasy KonsolaAdministatora polecenia wyświetlenia ekranu, w którym można dokonywać zmian w ogłoszeniach *Firm*.
- void pokazWidokZadanBiezacych()
Przekazania obiektowi klasy KonsolaAdministradora polecenia wyświetlenia ekranu, w którym *Administrator* obsługuje nowe, nieopłacone i nieaktualne ogłoszenia *Firm*.
- void pokazWidokKataloguBranz()
Przekazanie obiektowi klasy KonsolaAdministradora polecenia wyświetlenia ekranu, w którym *Administrator* wykonuje czynności związane z zarządzaniem katalogiem branż.
- void pokazWidokKataloguNajpopularniejszychBranz()
Przekazanie obiektowi klasy KonsolaAdministoratora polecenia wyświetlenia ekranu, w którym *Administrator* wykonuje czynności związane z zarządzaniem **Katalogiem najpopularniejszych branż**.
- void pokazWidokSzablonow()
Przekazanie obiektowi klasy KonsolaAdministradora polecenia wyświetlenia ekranu, w którym *Administrator* modyfikuje treść e-mail wysyłanych automatycznie przez system.
- void pokazWidokMapy()
Przekazanie obiektowi klasy KonsolaAdministradora polecenia wyświetlenia ekranu, w którym *Administrator* wykonuje czynności związane z zarządzaniem mapą.
- void wyswietl()
Wyświetlenie menu dostępnych *Administratorowi* opcji.

Podklasy:

- i. Klasa WidokOgloszenFirm

- void pokazWidokOgloszenFirm()
Przeddefiniowane metody tak, aby nie wykonywała żadnych działań.
- void wyswietlWidokOgloszen(
 (Collection of Ogloszenie) ogloszenia)
Wyświetlenie ekranu, w którym *Administrator* może dokonać edycji lub usunięcia dowolnego ogłoszenia *Firmy*.
- void edytujOgloszenie()
Wyświetlenie ekranu, w którym można edytować ogłoszenie wskazane przez *Administradora*.
- void zatwierdzOgloszenie()
Przekazanie obiektowi klasy *ZdalnaObslugaAdministradora* polecenia zatwierdzenia zmian w ogłoszeniu dokonanych przez *Administradora*.
- void usunOgloszenie()
Przekazanie obiektowi klasy *ZdalnaObslugaAdministradora* polecenia usunięcia ogłoszenia wskazanego przez *Administradora*.
- void filtrujOgloszenia()
Przekazanie obiektowi klasy *ZdalnaObslugaAdministradora* polecenia ściągnięcia i wyświetlenia ogłoszeń *Firm* spełniających kryteria podane przez *Administradora*.

ii. Klasa WidokZadanBiezacych

- void pokazWidokZadanBiezacych()
Przeddefiniowane metody tak, aby nie wykonywała żadnych działań.
- void wyswietlWidokZadanBiezacych()
Wyświetlenie ekranu, w którym *Administrator* może dokonać edycji albo usunięcia dowolnego nowego ogłoszenia *Firmy* lub przedłużenia terminu ogłoszenia nieaktualnego.
- void potwierdzAktualnosc()
Zleca obiektowi klasy *ZdalnaObslugaAdministradora* potwierdzenie aktualności ogłoszenia wskazanego przez *Administradora*.
- void edytujOgloszenie()
Wyświetlenie ekranu umożliwiającego edycję ogłoszenia wskazanego przez *Administradora*.
- void zmienOgloszenie()
Przekazanie obiektowi klasy *ZdalnaObslugaAdministradora* polecenia zatwierdzenia zmian w treści ogłoszenia wprowadzonych przez *Administradora*.
- void zatwierdzOgloszenie()
Zleca obiektowi klasy *ZdalnaObslugaAdministradora* zatwierdzenie nowego ogłoszenia wskazanego przez *Administradora*.
- void usunOgloszenie()

Zleca obiektowi klasy *ZdalnaObslugaAdministratora* usunięcie ogłoszenia wskazanego przez *Administradora*.

iii. Klasa *WidokZarzadzaniaKontem*

- `void pokazWidokZarzadzaniaKontem()`
Przeddefiniowane metody tak, aby nie wykonywała żadnych działań.
- `void wyswietlWidokZarzadzaniaKontem()`
Wyświetlenie ekranu, w którym *Administrator* może zmienić swoje hasło.
- `void zmienHaslo()`
Zlecenie obiektowi klasy *ZdalnaObslugaAdministratora* zmiany hasła *Administradora*.

iv. Klasa *WidokKataloguBranz*

- `void pokazKatalogBranz()`
Przeddefiniowanie metody tak, aby nie podejmowała żadnych działań.
- `void wyswietlKatalogBranz(Collection of Branza)`
Wyświetlenie ekranu umożliwiającego *Administratorowi* zarządzanie katalogiem branż.
- `void usunBranze()`
Zlecenie obiektowi klasy *ZdalnaObslugaAdministratora* usunięcia branży wskazanej przez *Administradora* z katalogu branż.
- `void dodajBranze()`
Zlecenie obiektowi klasy *ZdalnaObslugaAdministratora* dodania branży zdefiniowanej przez *Administradora* do katalogu branż.
- `void generujPlikKatalogu()`
Zlecenie obiektowi klasy *ZdalnaObslugaAdministratora* wygenerowania na serwerze pliku z katalogiem branż, z którego będą korzystać aplety i aplikacja *Administradora*.
- `void edytujBranze()`
Wyświetlenie ekranu umożliwiającego edycję branży.
- `void dodajBranzeDoGrupy()`
Zlecenie obiektowi klasy *ZdalnaObslugaAdministratora* dodania edytowanej branży do wskazanej przez *Administradora* grupy.
- `void usunBranzeZGrupy()`
Zlecenie obiektowi klasy *ZdalnaObslugaAdministratora* usunięcia edytowanej branży ze wskazanej przez *Administradora* grupy.
- `void zatwierdzBranze()`
Zlecenie obiektowi *ZdalnaObslugaAdministratora* zatwierdzenia zmian dokonanych przez *Administradora* we właściwościach branży.

v. Klasa *WidokKataloguNajpopularniejszychBranz*

- `void pokazWidokNajpopularniejszychBranz()`
Przeddefiniowanie metody tak, aby nie podejmowała żadnych działań.

- `void wyswietKatalogNajpopularniejszychBranz (Collection of NajpopularniejszaBranza) katalog)`
Wyświetlenie ekranu umożliwiającego *Administratorowi* zarządzanie **Katalogiem najpopularniejszych branż**.
- `void usunBranze()`
Usunięcie branży wskazanej przez *Administradora* z lokalnej kopii **Katalogu najpopularniejszych branż**.
- `void dodajBranze()`
Dodanie branży wskazanej przez *Administradora* do lokalnej kopii **Katalogu najpopularniejszych branż**.
- `void generujPlikKatalogu()`
Zlecenie obiektowi klasy *ZdalnaObslugaAdministradora* wygenerowania na serwerze pliku z **Katalogiem najpopularniejszych branż**, z którego będą korzystać aplety i aplikacja *Administradora*.
- `void edytujBranze()`
Wyświetlenie ekranu umożliwiającego edycję najpopularniejszej branży.
- `void zatwierdzBranze()`
Wprowadzenie zmian dokonanych przez *Administradora* we właściwościach najpopularniejszej branży w lokalnej kopii **Katalogu najpopularniejszych branż**.
- `void wyslijKatalogNajpopularniejszychBranz()`
Zlecenie obiektowi klasy *ZdalnaObslugaAdministradora* wysłania lokalnej kopii **Katalogu najpopularniejszych branż** w celu uaktualnienia oryginalnego katalogu.
- `void przesunWGore()`
Przesunięcie wybranej branży w górę w lokalnej kopii **Katalogu najpopularniejszych branż**.
- `void przesunWDol()`
Przesunięcie wybranej branży w dół w lokalnej kopii **Katalogu najpopularniejszych branż**.

vi. Klasa *WidokSzablonow*

- `void pokazWidokSzablonow()`
Przeddefiniowanie metody tak, aby nie podejmowała żadnych działań.
- `void wyswietlWidokSzablonow(Collection of Szablon)`
Wyświetlenie ekranu umożliwiającego *Administratorowi* edycję treści szablonów e-maili wysyłanych automatycznie przez System.
- `void pokazTrescSzablonu()`
Wyświetlenie na ekranie treści szablonu e-maila wskazanego przez *Administradora*.
- `void zmienTrescSzablonu()`
Przekazanie obiektowi klasy *ZdalnaObslugaAdministradora* pole-

cenia zachowania zmian wprowadzonych przez *Administradora* w treści szablonu e-maili wysyłanych automatycznie przez System.

vii. Klasa *WidokStatystyk*

- `void pokazWidokStatystyk()`
Przedefiniowanie metody tak, aby nie podejmowała żadnych działań.
- `void wywietlWidokStatystyk()`
Wyświetla ekran, w którym *Administrator* może obejrzeć statystyki związane z ogłoszeniami z bieżącego lub poprzedniego miesiąca.

viii. Klasa *WidokMapy*

- `void pokazWidokMapy()`
Przedefiniowanie metody tak, aby nie podejmowała żadnych działań.
- `void wyswietlWidokMapy()`
Wyświetlenie ekranu umożliwiającego zarządzanie mapą.
- `void dodajFragmentMapy()`
Przekazanie obiektowi klasy *ZdalnaObslugaAdministradora* polecenia dodania nowego fragmentu mapy wskazanego przez *Administradora*.
- `void usunFragmentMapy()`
Przekazanie obiektowi klasy *ZdalnaObslugaAdministradora* polecenia usunięcia fragmentu mapy wskazanego przez *Administradora*.
- `void dodajUlice()`
Przekazanie obiektowi klasy *ZdalnaObslugaAdministradora* polecenia dodania ulicy zdefiniowanej przez *Administradora*.
- `void edytujUlice()`
Wyświetlenie ekranu umożliwiającego edycję właściwości ulicy.
- `void zatwierdzUlice()`
Przekazanie obiektowi klasy *ZdalnaObslugaAdministradora* polecenia zatwierdzenia zmian wprowadzonych przez *Administradora* we właściwościach ulicy.
- `void usunUlice()`
Przekazanie obiektowi klasy *ZdalnaObslugaAdministradora* polecenia usunięcia wskazanej przez *Administradora* ulicy.
- `void dodajPoczatekFragmentuUlicy()`
Zaznaczenie na mapie punktu, gdzie będzie zaczynał się fragment ulicy.
- `void dodajKoniecFragmentuUlicy()`
Zaznaczenie na mapie punktu, gdzie będzie kończył się fragment ulicy.
- `void dodajFragmentUlicy()`
Przekazanie obiektowi klasy *ZdalnaObslugaAdministradora* polecenia dodania fragmentu ulicy zdefiniowanego przez *Administradora*.
- `void usunFragmentUlicy()`
Przekazanie obiektowi klasy *ZdalnaObslugaAdministradora* polecenia usunięcia fragmentu ulicy wskazanego przez *Administradora*.

- `void edytujFragmentUlicy()`
Rozpoczęcie edycji fragmentu ulicy wskazanego przez *Administradora*.
- `void zatwierdzFragmentUlicy()`
Przekazanie obiektowi klasy *ZdalnaObslugaAdministradora* polecenia zatwierdzenia zmian we właściwościach ulicy dokonanych przez *Administradora*.
- `void zmienWarstwe()`
Przekazanie obiektowi klasy *KonsolaAdministradora* polecenia ściągnięcia i przekazania do wyświetlania warstwy mapy wskazanej przez *Administradora*.
- `void pobierzNazwePliku()`
Wyświetla ekran, w którym *Administrator* wprowadza nazwę pliku, który będzie skojarzony z fragmentem mapy przez niego wskazanym.
- `void edytujWarstwe()`
Wyświetlenie ekranu umożliwiającego edycję właściwości warstwy.
- `void zatwierdzWarstwe()`
Zlecenie obiektowi klasy *ZdalnaObslugaAdministradora* aktualizacji informacji o warstwie.
- `void dodajWarstwe()`
Zlecenie obiektowi klasy *ZdalnaObslugaAdministradora* dodania nowej warstwy.

8.2.2 Warstwa sieci

Metody z klas z pakietu *ObslugaKlienta* działają asynchronicznie. Część metod z pakietu *ZdalnaObslugaAdministradora* może działać synchronicznie, gdyż wątki *Administradora* są uprzywilejowane.

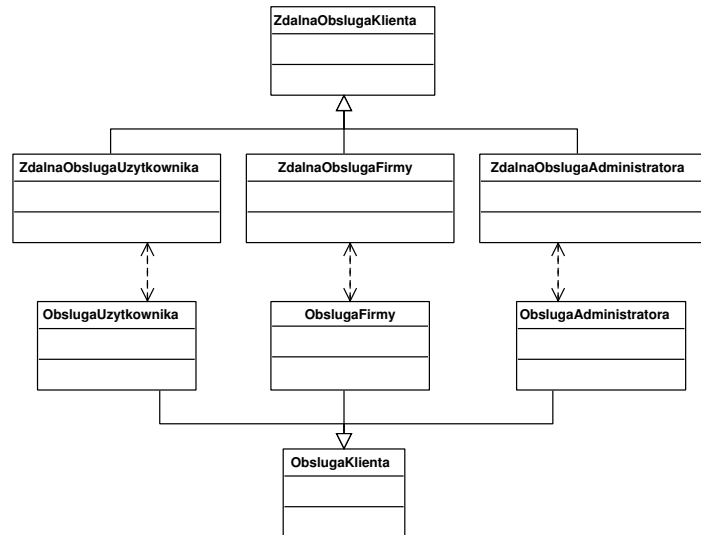
Typowy scenariusz funkcjonowania będzie następujący:

1. Interfejs wywołuje metodę *ZdalnejObslugiKlienta* o nazwie `pobierzX()`.
2. *ZdalnaObslugaKlienta* klienta wywołuje metodę `pobierzX()` w *ObsludzeKlienta* już po stronie serwera (i kończy się).
3. *ObslugaKlienta* zleca wykonanie odpowiedniej metody w logice biznesowej (i kończy się).
4. Gdy logika biznesowa zrealizuje zlecenie, informuje o tym i przekazuje wyniki do *ObslugiKlienta* za pomocą metody np. `pobranieXSukces()`.
5. *ObslugaKlienta* na serwerze przekazuje otrzymane informacje do *ZdalnejObslugiKlienta*, wywołując w niej metodę o takiej samej nazwie.
6. *ZdalnaObslugaKlienta* wywołuje odpowiednią metodę interfejsu.

Widać, że nazwy metod w klasach `ZdalnaObslugaKlienta` oraz `ObslugaKlienta` będą takie same.

Diagram klas dla warstwy sieci przedstawiony został na rysunku 8.4 na stronie 46.

Rysunek 8.4: Warstwa sieci



1. ObslugaUzytkownika

(a) Metody związane z wykonywaniem zapytań.

- `void zlecZapytanie(Kryteria kryteria)`
Zleca wykonanie zapytania.
- `void dostarczWynikiZapytania(Collection of Ogloszenie)`
Przekazuje wynik zapytania do interfejsu.
- `void zapytaniePorazka(Liczba blad)`
- `void zlecLokalizacje(Ogloszenie ogloszenie)`
Zleca wykonanie operacji zlokalizowania firmy na odpowiednim fragmencie mapy.
- `void dostarczLokalizacje(WarstwaMapy warstwa, FragmentMapy fragment)`
Przekazuje wynik lokalizowania.
- `void zlecenieLokalizacjiPorazka(Liczba blad)`

2. ObslugaFirmy

(a) Metody związane z zarządzaniem kontem firmy:

- `void zlecAutoryzacje(String nip, String haslo)`
Zleca autoryzację firmy za pomocą wprowadzonego hasła.
- `void autoryzacjaSukces ()`
- `void autoryzacjaPorazka(Liczba blad)`
Informacja o wyniku logowania do interfejsu.
- `void zlecZmianeHasla(String nip, String stare, String nowe)`
- `void zmianaHaslaSukces(Ogloszenie ogloszenie)`
- `void zmianaHaslaPorazka(Liczba blad)`
- `void przypomnienieHasla(String nip)`
- `void przypomnienieHaslaSukces()`
- `void przypomnienieHaslaPorazka(Liczba blad)`

(b) Metody związane z ogłoszeniami:

- `void zlecAktualizacjeOgloszenia(Ogloszenie nowe, Ogloszenie stare)`
Przekazuje nowe ogłoszenie, które ma zastąpić stare.
- `void aktualizacjaSukces(Ogloszenie ogloszenie)`
- `void aktualizacjaPorazka(Liczba blad)`
Informacja o wyniku: sukces lub porażka aktualizacji ogłoszenia. Porażka może nastąpić np. przy niepoprawnych danych. W przypadku sukcesu zwracamy pobrane dla firmy jej ogłoszenie.
- `void zlecNoweOgloszenie(Ogloszenie ogloszenie)`
- `void noweOgloszenieSukces()`
- `void noweOgloszeniePorazka(Liczba blad)`
Informacja o wyniku wstawiania nowego ogłoszenia: sukces lub porażka.
- `void zlecUsunOgloszenie(Ogloszenie ogloszenie)`
Zleca usunięcie ogłoszenia firmy.
- `void zlecenieUsunieciaOgloszeniaSukces()`
- `void zlecenieUsunieciaOgloszeniaPorazka(Liczba blad)`
- Metody do pobierania mini strony oraz statystyk związanych z danym ogłoszeniem nie są potrzebne, gdyż informacje te zawierają się w obiekcie klasy `Ogloszenie`.

3. Obsługa Administratora

(a) Metody związane z zarządzaniem kontem:

- `void zlecLogowanie(String haslo)`
Zleca autoryzację administratora za pomocą wprowadzonego hasła.
- `void logowanieSukces()`

- void logowaniePorazka(Liczba blad)
Informacja o wyniku logowania do interfejsu.
- void zmianaHasla(String stare, String nowe)
- void zmianaHaslaSukces()
- void zmianaHaslaPorazka(Liczba blad)
Informacja o wyniku operacji zmiany hasła.

(b) Metody związane z zarządzaniem zbiorami ogłoszeń:

- void pobierzOgloszenia(Kryteria kryteria)
Zleca pobranie wszystkich ogłoszeń.
- void pobierzNoweOgloszenia()
Zleca pobranie nowych ogłoszeń, tj. takich, które nie zostały jeszcze zaakceptowane.
- void pobierzNieaktualneOgloszenia()
Zleca pobranie ogłoszeń, którym upłynął termin wymaganej aktualizacji.
- void pobierzNieoplaconeOgloszenia()
Zleca pobranie ogłoszeń, które nie zostały opłacone, pomimo upłynięcia terminu płatności.
- void dostarczOgloszenia(
 (Collection of Ogloszenie) zestaw)
Informacja, że ogłoszenia zostały pobrane oraz zbiór pobranych ogłoszeń.
- void odbierzNoweOgloszenia(
 (Collection of Ogloszenie) zestaw)
- void odbierzNieaktualneOgloszenia(
 (Collection of Ogloszenie) zestaw)
- void odbierzNieoplaconeOgloszenia(
 (Collection of Ogloszenie) zestaw)
- void odbierzNowePorazka(Liczba blad)
- void odbierzNieaktualnePorazka(Liczba blad)
- void odbierzNieoplaconePorazka(Liczba blad)
- void pokazStatystyki(KrytStat kryteria)
Przekazuje zlecenie obliczenia statystyk według określonych kryteriów.
- void odbierzStatystyki(Statystyka stats)
- void odbierzStatystykiPorazka(liczba blad)

(c) Metody związane z zarządzaniem pojedynczym ogłoszeniem.

- void usunOgloszenie(Ogloszenie ogloszenie)
Zleca usunięcie ogłoszenia.
- void usuwanieSukces(Ogloszenie ogloszenie)

- `void usuwaniePorazka(Liczba blad)`
Informacja o wyniku operacji usuwania ogłoszenia. Funkcja ta musi przykazywać informację dotyczącą tego, którego ogłoszenia dotyczy usuwanie. Usuwanie ogłoszenia pełni także rolę *odrzuć* nowego ogłoszenia przez Administratora.
- `void modyfikujOgloszenie(Ogloszenie stare, Ogloszenie nowe)`
Zleca modyfikację ogłoszenia.
- `void modyfikacjaSukces(Ogloszenie ogloszenie)`
- `void modyfikacjaPorazka(Liczba blad)`
Informacja o wyniku operacji usuwania ogłoszenia.
- `void zatwierdzOgloszenie(Ogloszenie ogloszenie)`
Zleca operację zatwierdzenia ogłoszenia.
- `void zatwierdzanieSukces(Ogloszenie ogloszenie)`
- `void zatwierdzaniePorazka(Liczba blad)`
Informacja o wyniku operacji zatwierdzania.
- `void aktualizujOgloszenie(Ogloszeni ogloszenie)`
Zleca operację aktualizacji ogłoszenia.
- `void aktualizacjaSukces(Ogloszenie ogloszenie)`
- `void aktualizacjaPorazka(Liczba blad)`
Informacja o wyniku operacji aktualizacji.
- `void przedluzTerminOgloszenia(Ogloszenie ogloszenie)`
Zleca operację przedłużania terminu ogłoszenia.
- `void przedluzanieSukces(Ogloszenie ogloszenie)`
- `void przedluzaniePorazka(Liczba blad)`
Informacja o wyniku operacji zatwierdzania.

(d) Metody związane z zarządzaniem katalogami branż:

i. Operacje na pojedynczych branżach:

- `void dodanieBranzy(Branza branza, Branza matka)`
Zleca dodanie nowej branży do katalogu branż. Branża jest tworzona jako liść drzewa branż i podczepiana pod branżę matka.
- `void dodanieBranzySukces(Branza branza)`
- `void dodanieBranzyPorazka(Liczba blad)`
Informacja o wyniku operacji dodawania branży.
- `void usuwanieBranzy(Branza branza)`
Zlecenie wykonania operacji usunięcia branży.
- `void usuwanieBranzySukces(Branza branza)`
- `void usuwanieBranzyPorazka(Liczba blad)`
Informacja o wyniku operacji usuwania branży. Operacja może się nie po-

wieść, gdy np. nastąpiła próba usunięcia branży, która była grupą branż (miała podbranże).

- `void modyfikacjaBranzy(Branza branza)`
Zlecenie modyfikacji branży, np. jej nazwy lub opisu.
- `void modyfikacjaBranzySukces(Branza branza)`
- `void modyfikacjaBranzyPorazka(Liczba blad)`
Informacja o wyniku operacji modyfikacji branży.

ii. Operacje na grupach branż:

Analogiczne jak dla branż.

iii. Operacje na katalogu branż:

- `void zlecGenerowanieStatycznegoKatalogu()`
Zleca wygenerowanie statycznego katalogu branż i umieszczenie go w odpowiednim miejscu na serwerze.
- `void generowanieStatycznegoSukces()`
- `void generowanieStatycznegoPorazka(Liczba blad)`
Informacja o wyniku operacji generowania statycznego katalogu branż.

iv. Operacje dotyczące zarządzania **Katalogiem najpopularniejszych branż**:

- `void pobierzKatalogNajpop()`
Zlecenie pobierania katalogu.
- `void pobierzKatalogNajpopSukces(`
 `(Collection of NajpopularniejszaBranza) katalog)`
- `void pobierzKatalogNajpopPorazka(Liczba blad)`
Wyniki pobierania katalogu.
- `void zastapKatalogNaj(`
 `(Collection of NajpopularniejszaBranza) katalog)`
Zlecenie operacji wygenerowania statycznego katalogu najpopularniejszych branż i umieszczenia go na serwerze.
- `void zastepowanieNajSukces()`
- `void zastepowanieNajPorazka(Liczba blad)`
Informacja o wyniku zastępowania katalogu.

(e) Metody dotyczące zarządzania szablonami:

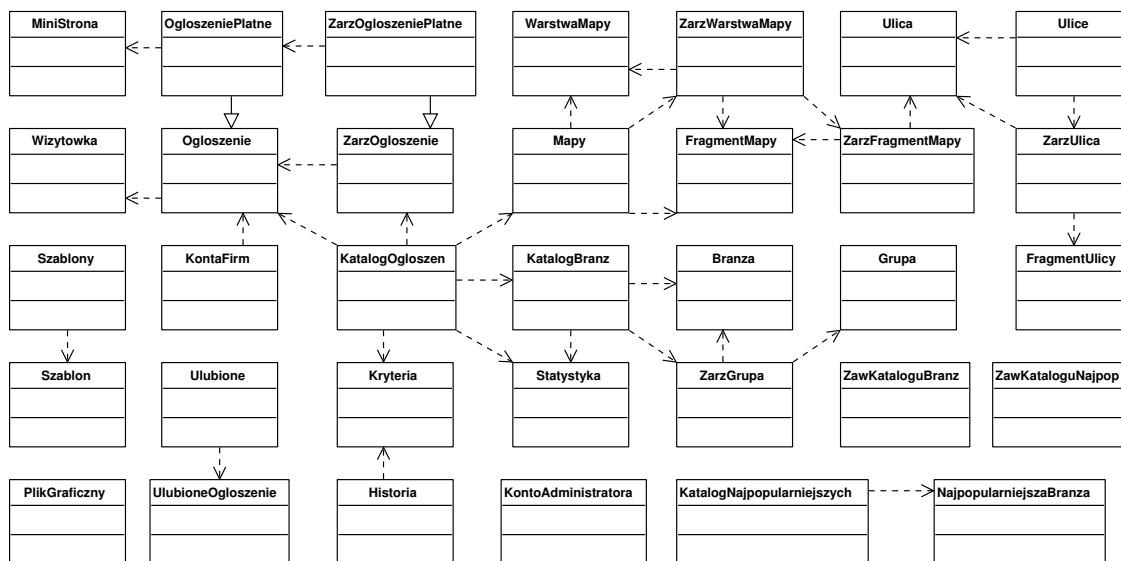
- `void pobierzSzablony()`
- `void pobranieSzablonuSukces(`
 `(Collection of Szablony) zestaw)`
- `void pobranieSzablonuPorazka(Liczba blad)`
- `void zmienSzablon(Szablon szablon)`
Modyfikuje szablon na serwerze, podmieniając jego starą wersję nową.
- `void zmianaSzablonuSukces()`
- `void zmianaSzablonuPorazka(Liczba blad)`

(f) Metody dotyczące administrowania mapą.

- void pobierzWarstwy()
- void odbierzWarstwy(
 (Collection of WarstwaMapy) warstwy)
- void pobierzWarstwyPorazka(Liczba blad)
- void dodajWarstwe(WarstwaMapy warstwa)
- void dodanieWarstwySukces(WarstwaMapy warstwa)
- void dodanieWarstwyPorazka(Liczba blad)
- void dodajFragment(FragmentMapy bazowy,
 FragmentMapy nowy, Liczba gdzie, PlikGraficzny obraz)
 Dodaje nowy fragment mapy w kierunku gdzie od bazowego.
- void dodanieFragmentuSukces(FragmentMapy fragment)
- void dodanieFragmentuPorazka(Liczba blad)
- void podmienFragment(
 FragmentMapy stary, FragmentMapy nowy,
 PlikGraficzny obraz)
- void podmianaFragmentuSukces(FragmentMapy nowy)
- void podmianaFragmentuPorazka(Liczba blad)
- void pobierzInfoOUlicyPunkt(Punkt p)
 Zleca wyszukanie informacji dotyczących ulicy, która znajduje się najbliżej
 wskazanego punktu na mapie.
- void dostarczInfoOUlicyPunkt(Ulica ulica)
- void pobierzInfoOUlicyPunktPorazka(Liczba blad)
- void pobierzUlice(String nazwa)
 Zleca wyszukanie informacji o ulicy o zadanej nazwie.
- void odbierzUlice(Ulica ulica,
 (Collection of FragmentUlicy) fragmenty)
- void pobierzUlicePorazka(Liczba blad)
- void dodajUlice(Ulica ulica)
- void dodanieUlicySukces()
- void dodanieUlicyPorazka(Liczba blad)
- void usunUlice(Ulica ulica)
- void usuwanieUlicySukces()
- void usuwanieUlicyPorazka(Liczba blad)
- void dodajFragmentUlicy(FragmentUlicy fragment)
- void dodanieFragmentuUlicySukces(
 FragmentUlicy fragment)
- void dodanieFragmentuUlicyPorazka(
 FragmentUlicy fragment, Liczba blad)

- void znajdzUliceWOkolicy(Punkt gl, Punkt dp)
Zleca wyszukanie ulic w prostokącie o lewym górnym rogu gl i prawym dolnym dp.
- void dostarczUliceZOkolicy(
(Collection of Ulica) zestaw)
- void dostarczUliceZOkolicyPorazka(Liczba blad)
- void pobierzFragmentXY(WarstwaMapy warstwa, Punkt p)
Zleca odnalezienie fragmentu mapy, który na zadanej warstwie zawiera zadany punkt.
- void odbierzFragmentXY(FragmentMapy fragment)
- void pobierzFragmentXYPorazka(Liczba blad)

Diagram klas dla warstwy logiki biznesowej przedstawiony został na rysunku 8.5 na stronie 52.



(a) Klasa Historia

- `(Collection of Kryteria) pokaz()`
Zwraca zapamiętane kryteria wyszukiwania.
- (b) Klasa `UlubioneOgloszenie`
Obiekty tej klasy będą przechowywać dane z ogłoszeń, które *Użytkownik* chce zapamiętać.
- (c) Klasa `Ulubione`
 - `void dodaj(UlubioneOgloszenie ogloszenie)`
Dodaje nowe ogłoszenie do „Ulubionych”.
 - `(Collection of UlubioneOgloszenie) pokaz()`
Zwraca zapamiętane Ulubione ogłoszenia.
 - `void zmien(UlubioneOgloszenie stare, UlubioneOgloszenie nowe)`
Zamienia w „Ulubionych” stare ogłoszenie na nowe.
 - `void usun(UlubioneOgloszenie ogloszenie)`
Usuwa ogłoszenie z „Ulubionych”.
 - `(Collection of String) pobierzKategorie()`
Generuje (na podstawie pamiętanego zestawu ulubionych ogłoszeń) zestaw używanych przez *Użytkownika* kategorii.

2. Pakiet `TransportowaneObiekty`

- (a) Klasa `Kryteria`
Zawiera szereg atrybutów, które określają kryteria wyszukiwania (NIP, nazwa firmy, data założenia, nazwa ulicy, promień wyszukiwania na mapie, punkt bazowy wyszukiwania na mapie, branża, informację ile ogłoszeń należy zawrzeć w wynikach oraz ile pierwszych pominąć) oraz akcesory do tych atrybutów.
- (b) Klasa `Ogloszenie`
Zawiera wszystkie informacje o ogłoszeniu (dane firmy, lokalizację na mapie, dane potrzebne do wygenerowania wizytówki). Posiada metodę:
`Wizytowka wygenerujWizytowke()`.
Klasa ta posiada jedną podklasę: `OgloszeniePlatne`, która dodatkowo przechowuje obiekty klasy `MiniStrona` oraz informacje o ogłoszeniu charakterystyczne dla ogłoszeń firm wymagających.
- (c) Klasa `MiniStrona`
- (d) Klasa `Wizytowka`
- (e) Klasa `Szablon`
- (f) Klasa `WarstwaMapy`
- (g) Klasa `FragmentMapy`
- (h) Klasa `Ulica`

- (i) Klasa `FragmentUlicy`
- (j) Klasa `Branza`
- (k) Klasa `Grupa`
- (l) Klasa `NajpopularniejszaBranza`
- (m) Klasa `PlikGraficzny`
- (n) Klasa `ZawKataloguBranz`
Zawiera zawartość całego katalogu branż.
- (o) Klasa `ZawKataloguNajpop`
Zawiera zawartość **Katalogu najpopularniejszych branż**.
- (p) Klasa `Statystyka`
Zawiera obliczone przez system statystyki najpopularniejszych branż i firm.
- (q) Klasa `KrytStat`
Zawiera kryteria do obliczenia statystyk najpopularniejszych branż i firm.

3. Pakiet `Ogloszenia`

- (a) Klasa `KatalogOgloszen`
 - `void noweOgloszenia()`
Odczytuje zestaw ogłoszeń, które zostały ostatnio dodane do systemu i wymagają akceptacji administratora.
 - `void nieaktualneOgloszenia()`
Odczytuje zestaw ogłoszeń, którym upłynął termin aktualizacji, a ich aktualność nie została potwierdzona.
 - `void nieoplaconeOgloszenia()`
Odczytuje zestaw ogłoszeń firm wymagających, którym upłynął (lub jest bliski) termin płatności.
 - `void wyborOgloszen(Kryteria kryteria)`
Odczytuje zestaw ogłoszeń, spełniających zadany zestaw kryteriów. Jeśli z kryteriów wynika, że ogłoszenia będą prezentowane na mapie, metoda ta stara się wybrać ogłoszenia tak, by można je było rozsądnie zaprezentować (stara się grupować firmy według lokalizacji na mapie). Dodatkowo, wywołana zostaje metoda `wybierzNajlepszyFragment` obiektu klasy `Mapy`, aby uzyskać informację, który fragment mapy będzie najlepszy do prezentacji wyników. Zestaw ogłoszeń oraz (ewentualnie) informacje o fragmencie mapy przekazuje do `ObslugiUzytkownika`.
 - `void usunOgloszenie(Ogloszenie ogloszenie)`
Usuwa z bazy danych zadane ogłoszenie wraz ze wszystkimi danymi z nim związanymi.
 - `ZarzOglosz zarzOglosz(Ogloszenie ogloszenie)`
Zwraca obiekt klasy `ZarzOglosz`, umożliwiający wykonywanie różnych operacji administracyjnych na danym ogłoszeniu.

- `Ogloszenie ogloszenieFirmy(String nip)`
Zwraca ogłoszenie danej firmy.
- `void obliczStatystyki(KrytStat kryteria)`
Oblicza statystyki najpopularniejszych branży i firm.
- `Statystyka najpopularniejszeFirmy(KrytStat kryteria)`
Znajduje najpopularniejsze firmy.

(b) Klasa `ZarzOglosz`

- `void potwierdzAktualnosc()`
Potwierdza aktualność danych w ogłoszeniu, przesuając datę ostatniej aktualizacji.
- `void zatwierdz()`
Zatwierdza ogłoszenie, umożliwiając jego publikowanie w wynikach wyszukiwania.
- `Ogloszenie dajTresc()`
Zwraca obiekt, zawierający całą treść ogłoszenia. Obiekt `Ogloszenie` może być transportowany do innych warstw systemu.
- `void zmienTresc(Ogloszenie nowe)`
Zapamiętuje w systemie nową treść ogłoszenia.

(c) Klasa `ZarzOgloszeniePlatne` (dziedziczy po `ZarzOgloszenie`)

- `void zmienMiniStrone(MiniStrona nowa)`
Zmienia mini stronę przypisaną do danego ogłoszenia płatnego.
- `void zmienNaBezplatne()`
Dokonyuje przekształcenia na ogłoszenie bezpłatne, usuwając dane właściwe ogłoszeniom płatnym.
- `void przedluzTermin(Data data)`
Przedłuża termin, do którego ogłoszenie jest opłacone.

4. Pakiet `Konto`

(a) Klasa `KontaFirm`

- `void utworzKonto(Ogloszenie ogloszenie, String haslo)`
Tworzy nowe konto dla Firmy na podstawie utworzonego ogłoszenia; zamieszcza ogłoszenie w bazie danych.
- `void autoryzacja(String nip, String haslo)`
Sprawdza poprawność hasła (zaszyfrowanego w polu `haslo`).
- `void przypomnijHaslo(String nip)`
Wykonuje odpowiednią procedurę przypominania zapomnianego przez firmę (i-identyfikowaną po NIP) hasła.
- `void zmienHaslo(String NIP, String stare, String nowe)`

Zmienia hasło dla konta firmy. Zarówno stare, jak i nowe są zaszyfrowanymi hasłami.

(b) Klasa KontoAdministradora

- void autoryzacja(String haslo)
Sprawdza poprawność hasła (zaszyfrowanego w polu haslo).
- void zmienHaslo(String stare, String nowe)
Zmienia hasło *Administradora*. Zarówno stare, jak i nowe są zaszyfrowanymi hasłami.

5. Pakiet Mapa

(a) Klasa Mapy

- void nowaWarstwa(WarstwaMapy warstwa)
Tworzy nową warstwę mapy.
- (Collection of WarstwaMapy) warstwy()
Zwraca zbiór wszystkich zdefiniowanych w systemie warstw mapy.
- ZarzWarstwaMapy zarzWarstwaMapy(WarstwaMapy warstwa)
Zwraca obiekt, umożliwiający zarządzanie warstwą mapy.
- FragmentMapy wybierzNajlepszyFragment(
(Collection of Ogloszenie) ogloszenia)
Znajduje fragment mapy najodpowiedniejszy do prezentacji zadanego zestawu ogłoszeń.

(b) Klasa ZarzWarstwaMapy

- void znajdzFragment(Punkt p)
Odnajduje fragment mapy, na którym znajduje się zadany punkt.
- void dodajFragment(FragmentMapy bazowy,
FragmentMapy nowy, Liczba gdzie, PlikGraficzny obraz)
Dodaje do warstwy nowy fragment mapy. Parametr gdzie określa, w którym kierunku (N, S, W, E), względem fragmentu bazowy ma zostać doklejony tworzony fragment.
- void podmienFragment(FragmentMapy stary,
FragmentMapy nowy, PlikGraficzny obraz)
Zastępuje stary fragment mapy nowym.
- ZarzFragmentMapy zarzFragmentMapy(
FragmentMapy fragment)
Zwraca obiekt, umożliwiający zarządzanie fragmentem mapy.

(c) Klasa ZarzFragmentMapy

- (Collection of Ulica) znajdzUlice()
Znajduje i zwraca ulice, które przebiegają przez dany fragment mapy.

(d) Klasa Ulice

- `ZarzUlica zarzUlica(Ulica ulica)`
Zwraca obiekt umożliwiający zarządzanie ulicą.
- `void znajdzUlice(String nazwa)`
Odnajduje ulicę o zadanej nazwie.
- `void usunUlice(Ulica ulica)`
Usuwa zadaną ulicę.
- `void dodajUlice(Ulica ulica)`
Dodaje do systemu zadaną ulicę.
- `void znajdzUliceWOkolicy(Punkt gl, Punkt dp)`
Odnajduje i zwraca wszystkie ulice, przebiegające przez obszar prostokąta o zadanych współrzędnych (gl – górny lewy wierzchołek, dp – dolny prawy wierzchołek).

(e) Klasa `ZarzUlica`

- `void wspolrzedneObiektu(Liczba nrDomu)`
Wylicza współrzędne zadanego numeru domu na danej ulicy.
- `void dodajFragment(FragmentUlicy fragment)`
Dodaje do systemu nowy fragment ulicy.
- `void usunFragment(FragmentUlicy fragment)`
Usuwa podany fragment.
- `void znajdzFragment(Liczba nrDomu)`
Odnajduje fragment ulicy, który obejmuje zadany numer domu.
- `void zmienNazwe(String nazwa)`
Zmienia nazwę ulicy na zadaną.

6. Pakiet `Branze`

(a) Klasa `ZarzGrupa`

- `Grupa dajTresc()`
Zwraca treść grupy (obiekt typu `Grupa`).
- `void podepnijBranze(Branza branza)`
W drzewie katalogu branż czyni zadaną branżę liściem podpiętym do danej grupy.
- `void odepnijBranze(Branza branza)`
Usuwa dowiązanie do zadanej branży od danej grupy.
- `void podgrupy()`
Odnajduje grupy będące podgrupami danej.
- `void podbranze()`
Odnajduje branże znajdujące się w danej grupie.

(b) Klasa `KatalogBranz`

- `ZarzGrupa zarzGrupa(Grupa grupa)`
Zwraca obiekt, który umożliwia zarządzanie grupą.
- `void dodajBranze(Branza branza)`
Tworzy nową branżę.
- `void usunBranze(Branza branza)`
Usuwa z katalogu branż zadaną branżę.
- `void zmienBranze(Branza stara, Branza nowa)`
Zmienia branżę.
- `void usunGrupe(Grupa grupa)`
Usuwa z katalogu branż zadaną grupę wraz ze wszystkimi jej podgrupami i podbranżami.
- `void znajdzBranze(String klucz)`
Znajduje branże, które odpowiadają zadanym słowom kluczowym.
- `void grupaGlowna()`
Znajduje zawartość korzenia katalogu branż.
- `void wygenerujStatycznyKatalogBranz()`
Tworzy obraz całego katalogu branż i zapisuje go do pliku.
- `void najpopularniejszeBranze(Statystyka stat, KrytStat kryteria)`
Na podstawie statystyk zebranych dla ogłoszeń oblicza, które branże są najpopularniejsze. Wynikami uzupełnia obiekt `stat`.

(c) Klasa `KatalogNajpopularniejszych`

- `void katalog()`
Znajduje zestaw branż, tworzących **Katalog najpopularniejszych branż**.
- `KatalogNajpopularniejszych nowyKatalog(Collection of NajpopularniejszaBranza) nowyKatalog`
Zastępuje istniejący **Katalog najpopularniejszych branż** zadanym nowym.
- `void wygenerujStatycznyKatalogNajpop()`
Tworzy obraz całego **Katalogu najpopularniejszych branż** i zapisuje go do pliku.

7. Pakiet Szablony

(a) Klasa `Szablony`

- `void dajSzablony()`
Przekazuje zestaw szablonów.
- `void zmienSzablon(Szablon nowy)`
Zapamiętuje nową wersję szablonu, zastępując nim starą.

8.2.4 Warstwa bazy danych

W systemie NIIKT z bazy danych korzysta się za pomocą interfejsu bazy danych API Javy – JDBC (ang. *Java DataBase Connectivity*). Funkcje pakietu Baza zostaną zrealizowane w języku zgodnym ze standardem SQL. Warstwa danych składa się z jednego pakietu, który zawiera definicje wyzwalaczy oraz – ewentualnie – procedur i funkcji składowanych.

1. Wyzwalacze (*triggers*)

W pakiecie Baza zostaną zaimplementowane następujące wyzwalacze

- (a) Wyzwalacz OGLOSZENIA_TELEFONY_TRIGG na tabeli OGLOSZENIA wykonywany przy wstawianiu do niej nowego wiersza lub modyfikacji pola TELEFONY, a także przy przekształcaniu ogłoszenia w ogłoszenie bezpłatne, zapewniający, że każda firma umieszczająca ogłoszenie bezpłatne posiada w bazie maksymalnie jeden numer telefonu.
- (b) Wyzwalacz OGLOSZENIA_MAIL_TRIGG na tabeli OGLOSZENIA wykonywany w momencie wstawiania nowego wiersza lub modyfikacji jednego z pól E_MAIL lub E_KONTAKT, sprawdzający czy wymienione pola są poprawnie sformatowanym adresem poczty elektronicznej.
- (c) Wyzwalacz OGLOSZENIA_HASLO_TRIGG na tabeli OGLOSZENIA wykonywany w momencie wstawiania nowego wiersza lub modyfikacji pola HASLO, sprawdzający dodatkowe wymogi dotyczące skomplikowości hasła, np. że musi ono zawierać przynajmniej jeden znak nie-alfanumeryczny lub przynajmniej dwie cyfry, lub być istotnie różne od starego hasła.
- (d) Wyzwalacz BRANZE_MATKA_TRIGG na tabeli BRANZE wykonywany przed usunięciem wiersza z tej tabeli, nie pozwalający usunąć branży, która posiada podbranże.

2. Sekwencje (*sequences*)

W pakiecie Baza zdefiniowane zostaną następujące sekwencje (ang. *sequences*) służące do generowania unikatowych identyfikatorów. Będą to:

- (a) PLATNE_SEQ – służąca do generowania identyfikatorów wierszy w tabeli OGLOSZENIA_PLATNE w polu NR,
- (b) TELEFONY_SEQ – służąca do generowania unikatowych identyfikatorów dla tabeli TELEFONY w polu ID,
- (c) BRANZE_SEQ – dla tabeli BRANZE w polu ID,
- (d) WMAPY_SEQ – identyfikatory warstwy mapy w tabeli WARSTWY_MAPY w polu ID,
- (e) FRMAPY_SEQ – identyfikatory fragmentów mapy w tabeli FRAGMENTY_MAPY w polu ID,
- (f) NAZWY_ULIC_SEQ – identyfikatory ulic w tabeli NAZWY_ULIC w polu ID,

(g) `FRAGMENTY_ULIC_SEQ` – identyfikatory ulic w tabeli `FRAGMENTY_ULIC` w polu `ID`.

3. Procedury i funkcje składowane.

Logika systemu powinna być skupiona w warstwie logiki biznesowej, dlatego najprawdopodobniej nie będziemy tworzyć żadnych procedur składowanych na bazie.

Dla przyspieszenia wykonywania operacji na bazie danych zastosujemy metodę przygotowanych zapytań (ang. *prepared statements*).

1. Przygotowane zapytanie dla wyszukiwania ogłoszenia w bazie.
2. Przygotowana instrukcja wstawiania nowego ogłoszenia do bazy.

8.3 Realizacja przypadków użycia

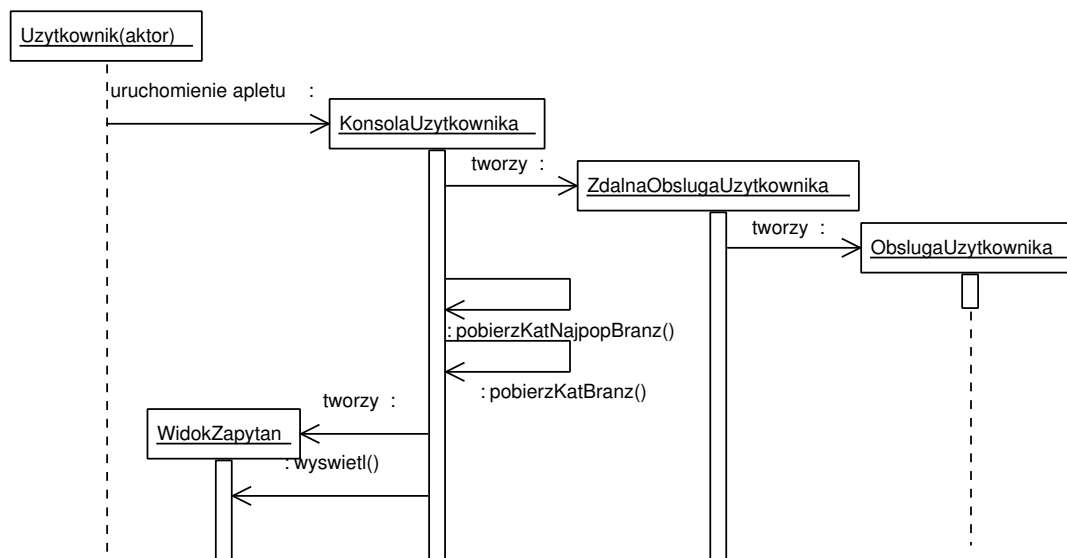
8.3.1 Rozpoczęcie pracy z Modułem Użytkownika

Przypadek użycia został przedstawiony na rysunku 8.6 na stronie 61.

Omówienie diagramu

Diagram przedstawia czynności, jakie musi wykonać system nim *Użytkownik* będzie mógł wykonać pierwsze polecenia. Muszą zostać stworzone odpowiednie obiekty, które będą obsługiwać zlecenia *Użytkownika*, a także potrzebny jest katalog branż oraz **Katalog najpopularniejszych branż**, które zostaną ściągnięte z serwera w postaci odpowiedniego pliku.

Rysunek 8.6: Rozpoczęcie pracy z Modułem Użytkownika



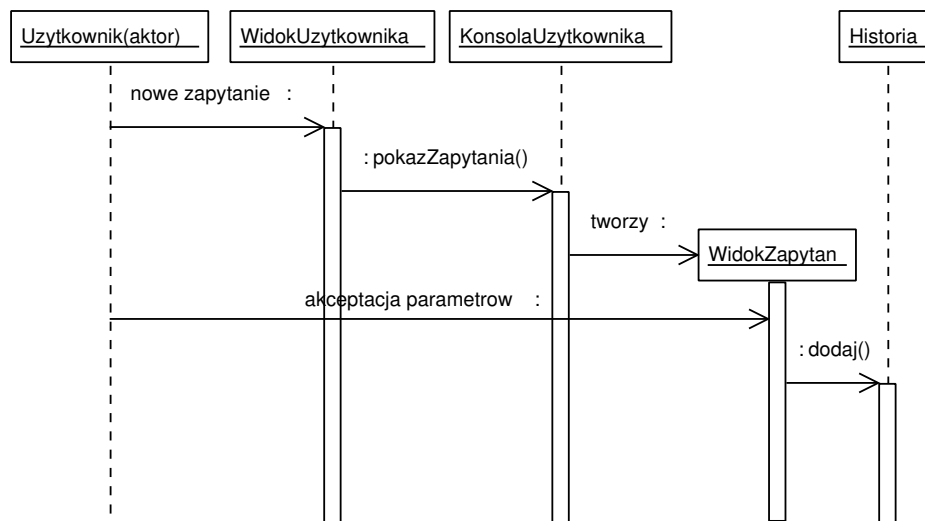
8.3.2 Ustalanie parametrów wyszukiwania

Przypadek użycia został przedstawiony na rysunku 8.7 na stronie 62.

Omówienie diagramu

Użytkownik ustala parametry wyszukiwania, które od razu zostają zapamiętane w *Historii*, gdyby *Użytkownik* chciał skorzystać z **Historii zapytań**. Po tym przypadku nastąpi *Wyświetlanie wyników wyszukiwania*, co zostało przedstawione na kolejnym diagramie.

Rysunek 8.7: Ustalanie parametrów wyszukiwania



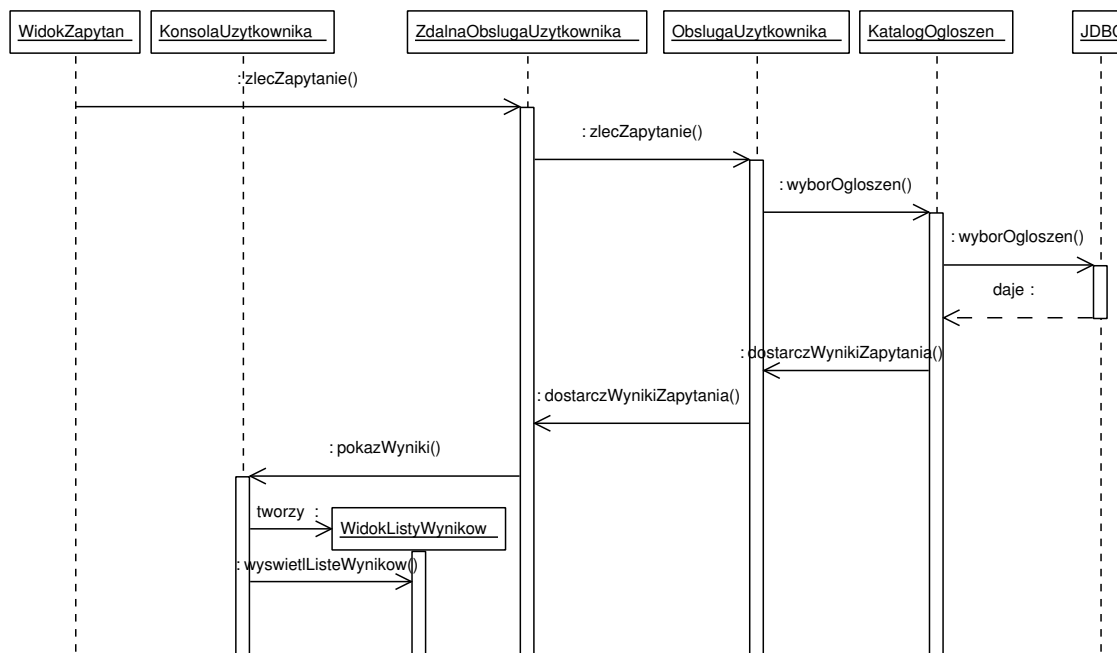
8.3.3 Wyświetlanie wyników wyszukiwania

Przypadek użycia został przedstawiony na rysunku 8.8 na stronie 63.

Omówienie

Zapytanie *Użytkownika* jest przesyłane do warstwy logiki, gdzie zostaje odpowiednio przetworzone. Jeśli *Użytkownik* zażądał wizualizacji na mapce, to warstwa logiki prześle do warstwy interfejsu odpowiedni fragment mapy do wyświetlenia, oprócz ogłoszeń interesujących *Użytkownika* firm. Na diagramie przedstawiona jest sytuacja, gdy *Użytkownik* chce oglądać wyniki wyłącznie w postaci listy.

Rysunek 8.8: Wyświetlanie wyników wyszukiwania



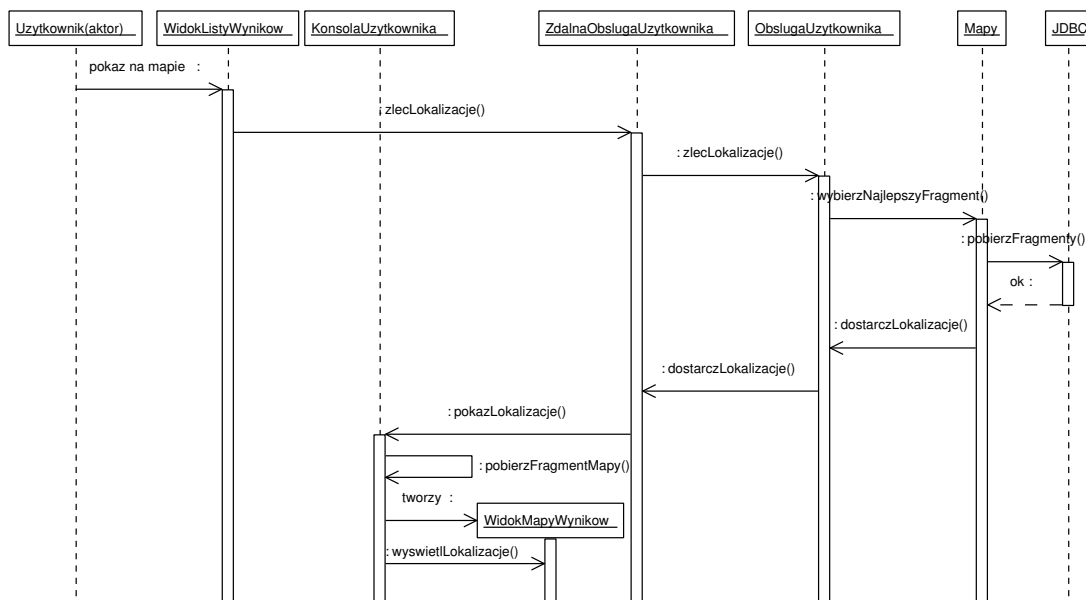
8.3.4 Lokalizacja firmy

Przypadek użycia został przedstawiony na rysunku 8.9 na stronie 64.

Omówienie

Gdy *Użytkownik* zażąda wizualizacji na mapce położenia konkretnej firmy, to system, w warstwie logiki, sprawdzi jaki fragment mapy jest potrzebny i prześle tę informację do warstwy interfejsu. Teraz zarządzająca wyświetleniem *KonsolaUzytkownika* będzie mogła ściągnąć z serwera odpowiedni plik z mapą.

Rysunek 8.9: Lokalizacja firmy



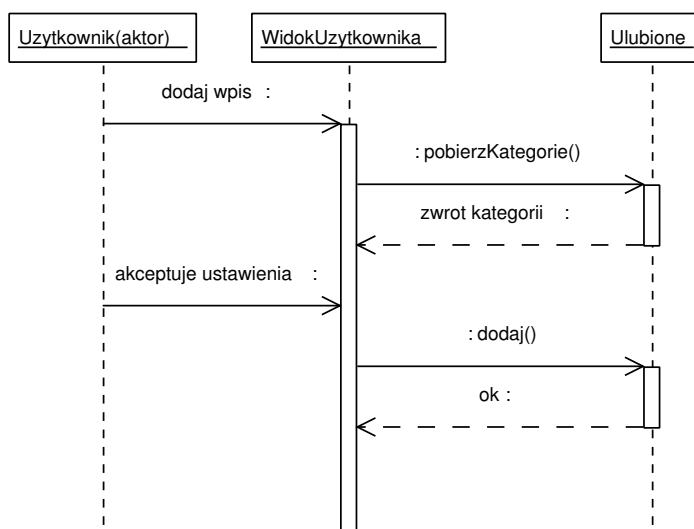
8.3.5 Dodanie wpisu do książki adresowej konta lokalnego

Przypadek użycia został przedstawiony na rysunku 8.10 na stronie 65.

Omówienie

Aby dodać wpis do „Ulubionych” *Użytkownik* określa kategorię wpisu (w celu pokazania już dostępnych kategorii trzeba sięgnąć do *Ulubionych*). Następnie wpis zostanie zapamiętany.

Rysunek 8.10: Dodanie wpisu do książki adresowej konta lokalnego



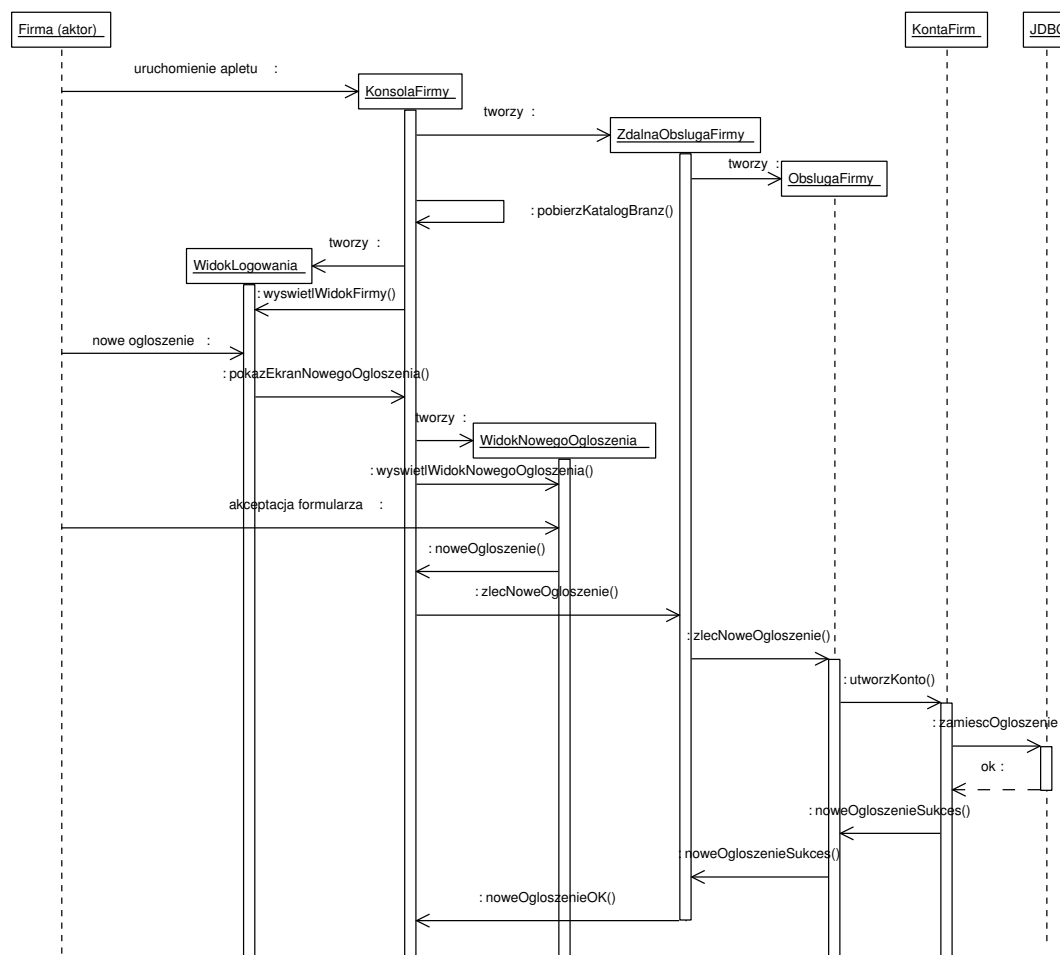
8.3.6 Zamieszczanie ogłoszenia

Przypadek użycia został przedstawiony na rysunku 8.11 na stronie 66.

Omówienie

Na diagramie przedstawiono sytuację, gdy nowa firma chce zamieścić ogłoszenie w Księżce NIIKT. Na początku system tworzy potrzebne do obsługi struktury. Następnie wyświetla formularz, który *Firma* wypełnia i akceptuje ustawienia. Teraz ogłoszenie zostaje przesłane do warstwy logiki, która umieści je w bazie jako *nowe ogłoszenie*, które nie będzie wyświetlane aż do momentu zatwierdzenia go przez *Administradora*.

Rysunek 8.11: Zamieszczanie ogłoszenia



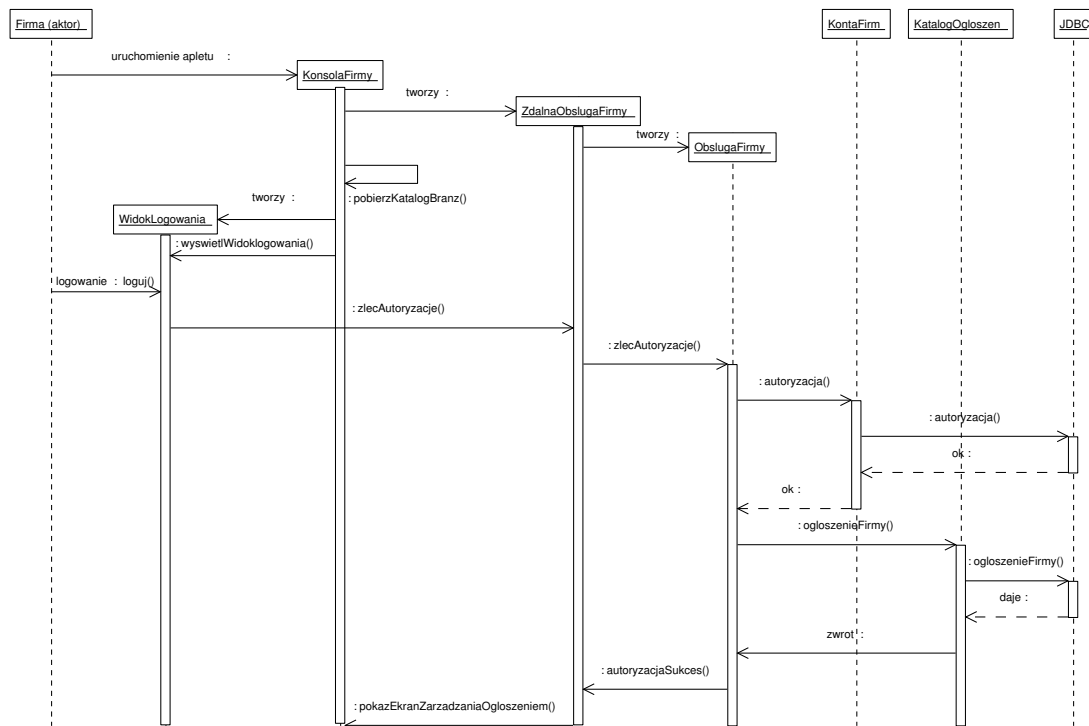
8.3.7 Logowanie firmy

Przypadek użycia został przedstawiony na rysunku 8.12 na stronie 67.

Omówienie

Na diagramie zostały przedstawione czynności jakie wykona system w przypadku, gdy logowanie się powiedzie i *Firma* uzyska dostęp do swojego konta.

Rysunek 8.12: Logowanie firmy



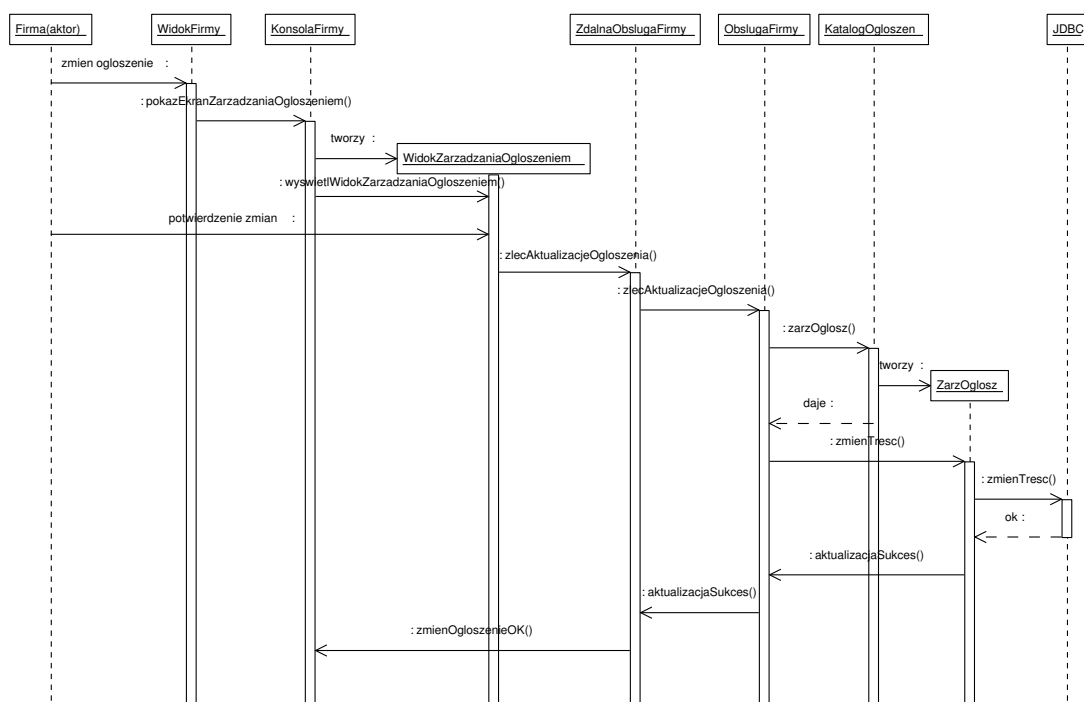
8.3.8 Aktualizacja/zmiana ogłoszenia

Przypadek użycia został przedstawiony na rysunku 8.13 na stronie 68.

Omówienie

Na diagramie zostały przedstawione czynności jakie wykona system w przypadku, gdy zmiana ogłoszenia firmy zakończy się sukcesem. Ponieważ w trakcie logowania zostaje ściągnięte ogłoszenie danej firmy, więc gdy chce ona dokonać zmian, `KonsolaFirmy` nie musi już prosić warstwy logiki o to ogłoszenie.

Rysunek 8.13: Aktualizacja/zmiana ogłoszenia



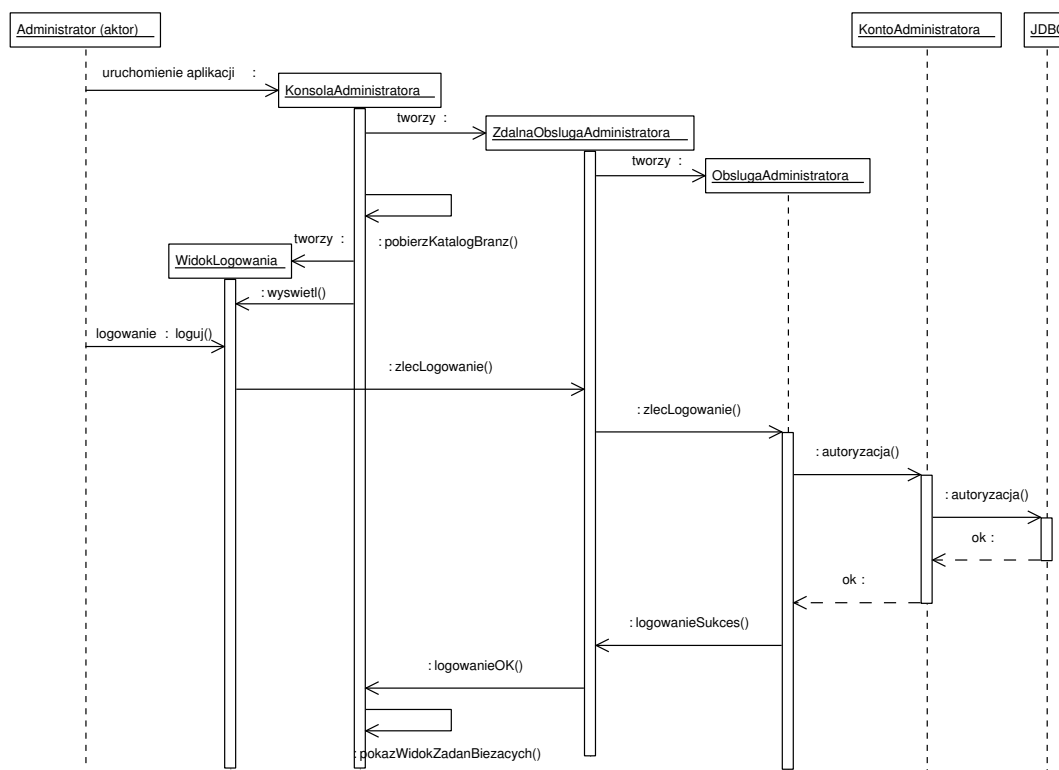
8.3.9 Logowanie administratora

Przypadek użycia został przedstawiony na rysunku 8.14 na stronie 69.

Omówienie

Diagram przedstawia sytuację, gdy *Administrator* zalogował się z sukcesem. Tuż po zalogowaniu zostaje wyświetlony panel zadań bieżących, którymi powinien zająć się *Administrator*.

Rysunek 8.14: Logowanie administratora



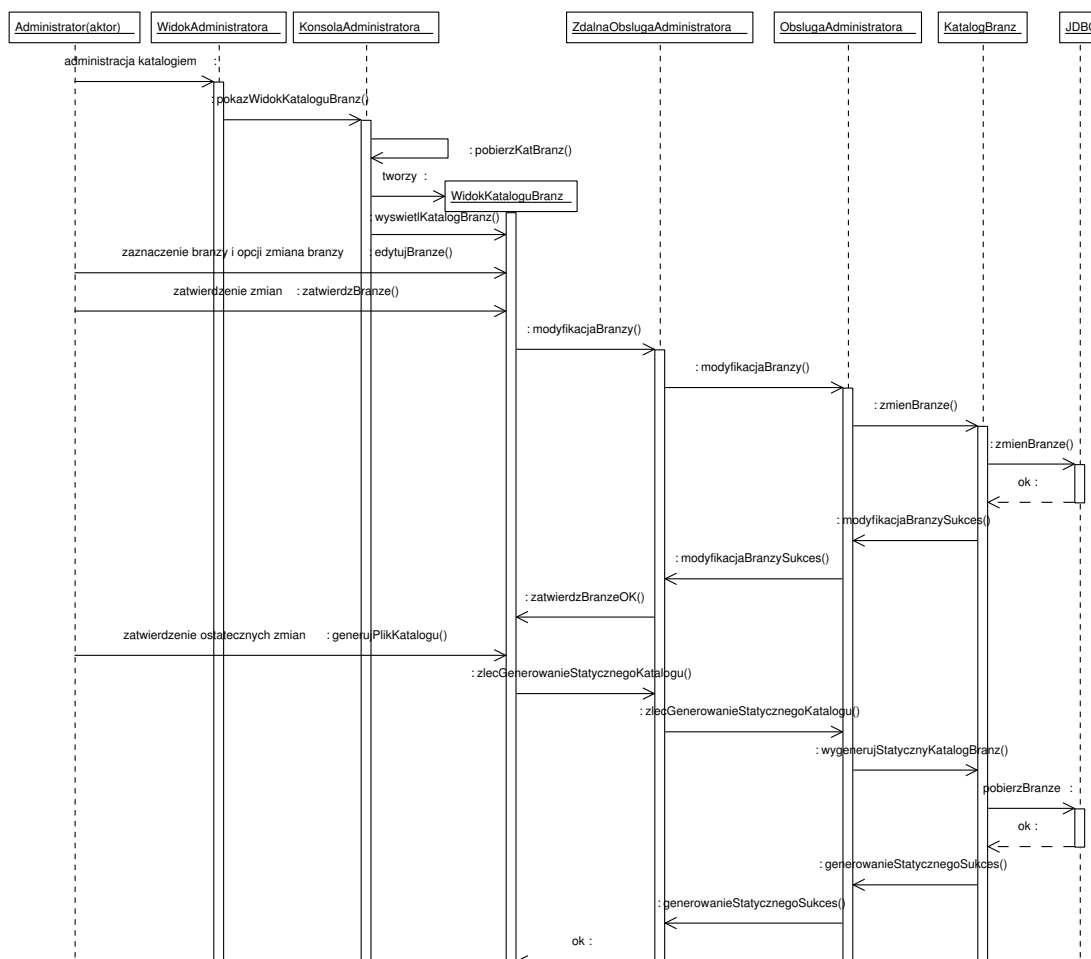
8.3.10 Dodanie/zmiana pozycji w katalogu branż

Przypadek użycia został przedstawiony na rysunku 8.15 na stronie 70.

Omówienie

Na diagramie została przedstawiona sytuacja, gdy *Administrator* zmienia konkretną branżę. W trakcie edycji wielokrotnie przechodzi po drzewie katalogu, korzystając przy tym z przypadku użycia *Przeglądanie katalogu branż* (co nie zostało przedstawione na diagramie). W trakcie dokonywania zmian są one przesyłane do warstwy logiki i zapamiętywane w bazie, a jednocześnie lokalna kopia katalogu jest aktualizowana. Gdy *Administrator* ostatecznie decyduje się na akceptację zmian, to generowany jest statyczny plik, z którego korzystają inni użytkownicy.

Rysunek 8.15: Dodanie/zmiana pozycji w katalogu branż



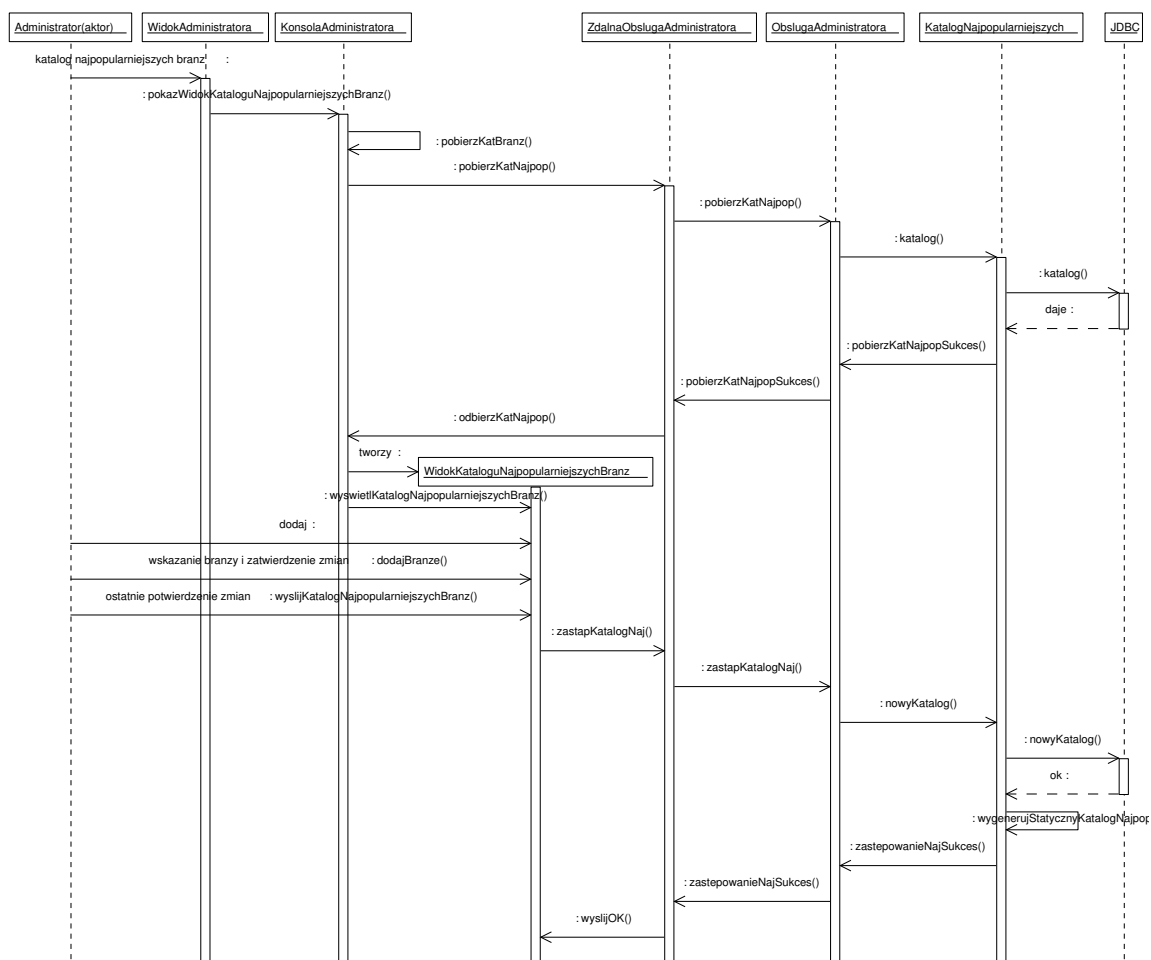
8.3.11 Dodanie/zmiana branży w najpopularniejszych branżach

Przypadek użycia został przedstawiony na rysunku 8.16 na stronie 71.

Omówienie

Na diagramie przedstawiona jest sytuacja, gdy *Administrator* dodaje branżę do **Katalogu najpopularniejszych branż**. Nim rozpocznie pracę z katalogiem ściągany jest plik z katalogiem branż oraz lista najpopularniejszych branż poprzez warstwę sieci. *Administrator* pracuje na lokalnej wersji katalogu najpopularniejszych branż, dopóki ostatecznie nie potwierdzi zmian. Dopiero wtedy są one przesyłane do warstwy logiki i jest generowany nowy statyczny katalog najpopularniejszych branż dla innych użytkowników.

Rysunek 8.16: Dodanie/zmiana branży w najpopularniejszych branżach



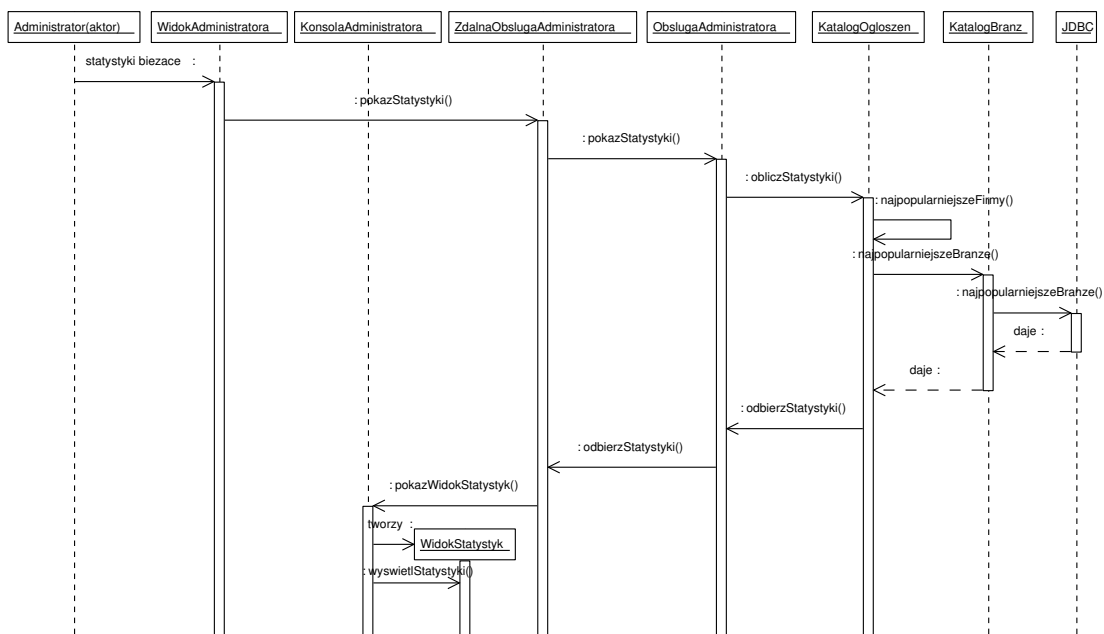
8.3.12 Wyświetlanie najpopularniejszych branż

Przypadek użycia został przedstawiony na rysunku 8.17 na stronie 72.

Omówienie

Na diagramie przedstawiono sytuację, gdy *Administrator* zażądał przedstawienia statystyk najpopularniejszych firm i branż.

Rysunek 8.17: Wyświetlanie najpopularniejszych branż



8.3.13 Zatwierdzenie nowego ogłoszenia

Przypadek użycia został przedstawiony na rysunku 8.18 na stronie 74.

Omówienie

Diagram przedstawia sytuację, gdy *Administrator* pozytywnie rozpatrzył nowe ogłoszenie. Gdy *Administrator* rozpoczyna pracę lub wchodzi do sekcji **Zadań bieżących** zostają ściągnięte wszystkie zadania do zrobienia, a więc nowe ogłoszenia do rozpatrzenia, a także ogłoszenia nieaktualne oraz nieopłacone. Pobieranie tych ogłoszeń odbywa się w sposób asynchroniczny (na diagramie zostały poszczególne czynności przedstawione sekwencyjnie, aby był on czytelniejszy).

8.3.14 Wprowadzanie mapy

Przypadek użycia został przedstawiony na rysunku 8.19 na stronie 75.

Omówienie

Diagram przedstawia przypadek, gdy *Administrator* pomyślnie dokleił fragment mapy na południe od bazowego. Gdy *Administrator* wybiera opcję pracy z mapą, to ściągane są informacje o wszystkich warstwach mapy oraz jeden fragment mapy – zawierający punkt (0,0). Następnie *Administrator* nawiguje po mapie, używając przypadku *Wyświetlanie mapy* (co nie zostało przedstawione na diagramie) i dochodząc dzięki temu do miejsca, w którym chce dokleić nowy fragment mapy.

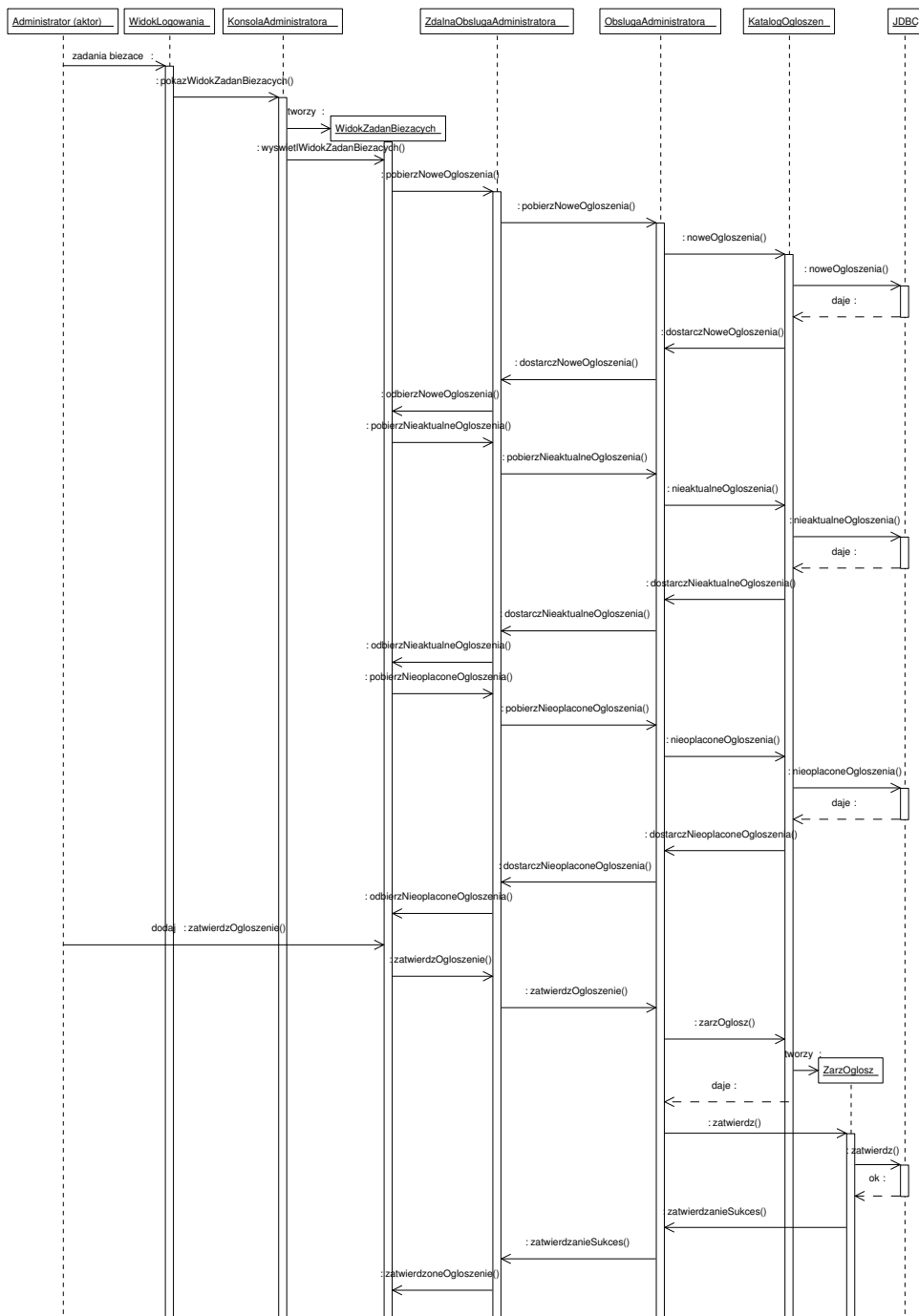
8.3.15 Dodanie/zmiana ulicy na mapie

Przypadek użycia został przedstawiony na rysunku 8.20 na stronie 76.

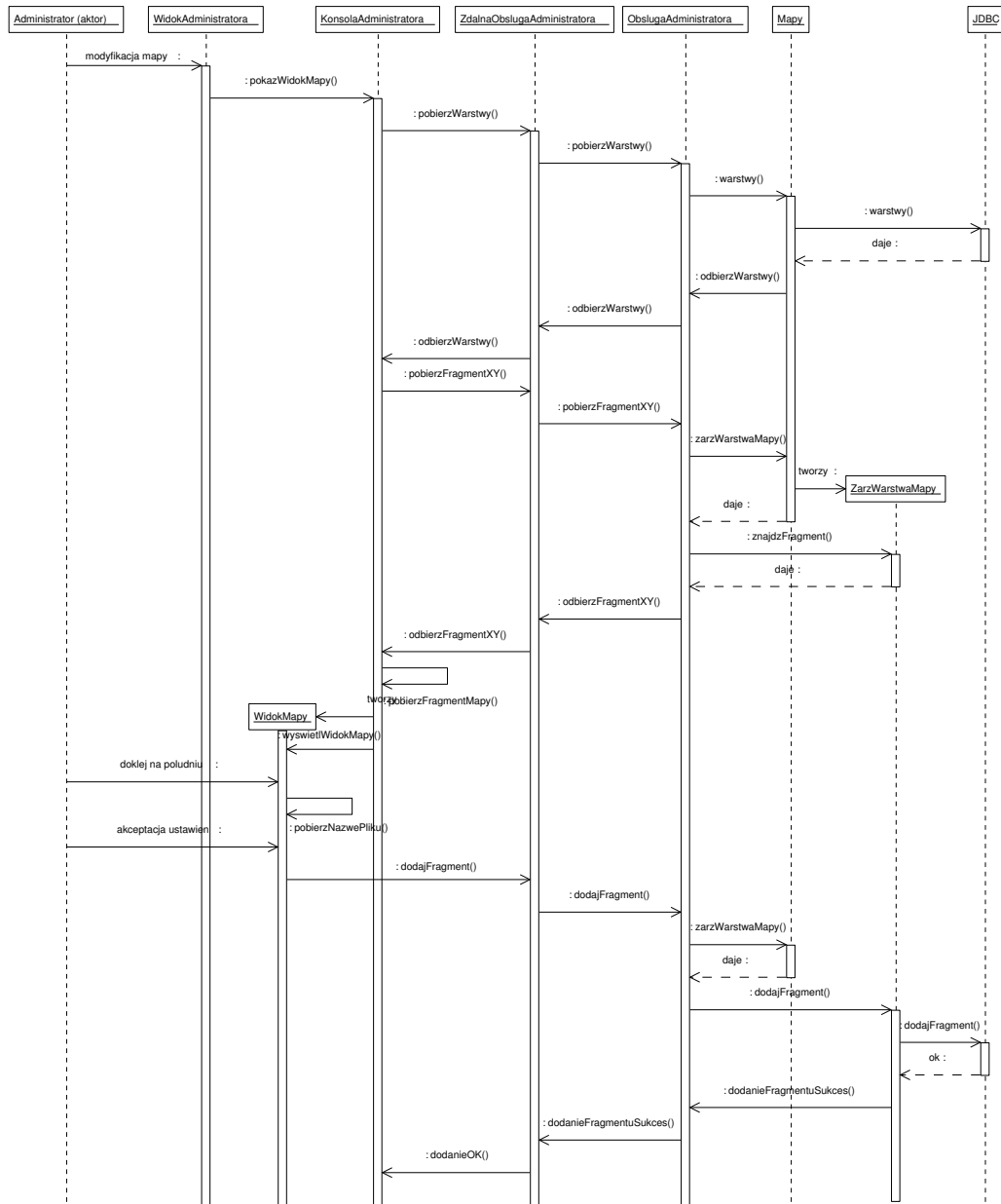
Omówienie

Diagram przedstawia sytuację, gdy *Administrator* dodał nowy fragment ulicy na jej końcu. Nim można było faktycznie rozpocząć pracę system ściągnął odpowiednie dane. Najpierw z logiki pobrany został obiekt *Ulica*. Następnie, znając współrzędne początku ulicy, *KonsolaAdministratora* zażądała odpowiedniego fragmentu mapy od warstwy logiki.

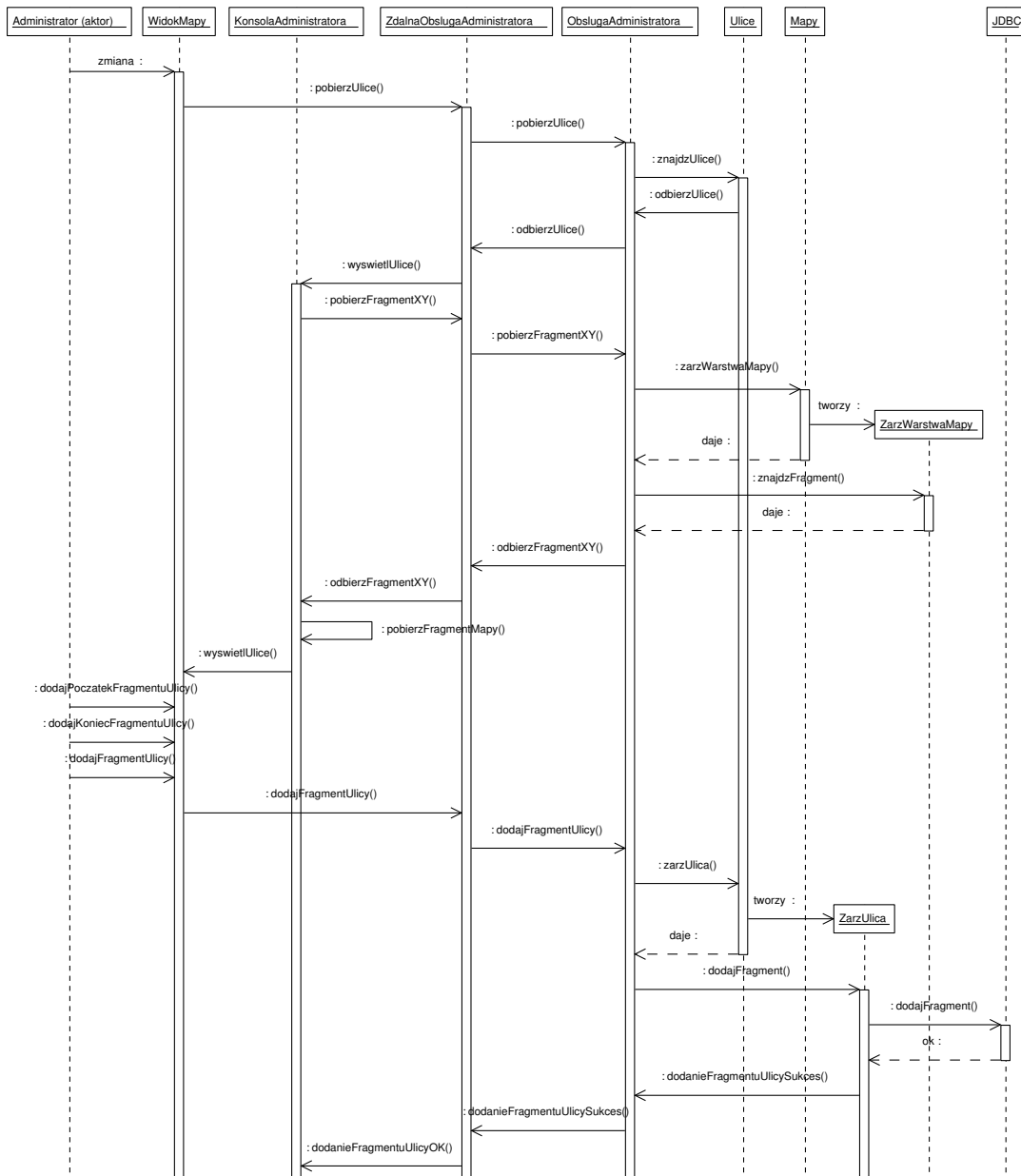
Rysunek 8.18: Zatwierdzenie nowego ogłoszenia



Rysunek 8.19: Wprowadzanie mapy



Rysunek 8.20: Dodanie/zmiana ulicy na mapie



Rozdział 9

Przechowywane dane

9.1 Omówienie

W systemie NIIKT dane przechowujemy na dwa sposoby:

1. Jako pliki na serwerze, co zostało omówione w punkcie [9.2](#).
2. W relacyjnej bazie danych, np. Oracle© lub PostgreSQL.
Oznacza to, iż dane zostaną zorganizowane w formie tabel o ustalonym z góry schemacie, w których wiersze identyfikowane są za pomocą wyróżnionej kolumny / kolumn – klucza głównego.

W opisie przechowywanych danych wyszczególniono zatem:

- najważniejsze tabele bazy,
- klucze główne tabel,
- istotne pola wraz z typem i opisem danych, które zawierają,
- klucze obce tabel.

O ile nie zaznaczono inaczej pole musi zostać wypełnione przy tworzeniu nowego wpisu do tabeli.

9.2 Dane nieprzechowywane w bazie

Nie wszystkie dane będą przechowywane w bazie danych. Dotyczy to następujących danych:

rodzaj danych	uzasadnienie
szablony do korespondencji automatycznej	rzadko modyfikowane, niedużych rozmiarów
pliki konfiguracyjne	potrzebne szybko, niedużych rozmiarów

hasło administratora	zakładamy, że w systemie istnieje dokładnie jeden superużytkownik
skompilowany katalog branż	potrzebny szybko i tylko wtedy, gdy się zmienił
skompilowany Katalog najpopularniejszych branż	j.w.
logo firm	grafikę wygodniej i efektywniej jest przechowywać w postaci plików
fragmenty mapy	j.w.

9.3 Ogłoszenie

Informacje wspólne dla ogłoszeń płatnych i bezpłatnych przechowywane są w tabeli OGLOSZENIA, scharakteryzowanej przez poniższe pola:

nazwa pola	opis	typ
NIP	numer NIP	liczba naturalna, wraz z polem OSOBOWOSC_PRAWNA klucz główny
OSOBOWOSC_PRAWNA	czy firma ma osobowość prawną	tak / nie
ULICA	ulica, na której jest firma	odwołanie do tabeli ULICE
NR_DOMU	numer domu na ulicy ULICA	liczba naturalna
NR_MIESZKANIA	numer mieszkania, opcjonalny	liczba naturalna
POCZTA	kod pocztowy	ciąg znaków
PLATNE	określa rodzaj ogłoszenia	puste (ogłoszenie bezpłatne) lub odwołanie do tabeli OGLOSZENIA_PLATNE
NAZWA	nazwa firmy	ciąg znaków
BRANZA	branża firmy	odwołanie do tabeli BRANZE
DATA_ZALOZENIA	data założenia firmy	data
TELEFONY	numery telefonów związane z firmą, opcjonalne	odwołanie do tabeli TELEFONY
FAX	numer faxu, opcjonalny	ciąg znaków
STRONA_INTERNETOWA	adres strony internetowej, opcjonalny	ciąg znaków
E_MAIL	adres poczty elektronicznej, opcjonalny	ciąg znaków zawierający znak @
E_KONTAKT	adres poczty elektronicznej	ciąg znaków zawierający znak @

RANKING	pozycja firmy w wewnętrznym rankingu	liczba rzeczywista
POPULAR-NOSC	liczba wyszukiń zw. z firmą w bieżącym miesiącu	liczba naturalna
POPULAR-NOSC_POPRZ	liczba wyszukiń zw. z firmą w poprzednim miesiącu	liczba naturalna
WSPOLRZEDNA_X	współrzędna x na mapie	liczba naturalna
WSPOLRZEDNA_Y	współrzędna y na mapie	liczba naturalna
DATA_AKTUALIZACJI	data ostatniej aktualizacji	data
HASLO	zaszyfrowane hasło	ciąg znaków
ZAACEPTOWANE	czy ogłoszenie zostało zaakceptowane	tak / nie
UWAGI	pomocnicze pole, opcjonalne	ciąg znaków

9.4 Ogłoszenie płatne

Dodatkowe informacje dotyczące ogłoszeń płatnych będą przechowywane w tabeli OGLOSZENIA_PLATNE, zawierającej następujące kolumny:

nazwa pola	opis	typ
NR	identyfikator wiersza tabeli	liczba naturalna, klucz główny
GODZINY	godziny otwarcia, opcjonalne	ciąg znaków
LOGO_GDZIE	położenie logo firmy na mini stronie	góra / dół / lewo / prawo / środek
USLUGI	opis usług firmy, opcjonalny	ciąg znaków
SLOWO_KLUCZOWE	słowo kluczowe związane z firmą	ciąg znaków
DATA_OPLACENIA	data, do kiedy ogłoszenie został opłacony	ciąg znaków

Każda firma ma możliwość wprowadzenia pewnej liczby słów kluczowych związanych z jej usługami. Liczba takich słów musi być ustalona oraz nieduża, aby wyszukiwanie było wystarczająco szybkie. Odpowiednia maksymalna liczba słów kluczowych przypadających na firmę zostanie ustalona po otrzymaniu przez *Gr0No3* testowych danych firm.

9.5 Telefony

W bazie danych przechowywane są następujące numery telefonów:

1. Maksymalnie jeden numer telefonu dla każdego ogłoszenia bezpłatnego.
2. Dowolnie wiele telefonów wraz z ich opisami dla ogłoszenia płatnego. W przypadku konwersji ogłoszenia płatnego w bezpłatne, brany jest telefon pierwszy na liście.

Tabela TELEFONY opisująca numery telefonów dla firm, które umieszczają w Książce ogłoszenie:

nazwa pola	opis	typ
ID	identyfikator telefonu dla firmy	liczba naturalna, klucz główny
ID_FIRMY	identyfikator firmy, do której należy telefon	odwołanie do tabeli OGŁOSZENIA, klucz obcy
NR	numer telefonu, może zawierać litery	ciąg znaków
OPIS	opis telefonu, np. sekretariat, rzecznik	ciąg znaków

9.6 Katalog branż

Katalog branż jest zorganizowany w drzewo. Każda branża może być nadbranżą lub/i liściem drzewa. Liść wewnętrzny jest grupą branż. Na przykład:

- GRUPA:Rozrywka → GRUPA:Zabawki → BRANŻA:Zabawki dla psa
- GRUPA:Zwierzęta → GRUPA:Akcesoria → GRUPA:Zabawki → BRANŻA:Zabawki dla psa

Tabela BRANZE jest listą wszystkich istniejących branż i grup branż.

nazwa pola	opis	typ
ID	identyfikator branży	liczba naturalna, klucz główny
NAZWA	nazwa branży	ciąg znaków
OPIS	opis branży	ciąg znaków
SLOWO_KLUCZOWE	słowo kluczowe opisujące branżę	ciąg znaków

Podobnie jak w tabeli OGŁOSZENIA, w tabeli BRANZE umieścimy kilka słów kluczowych odpowiadających danej branży.

Tabela DRZEWO_BRANZ koduje pokrewieństwo między branżami.

nazwa pola	opis	typ
SYN	identyfikator podbranży	odwołanie do pola ID tabeli BRANZE, klucz obcy
MATKA	identyfikator nadbranży	odwołanie do pola ID tabeli BRANZE, klucz obcy

9.7 Najpopularniejsze branże

Tabela NAJPOPULARNIEJSZE_BRANZE wyróżnia popularne branże:

nazwa pola	opis	typ
ID	branża	odwołanie do tabeli BRANZE, klucz główny
OPIS	opis branży	ciąg znaków
RANKING	sposób uporządkowania Najpopularniejszych branż	liczba naturalna

9.8 Mapa

9.8.1 Mapa

Do pamiętania informacji o mapie, używane są dwie tabele.

W tabeli WARSTWY_MAPY pamiętane są informacje na temat poszczególnych warstw mapy:

nazwa pola	opis	typ
ID	identyfikator warstwy	liczba naturalna, klucz główny
OPIS	opis warstwy, prezentowany użytkownikom (np. 1:500000)	ciąg znaków
SKALA	liczba, wykorzystywana przez program, reprezentująca skalę mapy	liczba zmiennoprzecinkowa
ROZMIAR_X	rozmiar (w pikselach) szerokości jednego fragmentu w tej warstwie	liczba naturalna
ROZMIAR_Y	rozmiar (w pikselach) wysokości jednego fragmentu w tej warstwie	liczba naturalna

W tabeli FRAGMENTY_MAPY pamiętane są informacje na temat fragmentów mapy dla poszczególnych warstw:

nazwa pola	opis	typ
ID	identyfikator fragmentu	liczba naturalna, klucz główny
ID_WARSTWY	identyfikator warstwy, z której pochodzi fragment	odwołanie do tabeli WARSTWY_MAPY
WSP_LG_X	współrzędna X lewego górnego rogu	liczba naturalna
WSP_LG_Y	współrzędna Y lewego górnego rogu	liczba naturalna
WSP_PD_X	współrzędna X prawego dolnego rogu	liczba naturalna
WSP_PD_Y	współrzędna Y prawego dolnego rogu	liczba naturalna

Współrzędne, o których mowa w tej tabeli, są współrzędnymi niezależnymi od skali warstwy mapy danego fragmentu – w punkcie o danych współrzędnych na każdej warstwie wypada ten

sam punkt fizyczny.

9.8.2 Ulice

Informacje o ulicach pamiętane są w dwóch tabelach.

Tabela NAZWY_ULIC:

nazwa pola	opis	typ
ID	identyfikator ulicy	liczba naturalna, klucz główny
NAZWA	nazwa ulicy	ciąg znaków

Tabela FRAGMENTY_ULIC:

nazwa pola	opis	typ
ID_ULICY	identyfikator ulicy	odwołanie do tabeli NAZWY_ULIC
ID	identyfikator fragmentu ulicy	liczba naturalna, klucz główny
NR_POCZ	numer domu na początku fragmentu tej ulicy	liczba naturalna
NR_KONC	numer domu na końcu fragmentu tej ulicy	liczba naturalna
WSP_POCZ_X	współrzędna X początku ulicy na mapie	liczba naturalna
WSP_POCZ_Y	współrzędna Y początku ulicy na mapie	liczba naturalna
WSP_KONC_X	współrzędna X początku ulicy na mapie	liczba naturalna
WSP_KONC_Y	współrzędna Y początku ulicy na mapie	liczba naturalna

Rozdział 10

Wydajność systemu

10.1 Omówienie

System został zaprojektowany tak, by umożliwić korzystanie z niego dużej liczbie *Użytkowników* jednocześnie. Liczbę tę szacujemy na kilka tysięcy, a bardziej precyzyjnie będzie można ją określić po przeprowadzeniu testów obciążeniowych.

10.2 „Wąskie gardła” systemu

10.2.1 Obciążenie serwera

Skalowalność systemu

Dużą skalowalność systemu NIIKT osiągnięto dzięki zaimplementowaniu systemu w języku obiektowym Java oraz dobrym udokumentowaniu systemu. Na przykład, dodanie nowych pól opisujących ogłoszenie nie jest trudne – dodanie jednego pola można wykonać bez reorganizacji bazy danych.

Optymalizacje przesyłu danych

Przesyłanie danych z serwera do klienta może okazać się bardzo wąskim gardłem systemu. Dlatego zastosowano następujące techniki minimalizujące ilość przesyłanych informacji:

1. Korzystanie z pamięci podręcznej komputera klienta.

Przesyłane do *Użytkownika* fragmenty mapy, logo firm i katalogi branż będą zapamiętywane w pamięci podręcznej przeglądarki (przy odpowiedniej jej konfiguracji).

2. Wysyłanie „na przód”.

Staramy się przewidzieć, które dane będą potrzebne „w przyszłości”, np. transportujemy do klienta większe sąsiadujące mapy lub cały katalog branż, gdyż prawdopodobnie klient zechce z nich szybko korzystać.

10.2.2 Optymalizacja zapytań na bazie

Dzięki optymalizacji realizacji zapytań na bazie oraz przesyłu danych do komputera klienta system ma jak najszybciej obsługiwać zlecenia *Użytkowników*. Szacujemy ten czas na kilka sekund, lecz dopiero testy wydajnościowe uściślą te oszacowania.

Wykorzystywane są standardowe metody optymalizacji zapytań wbudowane w bazę Oracle© lub PostgreSQL, które zostały wsparte w systemie poprzez:

1. Indeksy.

Założenie indeksów przede wszystkim na tabelach: TELEFONY, ULICE, OGLOSZENIA, OGLOSZENIA_PLATNE.

2. Grona.

Zorganizowanie bazy w postaci gron, tj. grupowanie firm z tej samej branży.

10.2.3 Dostrajanie bazy danych

W fazie projektowania i testów nie można przewidzieć dokładnie sposobu rozrastania się bazy danych. Rozwój bazy można monitorować po wdrożeniu systemu i, na tej podstawie, dostrajć ją do konkretnych wymagań rynku.

Dostrajanie nie jest konieczne, może jednak korzystnie wpłynąć na szybkość działania systemu. Powinno być dokonane przez osobę kompetentną, na przykład przez administratora.

10.2.4 Monitorowanie przepustowości łącz

Zalecane jest, by administrator nadzorował obciążenie łącza z siecią i w przypadku, gdyby badania sieci wykazały, iż łącze jest zajęte w więcej niż 85% będzie konieczne jego poszerzenie lub dostawienie nowego łącza.

Rozdział 11

Jakość

11.1 Omówienie

Aby NIIKT był systemem „wysokiej jakości” musi spełnić wszystkie z omówionych poniżej kryteriów jakości systemu. Większość aspektów jakości jest zależna tylko od implementacji systemu: jego rozszerzalność, przenośność, łatwość obsługi. Natomiast bezpieczeństwo i niezawodność zależą również od sposobu użytkowania i konserwacji systemu, dlatego omówienie tych punktów zawiera również zalecane metody zapewniania jakości systemu po jego wdrożeniu.

11.2 Aspekty jakości systemu NIIKT zależne tylko od implementacji

11.2.1 Rozszerzalność

System NIIKT jest łatwo rozszerzalny o nowe funkcjonalności, dzięki zastosowaniu techniki projektowania obiektowego oraz przejrzystej hierarchizacji komponentów.

11.2.2 Przenośność

Zaimplementowanie systemu w języku Java zapewnia jego niezależność od platformy. System nie wymaga rekompilacji przy przeniesieniu na inną maszynę. Niezawodność interfejsu graficznego oraz łatwa podmiana szat graficznych zostanie zapewniona przez zastosowanie biblioteki *Swing*.

Ponadto, dzięki wydzielonej warstwie danych, system jest niezależny od bazy danych oraz sposobu przechowywania danych.

11.2.3 Łatwość obsługi

Duży nacisk postawiono na takie zaprojektowanie interfejsu *Użytkownika* oraz *Firmy*, aby był on „przyjemny dla oka”, a także prosty i szybki, tak by nawet niedoświadczony *Użytkownik*

nie miał problemów z jego obsługą.

11.3 Aspekty jakości systemu NIIKT zależne od użytkowania systemu

11.3.1 Niezawodność

System został zaprojektowany tak, by mógł pracować bez przerw oraz niezawodnie. Jednak awarie (spowodowane na przykład przerwą w dostawie prądu, uderzeniem pioruna, itd.) mogą się zdarzyć niezależnie od tego, jak system jest zaimplementowany.

Aby uniemożliwić jakąkolwiek utratę danych zalecane są rozwiązania:

1. Regularna i częsta archiwizacja danych z bazy na innych nośnikach, np. płytach DVD. To rozwiązanie, przy rozrastającej się bazie, może się okazać niewystarczające.
2. Postawienie bazy *stand-by*, umożliwiającej ciągłe utrzymywanie aktualnej kopii bazy głównej. To rozwiązanie jest kosztowne, lecz skuteczne.
3. Przechowywanie dziennika powtórzeń *Redo Log* oraz dziennika wycofań *Undo Log* na innym dysku niż baza główna.
4. Zastosowanie klastrów RAID, co może zabezpieczyć nie tylko bazę danych, lecz również dane przechowywane w plikach na serwerze.

11.3.2 Bezpieczeństwo

Bezpieczny przesył danych nie jest kluczowym wymogiem dla systemu NIIKT, gdyż przesyłane dane są prawnie dostępne dla każdego. Bezpieczeństwo przechowywanych informacji w bazie jest jednak bardzo ważne, dlatego system zostanie zbudowany na sprawdzonym oprogramowaniu Oracle© lub PostgreSQL, które wspiera utrzymywanie dzienników oraz ścisłą kontrolę usuwania danych z bazy.

Nieodpowiedzialna administracja systemem może jednak zmniejszyć poziom bezpieczeństwa danych, np. poprzez ustawianie nieskomplikowanego hasła administratora.

Rozdział 12

Historia zmian

\$Log: sad.tex,v \$

Revision 1.65 2004/01/12 20:18:12 miej

QA: zatwierdzenie dokumentu.

Revision 1.64 2004/01/12 20:16:01 miej

QA: ostateczna korekta.

Revision 1.63 2004/01/12 18:57:16 miej

QA: dodałem obiekty ZawKataloguBranz, ZawKatalogu,
PlikGraficzny.

Revision 1.60 2004/01/11 21:40:25 miej

QA: prefinałna korekta całości.

Revision 1.59 2004/01/11 20:19:22 miej

UHI: komentarze do diagramów przypadków użycia.

Revision 1.58 2004/01/11 18:44:37 miej

QA: chyba już ostatnie zmiany merytoryczne w warstwach.

Revision 1.57 2004/01/11 18:19:20 miej

QA: weryfikacja merytoryczna warstwy logiki
+ kosmetyczne poprawki w innych warstwach.

Revision 1.56 2004/01/11 17:52:28 miej

PW: Przemowienie w warstwie interfejsu.

Revision 1.55 2004/01/11 17:40:22 miej

QA: drobne poprawki w implementacji warstwy interfejsu.

Revision 1.54 2004/01/11 17:28:24 miej
QA: weryfikacja merytoryczna metod w implementacji
warstwy sieci.

Revision 1.53 2004/01/11 16:31:40 miej
PW: KonsolaAdministratora, WidokMapy.

Revision 1.52 2004/01/11 12:03:37 miej
PW: Dodanie WidokuStatystyk do InterfejsuAdministratora.

Revision 1.51 2004/01/11 11:38:02 miej
PW: InterfejsUzytkownika: Konsola, WidokListyWynikow,
WidokMapyWynikow: nowe metody.

Revision 1.50 2004/01/10 22:48:12 miej
QA: podpiąłem diagramy klas dla warstwy interfejsu,
logiki i sieci.

Revision 1.49 2004/01/10 19:33:28 miej
PW: Male zmiany w Widokach*KataloguBranz.

Revision 1.48 2004/01/10 17:52:54 miej
QA: drobna korekta merytoryczna wersji 1.41 - 1.44.

Revision 1.47 2004/01/10 17:01:46 miej
PW: InterfejsFirmy: zmienOgloszenieOk,
pobierzKatalog*, noweOgloszenieOk.

Revision 1.46 2004/01/10 16:58:22 miej
QA: zmiana obsługi wyszukikań z wizualizacją na mapie
w implementacji warstwy logiki biznesowej.

Revision 1.45 2004/01/10 16:26:45 miej
QA: przywrócenie moich linijek skasowanych przez kogoś:/

Revision 1.44 2004/01/10 14:59:24 miej
EK: 10/01/04 Drobna poprawka dla Uli (zwracanie na
parametrze ogłoszenia przy logowaniu sie firmy)

Revision 1.43 2004/01/10 14:40:29 miej
EK: 10/01/04 Dodalam kilka zdan o JDBC w prezentacji systemu.
Dodalam tez korzystanie z prepared statements w bazie.

Revision 1.42 2004/01/10 13:33:36 miej
PW: Zmiany w Interfejsie
Administratora

Revision 1.41 2004/01/10 10:31:55 miej
EK: 10/01/04:

- * napisałam omówienie implementacji systemu
- * opis warstwy danych w implementacji
- * zreorganizowałam dekompozycję logiczną tak,
by była zgodna z implementacją
- * poprawki warstwy sieciowej – całkowita reorganizacja
metod dla branż i metod dla mapy

Revision 1.40 2004/01/09 20:57:57 miej
PW: Zmiany w WidokuZarządzaniaOgłoszeniem i
WidokuNowegoOgłoszenia.

Revision 1.39 2004/01/09 20:27:51 miej
PW: Procesy, Instalacja, Warstwa Interfejsu.

Revision 1.38 2004/01/09 18:48:22 miej
PW: Instalacja, Interfejs Administratora (została Mapa).

Revision 1.37 2004/01/09 16:24:06 miej
QA: korekta rozdziału Implementacja.Warstwa interfejsu
(Użytkownik i Firma).

Revision 1.36 2004/01/09 15:16:54 miej
QA: cała Logika biznesowa jest teraz przystosowana
do wywołań asynchronicznych.

Revision 1.35 2004/01/09 10:59:41 miej
PW: Przywróciłem moje zmiany.

Revision 1.34 2004/01/09 10:49:28 miej
QA: dostosowanie pakietów Ogłoszenia, Konto, Mapa i Szablony
do komunikacji asynchronicznej.

Revision 1.32 2004/01/09 01:28:35 miej
EK: 08/03/04 * poprawka opisu diagramu dekompozycji
* dodanie funkcjonalności Historia w użytkowniku
* poprawka przechowywanych danych w zw. z decyzją trzymania

logo na serwerze
* dodanie pola MAPA w FRAGMENTY_MAPY
EK: 08/03/04 * napisanie pełnego punktu Warstwy sieci

Revision 1.30 2004/01/06 23:11:25 miej
QA: drobne zmianki merytoryczne i kosmetyczne
w Implementacji Warstwy logiki biznesowej.

Revision 1.29 2004/01/06 09:13:01 miej
QA: poprawka \RequirePackage{hyperref}.

Revision 1.28 2004/01/05 19:18:20 miej
QA: - reorganizacja Logiki Biznesowej (wyodrębnienie obiektów,
które można przysyłać pomiędzy warstwami i obiektów,
mieszkających w logice, służących do zarządzania).
- drobna zmiana w przechowywaniu mapy.

Revision 1.27 2004/01/04 23:03:00 miej
QA: korekta dokumentu (iteracja #3).

Revision 1.26 2004/01/04 19:20:17 miej
EK: Nowe pola tabeli BRANZE i OGLOSZENIE_PLATNE.
Inne kosmetyczne.

Revision 1.16 2004/01/01 22:17:15 miej
QA: pierwsza korekta.

Revision 1.15 2003/12/30 21:24:06 miej
PW: Warstwa interfejsu: pakiet Firma.

Revision 1.14 2003/12/30 15:50:54 miej
PW: Warstwa interfejsu: pakiet Użytkownik.

Revision 1.13 2003/12/27 16:27:36 miej
EK: 27/12/03 Opis implementacji warstwy sieci.

Revision 1.12 2003/12/27 13:56:23 miej
UHI: dodany rozdział 4

Revision 1.11 2003/12/27 13:34:14 miej
EK: 27/12/03 Dodałam poprawki Michała.

Revision 1.5 2003/12/26 11:07:18 miej

Revision 1.4 2003/12/25 14:14:34 miej

EK: 25/12/03: zrobiłam sensowną dekompozycję logiczną
(gruntowne zmiany), teraz wszystkie uwagi Michała są
uwzględnione.

Revision 1.3 2003/12/24 14:52:30 miej

EK: 24/12/03: Uwzględniłam prawie wszystkie poprawki Michała.

Revision 1.2 2003/12/18 02:18:37 miej

EK: 12/12/03 Zmiana szablonu. Dodanie paru podpunktów.

EK: 12/12/03 Punkty: wprowadzenie, prezentacja, dane,
jakosc, wydajnosć

EK: 12/12/03 Przygotowanie punktów:
realizacja przypadków użycia i dekompozycja logiczna.

EK: 17/12/03 Dekompozycja logiczna, jeszcze bez diagramu.

Revision 1.1 2003/10/30 23:05:32 miej

QA: Pierwotna wersja szablonów.

Zawiera zunifikowane formatowanie stron.