Akademia Górniczo-Hutnicza
im. Stanisława Staszica w Krakowie

Wydział Elektrotechniki, Automatyki, Informatyki
i Inżynierii Biomedycznej

Praca magisterska

Internetowy system dynamicznego uzgadniania i optymalizacji terminów spotkań według podanych kryteriów i preferencji z wykorzystaniem czatbota

Dawid Gdański

Promotor: dr hab. Adrian Horzyk

Kraków, 20 września, 2015

Oświadczenie pracy autora

Oświadczam, świadomy odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonałem osobiście i samodzielnie, i nie korzystałem ze źródeł innych niż wymienione w pracy.

.....

Serdecznie dziękuję promotorowi dr hab. Adrianowi Horzykowi za nadzór postępów i cenne wskazówki podczas tworzenia niniejszej pracy magisterskiej.

Spis treści

1. Wstęp

W dzisiejszym świecie jesteśmy otoczeni wieloma rzeczami, których zadaniem jest pomoc, upraszczanie trudności i asystowanie nam podczas życia codziennego. Nieprzerwany proces ewolucyjny pozwala nam na tworzenie narzędzi i rozwiązań, dzięki którym jesteśmy w stanie stwierdzić, czy problemy i zadania, z którymi mamy do czynienia posiadają rozwiązania lub nie posiadają ich wcale. Wyraźny postęp jest widoczny w sektorze technologii informacyjnych, a przede wszystkim internetu. Obecnie stanowi on największy zbiór informacji i jest potężnym narzędziem pozwalającym na tworzenie kolejnych następstw, które mogą się okazać bardzo przydatne, a nawet niezbędne do swobodnej egzystencji. Warto zwrócić uwagę także na przenośność informacji. Dostęp do internetu stał się bardzo powszechny. Urządzenia mobilne są w stanie zapewnić nam dostęp do internetu z dowolnej lokalizacji, w której rozwinięta jest cywilizacja. Pozwala to na przyśpieszenie procesów związanych z pozyskiwaniem informacji oraz ich przetwarzaniem.

Obecnie istnieją inteligentne systemy, które realizują konkretne zadania
i przyczyniają się do przyśpieszenia przepływu informacji. Wydajność takich systemów jest coraz lepsza. Można stwierdzić, że internet i systemy oparte o jego działanie realizują założenia sieci semantycznej (ang. *Semantic Web*) określanej terminem **Web 3.0**. Coraz więcej tych systemów jest wyposażona w mechanizmy zdolne do inteligentnej interakcji z użytkownikiem. **Czatbot** (ang. *chatterbot*) to wirtualny asystent zdolny do prowadzenia rozmowy z człowiekiem w języku naturalnym z wykorzystaniem interfejsu tekstowego i/lub głosowego, mogącym zastąpić człowieka w przyszłości w wielu czynnościach. Rozwój czatbotów został zainspirowany w 1950 roku przez Alana Turinga, który zaproponował test mający na celu pośrednie udowodnienie inteligentnych zdolności maszyny. Pierwszym klasycznym czatbotem była "Eliza". Taki czatbot jest w stanie odpowiedzieć użytkownikowi na pytania odnośnie celu działania systemu, w którym jest zastosowany, a także inne pytania z nim związane.
Przy okazji czatbotów nasuwa się następujące pytanie: w jakim stopniu wirtualny rozmówca mógłby wyręczyć człowieka? Czy mógłby swoją pracą zaoszczędzić czasu pewnym osobom? Czy przyśpieszyłby tym samym pewne procesy, które bez jego istnienia musiałyby odbyć się z udziałem ludzkiej interwencji?

Zasadniczą kwestią tworzenia każdego systemu są odbiorcy, którzy mają z niego korzystać. Trudną i jednocześnie niezbędną rzeczą jest trafienie w potrzeby użytkowników. Nieprzemyślane decyzje zazwyczaj skutkują rozwojem bezużytecznych funkcjonalności. Te z kolei nakładają coraz więcej ograniczeń odnośnie swobody projektowania i implementacji systemu, nadając niewłaściwy kierunek postępowania, co ostatecznie może prowadzić do "śmierci" systemu. W związku z tym pojawia się pytanie: jak prawidłowo trafiać w potrzeby użytkowników? Jak ustrzegać się błędów, które z perspektywy czasu przyczynią się do spadku jakości systemu zamiast ją polepszać?

Celem tej pracy magisterskiej jest zaprojektowanie i implementacja systemu, który realizuje zadanie uzgadniania terminów spotkań przy uwzględnieniu zadanych przez użytkownika ograniczeń. System, który się na nich opiera, posiada czatbota zdolnego do

udzielania mu informacji na temat związany z realizowanym zadaniem. Czatbot przyjmuje rolę pośrednika osoby, z którą chcą się spotkać petenci. Staje się także doradcą petenta w sprawie uzgodnienia terminu z osobą, którą reprezentuje.

Etapy projektowania systemu są podzielone na 3 główne części:

- opis modelu biznesowego i architektury korporacyjnej systemu;
- specyfikacja systemu;
- implementacja oraz scenariusze testowe.

2. Założenia i definicje

2.1. Kryteria i preferencje

Wiadomo, że organizując spotkanie chcemy spotkać się z pewną grupą osób, które posiadają potrzebę odbycia takiego spotkania. Bezsensownym zatem jest zapraszanie osób, które nie skorzystają w żaden sposób z informacji wymienionych podczas jego trwania. Pojawia się problem kategoryzacji rozmówców. Logiczną propozycją selekcji odpowiednich uczestników jest wprowadzenie pewnych preferencji, na podstawie których będzie można podzielić osoby zainteresowane odpowiednią tematyką spotkania. Zbiór tych preferencji można określić mianem pewnego kryterium. Można stwierdzić zatem, że każde spotkanie, na odbycie którego zachodzi potrzeba, ma pewien zbiór kryteriów, które są zbiorami preferencji pewnej grupy osób.

Kierując się powyższym rozumowaniem, zdefiniowałem w systemie spotkania, które mają ściśle określone kryteria. Tymi kryteriami są preferencje użytkowników. Jako przykład, posłużyłem się tematyką tytułów naukowych i akademickch obowiązujących na uczelniach wyższych. Wyróżniłem tytuły:

- STUDENT - student uczelni;
- PHD - pracownik uczelni z tytułem doktora;
- PROFESSOR - pracownik uczelni z tytułem profesora.

2.2. Dynamizm uzgadniania terminów spotkań

Dynamizm uzgadniania terminów jest zinterpretowany w sposób, w którym metoda zarządzania spotkaniami polega na interakcji z czatbotem, który analizuje sekwencje słów wpisanych przez użytkownika i stara się odpowiedzieć na jego pytania, zaspokoić jego potrzeby.

2.3. Czatbot

Czatbot jest programem komputerowym, którego zadaniem jest prowadzenie konwersacji przy użyciu języka naturalnego, sprawiając wrażenie inteligentnego. Poprawnie stworzony, powinien odznaczać się cechą bycia jak najbardziej ludzkim. W 1950 r. Alan Turing zaproponował test [19] podczas badań nad sztuczną inteligencją. Polega on na przeprowadzeniu konwersacji przez człowieka (sędziego) z pozostałymi stronami. Jeżeli sędzia nie jest w stanie jednoznacznie stwierdzić czy ma do czynienia z maszyną, test uznaje się za pozytywny.

Czatbot pełni rolę negocjatora w systemie. Stanowi osobistego asystenta użytkownika, z którym chcą się umówić petenci. Kompetencje wirtualnego rozmówcy sprowadzają się do przedstawiania aktualnych terminów spotkań. Czatbot jest w stanie rozmawiać w języku naturalnym. Jego stopień inteligencji jest dobrany w taki sposób, aby był on w stanie dać petentowi właściwe zinterpretowaną odpowiedź dotyczącą jego pytania lub prośby.

2.3.1 Sztuczna inteligencja

Mając doświadczenie w projektowaniu systemów internetowych, można odnieść wrażenie, że z punktu widzenia programisty do stworzenia omawianego systemu wystarczy opanować zarządzanie relacyjnymi bazami danych oraz język programowania. Czatbot także mógłby interpretować pytania użytkowników i parować je z odpowiednimi wzorcami, generując odpowiedź. Faktycznie, jest to prawdziwe stwierdzenie, aczkolwiek tych wzorców musiałoby być naprawdę dużo i ich baza musiałaby się powiększać, gdyby wirtualny rozmówca dawał bezsensowne odpowiedzi na postawione przez użytkowników pytania. Następstwem rozrostu bazy wzorców byłby spadek wydajności systemu. Odpowiedzi generowałyby się coraz dłużej, system stawałby się nieoptymalny. Sztywne dopisywanie kolejnych wzorców byłoby daremne, w którymś momencie czas odpowiedzi serwera mógłby zostać przekroczony.

Pan dr hab. Adrian Horzyk w swojej książce pt. "Sztuczne systemy skojarzeniowe i asocjacyjna sztuczna inteligencja" [6] opisuje wiele nowoczesnych podejść w zakresie sortowania oraz przeszukiwania informacji. Odrzucone zostają klasyczne struktury danych takie jak: listy, zbiory mapy itd. ze względu na ich nieefektywność dla zastosowań omawianych w publikacji. Szeroko natomiast wykorzystywane są grafy oraz różnorodne kombinacje ich tworzenia. Mają one na celu odwzorowanie pracy ludzkiego mózgu. Często wykorzystywane zostaje także zjawisko kojarzenia, które w ludzkim umyśle jest głównym czynnikiem stopnia inteligencji. Biologiczny system skojarzeniowy BAS (ang. *Biological Associative System*) jest dynamiczną neuroaktywną strukturą mózgu i części układu nerwowego. Na jego podobieństwo tworzy się sztuczny system skojarzeniowy AAS (ang. *Artificial Associative System*). Podobnie do systemów biologicznych, AAS-y swoją grafową strukturą starają się odwzorować procesy kojarzenia i uzyskiwania informacji, jakie zachodzą w ludzkim mózgu.

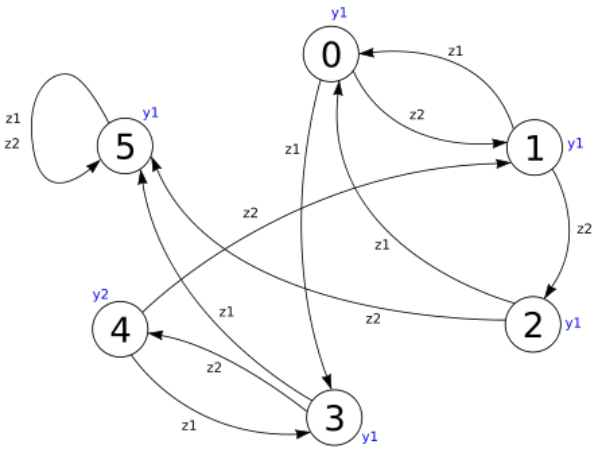
Ludzki mózg jest genialnie skonstruowanym mechanizmem efektywnie interpretującym i przetwarzającym dane [4]. Główną mechaniką ludzkiego umysłu nie jest jednak ciąg operacji obliczeniowych, które permutowane w inteligentny sposób potrafią jednoznacznie wygenerować odpowiedź. Ta mechanika opiera się na wiedzy formowanej z wielu jednostek informacji sprzężonych z wieloma sprecyzowanymi kontekstami informacji. Jednostki te, grupując się w coraz większe, kształtują zgeneralizowaną wiedzę oraz rozwijają kreatywność. Generalizacja jest nieodłącznym elementem modelowania wiedzy i inteligencji. W obecnym rozwoju sieci neuronowych posiadamy wiedzę na temat wielu rodzajów ich struktur, modeli neuronów oraz metod, które umożliwiają inicjację procesów aproksymacji, asocjacji, przewidywania, rozpoznawania oraz klasyfikacji. Sztuczne sieci neuronowe (ang. *Artificial Neural Network*) używają pewnych algorytmów, które nie istnieją w ludzkim mózgu. Biologiczne neurony natomiast wykorzystują wewnętrzne, plastyczne mechanizmy, które umożliwiają im nawiązywanie aktywnych połączeń, dzięki którym tworzą się coraz nowsze sekwencje generujące skojarzenia. W miarę pobudzania konkretnych aktywnych połączeń, zostają one utrwalane i uwzględniane przy przeszukiwaniu przyszłych sekwencji podczas generowania skojarzeń.

2.3.2 Grafy

Grafem nazywamy parę $G=(V,E)$, gdzie V jest niepustym zbiorem skończonym, natomiast E jest dowolnym podzbiorem zbioru nieuporządkowanych par elementów zbioru V. Elementy zbioru V nazywane są wierzchołkami (ang. *vertex*), zaś elementy zbioru

E krawędziami (ang. *edge*). Poszczególne wierzchołki grafu mogą być połączone w taki sposób, iż każda krawędź zaczyna się i kończy w którymś z wierzchołków. Grafy definiujemy za pomocą numerowania, etykietowania wierzchołków i krawędzi w zależności od potrzeb modelu, który tworzymy. W porównaniu do drzew, w grafach mogą występować pętle i cykle, krawędzie mogą posiadać wyznaczony kierunek (graf skierowany).

Grafy stanowią reprezentację danych, która spośród wszystkich istniejących rodzajów



Rysunek 2.1: Przykład grafu z występującymi cyklami.

najlepiej jest w stanie odwzorować mózg i zasadę jego działania. Neurony mogą być interpretowane jako wierzchołki, zaś synapsy jako krawędzie.

2.4. Potrzeby

Potrzeba oznacza poczucie niespełnienia czegoś, co jest niezbędne do życia, odpowiedniego funkcjonowania. Przede wszystkim potrzeby są tendencjami do postępowania. Oznacza to, że zawierają energię psychiczną, bez której nie moglibyśmy formułować żadnej myśli czy wykonywać żadnego gestu. Istnieją różne podziały potrzeb. Dr Adrian Horzyk w jednej ze swoich publikacji pt. "Negocjacje - Sprawdzone strategie" [5] w rozdziale nr 2 definiuje podział potrzeb człowieka w zależności od jego osobowości. Ten podział przewiduje:

- potrzeby fizjologiczne - związane nierozłącznie z konstrukcją ludzkiego ciała;
- potrzeby charakteru - związane według pewnych teorii biopsychologicznych ze specyficzną budową naszego mózgu;
- potrzeby intelektu - związane z budową i ze stopniem skomplikowania naszego mózgu;
- potrzeby duchowe - związane z pragnieniem czynienia dobra i zła, postępowania w pewnym stopniu niezależnie od fizjologii i osobowości.

W zależności od wyróżnionych potrzeb każdy typ utożsamiany jest z czterema rodzajami stanów:

- wzbudzenie potrzeby - zespół pewnych czynników skojarzonych z daną potrzebą generuje narastające impulsy w kierunku zrealizowania potrzeby;

- wygaszenie potrzeby - zespół pewnych czynników skojarzonych z daną potrzebą generuje słabnące impulsy w kierunku zrealizowania potrzeby;
- zaspokojenie potrzeby - pewne składniki zostały dostarczone lub pewne warunki spełnione, w wyniku czego potrzeba staje się mniej intensywna;
- pogłębienie potrzeby - pewne składniki zostały dostarczone lub pewne warunki spełnione, w wyniku czego potrzeba staje się bardziej intensywna.

Z punktu widzenia systemu potrzebą użytkownika jest chęć spotkania z konkretną osobą. Spełnienie potrzeby spowoduje osiągnięcie zamierzonego celu, wymiany informacji. Czatbot odgrywa rolę pomocnika, który pomaga tę potrzebę zaspokoić, uwzględniając jednocześnie kryteria spotkania w oparciu o preferencje użytkowników.

3. Model biznesowy systemu

Niniejszy rozdział ma na celu sprecyzowanie celu istnienia systemu jako produktu w ujęciu biznesowym. Podczas studiów 2 stopnia uczęszczałem na laboratoria specjalizacyjne prowadzone przez Pana prof. dra hab. inż. Jana Werewkę. Dzięki cyklowi laboratoriów zyskałem niezbędną wiedzę do tworzenia modeli biznesowych, tzw. modeli "Business Model Canvas" oraz warstw architektury korporacyjnej systemu: biznesowej, aplikacji i infrastruktry. Cykl laboratoriów ukończyłem pozytywnie.

3.1. Opis modelu "Business Model Canvas"

Opis modelu biznesowego, w skrócie, polega na zdefiniowaniu i opisaniu zasad jak przedsiębiorstwo powinno tworzyć, dostarczać i otrzymywać wynagrodzenie za produkt, który decyduje się dystrybuować. Model biznesowy "Business Model Canvas" (w skrócie BMC) został po raz pierwszy raz zaproponowany w 2010 roku [1]. Model ten jest szablonem strategii zarządzania oraz rozwijania wartości proponowanej i składa się z 4 głównych czynników, które go definiują.

3.1.1 Infrastruktura produktu

Pierwszym czynnikiem jest infrastruktura produktu. Zwraca ona uwagę na walory systemu, które dedykowane są użytkownikom. Infrastruktura definiuje:

1. Działania (ang. *Key Activities*), dzięki którym firma rozwijająca produkt zwiększa jego atrakcyjność poprzez rozszerzanie oferty.
2. Zasoby (ang. *Key Resources*), dzięki którym firma rozwija produkt lub usługę.
3. Sieć partnerską (ang. *Key Partnerships*), dzięki której możliwa jest analiza redukcji pewnymi ryzykami modelu biznesowego oraz optymalizacja operacji nabywca-dostawca; określa też potencjalnych strategicznych partnerów.

3.1.2 Oferta

Drugim czynnikiem jest są oferty (ang. *Value Propositions*). Oferta precyzuje zbiór produktów lub usług dedykowanych klientom. Przy okazji tworzenia ofert warto również uwzględnić takie oferty przedsiębiorstwa, które wyróżnią je na tle konkurencji lub wprowadzą innowację.

3.1.3 Klienci

Kolejnym czynnikiem są klienci. Na ten czynnik składają się takie zagadnienia, jak:

1. Podział klientów (ang. *Customer Segments*) - budowa efektywnego modelu biznesowego w dużej mierze zależy od grupy klientów, dla których produkt lub usługa jest przeznaczona.

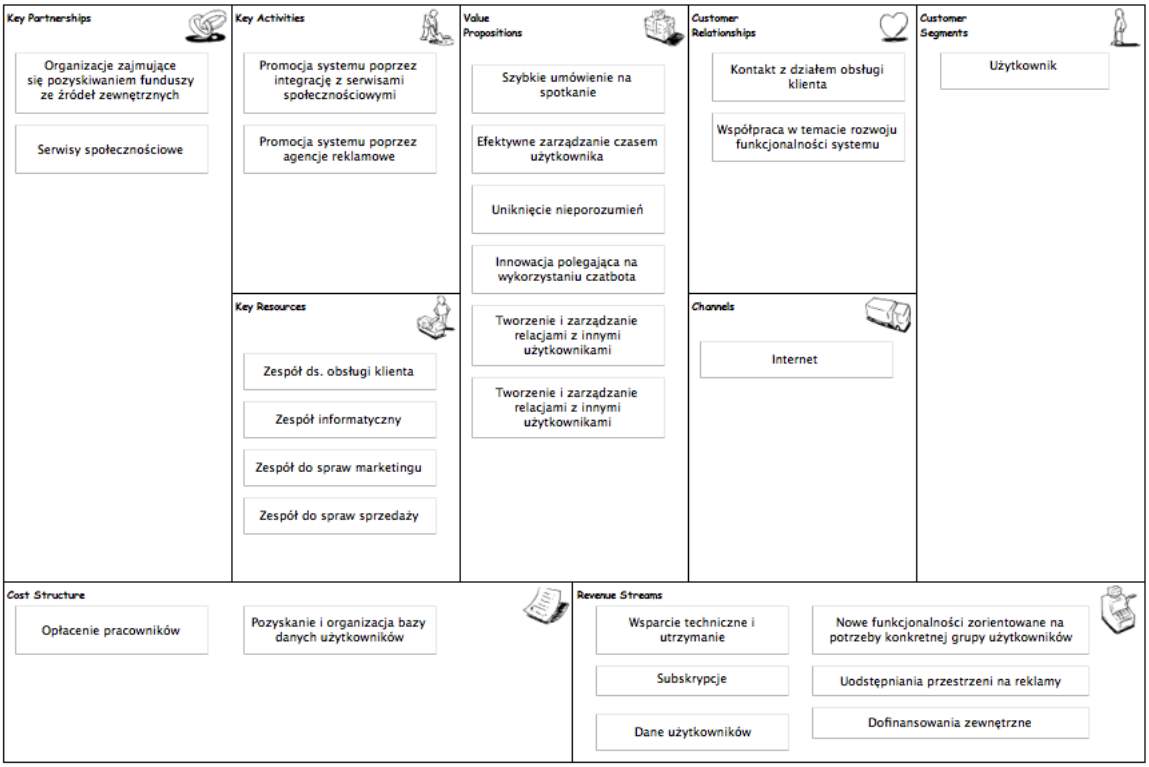
2. Kanały dystrybucji (ang. *Channels*) - firma tworząca produkt lub usługę dostarcza usługi lub produkt klientom poprzez różne kanały. Efektywne kanały dystrybucji w znacznym stopniu przyśpieszą świadomość użytkowników o produkcie lub usłudze oraz powiększającej się ofercie.

3. Relacje z klientami (ang. *Customer Relationships*) - respekt wśród klientów wpływa na świadomość o oferowanym produkcie lub usłudze oraz na poszerzanie grona nowych klientów.

3.1.4 Finanse

Ostatnim czynnikiem są finanse. Opisują one koszty (ang. *Cost Structure*), jakie przedsiębiorstwo jest zobowiązane ponieść by wdrożyć i utrzymywać produkt. Prezentują także przychody (ang. *Revenue Streams*), które generują poszczególne segmenty klientów.

3.1.5 Proponowany model biznesowy systemu



Rysunek 3.1: Przykładowy projekt Business Model Canvas dla opisywanego systemu.

Na powyższym schemacie widoczny jest przykładowy model biznesowy systemu. W sekcji działań skupiono się na promocji systemu. Opisywany system ma podobny charakter do popularnych serwisów społecznościowych. Rozsądnym krokiem jest zatem integracja z takimi serwisami. Drugą alternatywą jest promocja poprzez agencje reklamowe. W sekcji zasobów zaproponowano grupy odpowiedzialne za różne aspekty w sensie utrzymania, rozwoju oraz promocji systemu. W sieci partnerskiej zaproponowano kolaborację z instytucjami pozyskującymi fundusze pochodzące ze źródeł zewnętrznych (np. z Unii

Europejskiej) oraz podobnymi serwisami społecznościowymi.

W sekcji ofert zaproponowano rozwiązania, które mają przede wszystkim zoptymalizować czas potrzebny na umówienie się na spotkanie oraz wprowadzić element innowacyjności, dający przewagę nad systemami konkurencji.

W podziale klientów zdefiniowano "użytkownika" jako jedynego przedstawiciela tego rodzaju. Wynika to z faktu, iż specyfika systemu pod względem ról, jakie można przypisać w podziale klientów jest transparentna. Użytkownikiem systemu może być każda osoba. Proponowanymi elementami w temacie relacji z użytkownikiem jest łatwość w kontakcie z działem obsługi klienta oraz współpraca z klientami w kontekście rozwoju systemu. Proponowany kanał dystrybucji to Internet.

W sekcji finansów zaproponowano generalne koszty, jakie przedsiębiorstwo jest zobowiązane ponieść w ramach wynagrodzenia pracowników. W związku z faktem, iż dane użytkowników są głównym źródłem dochodów, istotne jest także zadbanie o wysokiej klasy sprzęt i oprogramowanie. Dochodami natomiast są: model subskrypcyjny systemu, wspomniane dane użytkowników, wsparcie techniczne systemu rzutujące na poszerzanie gamy klientów, a także opcja dodawania funkcjonalności za dopłatą dla konkretnych grup użytkowników.

Rysunek poglądowy został wykonany w standardzie *Archimate* ® [15].

3.2. Korporacyjny model architektury systemu

Podczas projektowania produktu informatycznego oprócz modelu biznesowego warto także rozważyć stworzenie korporacyjnego modelu architektury systemu. Pan dr hab. inż. Andrzej Sobczak w swoim artykule na temat modeli i metamodeli architektury korporacyjnej [18] definiuje znaczenie samego terminu i dzieli go na różne typy. Typy te, łączone ze sobą, przedstawiają spójną całość oraz obrazują różne aspekty specyficzne dla danego typu modelu.

Definicja architektury korporacyjnej mówi, że jest ona formalną reprezentacją właściwości korporacji. Według definicji zawartej w artykule Pana dra Andrzeja Sobczaka jest ona interpretowana, jako "strategiczny zasób informacyjny, w ramach którego określona jest misja korporacji, zasoby techniczne i informacje potrzebne do realizacji tej misji oraz proces przejścia mający na celu implementację nowych rozwiązań technicznych na skutek zachodzących zmian strategicznych".

Całkowity model architektury jest definiowany poprzez zespół celów, jakimi kieruje się korporacja. W doborze konkretnych typów modeli panuje pewna dowolność. Dzieje się tak, ponieważ różni interesariusze wyrażają odmienne troski, które zebrane w całość generują konkretny typ. Mark Lankhorst w swojej książce pt. "Enterprise Architecture at Work" [8] definiuje podział na następujące typy modeli:

- model statyczny - reprezentowany przez obiekty, na których wykonywane są konkrentne działania lub które wykorzystywane są w konkretnych działaniach;

- model dynamiczny - przypisane do obiektów modelu statycznego; definiują, kto jest odpowiedzialny za realizację danego działania;
- model odnoszący się do indywidualnych elementów - opisuje usługi realizowane przez pojedyncze elementy strukturalne;
- model odnoszący się do interakcji - prezentuje usługi świadczone przez współpracujące elementy strukturalne.

Istotną kwestią w etapie tworzenia modelu architektury jest wyznacznik jakości. Dobrze zdefiniowany model rzutuje na przyszłe sukcesy przedsiębiorstwa i pozwala na jego dalszą rozbudowę.

3.2.1 Wzorzec architektoniczny

W inżynierii oprogramowania wzorzec architektoniczny [3] to sprawdzony i uznany sposób rozwiązania określonego problemu w obrębie architektury oprogramowania. Głównymi korzyściami z tworzenia wzorców architektonicznych są:

- rozwiązanie określonego problemu w obrębie architektury oprogramowania;
- podzielenie systemu na elementy mające mniejsze odpowiedzialności;
- zdefiniowanie i przypisanie odpowiedzialności elementom systemu;
- określenie zasad komunikacji między elementami systemu.

Istnieje wiele wzorców architektonicznych, według których projektowane są systemy. Najbardziej powszechne z nich to:

- architektura wielowarstwowa - jest to architektura typu klient-serwer, w której zdefiniowane są niezależnie: interfejs użytkownika, przetwarzanie oraz składowanie danych. Te trzy komponenty mogą dzielić się na mniejsze osobne warstwy mogące być rozwijane niezależnie od siebie;
- architektura trójwarstwowa - jest to architektura typu klient-serwer, w której komponenty: interfejs użytkownika, przetwarzanie danych oraz składowanie danych rozwijane są w postacie osobnych modułów;
- architektura Model-Widok-Kontroler (ang. *Model-View-Controller*) - jest to architektura, która zakłada podział aplikacji na trzy części: model (pewna reprezentacja problemu lub logiki aplikacji), widok (opisuje jak wyświetlić pewną część modelu w ramach interfejsu użytkownika), kontroler (przyjmuje dane wejściowe od użytkownika i reaguje na jego poczynania zarządzając aktualizacją modelu oraz odświeżenie widoku;
- architektura zorientowana na usługi (ang. *Service-Oriented Architecture*) - idea tworzenia systemów informatycznych, w której główny nacisk stawia się na definiowanie usług, które spełnią pewne wymagania użytkownika;
- wywołanie niejawne - sposób projektowania oprogramowania, w którym system konstruuje się wokół obsługi zdarzeń, używając formy wywołania zwrotnego.

3.2.2 Wstępny opis proponowanego modelu architektury biznesowej

Wraz z modelem biznesowym zaproponowałem trójwarstwowy model architektury systemu skupiający się na warstwach: motywacji, aplikacji i infrastruktury. Projekt został wykonany w standardzie *ArchiMate* ® [15].

3.2.3 Warstwa motywacji

Warstwa motywacji składa się z elementów i połączeń między nimi, które odpowiednio dobrze zaprojektowane, zestawiają i obrazują cele firmy, potencjalne zagrożenia, a także wymagania i istniejące ograniczenia.
Istotnym w warstwie motywacji jest zrozumienie czynników sterujących (ang. *driver*), które wpływają na architekturę korporacyjną. Czynniki sterujące dzielą się na dwa typy: wewnętrzne i zewnętrzne. Wewnętrzne czynniki, tzw. troski (ang. *concerns*), są skojarzone z interesariuszami (ang. *stakeholder*), którzy reprezentują indywidualności lub pewne grupy osób. Przykładem czynników wewnętrznych jest: satysfakcja klienta, zgodność
z prawem, dochodowość. Zasadniczo, każdy biznes musi określić swoje wewnętrzne troski i w odpowiedni sposób zadbać o ich wypełnienie. Zewnętrzne czynniki wskazują na sytuacje, których przedsiębiorstwo musi być świadome podczas dążenia do zamierzonych celów, np. rozwój konkurencji.

Właściwe motywacje są reprezentowane przez cele (ang. *goal*), pryncypia (ang. *principle*), wymagania (ang. *requirements*) i ograniczenia (ang. *constraints*). Cele reprezentują pożądane rezultaty, które interesariusz chce osiągnąć, np. zwiększenie zadowolenia klienta. Pryncypia oraz wymagania odwzorowują pożądane właściwości konkretnego rozwiązania do osiągnięcia celu. Pryncypia stanowią wyznaczniki generujące wszystkie możliwe rozwiązania w obrębie pewnego kontekstu. Na przykład pryncypium "System powinien być maksymalnie przyjazny użytkownikowi" wraz z wymaganiem "Innowacyjny sposób umówienia na spotkanie" może generować rozwiązania do osiągnięcia celu - "Pozyskanie klientów konkurencji". Wymagania obrazują formalne stwierdzenia potrzeb wyrażanych przez interesariuszy, które muszą zostać spełnione przez architekturę systemu.

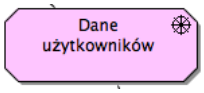
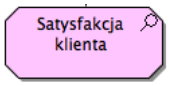
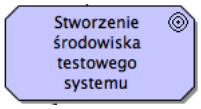
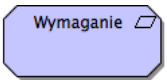
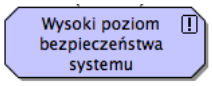
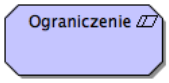
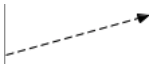
Nazwa	Oznaczenie	Funkcja
Interesariusz		Rola jednostki, zespołu lub organizacji (lub jej pochodnej), która może pozostawać pod wpływem lub mieć wpływ na wynik architektury. Przykład: zarząd, klient.
Czynnik sterujący		Element tworzący, motywujący lub napędzający zmiany w organizacji. Przykład: zmiany gospodarcze, notowania giełdowe.
Ocena		Obrazuje silne i słabe strony, szanse oraz zagrożenia z punktu widzenia pewnego obszaru zainteresowania. Przykład: innowacyjne rozwiązanie problemu, niezadowolenie klienta z poziomu usług dostarczanych przez firmę.
Cel		Stan, do którego dąży interesariusz.
Wymaganie		Definiują zbiór właściwości, które są potrzebne do osiągnięcia końcowych celów określanych przez element typu “Cel”.
Pryncypium		Opisują ściśle powiązane właściwości wszystkich systemów w obrębie tego samego kontekstu. Powiązane ściśle z elementami typu “Cel” oraz “Wymaganie”
Ograniczenie		Restrykcja informująca o sposobie zrealizowania systemu.
Relacja realizacji		Umożliwia opisywanie relacji między różnymi elementami, np. element “Pryncypium” może realizować element “Cel”.
Relacja wpływu		Umożliwia opisywanie wpływu jaki element motywacji może mieć w stosunku do innych elementów motywacji.

Tabela 3.1: Elementy i połączenia warstwy motywacji.

3.2.4 Warstwa biznesu

Struktura warstwy biznesu odnosi się do struktury organizacji, na którą składają się pewne jednostki wraz z zależnościami między nimi. Jednostki organizacji dzielimy na aktywne i pasywne. Aktywnymi jednostkami są podmioty takie jak: aktor biznesowy lub rola biznesowa. Definiują one zachowanie procesów i funkcji biznesowych. Aktorami biznesowymi mogą być indywidualne osoby (klienci lub pracownicy) lub grupy ludzi (jednostki organizacyjne) oraz zasoby mające niezmienny status w obrębie organizacji (np. oddział badań

i rozwoju, oddział marketingowy).

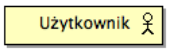
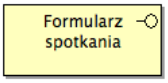
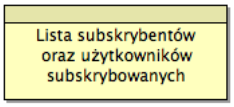
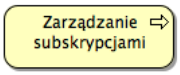
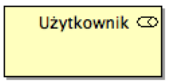
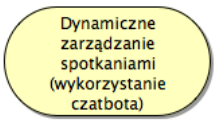
Nazwa	Oznaczenie	Funkcja
Aktor biznesowy		Jest elementem strukturalnym. Jednostka mająca wpływ na wywołanie określonego zachowania systemu, np. klient, administrator systemu.
Interfejs biznesowy		Jest elementem strukturalnym. Zdefiniowany jako punkt dostępu, który dostarcza funkcjonalność realizowaną przez usługę biznesową.
Obiekt biznesowy		Jest elementem strukturalnym. Odwzorowuje element informacyjny, który jest istotny z poziomu biznesowego
Proces biznesowy		Jest elementem zachowania. Jego zadaniem jest zdefiniowanie sekwencji działań biznesowych.
Rola biznesowa		Jest elementem strukturalnym. Jest to odpowiedzialność, jaką wywołuje aktor biznesowy w celu osiągnięcia pożądanego zachowania systemu.
Usługa biznesowa		Jest elementem zachowania. Jest to usługa, która realizuje wymagania biznesowe klienta.

Tabela 3.2: Najczęściej wykorzystywane elementy i połączenia warstwy biznesu.

3.2.5 Warstwa aplikacji

Warstwa aplikacji składa się w dużej mierze z komponentów aplikacyjnych. Komponentem aplikacyjnym jest modularne i reużywalne oprogramowanie systemowe, które otacza (enkapsuluje) swoje zachowanie oraz dane poprzez interfejsy widoczne na zewnątrz. Komponent aplikacji może być przypisany do jednej lub wielu funkcji aplikacji, procesów lub funkcji biznesowych. Posiada on także jeden lub więcej interfejsów, które definiują jego funkcjonalność. Interfejsy aplikacji mogą być używane przez wiele jej komponentów.

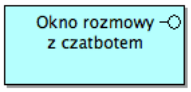
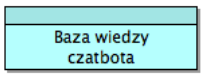
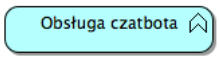
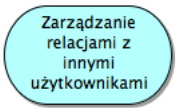
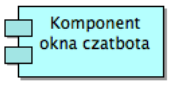
Nazwa	Oznaczenie	Funkcja
Interfejs aplikacji		Jest elementem strukturalnym. Obrazuje, w jaki sposób pewna funkcjonalność może być wykorzystywana przez inne komponenty w obrębie systemu.
Obiekt danych		Jest elementem strukturalnym. Funkcje aplikacji wykorzystują ten obiekt ze względu na informacje, jakie dostarcza.
Funkcja aplikacji		Jest elementem zachowania. Opisuje wewnętrzne zachowanie komponentu systemu.
Usługa aplikacji		Realizowana przez jedną lub więcej funkcji aplikacji. Opcjonalnie może operować na obiekcie danych. Powinna dostarczać funkcjonalność użyteczną z punktu widzenia użytkownika.
Komponent aplikacji		Część oprogramowania systemowego obejmująca dane oraz zestaw interfejsów, przez który je udostępnia.

Tabela 3.3: Najczęściej wykorzystywane elementy warstwy aplikacji.

3.2.6 Warstwa infrastruktury (technologiczna)

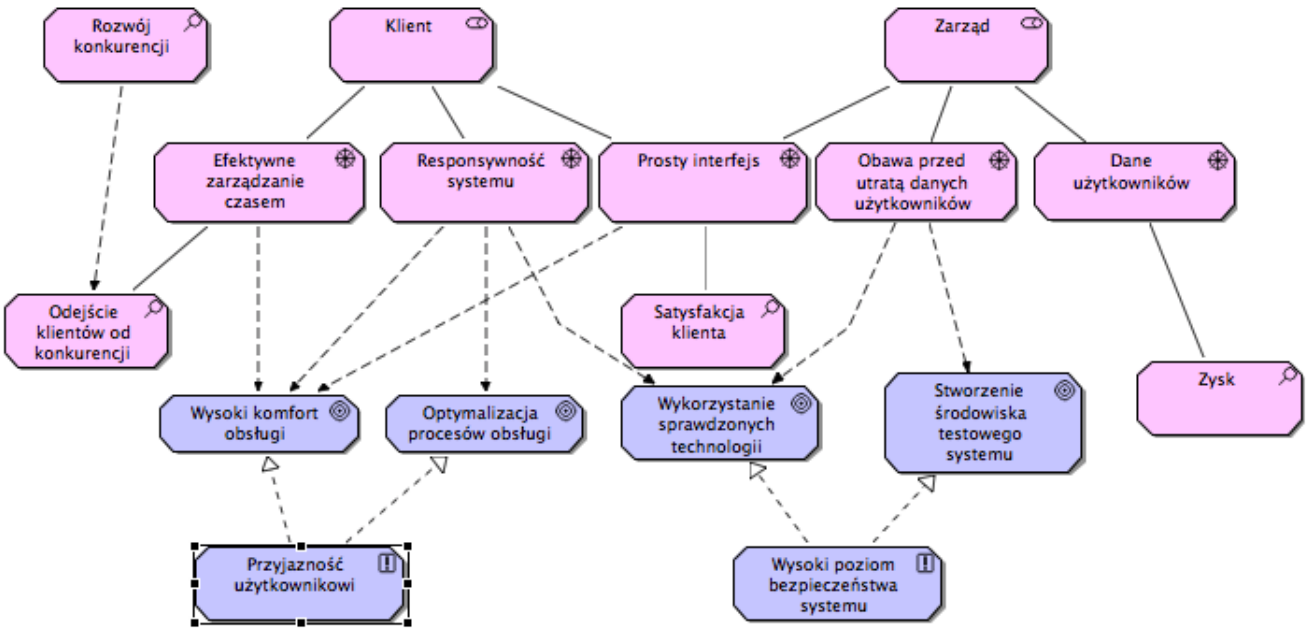
Warstwa infrastruktury opisuje część technologiczną systemu. Głównymi komponentami w tej warstwie są węzły (ang. *node*). Węzły można podzielić na dwa typy: fizyczne urządzenie i oprogramowanie systemowe. Oprogramowanie systemowe jest komponentem, który działa dzięki fizycznemu urządzeniu. Urządzenie opisuje fizyczny zasób, w którym oprogramowanie może zostać zainstalowane i uruchamiane.

Nazwa	Oznaczenie	Funkcja
Artefakt	Graf danych użytkowników	Jest elementem informacyjnym. Odwzorowuje element będący fizycznym podzbiorem danych, np. plik, tabela bazy danych, skrypt.
Węzeł	Serwer aplikacji	Jest elementem strukturalnym. Definiowany jako zasób obliczeniowy zdolny do przechowywania lub wdrażania działania artefaktów.
Urządzenie	SUSE Linux Enterprise Server 11	Zasób sprzętowy zdolny do zarządzani artefaktami.
Usługa	Zarządzanie relacjami z innymi użytkownikami	Jest elementem zachowania. Udostępnia funkcjonalność punktu węzłowego.
Oprogramowanie systemu	JBoss Enterprise Edition	Jest elementem strukturalnym. Zdefiniowane jako oprogramowanie tworzące środowisko uruchomieniowe specyficznych komponentów i obiektów.

Tabela 3.4: Elementy warstwy infrastruktury.

3.2.7 Proponowany korporacyjny model architektury systemu

W ostatniej części opisującej korporacyjny model architekturalny stworzyłem przykład takiego modelu. Poniższy przykład składa się z trzech rysunków poglądowych opisujących kolejno omawiane warstwy.



Rysunek 3.2: Proponowana warstwa motywacji.

Jako dwóch głównych interesariuszy wyróżniłem klienta systemu oraz zarząd przedsiębiorstwa dystrybuującego system. Każdy z interesariuszy posiada własne czynniki sterujące, które w konsekwencji prowadzą do konkretnych celów.

Z punktu widzenia klienta istotnymi czynnikami sterującymi są: efektywne zarządzanie czasem, responsywność systemu oraz prosty interfejs. Natomiast zarząd przedsiębiorstwa dystrybuującego system kieruje się takimi czynnikami jak: dane użytkowników, obawa przed utratą danych użytkowników. Częścią wspólną obu interesariuszy jest prosty i intuicyjny interfejs systemu.

Analizując diagram od strony pryncypiów, dwa najbardziej wyszczególnione elementy to przyjazność użytkownikowi oraz wysoki poziom bezpieczeństwa danych. Przyjazność systemu jest realizowana poprzez cele: wysoki komfort oraz optymalizacja procesów obsługi. Bezpieczeństwo znajduje swoją realizację w wykorzystaniu sprawdzonych technologii oraz przykładaniu dużej uwagi do integracji ze środowiskiem testowym potwierdzającym teoretycznie jego poziom.

Na cel komfortu obsługi składają się wszystkie czynniki sterujące interesariusza "Klient". Na optymalizację procesów obsługi ma wpływ responsywność systemu. Wykorzystanie sprawdzonych technologii determinują responsywność systemu oraz obawa przed utratą danych użytkowników. Drugi z determinantów także wpływa na stworzenie środowiska testowego systemu.

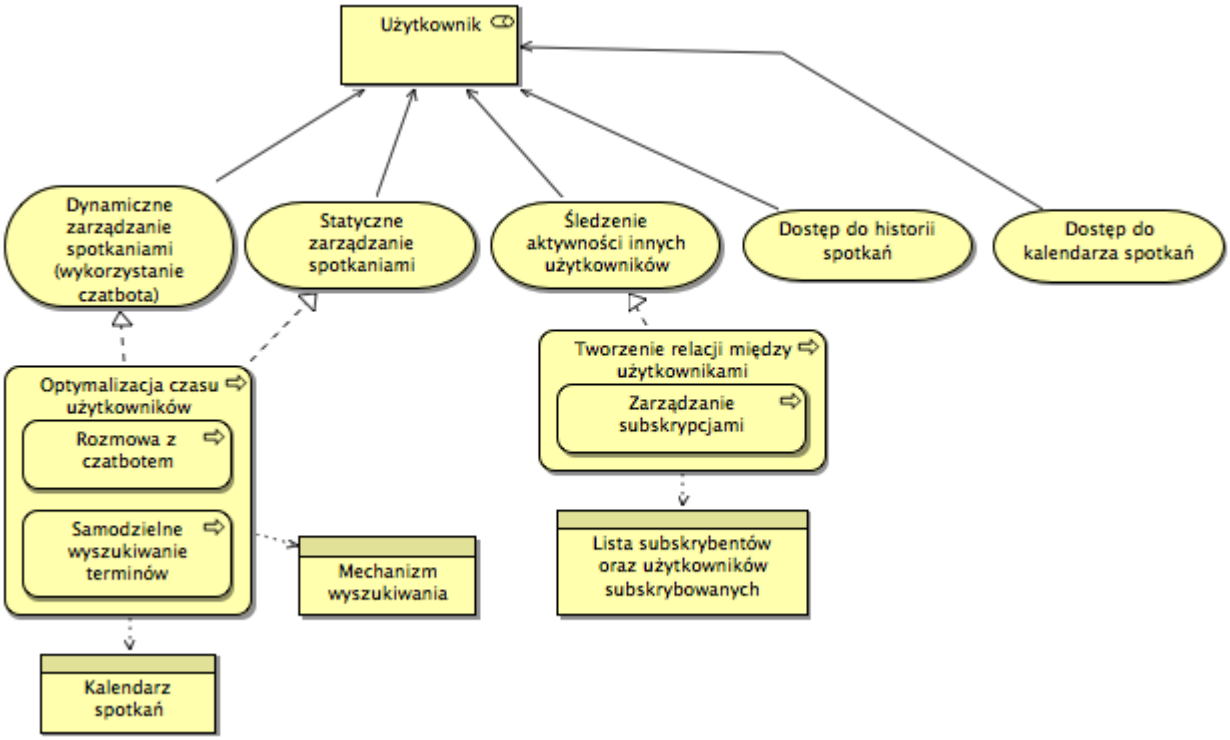
W warstwie motywacji wyszczególniono oceny połączone bezpośrednio z czynnikami sterującymi. Pierwsza z nich, odejście klientów od konkurencji, jest ściśle powiązana z efektywnym zarządzaniem czasem. Ten czynnik sterujący można powiązać z jedną z głównych funkcjonalności systemu. Wobec tego należy upatrywać napływu nowych klientów, kiedy ten czynnik zostanie dobrze zinterpretowany i przełożony na cel, który realizuje. Niezależnie, należy też brać pod uwagę rozwiązania konkurentów. Ich rozwój może wpływać na zahamowanie przyrostu napływu nowych klientów.

Druga ocena jest utożsamiana z finansowymi korzyściami płynącymi ze stworzenia systemu. Wartością dla przedsiębiorstwa jest zysk, który ma być generowany na podstawie danych zarejestrowanych użytkowników.

W warstwie biznesowej wyszczególniłem użytkownika, jako jedyną rolę biznesową. Usługi biznesowe, które mogą być przez niego używane to:

- dynamiczne zarządzanie spotkaniami z wykorzystaniem czatbota;
- statyczne zarządzanie spotkaniami poprzez interfejs użytkownika;
- śledzenie aktywności innych użytkowników;
- dostęp do historii spotkań w interfejsie użytkownika;
- dostęp do kalendarza spotkań w interfejsie użytkownika.

Analizując diagram od strony obiektów biznesowych, możemy wyszczególnić obiekt kalendarza spotkań, mechanizmu wyszukiwania oraz listy subskrybentów i użytkowników subskrybowanych. Procesy biznesowe, które z nich korzystają, realizują z ich udziałem część usług opisanych powyżej. Proces optymalizacji czasu użytkowników obejmuje dwa



Rysunek 3.3: Proponowana warstwa biznesowa.

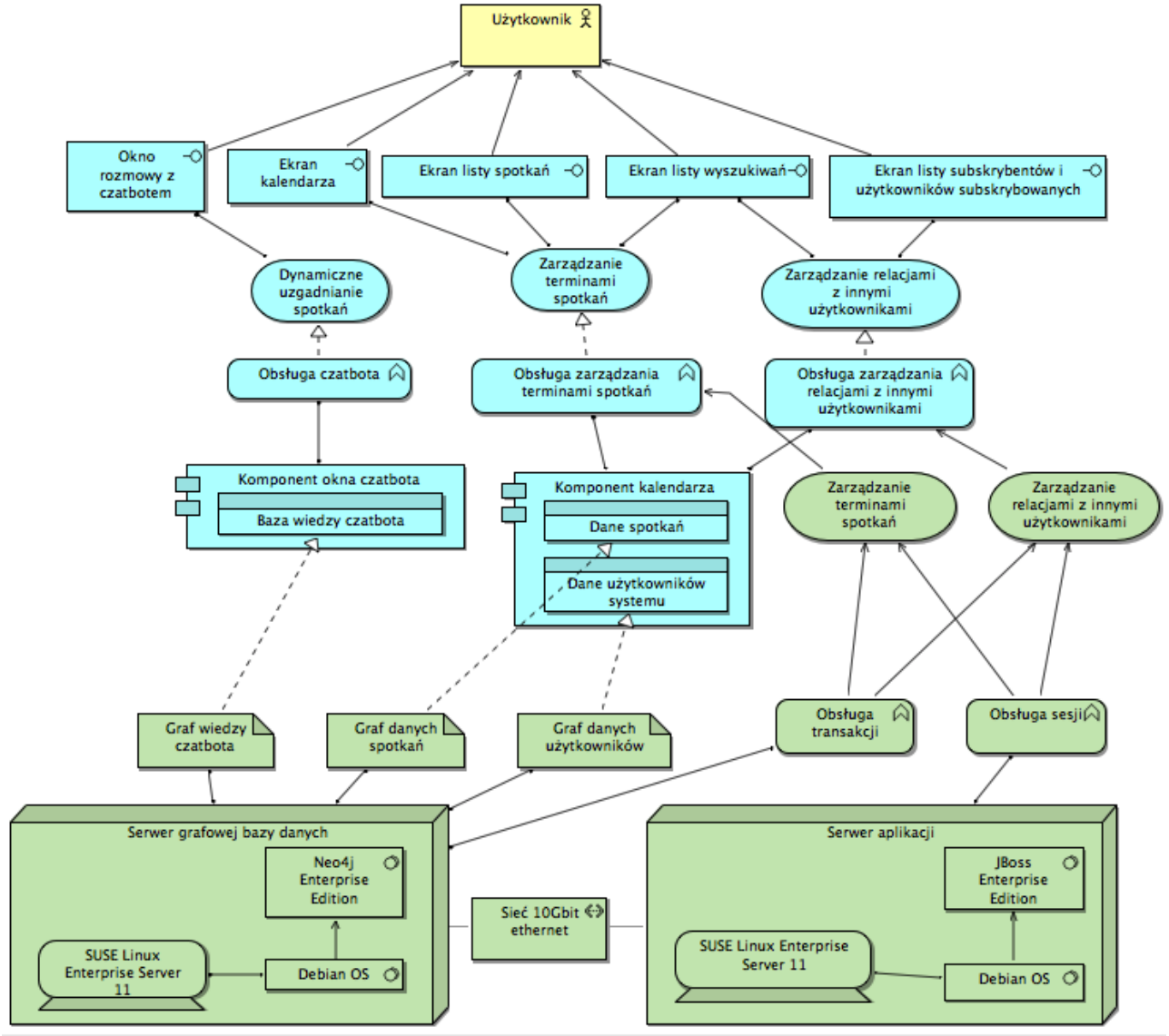
podprocesy: rozmowę z czatbotem oraz samodzielne wyszukiwanie dogodnych terminów spotkań. Realizację tego procesu znajdujemy w usługach dynamicznego zarządzania spotkaniami z użyciem czatbota oraz statycznego zarządzania przez użytkownika. Proces tworzenia relacji między użytkownikami jest realizowany przez usługę śledzenia aktywności innych użytkowników. Proces ten korzysta z obiektu biznesowego listy subskrybentów oraz użytkowników subskrybowanych.

Warstwa aplikacji jest zorientowana na jednego aktora biznesowego, którym jest użytkownik systemu. Aplikacja dostarcza interfejsy dedykowane użytkownikowi. Są nimi:

- okno rozmowy z czatbotem;
- ekran kalendarza;
- ekran listy spotkań;
- ekran listy wyszukiwań;
- ekran listy subskrybentów i użytkowników subskrybowanych.

Poszczególne interfejsy są powiązane z usługami. Wyszczególnione usługi to:

- obsługa czatbota - powiązana z oknem rozmowy z czatbotem;
- zarządzanie terminami spotkań - powiązana z ekranami kalendarza, listy spotkań oraz listy wyszukiwań;
- zarządzanie relacjami z innymi użytkownikami - powiązana z ekranem listy wyszukiwań oraz ekranem listy subskrybentów i użytkowników subskrybowanych.



Rysunek 3.4: Proponowana warstwa aplikacji połączona z warstwą infrastruktury.

Usługi znajdują realizację poprzez funkcje i skojarzone z nimi komponenty. Wyszczególnione funkcje to:

- obsługa czatbota - realizuje usługę dynamicznego uzgadniania spotkań;
- obsługa zarządzania terminami spotkań - realizuje usługę zarządzania terminami spotkań;
- obsługa zarządzania relacjami z innymi użytkownikami - realizuje usługę zarządzania relacjami z innymi użytkownikami.

Komponenty, które stanowią podstawę warstwy aplikacji, składają się z:

- komponentu obsługi czatbota. Bezpośrednio jest do niego także obiekt danych stanowiący bazę wiedzy czatbota;

- komponentu kalendarza. Ten komponent operuje na takich obiektach danych jak detale spotkań oraz dane użytkowników systemu.

Zaczynając analizę diagramu od warstwy infrastruktury, należy wziąć pod uwagę, że dwa główne węzły przedstawiają środowiska sprzętowe z oprogramowaniem, na którym działa system. Jeden z nich opisuje serwer bazy danych systemu. Proponowany system do zarządzania serwerami danych to dystrybucja systemu "Linux" o nazwie SUSE Linux (`https://www.suse.com/`). Element persystencji środowiska reprezentuje grafowa dystrybucja bazy danych. Ten element jest także swego rodzaju innowacją ze względu na odejście od tradycyjnego modelu danych opartego na tabelach relacyjnych. Baza danych, oprócz podstawowej funkcjonalności przechowywania danych systemu, jest także bazą wiedzy dla czatbota. Jako system operacyjny wybrano dystrybucję systemu "Linux" o nazwie Debian (`https://www.debian.org/index.html`). Węzeł bazy danych systemu skojarzony jest z artefaktami, które z kolei bezpośrednio łączą się z komponentami aplikacji, dostarczając im informacje przetworzone w struktury odpowiednie do dalszej pracy na nich. Dodatkowo z węzłem bazy danych jest skojarzona funkcja obsługi transakcji, której używają usługi infrastruktury: usługa zarządzania terminami spotkań oraz usługa zarządzania relacjami z innymi użytkownikami. Z technicznego punktu widzenia funkcja ta odpowiada za niedopuszczenie do powstania zakleszczenia (ang. *deadlock*) podczas odpowiedzi na prośbę dostępu do danych.

Drugi z węzłów opisuje serwer aplikacji. Do obsługi serwerów danych zaproponowany został system identyczny do tego w węźle bazy danych. Podobnie z systemem operacyjnym, który posiada, jest to także dystrybucja systemu "Linux" o nazwie Debian. Jako kontener aplikacji wybrałem rozwiązanie o nazwie JBoss ze względu na wsparcie dla wielu platform systemowych(`http://www.jboss.org/`). Istotne jest zadbanie o sprawną komunikację między węzłami, toteż proponowanym kanałem komunikacyjnym jest sieć typu Ethernet o przepustowości 10 gigabitów na sekundę. Węzeł serwera aplikacji jest skojarzony z funkcją obsługi sesji. Ta z kolei, podobnie do funkcji obsługi transakcji, odpowiada za przetrzymywanie sesji użytkownika działającego w systemie.

Elementami spinającymi warstwę aplikacji z warstwą infrastruktury są artefakty oraz interfejsy infrastruktury. Wyszczególnione artefakty danych to:

- graf wiedzy czatbota - dostarcza przetworzone dane komponentowi aplikacji o nazwie "Baza wiedzy czatbota";
- graf danych spotkań - dostarcza przetworzone dane komponentowi aplikacji o nazwie "Dane spotkań";
- graf wiedzy użytkowników - dostarcza przetworzone dane komponentowi aplikacji o nazwie "Dane użytkowników systemu".

Istotne z punktu widzenia infrastruktury interfejsy to:

- zarządzanie terminami spotkań - wykorzystuje obsługę transakcji oraz sesji. Jest wykorzystywany przy funkcji aplikacji obsługującej zarządzanie terminami spotkania;
- zarządzanie relacjami z innymi użytkownikami - wykorzystuje obsługę transakcji oraz sesji. Jest wykorzystywany przy funkcji aplikacji obsługującej zarządzanie relacjami z użytkownikami.

4. Specyfikacja systemu

Niniejszy rozdział prezentuje specyfikację techniczną systemu - przedstawia zasadę działania, zależności między podsystemami i ich zachowanie. Podczas studiów 2 stopnia uczęszczałem na przedmiot pt.: "Specyfikacja i systemy wspomagania projektowania oprogramowania" prowadzony przez Pana dra inż. Piotra Szweda. Dzięki temu przedmiotowi zyskałem wiedzę potrzebną do stworzenia specyfikacji wymagań oraz projektowania oprogramowania.

Specyfikacja projektu wyznacza techniczne aspekty projektu oraz stanowi formalną reprezentację prac, które muszą zostać wykonane. Opisuje także serię przypadków użycia, możliwych do urzeczywistnienia dzięki funkcjonalnościom zawartym w projekcie.

4.1. Ogólny opis systemu

Wizją jest stworzenie konkurencyjnego systemu do zarządzania spotkaniami. Odbiorcami są użytkownicy ze środowiska akademickiego. Wdrożenie tego typu systemu pozwoli na:

- przyspieszenie procesów związanych z zarządzaniem spotkaniami przez użytkownika;
- dokonywanie selekcji osób uprawnionych od odbycia spotkania na podstawie ich preferencji;
- wygodne zarządzanie relacjami z innymi użytkownikami;
- bardziej optymalne wykorzystanie czasu.

4.1.1 Cel systemu

Celem systemu jest możliwość uzgadniania terminów spotkań i w ten sposób unikanie nieporozumień oraz wydajne zarządzanie czasem jego użytkownikom. Użytkownik, z którym chcą się umówić inni użytkownicy, definiuje spotkania, określając w nich kryteria. Kryteria są ściśle powiązane z preferencjami użytkowników. Jeżeli termin nie istnieje lub nie jest możliwy do urzeczywistnienia spotkania, zostają podjęte kroki mające na celu doprowadzenie do spotkania lub odmowa zależnie od woli użytkownika, z którym chcą się umówić inne osoby. System wykorzystuje czatbota do prowadzenia konwersacji z użytkownikami w języku naturalnym. Zadaniem czatbota jest:

- odpowiadanie na pytania użytkownika w języku polskim;
- walidacja w postaci informacji w języku naturalnym czy petent może zostać przypisany do spotkania;
- generowanie odpowiedzi na pytania dotyczące terminów spotkań.

4.1.2 Udziałowcy i użytkownicy

- Użytkownik systemu - klient przedsiębiorstwa dystrybuującego system.

4.1.3 Podstawowe cele udziałowców i użytkowników

Użytkownik systemu:

- zarządzanie spotkaniami w formie statycznej poprzez wyszukiwanie i przypisywanie się do spotkań;
- zarządzanie spotkaniami w formie dynamicznej poprzez rozmowę z czatbotem w języku naturalnym;
- tworzenie i usuwanie spotkań;
- zarządzanie relacjami z innymi użytkownikami.

4.1.4 Granice systemu

Granice systemu otaczają przypadki użycia. W notacji UML granicami systemu są aktorzy. Wyróżniamy następujących aktorów:

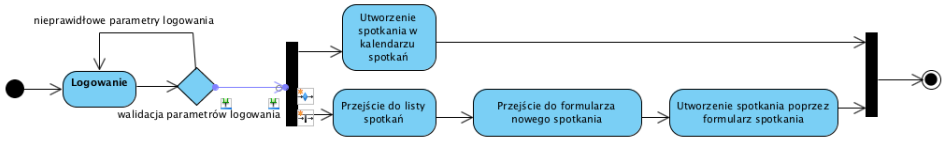
- użytkownik - klient przedsiębiorstwa;
- system.

4.1.5 Lista możliwości (funkcji systemu)

Lista możliwości prezentuje możliwe przejścia między stanami systemu w celu osiągnięcia pożądanego celu. Może być reprezentowana poprzez wylistowanie sekwencji konkretnych podczynności realizujących czynność nadrzędną lub za pomocą diagramu aktywności.

Definiowanie spotkań:

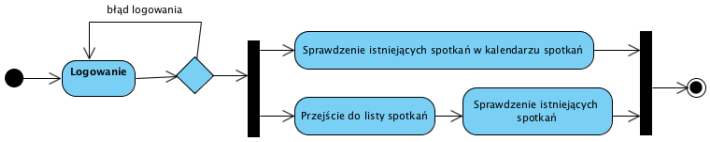
- użytkownik loguje się do systemu;
- opcja nr 1: użytkownik tworzy spotkanie w kalendarzu spotkań;
- opcja nr 2: użytkownik tworzy spotkanie poprzez formularz spotkania.



Rysunek 4.1: Definiowanie spotkań: diagram aktywności.

Sprawdzanie dostępności spotkań:

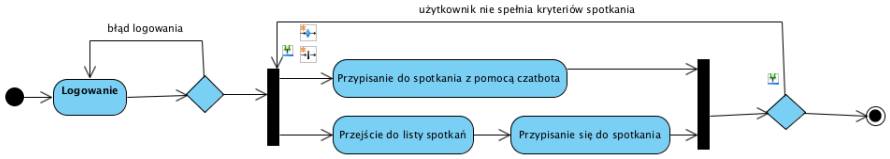
- użytkownik loguje się do systemu;
- opcja nr 1: użytkownik sprawdza spotkanie poprzez kalendarz spotkań;
- opcja nr 2: użytkownik sprawdza spotkanie na liście spotkań.



Rysunek 4.2: Sprawdzanie dostępności spotkań: diagram aktywności.

Przypisanie się do spotkania:

- użytkownik loguje się do systemu;
- opcja nr 1: użytkownik przypisuje się do spotkania na liście spotkań;
  - jeżeli użytkownik spełnia kryteria spotkania, zostaje przypisany;
  - jeżeli użytkownik nie spełnia kryteriów spotkania, nie zostaje przypisany.
- opcja nr 2: użytkownik przypisuje się do spotkania za pośrednictwem czatbota;
  - jeżeli użytkownik spełnia kryteria spotkania, zostaje przypisany;
  - jeżeli użytkownik nie spełnia kryteriów spotkań, nie zostaje przypisany.



Rysunek 4.3: Sprawdzanie dostępności spotkań: diagram aktywności.

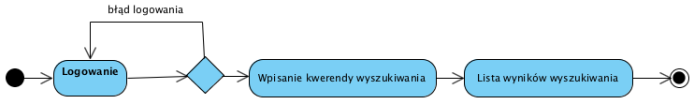
Wypisanie się ze spotkania:

- użytkownik loguje się do systemu;
- opcja nr 1: użytkownik wypisuje się ze spotkania na liście spotkań;
- opcja nr 2: użytkownik wypisuje się ze spotkania za pośrednictwem czatbota.

Rysunek 4.4: Wypisanie się ze spotkania: diagram aktywności.

Wyszukiwanie:

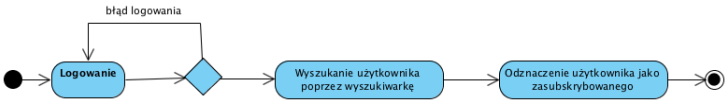
- użytkownik loguje się do systemu;
- użytkownik wyszukuje żądane wyniki poprzez okno wyszukiwarki.



Rysunek 4.5: Wyszukiwanie: diagram aktywności.

Zarządzanie relacjami z innymi użytkownikami

- użytkownik loguje się do systemu;
- użytkownik subskrybuje lub odznacza subskrypcję innego użytkownika po znalezieniu użytkowników w opcji "Szukaj".



Rysunek 4.6: Zarządzanie relacjami z innymi użytkownikami: diagram aktywności.

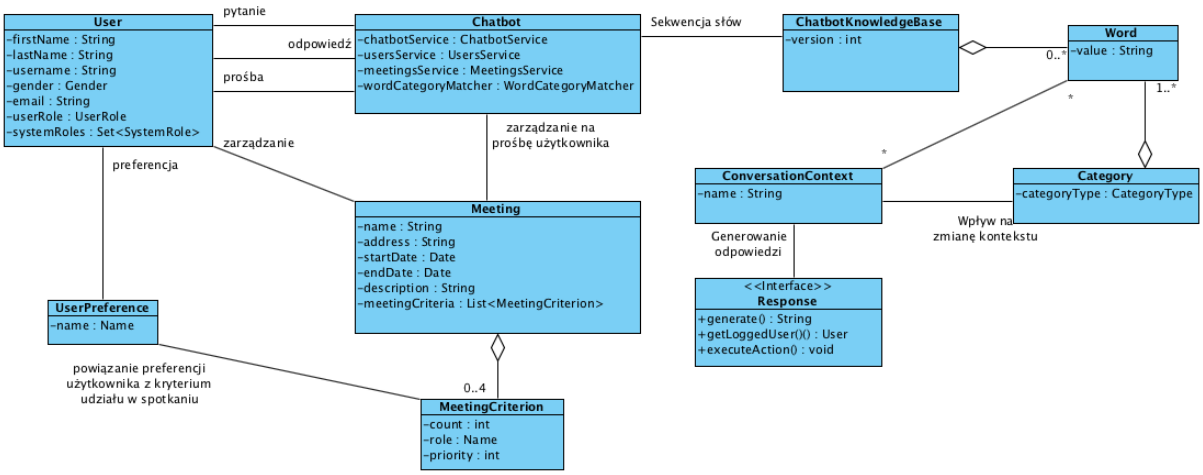
4.1.6 Analiza obiektów biznesowych (dziedziny)

Model dziedziny jest diagramem klas [7] należących do dziedziny biznesowej, która będzie realizowana przez przyszły system. Można powiedzieć, że jest to zbiór obiektów, którymi będzie operował przyszły użytkownik systemu. Model dziedziny nie zawiera klas realizujących logikę systemu lub schematów interfejsów użytkownika. Celem jego tworzenia jest dostarczenie prostego prototypu jego klientowi. Poniżej przedstawiona jest przykładowa analiza dziedziny opisywanego systemu.

Słownik pojęć:

- Użytkownik (kod: User) - klient przedsiębiorstwa dystrybuującego system; osoba zarządzająca swoimi spotkaniami;
- Czatbot (kod: Chatbot) - wirtualny pomocnik Użytkownika asystujący przy zarządzaniu spotkaniami;
- Baza wiedzy czatbota (kod: ChatbotKnowledgeBase) - graf sekwencji słów oraz kontekstów wypowiedzi dający podstawę do prawidłowej interpretacji zapytania użytkownika przez czatbota;
- Kontekst wypowiedzi (kod: ConversationContext) - interpretacja wpisanej wypowiedzi skojarzona z konkretną potrzebą;
- Subskrypcja (kod: Subscribe) - relacja dwóch użytkowników wynikająca ze wspólnych zainteresowań, potrzeb;
- Spotkanie (kod: Meeting)- element zawierający tematykę, czas trwania, kryteria oraz liczbę przypisanych osób z określonymi preferencjami mającymi na celu zaspokojenie wspólnych potrzeb czy dojście do konkretnych wniosków;

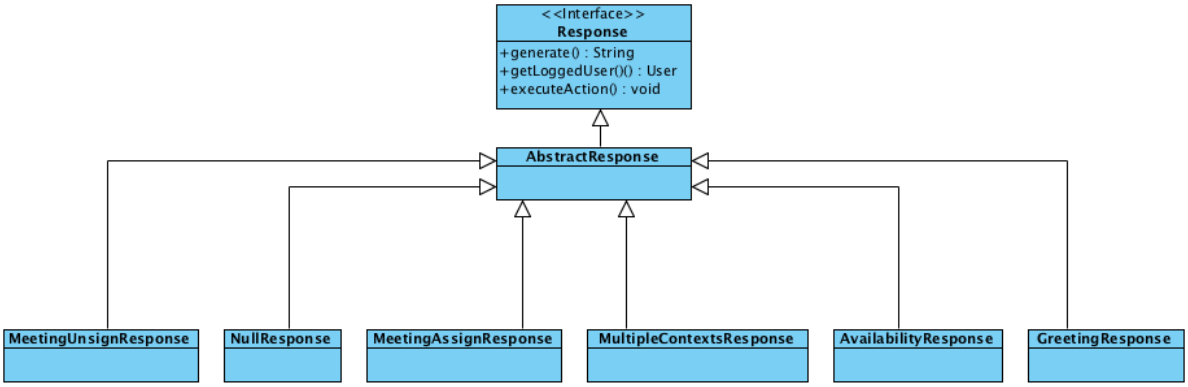
- Kryterium spotkania (kod: MeetingCriterion) - warunek, który musi być spełniony, by użytkownik mógł wziąć udział w spotkaniu;
- Kategoria słowna (kod: Category) - kategoria wpływająca na generowanie kontekstu wypowiedzi;
- Odpowiedź (kod: Response) - wynik interpretacji ciągu słów wygenerowany przez czatbota;
- Preferencja (kod: UserPreference) - pewna właściwość specyficzna dla jednego użytkownika lub grupy użytkowników.



Rysunek 4.7: Dziedzina systemu - ogólny zarys.

Typy odpowiedzi generowane przez czatbota:

- Przypisanie do spotkania (kod: MeetingAssignResponse) - odpowiedź, która generuje tekst mówiący o przypisaniu użytkownika do konkretnego spotkania;
- Wypisanie ze spotkania (kod: MeetingUnsignResponse) - odpowiedź, która generuje tekst mówiący o wypisaniu użytkownika z konkretnego spotkania;
- Odpowiedź na błędny tekst (kod: NullResponse) - odpowiedź wygenerowana, gdy czatbot nie zrozumiał kontekstu wypowiedzi;
- Wiele kontekstów zinterpretowanych (kod: MultipleContextsResponse) - odpowiedź generowana na skutek zrozumienia wypowiedzi w więcej niż w jednym kontekście wypowiedzi;
- Odpowiedź na pytanie o dostępność (kod: AvailabilityResponse) - odpowiedź na pytanie o dostępne spotkania konkretnego użytkownika;
- Pozdrowienie (kod: GreetingResponse) - odpowiedź na pozdrowienie przez użytkownika.



Rysunek 4.8: Dziedzina systemu - typy odpowiedzi czatbota.

4.1.7 Atrybuty klas obiektów biznesowych

Użytkownik (kod: User)

- firstName - imię użytkownika;
- lastName - nazwisko użytkownika;
- username - nazwa użytkownika;
- gender - płeć użytkownika;
- email - e-mail użytkownika;
- userPreference - preferencja użytkownika;
- systemRoles - role systemowe użytkownika.

Czatbot (kod: Chatbot)

- chatbotService - serwis danych czatbota;
- usersService - serwis danych użytkownika;
- meetingsService - serwis danych spotkań;
- wordCategoryMatcher - obiekt przyrównujący słowa do poszczególnych kategorii.

Baza wiedzy czatbota (kod: ChatbotKnowledgeBase)

- version - wersja bazy danych czatbota.

Słowo (kod: Word)

- value - wartość słowa.

Kategoria (kod: Category)

- categoryType - kategoria słowa.

Kontekst konwersacji (kod: ConversationContext)

- name - nazwa kontekstu konwersacji.

Spotkanie (kod: Meeting)

- name - nazwa spotkania;
- address - adres spotkania;
- startDate - data rozpoczęcia spotkania;
- endDate - data zakończenia spotkania;
- description - opis spotkania;
- meetingCriteria - kryteria spotkania.

Kryterium spotkania (kod: MeetingCriterion)

- count - kryterium liczebnościowe spotkania;
- preference - kryterium preferencji uczestnika spotkania;
- priority - kryterium priorytetu preferencji uczestnika spotkania.

Preferencja użytkownika (kod: UserPreference)

- name - nazwa preferencji użytkownika.

Subskrypcja (kod: Subscribe)

- id - identyfikator subskrypcji;
- follower - subskrybent;
- followedBy - użytkownik subskrybowany.

4.1.8 Specyfikacja wymagań oprogramowania

Specyfikacja wymagań definiuje przypadki użycia systemu, wprowadzając [2] gramatyczną formę opisu przypadku, w którym aktor osiąga pożądany rezultat lub przekazuje informacje innemu aktorowi. Wymagania mogą także przybrać formę diagramu przejść.

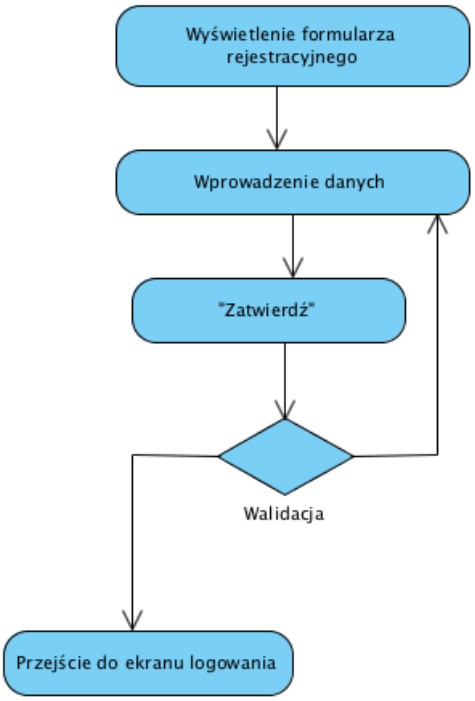
Definiując przypadek użycia, narasta sporo trudności z wyceną, czy jest on wart rozważania lub kiedy może zostać uznany za skończony. W związku z tym należy jasno określić:

- zakres, czyli jaka część systemu jest rozważana w przypadku;
- aktor, czyli kto bierze udział w przypadku;
- poziom przypadku, czyli czy aktor w systemie odgrywa dużą rolę.

Zdefiniowany przypadek użycia tworzy kontrakt pomiędzy określonym zachowaniem systemu a aktorem biorącym z nim udział.

Poniższa specyfikacja jest proponowaną dla opisywanego systemu. Specyfikację wymagań przedstawiam w postaci identycznej do tej, która zaprezentowana była podczas cyklu laboratoriów [16].

Proces zakładania konta użytkownika



Rysunek 4.9: Proces zakładania konta użytkownika.

Aktorzy:	Użytkownik, System.
Zakres:	Interfejs użytkownika dla zakładania konta.
Poziom:	Systemowy
Udziałowcy i ich cele:	Użytkownik chce utworzyć konto w systemie.
Zdarzenie wyzwalające (trigger) :	Użytkownik rejestruje się w formularzu tworzenia konta, wysyłając swoje dane.
Warunki wstępne:	Użytkownik posiada przewidziane przez system preferencje.
Warunki końcowe dla sukcesu:	Zostaje utworzone nowe konto użytkownika. Użytkownik zostaje przekierowany do ekranu logowania.
Warunki końcowe dla niepowodzenia:	W przypadku istnienia konta zarejestrowanego na wpisany e-mail pojawia się stosowna informacja. W przypadku błędnego wypełnienia formularza, zostaje wyświetlona informacja walidacyjna pod każdym błędnie wypełnionym polem.

Tabela 4.1: Specyfikacja wymagań dla procesu zakładania konta użytkownika.

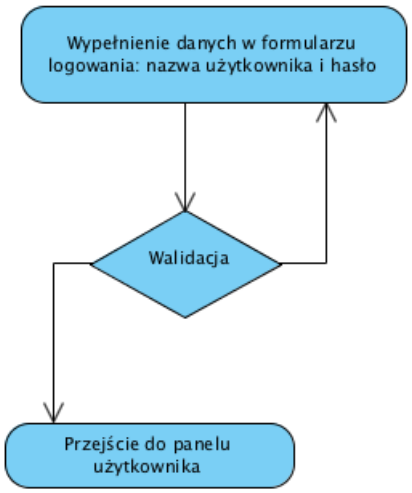
Scenariusz główny

1. System wyświetla formularz danych użytkownika.
2. Użytkownik wprowadza dane: imię, nazwisko, e-mail, płeć, nazwę użytkownika.
3. Użytkownik zatwierdza wprowadzone dane.
4. System waliduje wprowadzone dane.
5. System tworzy konto użytkownika.

Scenariusz alternatywny
Punkt 4 scenariusza głównego - walidacja prośby stworzenia konta:

- Użytkownik niepoprawnie wprowadził wymagane dane. System podświetla źle wypełnione pola. Powrót do 2 scenariusza głównego.
- Użytkownik wprowadził nazwę użytkownika, która jest już zarejestrowana w systemie. System wyświetla informację o istnieniu konta z wpisaną nazwą użytkownika. Powrót do punktu 2 scenariusza głównego.

Proces logowania się użytkownika do systemu



Rysunek 4.10: Proces logowania użytkownika.

Aktorzy:	Użytkownik, System.
Zakres:	Interfejs użytkownika dla logowania.
Poziom:	Systemowy
Udziałowcy i ich cele:	Użytkownik chce zalogować się do systemu.
Zdarzenie wyzwalające (trigger) :	Użytkownik wypełnia wymagane dane: nazwę użytkownika i hasło.
Warunki wstępne:	Użytkownik posiada konto w systemie.
Warunki końcowe dla sukcesu:	Użytkownik zostaje przekierowany do panelu użytkownika.
Warunki końcowe dla niepowodzenia:	W przypadku błędnych danych logowania system wyświetla stosowny komunikat.

Tabela 4.2: Specyfikacja wymagań dla procesu logowania użytkownika.

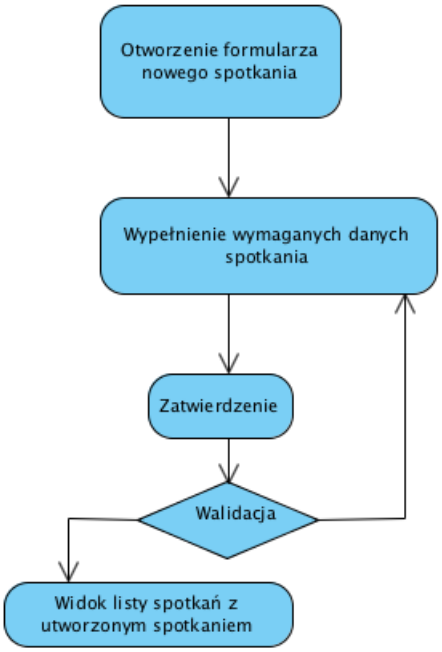
Scenariusz główny

1. System wyświetla formularz logowania użytkownika.
2. Użytkownik wprowadza dane: nazwę użytkownika i hasło.
3. Użytkownik zatwierdza wprowadzone dane.
4. System waliduje wprowadzone dane.
5. System loguje użytkownika, przenosząc go do panelu użytkownika.

Scenariusz alternatywny
Punkt 4 scenariusza głównego:

- Logowanie z nieprawidłowymi danymi powoduje ponowne wyświetlenie informacji o popełnionym błędzie. Scenariusz powraca do punktu 1 scenariusza głównego.

Proces statycznego tworzenia spotkania



Rysunek 4.11: Proces statycznego tworzenia spotkania.

Aktorzy:	Użytkownik, System.
Zakres:	Widok listy spotkań
Poziom:	Systemowy
Udziałowcy i ich cele:	Użytkownik chce przypisać się do spotkania, które go interesuje.
Zdarzenie wyzwalające (trigger):	Użytkownik zaznacza opcję umożliwiającą przypisanie do spotkania.
Warunki wstępne:	Użytkownik posiada odpowiednie preferencje do udziału w spotkaniu.
Warunki końcowe dla sukcesu:	Użytkownik zostaje przypisany do spotkania.
Warunki końcowe dla niepowodzenia:	System wyświetla informację o niemożności przypisania do spotkania.

Tabela 4.3: Specyfikacja wymagań dla procesu statycznego przypisywania się spotkania.

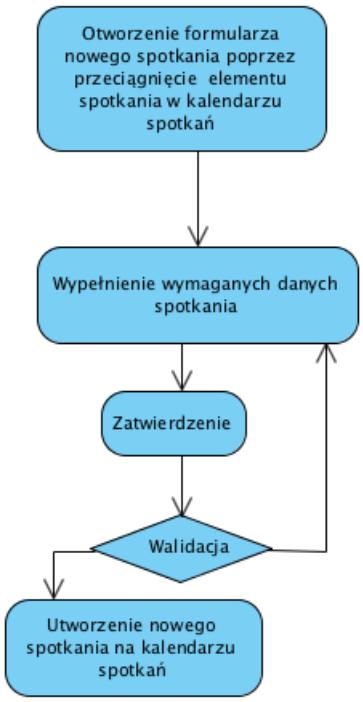
Scenariusz główny

1. Użytkownik przechodzi na listę spotkań.
2. Użytkownik przypisuje się do spotkania, zaznaczając odpowiednią do tego opcję.
3. System waliduje prośbę użytkownika.
4. Użytkownik zostaje przypisany do spotkania.

Scenariusz alternatywny
Punkt 3 scenariusza głównego:

- Użytkownik nie posiada wymaganych preferencji do udziału w spotkaniu.
- Liczba uczestników spotkania osiągnęła maksymalny limit.

Proces dynamicznego tworzenia spotkania



Rysunek 4.12: Proces statycznego tworzenia spotkania.

Aktorzy:	Użytkownik, System.
Zakres:	Widok kalendarza w systemie.
Poziom:	Systemowy
Udziałowcy i ich cele:	Użytkownik chce stworzyć spotkanie.
Zdarzenie wyzwalające (trigger) :	Użytkownik definiuje spotkanie w kalendarzu, następnie wpisuje informacje dotyczące spotkania.
Warunki wstępne:	Użytkownik ma konto w systemie.
Warunki końcowe dla sukcesu:	Użytkownik tworzy nowe spotkanie.
Warunki końcowe dla niepowodzenia:	System wyświetla informację wyjaśniającą, dlaczego spotkanie nie mogło zostać utworzone.

Tabela 4.4: Specyfikacja wymagań dla procesu dynamicznego przypisywania się spotkania.

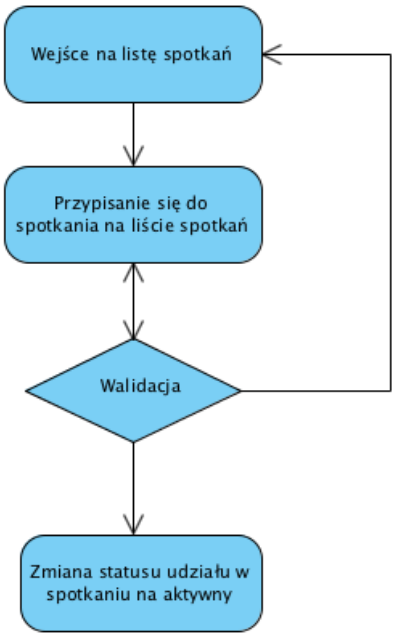
Scenariusz główny

1. Użytkownik przechodzi do widoku kalendarza spotkania.
2. Użytkownik definiuje spotkanie w kalendarzu oraz wpisuje wymagane informacje.
3. System waliduje zatwierdzone spotkanie.
4. Użytkownik tworzy spotkanie.

Scenariusz alternatywny
Punkt 3 scenariusza głównego.

- Użytkownik niepoprawnie wypełnił dane spotkania. System wyświetla informację o tym zdarzeniu. Następuje powrót do punktu 1 scenariusza głównego.
- Użytkownik organizuje spotkanie w czasie, gdy chce stworzyć kolejne spotkanie. System waliduje tę aktywność i informuje o tym zdarzeniu. Następuje powrót do punktu 1 scenariusza głównego.

Proces statycznego przypisywania się do spotkania



Rysunek 4.13: Proces statycznego tworzenia spotkania.

Aktorzy:	Użytkownik, System.
Zakres:	Widok listy spotkań.
Poziom:	Systemowy
Udziałowcy i ich cele:	Użytkownik chce przypisać się do spotkania.
Zdarzenie wyzwalające (trigger):	Użytkownik zaznacza opcję umożliwiającą przypisanie go do spotkania.
Warunki wstępne:	Użytkownik posiada konto w systemie.
Warunki końcowe dla sukcesu:	Użytkownik zostaje przypisany do spotkania.
Warunki końcowe dla niepowodzenia:	System wyświetla informację z błędem informującym dlaczego nie może zostać przypisany do spotkania.

Tabela 4.5: Specyfikacja wymagań dla procesu statycznego przypisywania się spotkania.

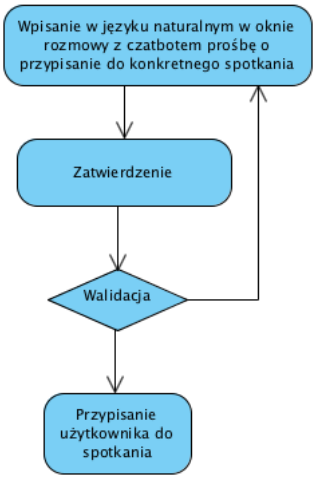
Scenariusz główny

1. Użytkownik przechodzi do listy spotkań.
2. Użytkownik zaznacza opcję umożliwiającą przypisanie do spotkania.
3. System waliduje zaznaczenie opcji przypisania do spotkania.
4. Użytkownik zostaje przypisany do spotkania.

Scenariusz alternatywny
Punkt 3 scenariusza głównego:

- Użytkownik nie ma wystarczających uprawnień do wzięcia udziału w spotkaniu. System waliduje tę prośbę informacją o przyczynie, dlaczego użytkownik nie może zostać przypisany do spotkania. Następuje powrót do punktu 1 scenariusza głównego.

Proces dynamicznego przypisywania się do spotkania



Rysunek 4.14: Proces dynamicznego przypisywania się do spotkania.

Aktorzy:	Użytkownik, System.
Zakres:	Widok okna rozmowy z czatbotem.
Poziom:	Systemowy
Udziałowcy i ich cele:	Użytkownik chce przypisać się do spotkania.
Zdarzenie wyzwalające (trigger):	W oknie rozmowy z czatbotem użytkownik wpisuje prośbę o wpisanie go na listę uczestników.
Warunki wstępne:	Użytkownik posiada konto w systemie.
Warunki końcowe dla sukcesu:	Czatbot wyświetla informację o przypisaniu użytkownika do spotkania.
Warunki końcowe dla niepowodzenia:	Czatbot wyświetla informację o przyczynie błędu podczas przypisywania do spotkania. Użytkownik nie zostaje przypisany do spotkania.

Tabela 4.6: Specyfikacja wymagań dla procesu dynamicznego przypisywania się do spotkania.

Scenariusz główny

1. Użytkownik przechodzi do widoku okna rozmowy z czatbotem.
2. Użytkownik wpisuje w języku naturalnym prośbę o przypisanie do spotkania.
3. Czatbot interpretuje wpisaną prośbę.
4. Czatbot odpowiada użytkownikowi, potwierdzając przypisanie do spotkania.

Scenariusz alternatywny
Punkt 3 scenariusza głównego.

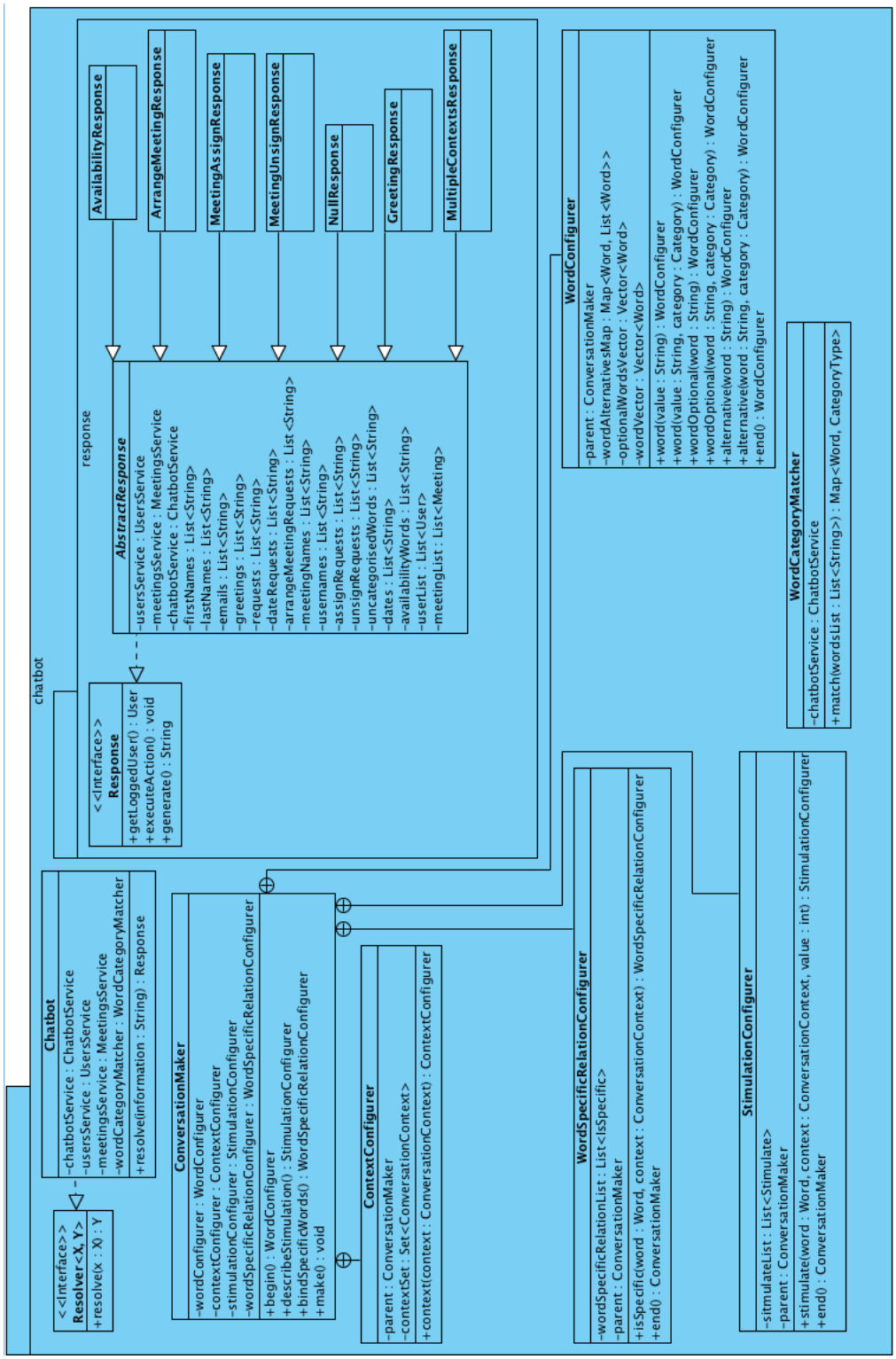
- W przypadku błędnej interpretacji czatbot powiadamia użytkownika o próbie sformułowania prośby innymi słowami. Powrót do punktu 1 scenariusza głównego.
- W przypadku poprawnie zinterpretowanej prośby, jeżeli użytkownik nie posiada odpowiednich preferencji do uczestniczenia w spotkaniu, czatbot odpowiada stosowną informacją. Powrót do punktu 1 scenariusza głównego.

4.2. Architektura systemu

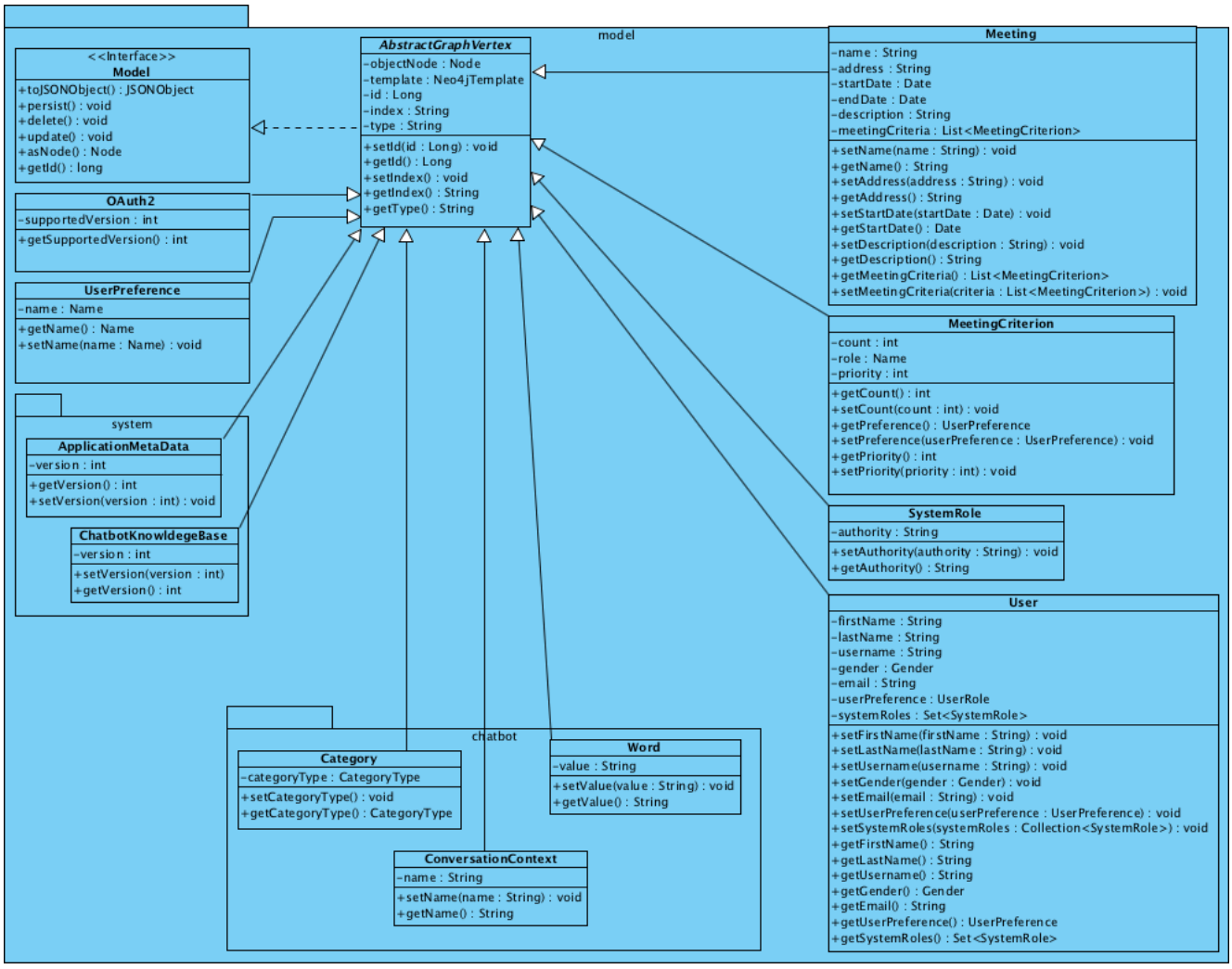
Architektura systemu jest jest oparta na wzorcu architektonicznym "Model-Widok-Kontroler" (ang. *Model-View-Controller*, w skrócie MVC). Architektura "Model-Widok-Kontroler" [20] definiuje obiekty trzech rodzajów. Model jest obiektem aplikacji, Widok obejmuje reprezentację widoczną na ekranie, natomiast Kontroler określa zachowanie interfejsu na działanie użytkownika. Domeną MVC jest rozdzielenie widoków i modeli oraz ustanowienie subskrypcji oraz mechanizmu powiadomień służącego do komunikowania się między nimi. Zmiana w modelu determinuje powiadomienie widoku, który odpowiednio się do niej dostosowuje.

4.2.1 Klasy w systemie

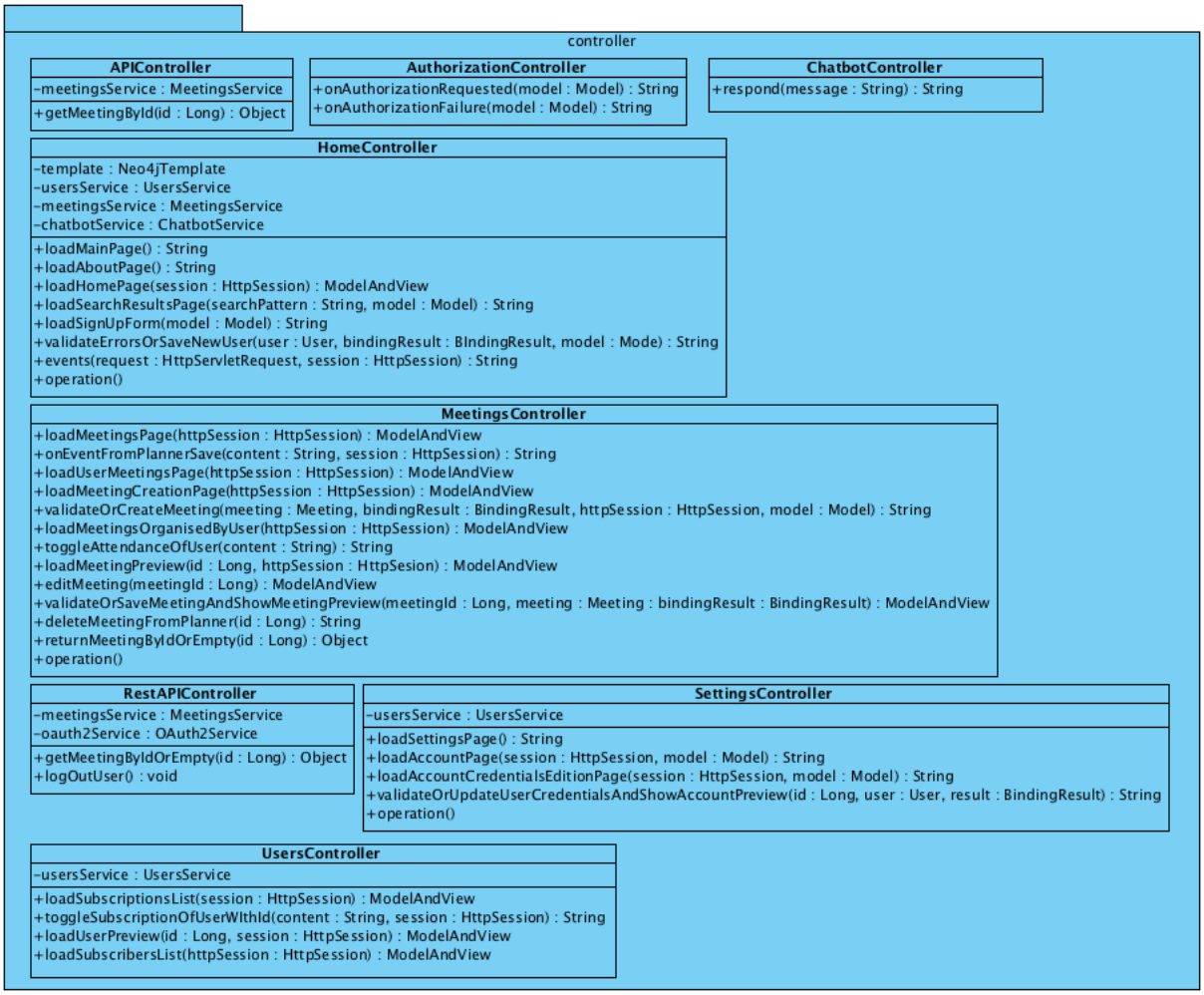
System został stworzony w języku obiektowym. W niniejszym rozdziale prezentuję najważniejsze pakiety klas realizujące pożądane działanie systemu. Jest to pakiet klas czatbota, modeli i serwisów bazodanowych oraz kontrolerów.



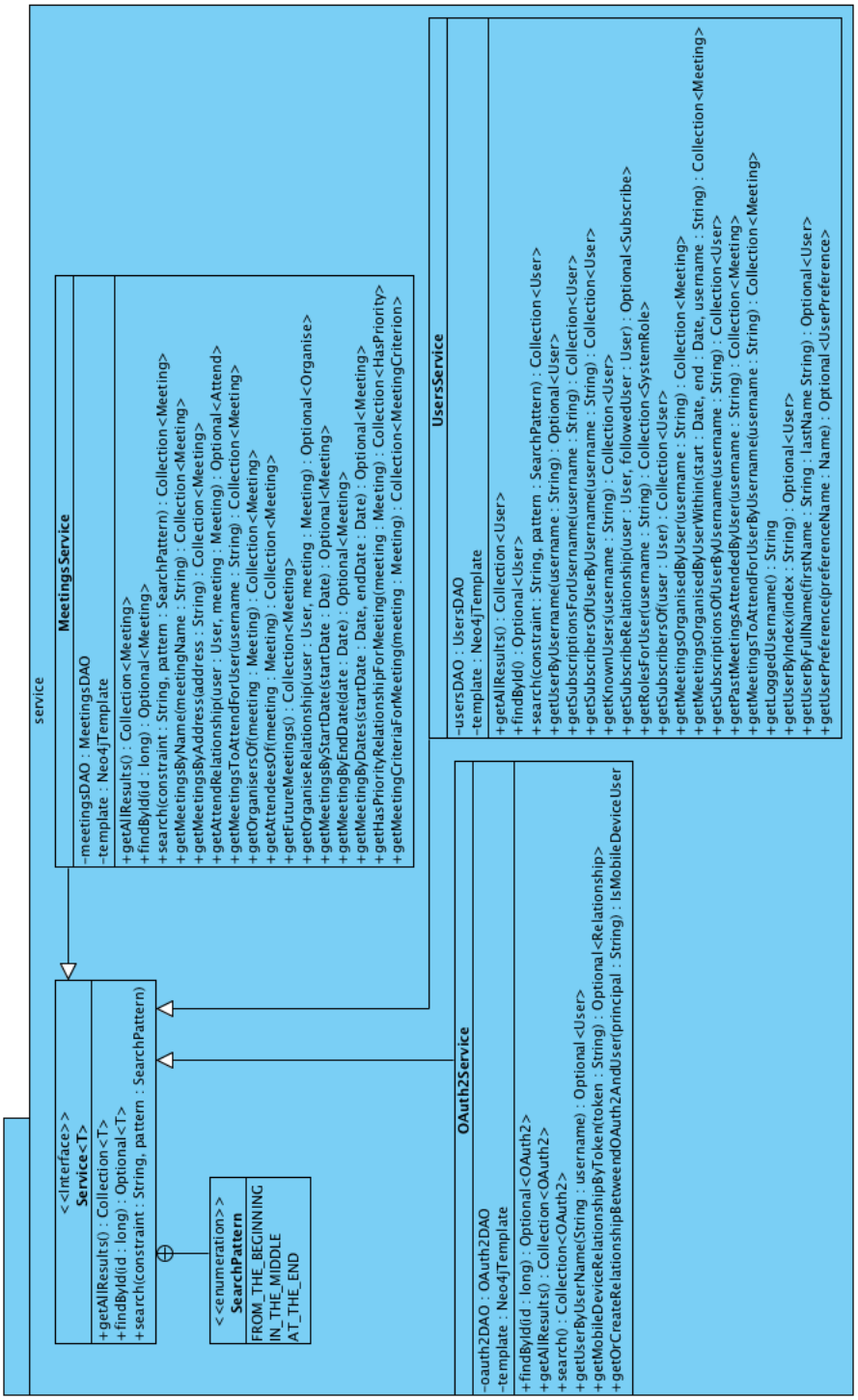
Rysunek 4.15: Pakiet klas składających się na funkcjonalność czatbota w systemie.



Rysunek 4.16: Pakiet klas opisujących modele (wierzchołki grafowej bazy danych) wykorzystywane w systemie.



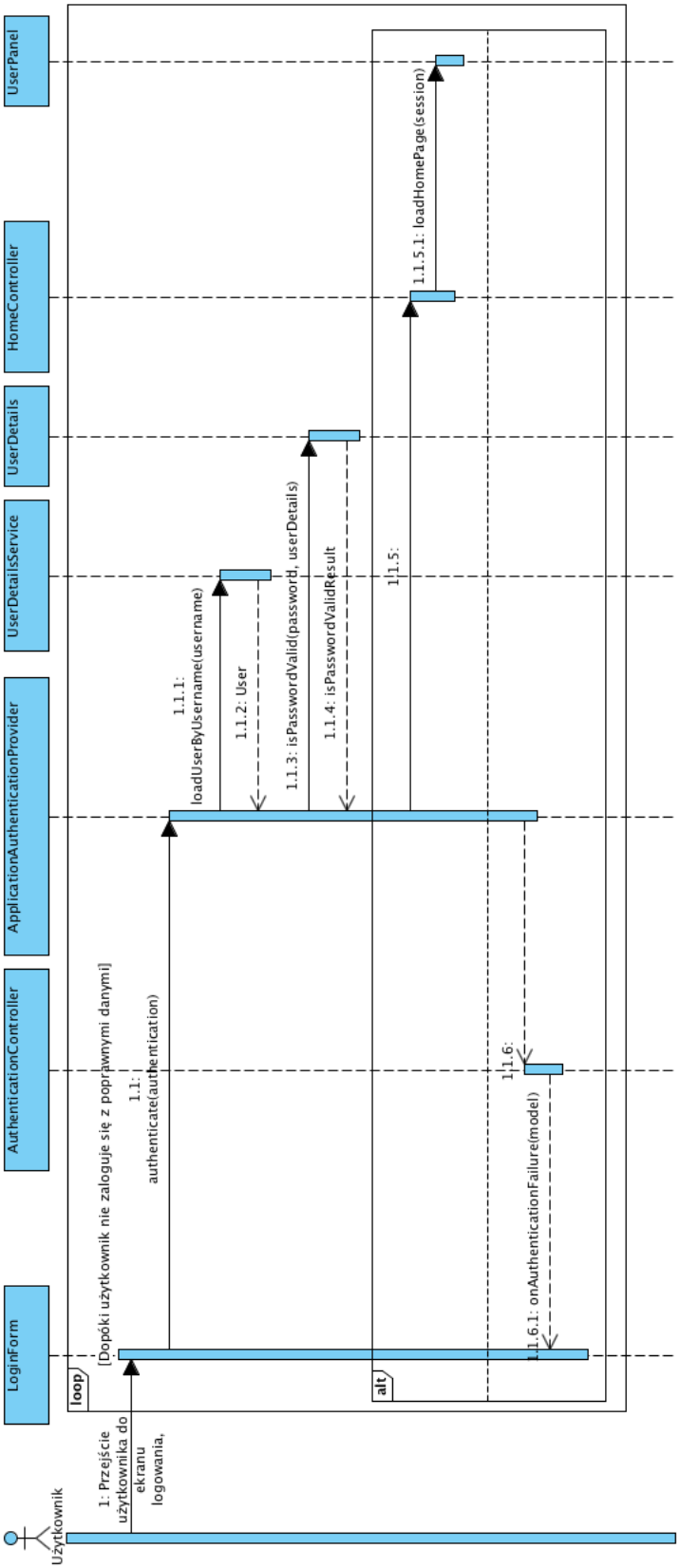
Rysunek 4.17: Pakiet klas opisujących kontrolery obsługujące interakcje użytkownika z systemem.



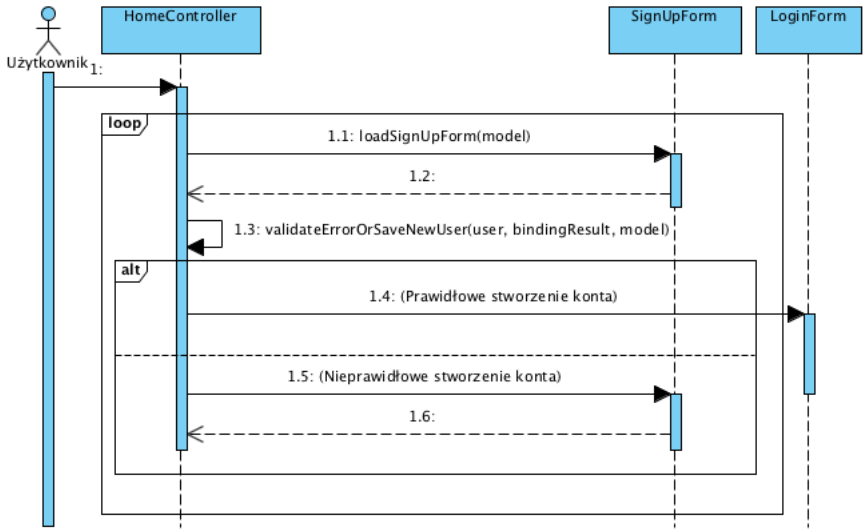
Rysunek 4.18: Pakiet klas opisujących serwisy bazodanowe dostarczające dane.

4.2.2 Interakcje pomiędzy klasami w systemie

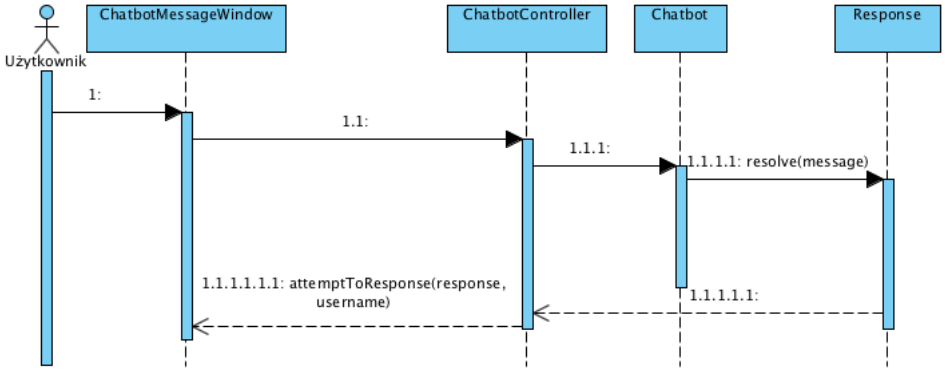
Interakcje pomiędzy obiektami są przedstawione w niniejszym rozdziale jako diagramy sekwencji procesów opisujących działania poszczególnych komponentów systemu.



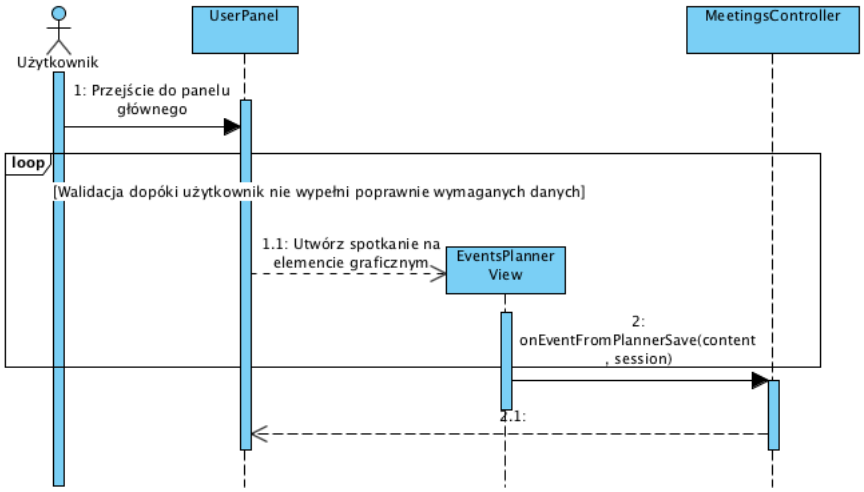
Rysunek 4.19: Diagram sekwencji przedstawiający proces logowania użytkownika.



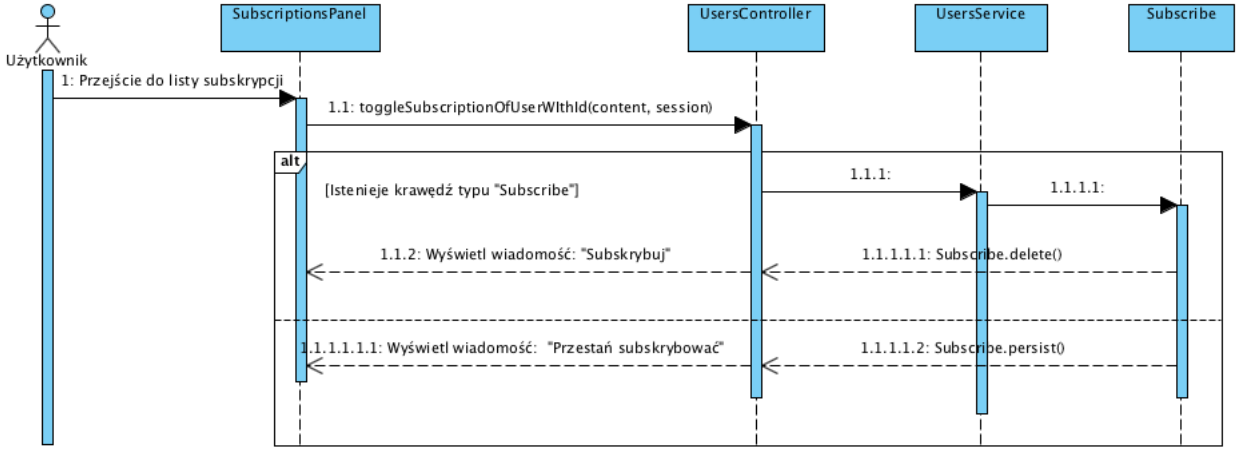
Rysunek 4.20: Diagram sekwencji przedstawiający proces tworzenia nowego konta użytkownika.



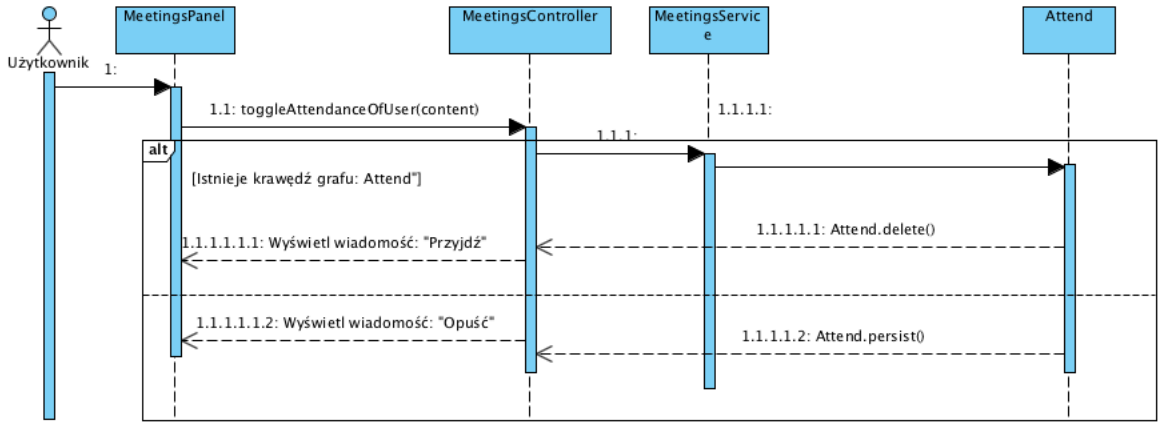
Rysunek 4.21: Diagram sekwencji przedstawiający proces rozmowy z czatbotem.



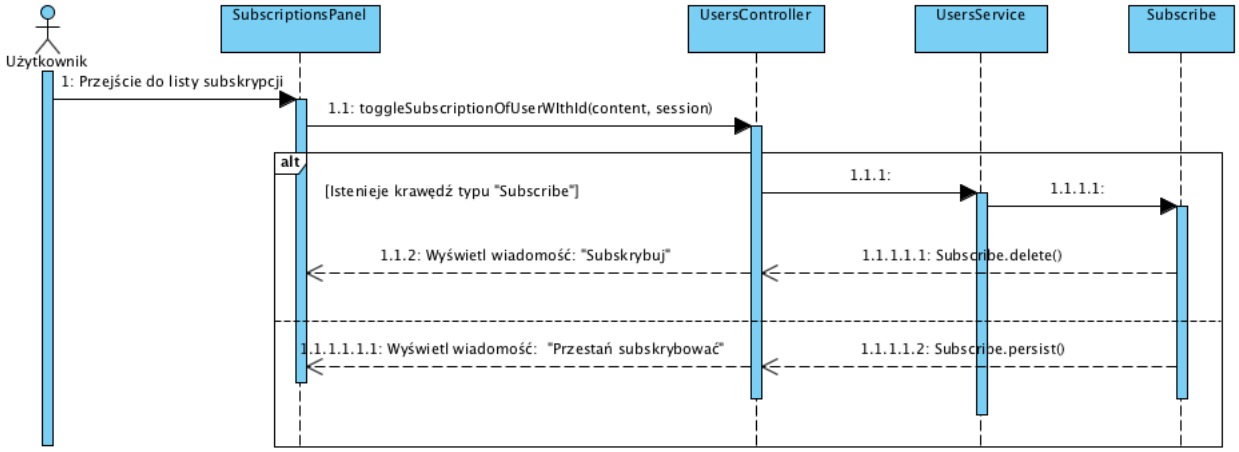
Rysunek 4.22: Diagram sekwencji przedstawiający "dynamiczny" proces tworzenia spotkania z poziomu kalendarza spotkań.



Rysunek 4.23: Diagram sekwencji przedstawiający "statyczny" proces tworzenia spotkania z poziomu formularza spotkania.



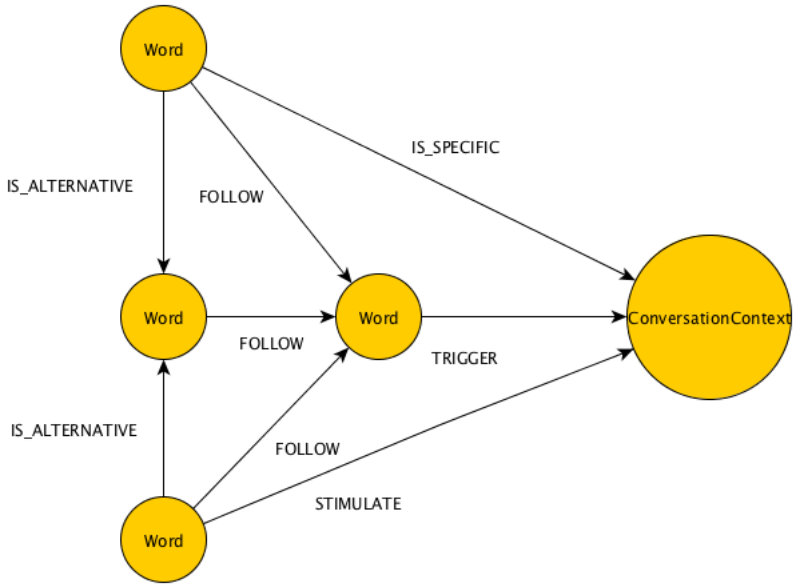
Rysunek 4.24: Diagram sekwencji przedstawiający proces "statycznego" zapisywania lub wypisywania się użytkownika ze spotkania.



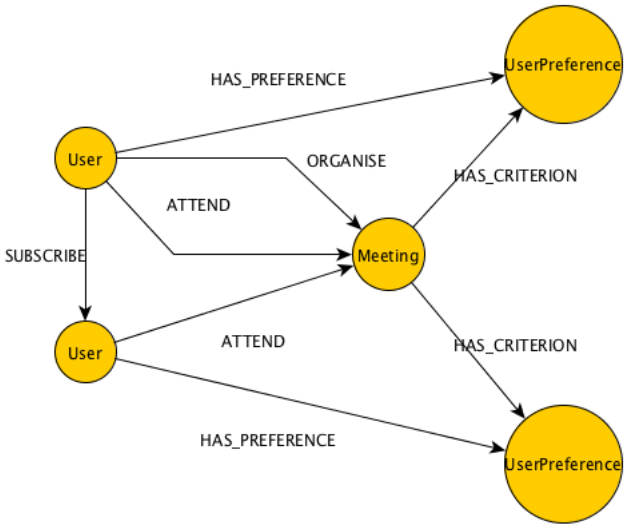
Rysunek 4.25: Diagram sekwencji przedstawiający proces subskrypcji lub desubskrypcji przez użytkownika.

4.2.3 Projekt bazy danych

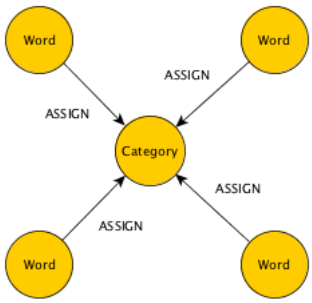
Dane w systemie są przechowywane w grafowej bazie danych. Wybór tego typu bazy znacząco ułatwił implementację mechanizmu czatbota polegającą na interpretacji prośby lub pytania użytkownika oraz wygenerowania sensownej odpowiedzi. Baza wiedzy czatbota obsługująca interakcje czatbota z użytkownikiem została zaprojektowana w taki sposób, by częściowo odwzorować zasadę działania ludzkiego mózgu przy generowaniu odpowiedzi na postawioną wiadomość.



Rysunek 4.26: Graf przedstawiający bazę wiedzy czatbota.



Rysunek 4.27: Graf przedstawiający relacje między użytkownikami oraz ich preferencjami w połączeniu ze spotkaniami, którym ich preferencje odpowiadają.



Rysunek 4.28: Graf przedstawiający relacje pomiędzy bazą słów i kategorii, do których są przypisane.

Wyjaśnienia typów krawędzi pomiędzy wierzchołkami grafu:

- *IS_ALTERNATIVE* - krawędź pomiędzy wierzchołkami grafu typu *Word*. Opisuje relację, w której dane słowo może być użyte alternatywnie do danego w określonej wypowiedzi.
- *FOLLOW* - krawędź pomiędzy wierzchołkami grafu typu *Word*. Opisuje relację, w której pewne słowo następuje w sekwencji po innym słowie.
- *IS_SPECIFIC* - krawędź pomiędzy wierzchołkami grafu typu *Word* i *ConversationContext*. Opisuje relację, w której pewne słowo występuje często w danym kontekście wypowiedzi.
- *TRIGGER* - krawędź pomiędzy wierzchołkami grafu typu *Word* i *ConversationContext*. Opisuje relację, w której słowo wyzwala określony kontekst wypowiedzi.
- *STIMULATE* - krawędź pomiędzy wierzchołkami grafu typu *Word* i *ConversationContext*. Opisuje relację, w której słowo stymuluje pobudzenie konkretnego kontekstu wypowiedzi.
- *SUBSCRIBE* - krawędź pomiędzy wierzchołkami grafu typu *User*. Opisuje relację, w której jeden użytkownik subskrybuje aktywności innego użytkownika.
- *HAS_PREFERENCE* - krawędź pomiędzy wierzchołkami grafu typu *User* i *UserPreference*. Opisuje relację, w której użytkownik posiada określoną preferencję zdefiniowaną w systemie.
- *HAS_CRITERION* - krawędź pomiędzy wierzchołkami grafu typu *Meeting* i *UserPreference*. Opisuje relację, w której spotkanie wymaga posiadania przez użytkownika określonej preferencji. Ta preferencja jest kryterium udziału w spotkaniu.
- *ORGANISE* - krawędź pomiędzy wierzchołkami grafu typu *User* i *Meeting*. Opisuje relację, w której użytkownik organizuje spotkanie.
- *ATTEND* - krawędź pomiędzy wierzchołkami grafu typu *User* i *Meeting*. Opisuje relację, w której użytkownik bierze udział w spotkaniu.
- *ASSIGN* - krawędź pomiędzy wierzchołkami grafu typu *Word* i *Category*. Opisuje relację, w której słowo jest przypisane do zdefiniowanej kategorii.

4.2.4 Baza wiedzy czatbota

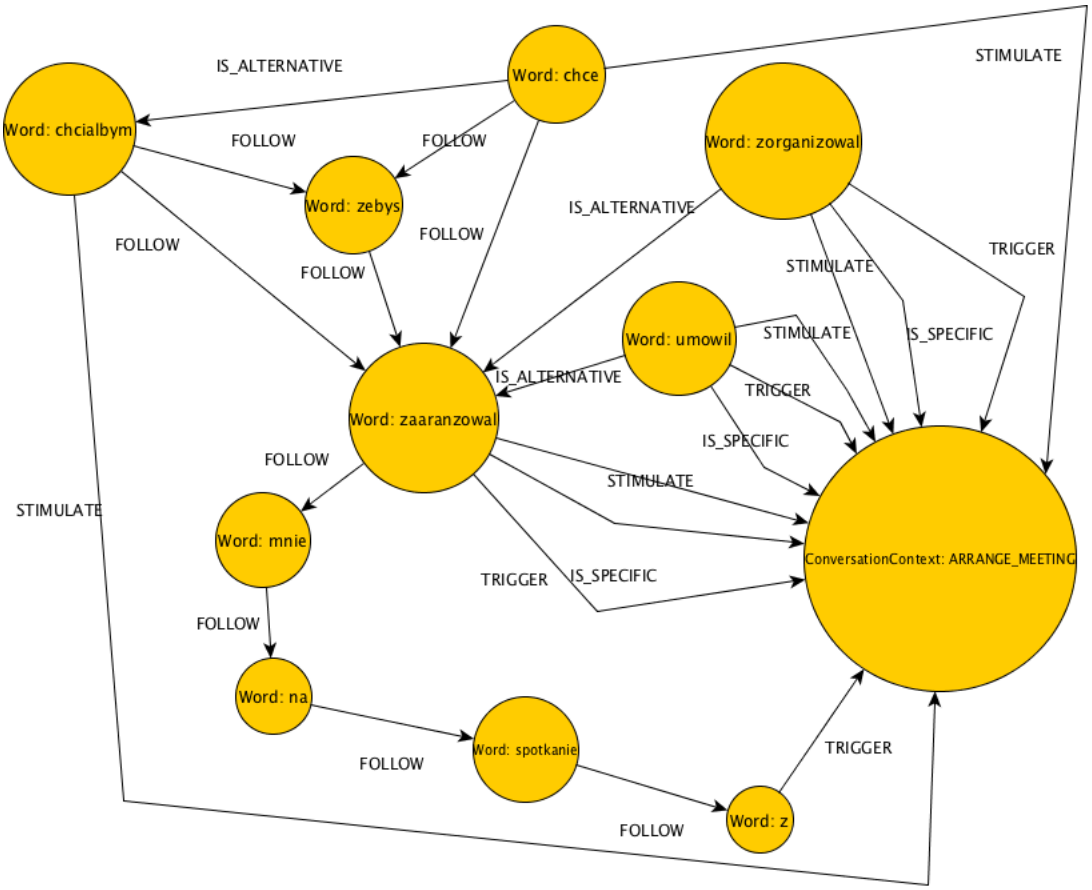
Czatbot posiada bazę słów połączonych różnymi relacjami pomiędzy sobą według schematów wyżej pokazanych. Nie mniej jednak, baza danych posiada pewne ograniczenia ze względu na swój rozmiar. Innymi słowy, mogą zdarzyć się sekwencje słów, które zostaną błędnie zinterpretowane przez czatbota lub zostaną odebrane jako wiadomości z niewiadomym kontekstem. Aczkolwiek, mechanizm poszerzania bazy danych został przewidziany i zaimplementowany. Całość odbywa się w klasie "ConversationMaker". Poniżej przedstawiony jest przykład użycia klasy "ConversationMaker":

```
ConversationContext context = new ConversationContext(ConversationContext.ARRANGE_MEETING);

new ConversationMaker(context)
        .begin()
        .wordOptional("chcialbym")
        .alternative("chce")
        .wordOptional("zebys")
        .word("zaaranzowal")
        .alternative("umowil")
        .alternative("zorganizowal")
        .wordOptional("mnie")
        .wordOptional("na")
        .wordOptional("spotkanie")
        .wordOptional("z")
        .end()
        .describeStimulation()
        .stimulate(new Word("chcialbym"), context, STIMULATION_VALUE_LOW)
        .stimulate(new Word("chce"), context, STIMULATION_VALUE_LOW)
        .stimulate(new Word("zaaranzowal"), context, STIMULATION_VALUE_HIGH)
        .stimulate(new Word("zorganizowal"), context, STIMULATION_VALUE_HIGH)
        .stimulate(new Word("umowil"), context, STIMULATION_VALUE_HIGH)
        .end()
        .bindSpecificWords()
        .isSpecific(new Word("zaaranzowal"), context)
        .isSpecific(new Word("zorganizowal"), context)
        .isSpecific(new Word("umowil"), context)
        .end()
        .make();
```

Rysunek 4.29: Procedura tworzenia scenariusza prośby użytkownika o zaaranżowanie spotkania przez czatbota.

Powyższa procedura przedstawia tworzenie sekwencji słów, która mapowana jest na kontekst, według którego czatbot ma zaaranżować spotkanie między użytkownikami. Kod generuje podgraf, który jest brany pod uwagę przy tworzeniu odpowiedzi czatbota na prośbę lub pytanie użytkownika. Podgraf wygląda w następujący sposób:



Rysunek 4.30: Wygenerowany podgraf wypowiedzi na procedurę tworzenia scenariusza prośby użytkownika o zaaranżowanie spotkania przez czatbota (4.29).

System w obecnej wersji nie posiada funkcjonalności, dzięki której czatbot mógłby się uczyć rozpoznawać kontekst wypowiedzi. Aczkolwiek, dzięki obecnie zaimplementowanym mechanizmom większość aspektów, które umożliwiałyby naukę czatbota jest obsłużona. Stosunkowo prostą funkcjonalnością mogłoby być wersjonowanie bazy wiedzy czatbota połączone z logowaniem nierozpoznanych pytań lub próśb użytkowników. Obsłużenie tych wypowiedzi mogłoby być umożliwiane w kolejnych dystrybucjach.

Wszystkie rysunki zostały wykonane z pomocą darmowego oprogramownia *Visual Paradigm for UML Community Edition 11.0* ® [13] oraz *yEd Graph Editor* ® [14].

5. Implementacja

Ostatni rozdział opisywanego systemu dotyczy praktycznej części polegającej implementacji systemu. Opisuję w nim techniczne aspekty napotkanych problemów i scenariusze testowe systemu.

5.1. Czatbot

Czatbot stanowi istotny element systemu. W niniejszym podrozdziale opisuję zaimplementowaną mechanikę działania wirtualnego rozmówcy oraz przedstawiam przykłady jego działania.

Implementacja inteligencji czatbota opiera się na wspomnianej w poprzednim rozdziale grafowej bazie wiedzy. Wirtualny rozmówca nie jest w stanie porozmawiać z użytkownikiem na każdy temat, jednak w kwestii doradzenia, uzgodnienia, czy zaaranżowania spotkania służy swoją inteligencją i potrafi prawidłowo zareagować na jego potrzeby. Czatbot w obecnej postaci nie zapamiętuje kolejnych wypowiedzi użytkownika. Analizuje wiadomość pod kątem doboru kontekstu wypowiedzi i stara się wykonać tyle pracy za swojego rozmówcę, ile jest to możliwe i sensowne.

5.1.1 Zaimplementowane algorytmy wykorzystywane przez czatbota

Algorytm 1 Procedura generowania odpowiedzi przez czatbota na postawione pytanie.

```
1: procedure RESOLVE
2:     message ← normalize(userMessage)
3:     wordsStringList ← splitIntoNormalizedWords(message)
4:     wordsCategoryMap ← matchWordsToCategories(wordsStringList)
5:     try:
6:     conversationContext ← searchConversationContext(wordsCategoryMap)
7:     return generateAnswer(conversationContext, wordsCategoryMap, message)
8:     catch(MultipleContextsRecognizedException):
9:     return generateQuestionAsAResultOfMultipleContextsRecognized(recognizedContexts, wordsCategoryMap, message).
```

Wyjaśnienie:
Powyższy algorytm realizuje generowanie odpowiedzi przez czatbota. W pierwszym kroku tekst jest normalizowany. Normalizacja polega na usunięciu polskich znaków. Znaki białe są zastępowane znacznikami. W kolejnym kroku ciąg znormalizowanych słów jest transformowany do listy słów. Transformacja polega na pocięciu tekstu według znaczników wprowadzonych w miejsce białych znaków podczas normalizacji. Następnie, z listy znormalizowanych słów generowana jest struktura klucz-wartość, gdzie kluczami są znormalizowane słowa, natomiast wartościami kategorie, do których poszczególne słowa są przypi-

sane. Na podstawie wygenerowanej struktury klucz-wartość wyszukuje odpowiedni kontekst wypowiedzi i zwraca odpowiednio dobraną odpowiedź.

Algorytm 2 Algorytm rozpoznawania kontekstu wypowiedzi przez czatbota.

```
procedure searchConversationContext
    categoryTypeList ← getValues(wordCategoryMap)
    if isGreeting(categoryTypeList) then
        return newConversationContext(GREETING)
    if isUnrecognizedWordsFoundOnly(categoryTypeList) then
        return newConversationContext(UNKNOWN_QUESTION)
     stimulatedContextMap ← newStimulatedContextMap
    for entry : wordCategoryMap.entrySet() do
        word ← entry.getKey()
        pathSet ← traverseGraphFromWordEndingWithTriggerRelationship(word)
        for path : pathSet do
            contextNode ← path.endNode()
            conversationContext ← newConversationContext(contextNode.getName())
            if ! stimulatedContextMap.containsKey(conversationContext) then
                stimulatedContextMap.put(conversationContext, 0.0)
            nodeList ← toList(path.nodes()) nodeList.remove(nodeList.size() -1);
            for node : nodeList do
                pathWord ← newWord(node.getName())
                stimulusOptional ← findStimulus(pathWord, conversationContext)
                if stimulusOptional.isPresent() then
                    stimulatedContextMap.put(
                        conversationContext,
                        stimulatedContextMap.get(conversationContext) +
                        stimulusOptional.get().getStimulationValue())
                if isSpecificForConversationContext(word, conversationContext) then
                        stimulatedContextMap.put(conversationContext,
                        stimulatedContextMap.get(conversationContext) * 1.5)
        sortedContextMap ← sortByValuesInAscOrder(stimulatedContextMap)
    throwIfMultipleContextsRecognized(sortedContextMap)
    conversationContextList ← toList(sortedContextMap.keySet())
    if conversationContextList.isEmpty() then
        return null
    else
        return conversationContextList.get(0)
```

Wyjaśnienie:
Powyższy algorytm dopasowuje kontekst wypowiedzi na podstawie otrzymanej struktury klucz-wartość gdzie kluczami są słowa, natomiast wartościami ich kategorie. W pierwszym kroku następuje sprawdzenie warunku, czy wiadomość użytkownika jest pozdrowieniem. Jeżeli warunek jest prawdziwy, następuje zwrócenie kontekstu wypowiedzi: pozdrowienie. W kolejnym kroku następuje sprawdzenie warunku, czy wartości (kategorie słów) otrzymanej struktury są nieznane. Jeżeli warunek jest prawdziwy, następuje zwrócenie kontekstu wypowiedzi: niepoprawnie sformułowana wypowiedź. Następnie dla wszystkich zna-

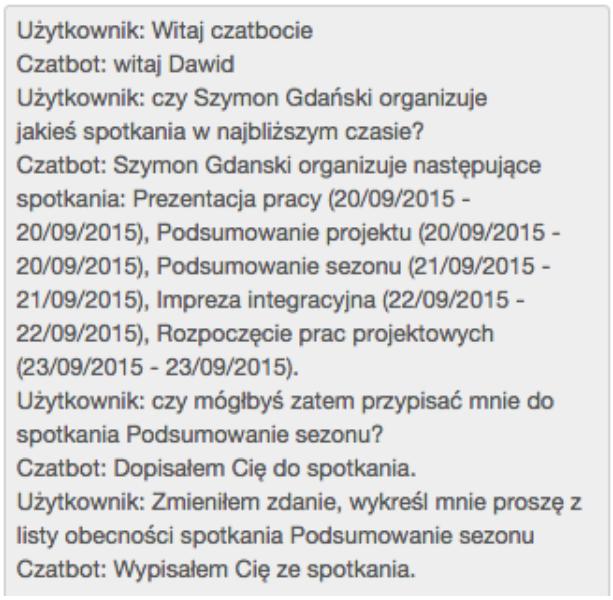
lezionych słów wyszukuję ciąg słów (wierzchołków typu *Word*) połączonych krawędziami typu *FOLLOW*, w którym ostatni wierzchołek jest typu *ConversationContext* połączony krawędzią typu *TRIGGER*. Dla znalezionych ciągów słów sprawdzam, z jaką wartością każde słowo w ciągu pobudza kontekst wypowiedzi i sumuję je. Dla znalezionych ciągów słów sprawdzam także, czy każde słowo jest specyficzne dla danego kontekstu wypowiedzi. W przypadku znalezienia takiego słowa mnożę dotychczasowy zsumowany rezultat ze współczynnikiem 1.5. Po wykonaniu zliczania pobudzeń kontekstów sortuję je malejąco według sum pobudzeń kontekstów. W przypadku znalezienia dwóch kontekstów wypowiedzi z tą samą wartością, waliduję wyjątkiem mówiącym o niejednostronnej interpretacji rozpoznanej wiadomości. Jeżeli posortowana lista jest pusta, zwracam obiekt pusty. W przeciwnym wypadku zwracany jest kontekst wypowiedzi na pozycji 0 - z największą sumą współczynników stymulujących.

5.1.2 Przykłady rozmów z czatbotem

Pytanie o dostępne spotkania:
Przykład 5.1 przedstawia rozmowę użytkownika z czatbotem, w której użytkownik pyta o dostępne spotkania użytkownika Szymona Gdańskiego. W pytaniu specyfikuje imię i nazwisko organizatora. Czatbot interpretuje pytanie i dobiera na podstawie sekwencji słów odpowiedni kontekst wypowiedzi. Następnie sprawdza, czy użytkownicy są powiązani ze sobą relacją subskrypcji. Subskrypcja w którąkolwiek stronę istnieje (tzn. użytkownik subskrybuje Pana Szymona Gdańskiego lub na odwrót), więc czatbot wyszukuje wszystkie przyszłe spotkania Pana Szymona Gdańskiego, których kryteria przewidują uczestnictwo użytkownika z jego preferencjami.

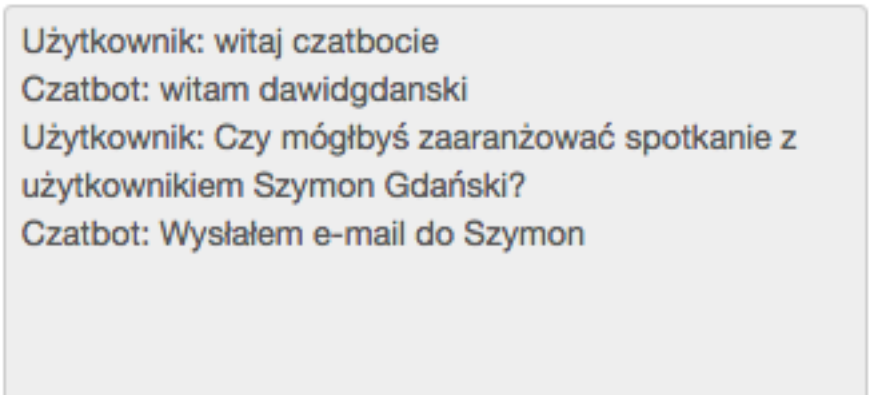
Prośba o przypisanie do spotkania:
Użytkownik, wiedząc o spotkaniach Pana Szymona, prosi czatbota o przypisanie jednego z wylistowanych podając pełną nazwę spotkania. Czatbot wykonuje sprawdzenie, czy preferencje użytkownika spełniają kryteria spotkania. W tym przypadku tak jest, zatem czatbot przypisuje użytkownika do spotkania i informuje go o tym.

Prośba o wypisanie ze spotkania:
Użytkownik, będąc przypisanym do spotkania, prosi czatbota o wypisanie z niego. W swej wypowiedzi podaje nazwę spotkania. Czatbot znajduje wspomniane spotkanie i wypisuje użytkownika z niego.



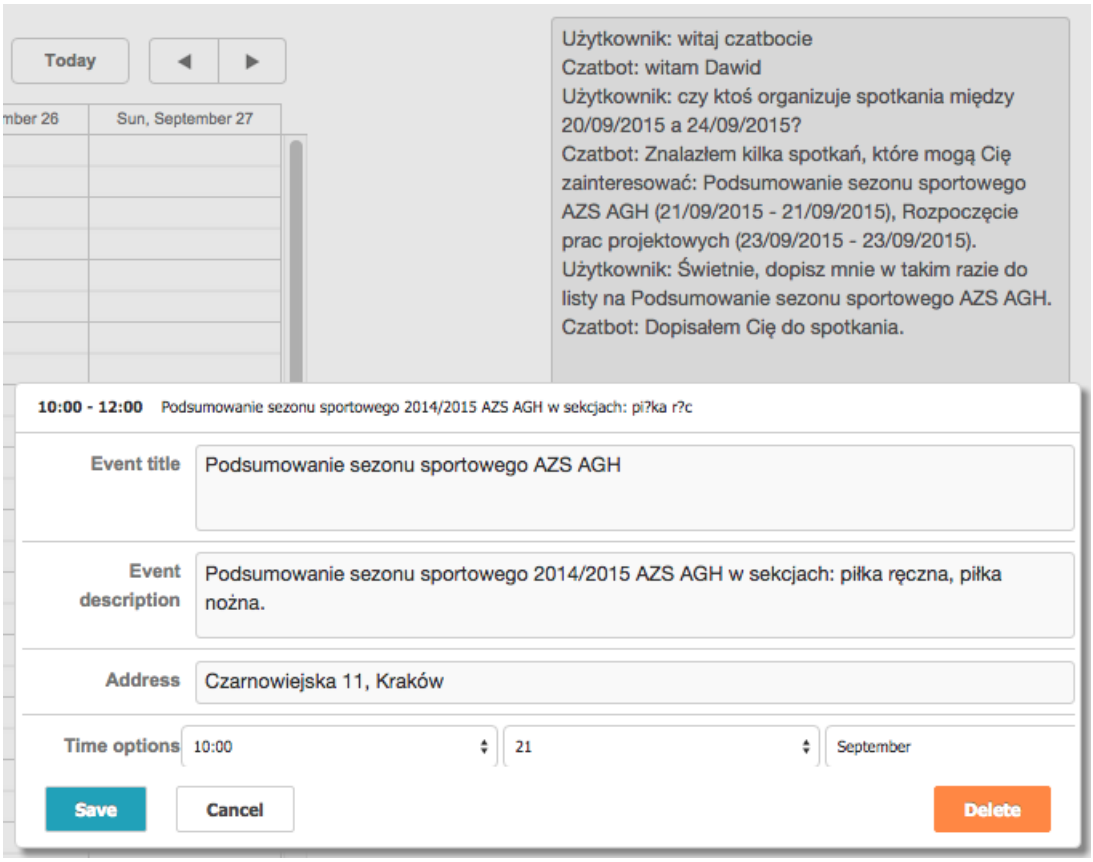
Rysunek 5.1: Konwersacja z czatbotem - przypisanie i wypisanie użytkownika ze spotkania po uprzednim dowiedzeniu się jakie spotkania organizuje inny użytkownik.

Zaaranżowanie spotkania:
Przykład 5.2 prezentuje rozmowę użytkownika z czatbotem, w której użytkownik chce soptkać się z użytkownikiem Szymonem Gdańskim. Czatbot aranżuje spotkanie poprzez wysłanie wiadomości e-mail do Szymona Gdańskiego o treści:
"Witaj (imię odbiorcy),
(imię użytkownika proszącego o spotkanie) wyraził potrzebę spotkania się z Tobą.
Rozważ tę prośbę i utwórz spotkanie bądź skontaktuj się z (imię użytkownika proszącego o spotkanie).
Jego e-mail to (e-mail użytkownika proszącego o spotkanie).
Oto prośba, którą otrzymałem: (pełna prośba użytkownika proszącego o spotkanie).
Twój asystent.".



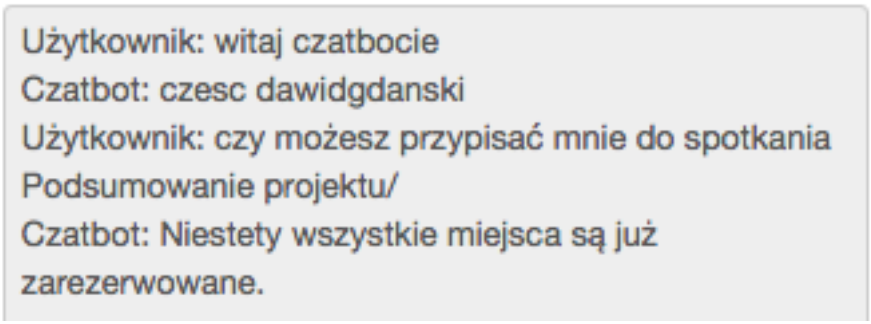
Rysunek 5.2: Wysłanie maila w imieniu jednego użytkownika do innego przez czatbota w celu zaaranżowania spotkania między nimi.

Pytanie o terminy spotkań:
Przykład 5.3 demonstruje odpowiedź czatbota na pytanie użytkownika o dostępne spotkania w konkretnym przedziale czasowym. Czatbot analizuje wypowiedź, znajdując w niej date i zawęża listę spotkań, którymi zainteresowany jest użytkownik. W odpowiedzi podaje spotkania odpowiadające preferencjom użytkownika wraz z terminami mieszczącymi się w wymaganiach czasowych postawionych przez swojego rozmówcę.



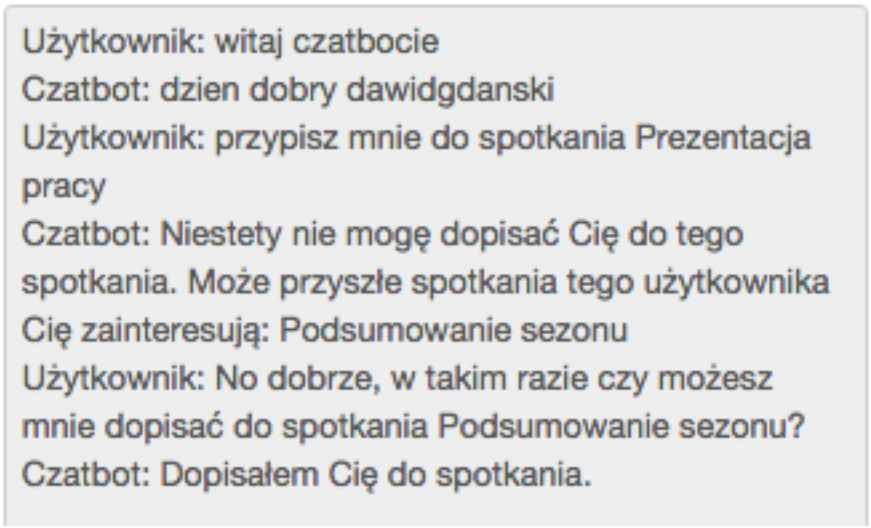
Rysunek 5.3: Odpowiedź użytkownika na pytanie o dostępne spotkania w konkretnym przedziale czasowym i przypisanie go do wybranego spotkania.

Odmowa przypisania ze względu na brak wolnych miejsc:
Przykład 5.4 przedstawia odmowę czatbota na prośbę przypisania użytkownika do spotkania z powodu braku wolnych miejsc.



Rysunek 5.4: Odmowa przypisania użytkownika do spotkania przez czatbota ze względu na brak wolnych miejsc.

Odmowa przypisania użytkownika ze spotkania ze względu na brak wymaganych preferencji określonych w kryteriach spotkania:
Przykład 5.5 prezentuje konwersację użytkownika z czatbotem, w której odmawia przypisania go do spotkania na jego prośbę. Powodem jest brak posiadania odpowiednich preferencji, które uwzględnione są w kryteriach spotkania. Jednocześnie czatbot wyszukuje przyszłe spotkania, kryteriom których preferencje użytkownika odpowiadają i sugeruje mu wzięcie w nich udziału.

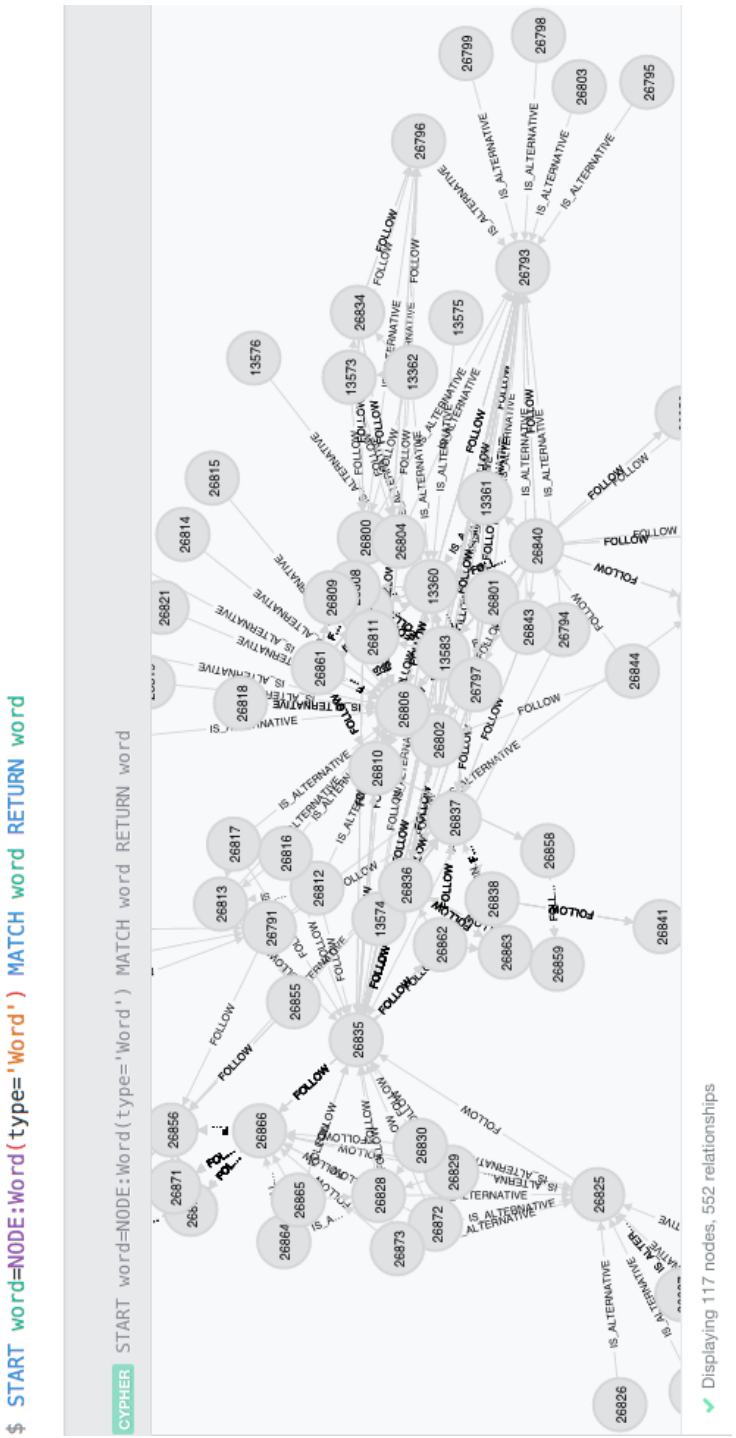


Rysunek 5.5: Odmowa przypisania użytkownika do spotkania przez czatbota ze względu na brak wymaganych preferencji. Jednoczesne zaproponowanie innego spotkania, którego kryterium przewiduje udział użytkowników z takimi preferencjami.

5.1.3 Ograniczenia inteligencji czatbota

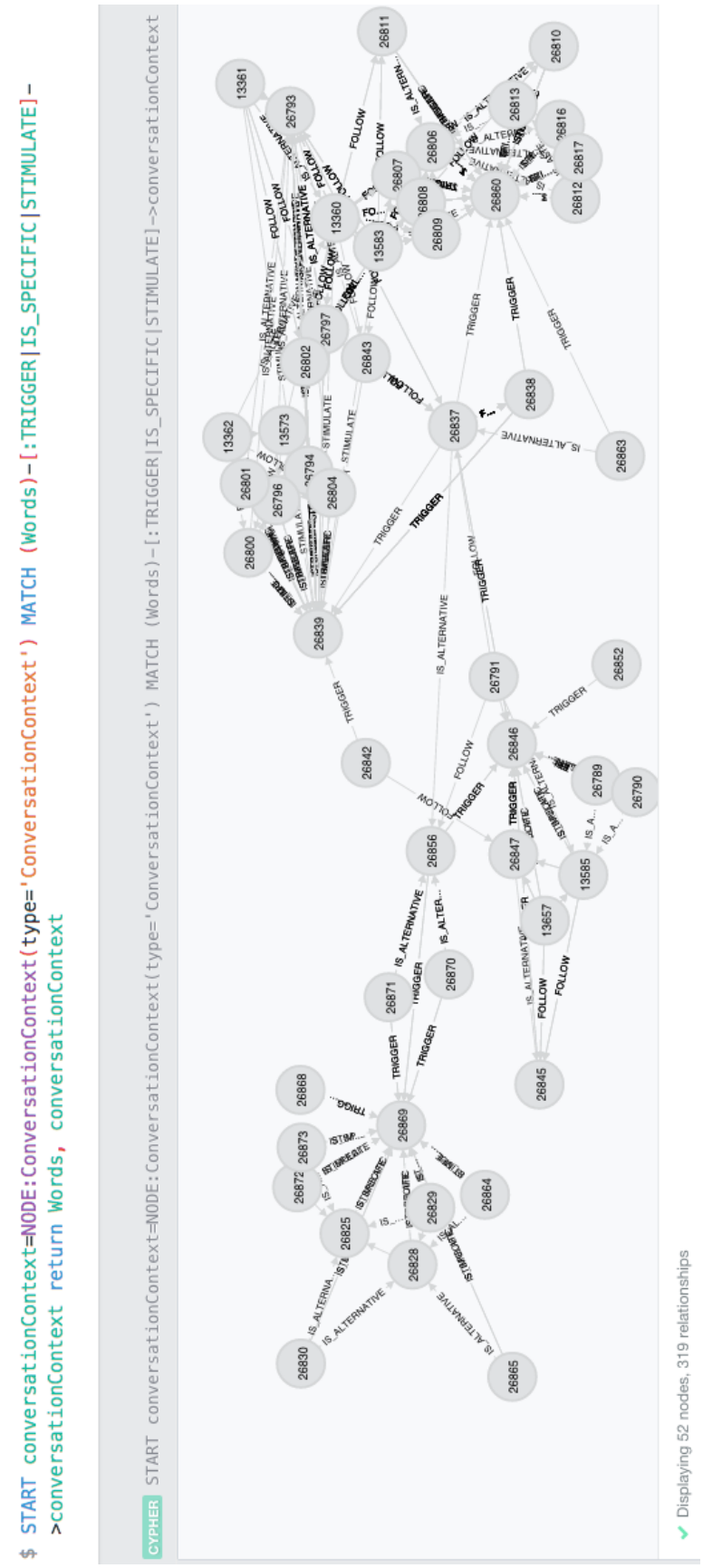
Czatbot, w zaimplementowanej wersji, potrafi inteligentnie odnieść się do potrzeb użytkownika w zakresie tematyki uzgadniania terminów spotkań. Umożliwia także wystąpienie

w imieniu petenta o zorganizowanie spotkania z innymi oraz jest w stanie zarządzać miejscami wyznaczonymi przez organizatorów spotkania wpisując lub wypisując z nich zainteresowanych użytkowników. Jednakże, posiada on ograniczenia ze względu na rozmiar swojej bazy wiedzy. W zaimplementowanej wersji baza ta posiada łącznie 117 wierzchołków typu *Word*. Wierzchołki te połączone są ze sobą krawędziami typu *FOLLOW* i *IS_ALTERNATIVE* w łącznej liczbie 552.



Rysunek 5.6: Konsola bazy danych Neo4j - graf bazy wiedzy przedstawiający statystyczne dane dotyczące wierzchołków typu *Word* i połączeń między nimi na odpowiednie zapytanie.

Baza wiedzy posiada 4 wierzchołki typu *ConversationContext*, które są połączone z wierzchołkami typu *Word* w liczbie 52. Połączenia między tymi wierzchołkami są typu *STIMULATE*, *TRIGGER* i *IS_SPECIFIC* oraz występują liczbie 319.



Rysunek 5.7: Konsola bazy danych Neo4j - graf bazy wiedzy przedstawiający statystyczne dane dotyczące liczby wierzchołków typu *Word* i *ConversationContext* oraz połączeń między nimi na odpowiednie zapytanie.

Na podstawie powyższych danych można stwierdzić, iż graf złożony z kilkuset wierzchołków i kilkukrotnie wyższą liczbą połączeń między nimi jest w stanie nadać wirtualnemu rozmówcy zdolność poradzenia sobie z interpretacją pewnej grupy potrzeb w konkretnych sytuacjach. Uwidacznia się w tym momencie analogia do ludzkiej zdolności zapamiętywania i kojarzenia informacji. Proces uczenia się przez ludzki umysł polega na tworzeniu coraz nowszych jednostek przetwarzania informacji - neuronów. Im wiedza ta jest częściej utrwalana, tym neurony odpowiedzialne za jej przetwarzanie są mocniejsze. Aczkolwiek, nawet najsilniejsze, nie połączone ze sobą neurony nie stworzą w pełni inteligentnego systemu zdolnego do efektywnego kojarzenia faktów. Kojarzenie jest procesem wiązania ze sobą danych wraz z ich kombinacjami i układami.
Pan dr hab. Adrian Horzyk, w jednym ze swoich dzieł na temat sztucznej inteligencji [6] przypisuje rolę procesu formowania się skojarzeń synapsom, czyli aktywnym połączeniom między neuronami. Podczas procesu kojarzenia, na podstawie tzw. kontekstu powiązania wiedzy [6] generowane są impulsy między neuronami przewodzone przez synapsy. Im bardziej "utrwalona" synapsa, tym mocniejsze skojarzenia zespolone z konkretnymi neuronami kolejkują się i priorytetują tworząc mechanizm reakcji zależny od kontekstu sytuacji.

Czatbot w obecnej postaci nie potrafi:

- zapamiętać poprzednich kontekstów wypowiedzi w rozmowie z użytkownikiem;
- oceniać jakości swoich odpowiedzi w rozmowie z użytkownikiem;
- uczyć się rozpoznawać nowe konteksty wypowiedzi w przypadku, gdy nie uda mu się dopasować poznanego wcześniej kontekstu do sekwencji słów;
- odnieść się do swojego rozmówcy personalnie, biorąc pod uwagę jego preferencje.

5.2. Wykorzystane oprogramowanie i licencja projektu

Opis	Nazwa
Język programowania	Java SE wersja 7, Oracle [12]
Kontener systemu	Apache Tomcat, wersja 8 [9]
Platforma programistyczna	Spring Framework, wersja 4.0.0 [9]
Baza danych	Neo4j - wersja "community" [11]
Kalendarz spotkań	DHTMLX JavaPlanner - wersja Standard [11]
Testy projektu	JUnit [10]

Tabela 5.1: Tabela ilustrująca najbardziej istotne technologie wykorzystane do stworzenia systemu.

Do stworzenia systemu wykorzystałem także inne biblioteki ułatwiające proces implementacji. Wszystkie biblioteki można zobaczyć w załączniku z kodem źródłowym. Licencje wykorzystanych przeze mnie, są umieszczone w bibliografii pracy magisterskiej [12] [9] [11].

Kod źródłowy stanowi część praktyczną pracy magisterskiej. Ze względu na licencje wykorzystanych technologii projekt nie może być dystrybuowany komercyjnie. Zezwalam na wykorzystanie projektu w celach dydaktycznych.

5.3. Scenariusze testowe

Test 1: umówienie spotkania w z wyznaczonym użytkownikiem przez czatbota.

Warunki początkowe:

1. Użytkownik A jest zarejestrowany w systemie.
2. Użytkownik B jest zarejestrowany w systemie.
3. Użytkownik A organizuje spotkanie.

Akcja:

Użytkownik B wpisuje wiadomość z prośbą o umówienie na spotkanie, np.: "Witaj, czy mógłbyś umówić mnie z użytkownikiem A na spotkanie?".

Rezultat:

Czatbot wysyła e-mail do użytkownika A z powiadomieniem, że zaszła potrzeba stworzenia spotkania z udziałem użytkowników o preferencjach użytkownika B.

Test 2: odpowiedź czatbota na pytanie o dostępne spotkania odpowiadające preferencjom użytkownika.

Warunki początkowe:

1. Użytkownik A jest zarejestrowany w systemie.
2. Są dostępne spotkania odpowiadające preferencjom użytkownika A.

Akcja:

Użytkownik zadaje pytanie o dostępne spotkania, np.: "Czy możesz podać mi dostępne spotkania?".

Rezultat:

Czatbot odpowiada użytkownikowi, podając listę przyszłych spotkań, do których może się przypisać.

Test 3: przypisanie użytkownika do spotkania przez czatbota.

Warunki początkowe:

1. Użytkownik A posiada konto w systemie.
2. Istnieje spotkanie B, w kryteriach którego są przewidziane preferencje użytkownika A.

Akcja:

Użytkownik prosi czatbota o przypisanie go do spotkania, np.: "Czy mógłbyś przypisać mnie do spotkania B?".

Rezultat:

Czatbot generuje odpowiedź, przypisując użytkownika do spotkania na podstawie jego preferencji.

Test 4: odmowa przypisania użytkownika do spotkania przez czatbota ze względu na brak preferencji przewidzianych w kryteriach spotkania.

Warunki początkowe:

1. Użytkownik A posiada konto w systemie.
2. Istnieje spotkanie B, w kryteriach którego nie są przewidziane preferencje użytkownika A.

Akcja:

Użytkownik prosi czatbota o przypisanie go do spotkania, np.: "Dopisz mnie proszę do spotkania B"

Rezultat:

Czatbot waliduje zinterpretowaną prośbę, podając przyczynę, dlaczego przypisanie użytkownika do spotkania się nie powiodło.

Test 5: odmowa przypisania użytkownika do spotkania przez czatbota ze względu na brak brak miejsc.

Warunki początkowe:

1. Użytkownik A posiada konto w systemie.
2. Istnieje spotkanie B, w kryteriach którego są przewidziane preferencje użytkownika A, natomiast wszystkie miejsca są już zarezerwowane.

Akcja:

Użytkownik prosi czatbota o przypisanie go do spotkania, np.: "Dopisz mnie proszę do spotkania B"

Rezultat:

Czatbot waliduje zinterpretowaną prośbę, podając przyczynę, dlaczego przypisanie użytkownika do spotkania się nie powiodło.

Test 6: wypisanie użytkownika ze spotkania.

Warunki początkowe:

1. Użytkownik A posiada konto w systemie.
2. Istnieje spotkanie B, do którego użytkownik A jest przypisany.

Akcja:

Użytkownika prosi czatbota o wypisanie go ze spotkania, np.: "Czy mógłbyś wypisać mnie ze spotkania B?".

Rezultat:

Czatbot wypisuje użytkownika ze spotkania i generuje powiadomienie o wypisaniu.

Test 7: odpowiedź na pytanie o dostępne terminy spotkań.

Warunki początkowe:

1. Użytkownik A posiada konto w systemie.
2. Istnieje spotkanie B, którego kryteria przewidują uczestnictwo spotkania.

Akcja:

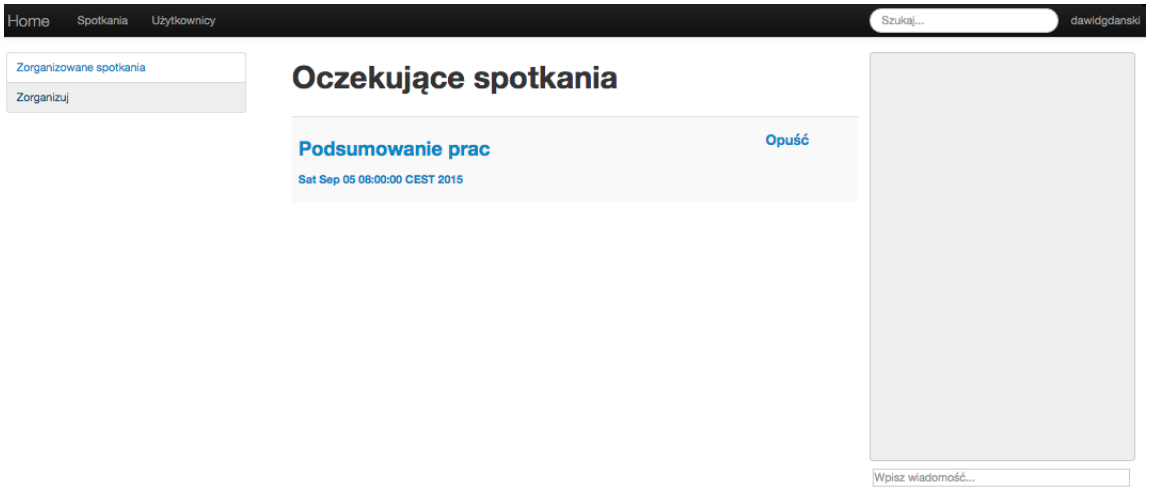
Użytkownika pyta czatbota o terminy spotkań, np.:

- "Czy jest ktoś, kto organizuje spotkania pomiędzy 01-10-2015, a 01-20-2015?";
- "Czy jest ktoś, kto organizuje spotkania w okolicach 05/10/2015?"

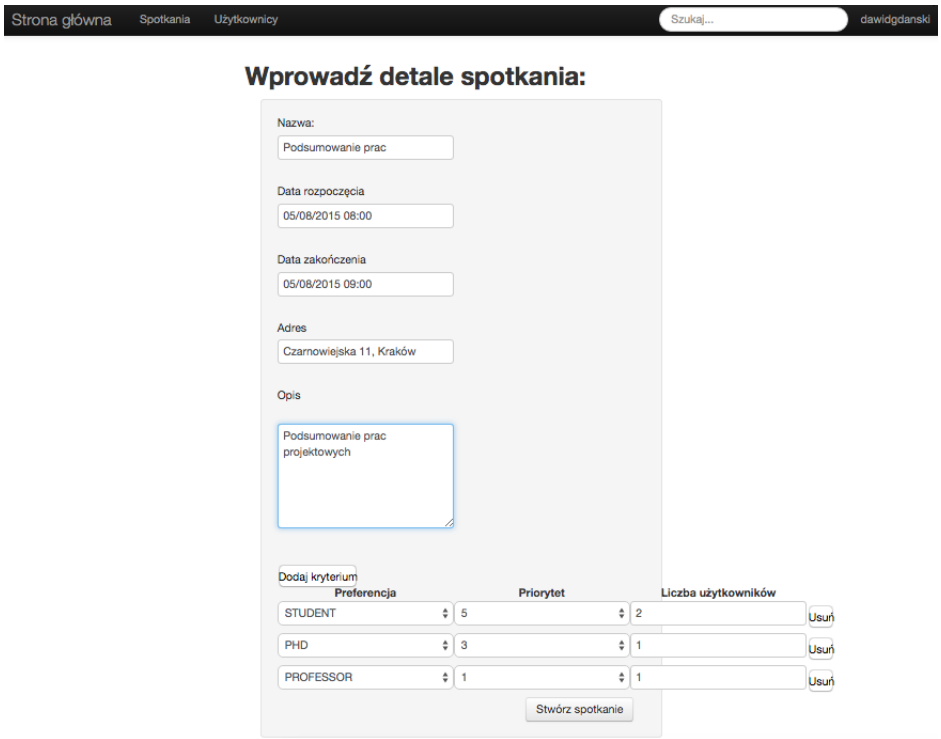
Rezultat:

Czatbot przeszukuje bazę spotkań i podaje informacje odnośnie tych, do których użytkownik może się przypisać w ramach wyznaczonych dat. W przypadku podania pojedynczej daty, czatbot podaje dostępne spotkania, począwszy od czasu wpisania pytania do wyznaczonej daty. Podanie daty przeszłej nie jest akceptowane przez czatbota.

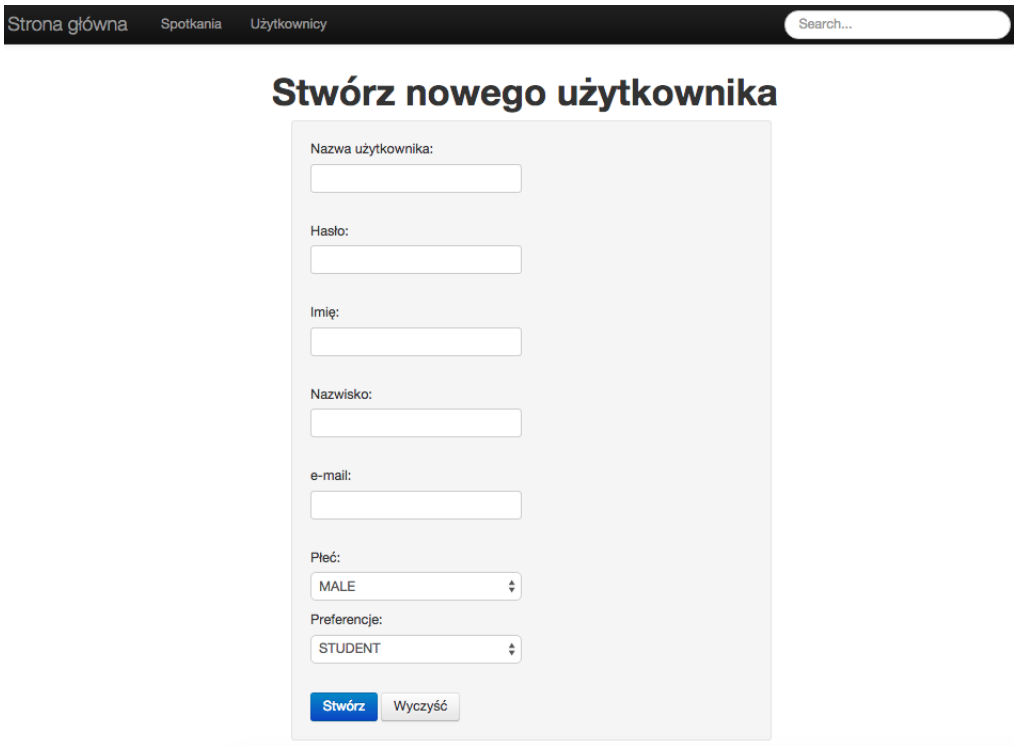
5.4. Wizualizacje systemu



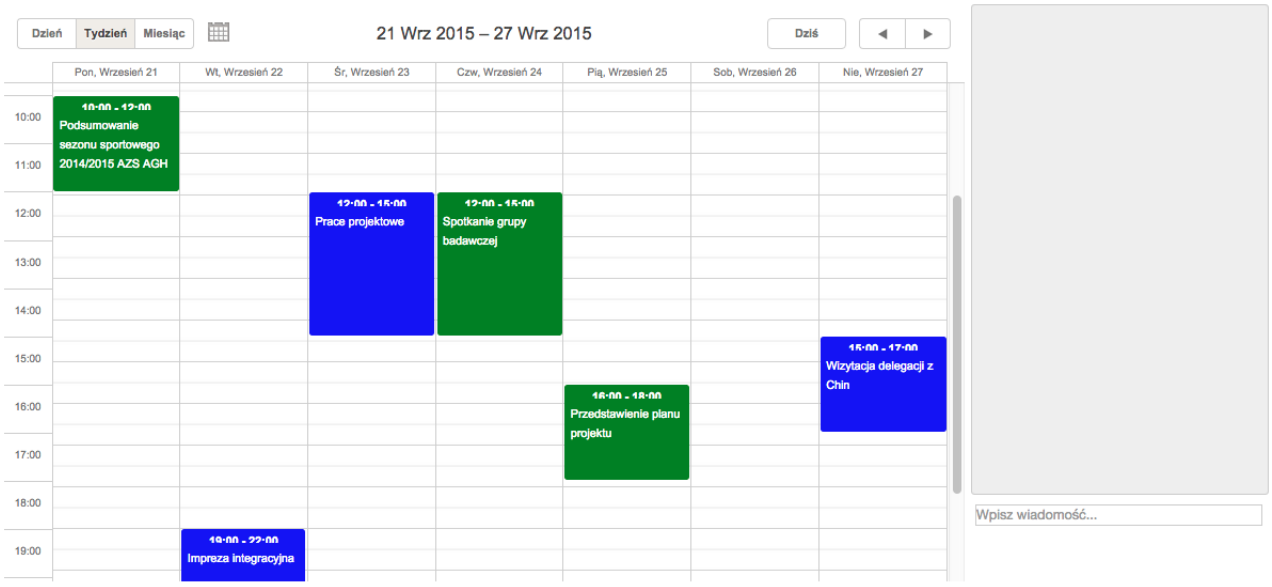
Rysunek 5.8: Widok listy oczekujących spotkań, do których użytkownik jest przypisany.



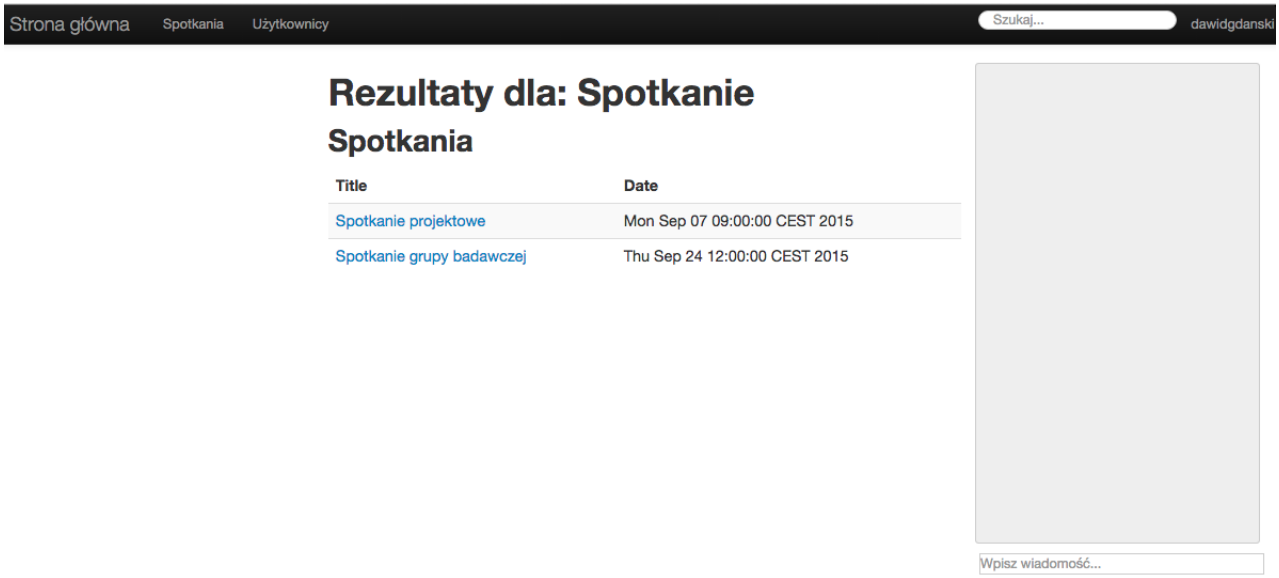
Rysunek 5.9: Widok tworzenia spotkania.



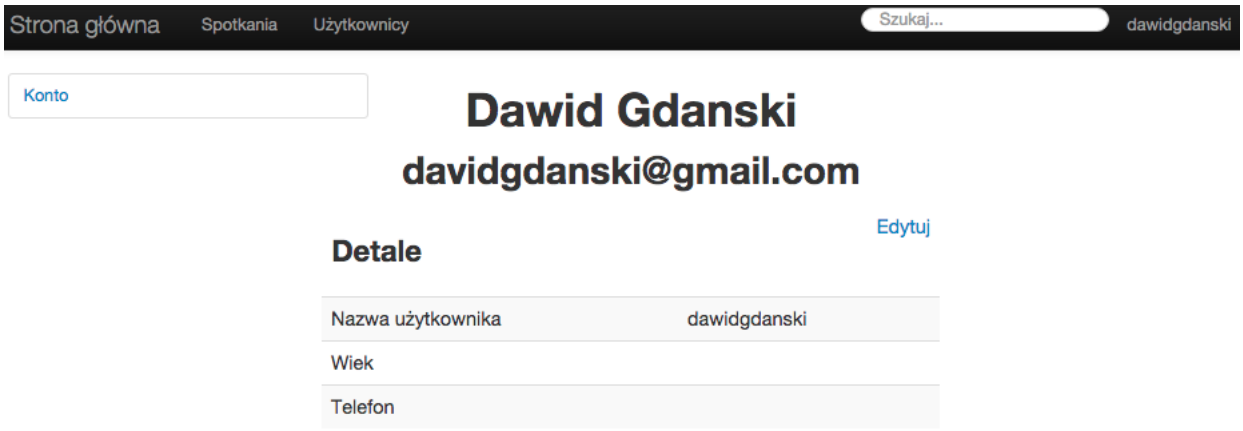
Rysunek 5.10: Widok tworzenia konta użytkownika.



Rysunek 5.11: Widok planera spotkań wraz z konsolą rozmów czatbotem. Zielonym kolorem zaznaczone są spotkania, do których użytkownik jest przypisany. Niebieski kolor symbolizuje spotkania, do których użytkownik jeszcze nie przypisał się lub nie może być przypisany.



Rysunek 5.12: Widok wyszukiwanych rezultatów wraz z konsolą rozmów czatbotem.



Rysunek 5.13: Widok wyszukiwanych rezultatów wraz z konsolą rozmów czatbotem.

6. Zakończenie

6.1. Wnioski

Zadaniem tej pracy magisterskiej było stworzenie systemu, dzięki któremu jego użytkownicy będą mogli zarządzać i uzgadniać terminy spotkań. System jednocześnie miał dokonać pewnych optymalizacji pod względem dopasowania preferencji użytkownika z kryteriami spotkań, generując odpowiedź mówiącą o tym, czy zapis do spotkania jest możliwy.

Projektowanie systemu rozpocząłem od stworzenia modelu biznesowego systemu dla przedsiębiorstwa, które mogłoby dystrybuować system jako usługę. Zaproponowałem przykład w oparciu o model "Business Model Canvas". Jako dopełnienie stworzyłem także propozycję korporacyjnego modelu architektury systemu składającego się z 3 warstw: biznesowej, aplikacji i technologicznej. Model biznesowy "Business Model Canvas" dostarcza szablonu, który obrazuje, jak poszczególne obszary biznesowe przedsiębiorstwa są powiązane ze sobą [1]. Pozwala zilustrować słabości i mocne strony biznesplanu jego twórcom. Innymi słowy, twórca na pytanie "co, jeśli..." otrzymuje odpowiedź w postaci wypełnionego szablonu strategii biznesowej firmy. Umożliwia on także dokonywanie pewnych przewidywań jego twórcom mówiącym, jak przedsiębiorstwo zareaguje na konkretny scenariusz. Kluczowym pryncypium modelu BMC jest położenie w większym stopniu nacisku na czynnik jakościowy niż na ilościowy proponowanej oferty.
Korporacyjny model architekturowy jest diagramem strategii przedsiębiorstwa. Swoją postacią obejmuje obszary z zakresu planów biznesowych, wizji dotyczących oprogramowania oraz decyzji w zakresie doboru technologii.

Następnie opracowałem specyfikację techniczną jako teoretyczny wstęp do implementacji systemu. Specyfikacja technicza stanowi potwierdzenie obecnego stanu projektu. Odgrywa także istotną rolę pomiędzy osobami ze środowiska odpowiedzialnego za rozwój produktu z osobami z innych środowisk współpracujących w obrębie tego samego przedsiębiorstwa. Dla opisywanego systemu stworzyłem specyfikację według schematu przedstawionego na laboratorium, a w nim:

1. Zdefiniowałem opis i cel systemu.
2. Wyróżniłem udziałowców i użytkowników oraz ich cele.
3. Przedstawiłem granice i możliwości systemu.
4. Przeprowadziłem analizę obiektów biznesowych.
5. Opisałem specyfikację wymagań oprogramowania.
6. Opisałem architekturę systemu.
7. Przedstawiłem architekturę bazy danych stanowiącej bazę obiektów biznesowych oraz bazę wiedzy czatbota.

W ostatnim kroku zaimplementowałem system według specyfikacji technicznej z częściowym zachowaniem zasad z modelu biznesowego. W systemie obecny miał być także element czatbota, który realizuje zadanie uzgadniania i optymalizacji terminu spotkania.

Ten element stworzyłem w oparciu o grafową bazę wiedzy. Algorytm interpretacji wypowiedzi użytkownika zaimplementowałem w sposób zaprojektowany według uzgodnień z promotorem. Polega on na pobudzaniu grafu sekwencji zapamiętanych przez czatbota słów w taki sposób, by uzyskać właściwy kontekst wypowiedzi i skutecznie zareagować na potrzebę swojego rozmówcy. Innymi słowy, czatbot posiada uproszczoną strukturę zbliżoną do ludzkiego mózgu, w której zachodzi proces kojarzenia i zaspokajania potrzeb użytkowników. Termin "uzgodnić" oznacza: doprowadzić do stanu, w którym dwie równoprawne strony zgadzają się co do podjęcia obupólnej decyzji biorąc pod uwagę argumenty każdej z nich [17]. Czatbot, jako członek procesów uzgadniania potrafi:

1. Pozwolić na zapisanie użytkownika do spotkania pod warunkiem spełnienia odpowiednich kryteriów.

2. Zabronić zapisania użytkownika do spotkania, gdy ten nie spełnia jego kryteriów.

3. Poprosić w imieniu swojego reprezentanta innych użytkowników do rozważenia stworzenia wspólnego spotkania.

4. Zasugerować alternatywne propozycje w przypadku, gdy nie jest przewidziane miejsce dla użytkownika w spoktaniu, w którym chciałby wziąć udział.

5. Zaproponować spotkania odpowiadające preferencjom użytkownika w konkretnych terminach lub przedziałach czasowych.

Dbając o jakoś stworzonego rozwiązania, stworzyłem także testy do grafowej bazy danych sprawdzające niektóre jej komponenty systemu w izolacji od środowiska, w którym działają. Kod źródłowy stanowi załącznik pracy magisterskiej.

Odpowiadając na pytanie postawione we wstępie pracy magisterskiej: jak rozwijać projekty informatyczne, podejmując jednocześnie trafne decyzje odnośnie ich rozwoju? Jak unikać niepotrzebnych błędów skutkujących spowolnieniem lub wstrzymaniem postępu prac oraz brakiem zadowolenia ze strony klientów? Błędy są naturalną konsekwencją czynionych prac. Aczkolwiek, w opisanym schemacie projektowania systemu połączyłem poznane metody, dzięki czemu uważam, że liczba błędów może zostać sukcesywnie zredukowana do mniejszej liczby. Istotą opisanego schematu jest dogłębna wiedza na temat rynku klientów, rozwiązań konkurencji wraz z ich słabościami i walorami. Na tej podstawie można stworzyć specyfikację techniczną produktu, która zrealizowana praktycznie przyniesie korzyść przedsiębiorstwu.

6.2. Podsumowanie

W życiu codziennym otaczamy się wieloma rozwiązaniami informatycznymi oraz elektronicznymi. Można odnieść wrażenie, że w miarę postępu technologicznego przyzwyczajamy się do systemów, które dostarczają pewne usługi, odciążając nas tym samym od niepotrzebnej straty czasu. Wobec tego niezwykle istotnym dla przedsiębiorstw informatycznych dostarczających te usługi jest dbanie o ich właściwy rozwój oraz wychodzenie naprzeciw potrzebom klientów. Można stwierdzić, że im więcej reakcji na potrzeby odbiorców, tym oferta jest bardziej elastyczna. Potrzeby ludzkie w dzisiejszym świecie są bardzo zróżnicowane. W wielu kwestiach jesteśmy zmuszeni zaspokajać je osobiście, jednak postęp technologiczny pozwala na wyręczenie nas z tego w coraz większej mierze i zaoszczędzenie naszego czasu. Zaawansowane rozwiązania sztucznej inteligencji są w stanie domyślić się w

wielu kwestiach, co jest naszym celem oraz dobierają najlepsze rozwiązania, które spełnią nasze oczekiwania. Praca magisterska stanowi podsumowanie wiedzy, którą zdobyłem podczas studiów II stopnia i zastosowałem do rozwiązania parktycznego problemu, jakim było stworzenie i udokumentowanie opisanego systemu.

Bibliografia

[1] *Business Model Generation: A Handbook for Visionaries, Game Changers, and Challengers*. John Wiley i Sons, Inc., 2010.

[2] Alistair Cockburn. *Writing Effective Use Cases*. URL: `http://home.agh.edu.pl/~pszwed/pub/uml/weuc0002extract.pdf`.

[3] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2002.

[4] Adrian Horzyk. „Innovative Types and Abilities of Neural Networks Based on Associative Mechanisms and a New Associative Model of Neurons". *Referat na zaproszenie na międzynarodowej konferencji ICAISC 2015* (2015.), str. 26–38.

[5] Adrian Horzyk. *Negocjacje - Sprawdzone strategie*. Kraków: Samo Sedno, 2012.

[6] Adrian Horzyk. *Sztuczne systemy skojarzeniowe i asocjacyjna sztuczna inteligencja*. Warszawa: Akademicka Oficyna Wydawnicza EXIT, 2013.

[7] RedPill szkolenia IT konsulting. „Ciąg artykułów - "Akademia UML"". *Software Developer's Journal nr 4* (2009.). URL: `http://www.redpill.com.pl/artykuly/akademia/`.

[8] Mark Lankhorst. *Enterprise Architecture at Work*. Springer, 2009.

[9] *Licencja Apache Software Foundation*. URL: `http://www.apache.org/licenses/`.

[10] *Licencja Framework'u do testów JUnit*. URL: `https://github.com/junit-team/junit/blob/master/LICENSE-junit.txt`.

[11] *Licencja "GNU General Public License", wersja 3*. URL: `http://www.gnu.org/licenses/gpl-3.0.en.html`.

[12] *Licencja języka Java firmy Oracle*. URL: `http://www.oracle.com/technetwork/java/javase/terms/license/index.html`.

[13] *Licencja oprogramowania Visual Paradigm for UML CE 11*. URL: `http://www.visual-paradigm.com/editions/community.jsp`.

[14] *Licencja programu yEd Graph Editor*. URL: `http://www.yworks.com/products/yed/license.html`.

[15] *Licencja standardu ArchiMate ®*. URL: `https://www.opengroup.org/archimate/2.1/corp_index.htm`.

[16] Dr inż Piotr Szwed. *Specyfikacja i systemy wspomagania oprogramowania*. URL: `http://home.agh.edu.pl/~pszwed/wiki/doku.php?id=specif:projekt`.

[17] *Słownik języka polskiego*. URL: `http://sjp.pwn.pl/`.

[18] Dr Andrzej Sobczak. „Modele i metamodele w architekturze korporacyjnej". *Projekt badawczy nr N115 010 32/01143 finansowany ze środków Ministerstwa Nauki i Szkolnictwa Wyższego* ().

[19] Alan Mathison Turing. „Computing machinery and intelligence". (1950.).

[20] Erich Gama; Richard Helm; Ralph Johnson; John Vlissides. *Wzorce projektowe: Elementy oprogramowania obiektowego wielokrotnego użytku*. Gliwice: Helion, 2010.