

Spis treści

1	Wstęp	5
1.1	Wprowadzenie	5
1.2	Cel pracy	5
2	Analiza rynku	6
2.1	Podobne rozwiązanie - uShip	6
2.2	Podobne rozwiązanie - Oferteo	7
2.3	Podobne rozwiązanie - Allegro	9
2.4	Projektowana aplikacja	9
2.5	Podsumowanie analizy	9
3	Omówienie koncepcji systemu	10
3.1	Ogólny opis projektu	10
3.2	Docelowi użytkownicy	10
3.3	Cele funkcjonowania systemu	10
3.4	Opis funkcjonalności	10
3.4.1	Wymagania funkcjonalne	10
3.4.2	Wymagania niefunkcjonalne	11
4	Analiza dostępnych narzędzi	12
4.1	Baza danych	12
4.2	Serwer	12
4.3	Aplikacja kliencka	12
4.4	Podsumowanie wyboru technologii	13
5	Projekt systemu	14
5.1	Architektura systemu	14
5.2	Diagramy klas	15
5.3	Diagramy związków encji	16
5.4	Diagramy przypadków użycia	17
5.5	Diagramy aktywności	22
5.6	Diagramy sekwencji	24
5.7	Opis mechanizmów bezpieczeństwa	26
5.8	Opis <i>endpointów</i>	28
5.8.1	<i>Backend</i>	28
5.8.2	<i>Frontend</i>	29
5.9	Szkice i funkcjonalność aplikacji klienckiej	31
5.9.1	Logowanie	31
5.9.2	Wyszukiwanie aukcji	32
5.9.3	Rejestracja nowego użytkownika	33
5.9.4	Panel użytkownika	34
5.9.5	Aukcja	35
5.9.6	Stwórz nową aukcję	36
5.9.7	Aktywacja konta	37
5.9.8	Wspólne elementy	38
6	Implementacja systemu	39
6.1	Implementacja bazy danych	39
6.2	Implementacja serwera	44
6.2.1	Struktura projektu	44
6.2.2	Konfiguracja projektu - Maven	45
6.2.3	Encje	47
6.2.4	Obiekty dostępu danych	49

6.2.5	Serwisy danych	51
6.2.6	Serwis mailowy	52
6.2.7	Udostępnianie danych za pomocą kontrolera	54
6.2.8	Przykładowe obiekty transferu danych - DTO	55
6.2.9	Dane inicjujące	56
6.2.10	Przykład uruchomienia serwera	58
6.3	Implementacja aplikacji klienckiej	59
6.3.1	Struktura projektu	59
6.3.2	Integracja z serwerem	60
6.3.3	Główny moduł aplikacji	62
6.3.4	Logowanie	63
6.3.5	Widok dostępnych aukcji	67
6.3.6	Implementacja wielojęzyczności	68
6.3.7	Uruchomienie aplikacji klienckiej	72
7	Użytkowanie aplikacji	73
7.0.1	Logowanie i wylogowywanie	73
7.0.2	Stworzenie nowej aukcji	75
7.0.3	Stworzenie nowego użytkownika	76
7.0.4	Wyświetlenie aukcji	77
8	Wdrożenie i ocena jakości	78
8.1	Podstawowe wymagania instalacji systemu	78
8.1.1	Instalacja Javy	78
8.1.2	Instalacja Maven'a	78
8.1.3	Instalacja nodeJS	78
8.1.4	Instalacja i konfiguracja bazy danych	78
8.2	Instalacja i budowa aplikacji	79
8.3	Uruchomienie aplikacji	80
8.4	Testy aplikacji	80
8.4.1	Istniejące testy	80
8.4.2	Testy które powinny zostać zaimplementowane	80
8.5	Weryfikacja	81
8.6	Walidacja	81
9	Podsumowanie	82
9.1	Technologie	82
9.2	Realizacja pracy	82
9.3	Wnioski	82
9.4	Dalsze możliwości rozwoju systemu	83
	Literatura	84
A	Opis załączonej płyty CD/DVD	85

Spis rysunków

1	uShip - widok dostępnych aukcji	6
2	uShip - statystyki zapytań o stronę w serwisie GoogleTrends	7
3	Oferteo - widok dostępnych kategorii	8
4	Oferteo - widok dostępnych aukcji	8
5	Architektura systemu	14
6	Diagram klas	15
7	Diagram związków encji	16
8	Diagram przypadków użycia - użytkownik aplikacji	17

9	Diagram przypadków użycia - moderator aplikacji	20
10	Diagram aktywności - wycofanie aukcji	22
11	Diagram aktywności - wystawienie aukcji	22
12	Diagram aktywności - rejestracja konta	23
13	Diagram sekwencji - logowanie	24
14	Diagram sekwencji - rejestracja użytkownika	25
15	Diagram sekwencji - anulowanie aukcji	26
16	Mapa <i>endpointów</i> aplikacji klienckiej	30
17	Szkice aplikacji - logowanie użytkownika	31
18	Szkice aplikacji - wyszukiwanie aukcji	32
19	Szkice aplikacji - rejestracja nowego użytkownika	33
20	Szkice aplikacji - panel użytkownika	34
21	Szkice aplikacji - aukcja	35
22	Szkice aplikacji - stwórz nową aukcję	36
23	Szkice aplikacji - aktywacja konta	37
24	Szkice aplikacji - wspólne elementy	38
25	Implementacja bazy danych - porównanie widoku bazy Java - SQL	39
26	Implementacja bazy danych - porównanie widoku encji Java - SQL	41
27	Implementacja serwera - struktura projektu	44
28	Implementacja serwera - przykładowy e-mail	54
29	Wyciąg z konsoli po uruchomieniu serwera	58
30	Implementacja aplikacji klienckiej - struktura projektu	59
31	Implementacja aplikacji klienckiej - przykładowe zapytanie <i>HTTP</i>	61
32	Wyciąg z konsoli po uruchomieniu aplikacji klienckiej	72
33	Użytkowanie aplikacji - logowanie	73
34	Użytkowanie aplikacji - wylogowanie	74
35	Użytkowanie aplikacji - stworzenie nowej aukcji	75
36	Użytkowanie aplikacji - stworzenie nowego użytkownika	76
37	Użytkowanie aplikacji - aktywacja nowego użytkownika	76
38	Użytkowanie aplikacji - wyświetlanie aukcji	77
39	Użytkowanie aplikacji - wyświetlenie konkretnej aukcji	77

Spis listingów

1	Konfiguracja dostępu do bazy danych przy użyciu <i>Hibernate</i> - XML	42
2	Konfiguracja dostępu do bazy danych przy użyciu <i>Hibernate</i> - Java	43
3	Implementacja serwera - konfiguracja <i>Maven'a</i>	46
4	Implementacja serwera - encje	47
5	Implementacja serwera - wspólne metody	49
6	Implementacja serwera - obiekt dostępu danych użytkownika	50
7	Implementacja serwera - serwis użytkownika	51
8	Implementacja serwera - serwis e-mail	52
9	Implementacja serwera - konfiguracja serwisu e-mail	53
10	Implementacja serwera - kontroler tworzący nowego użytkownika	55
11	Implementacja serwera - przykładowy obiekt transferu danych - aukcja	56
12	Implementacja serwera - przykładowy dane inicjujące	57
13	Implementacja serwera - klasa uruchamiająca	58
14	Implementacja aplikacji klienckiej - integracja z serwerem	60
15	Implementacja aplikacji klienckiej - główny moduł aplikacji	62
16	Implementacja aplikacji klienckiej - router	62
17	Implementacja aplikacji klienckiej - logowanie <i>HTML</i>	64
18	Implementacja aplikacji klienckiej - logowanie <i>CSS</i>	65
19	Implementacja aplikacji klienckiej - logowanie <i>Component</i>	66
20	Implementacja aplikacji klienckiej - lista aukcji <i>HTML</i>	67

21	Implementacja aplikacji klienckiej - lista aukcji <i>Component</i>	68
22	Implementacja aplikacji klienckiej - implementacja wielojęzyczności - <i>HTML</i>	68
23	Implementacja aplikacji klienckiej - implementacja wielojęzyczności - <i>CSS</i>	69
24	Implementacja aplikacji klienckiej - implementacja wielojęzyczności - <i>Component</i>	70
25	Implementacja aplikacji klienckiej - implementacja wielojęzyczności - <i>Serwis</i>	71

Spis tabel

1	Podsumowanie analizy rynku	9
---	--------------------------------------	---

1 Wstęp

1.1 Wprowadzenie

Praca ta polega na zaprojektowaniu i zaimplementowaniu aplikacji internetowej wspomagającej aukcyjną organizację usług przewozu towarów. Pierwsza część tematu czyli *"Aplikacja internetowa"* oznacza system do którego użytkownicy mają dostęp poprzez przeglądarkę internetową poprzez sieć internetową. Druga część czyli *Wspomagającą aukcyjną organizację usług przewozu towarów* oznacza, iż główną funkcjonalnością będzie możliwość licytacji usługi przewozu towaru - jedna osoba tworzy aukcję, a inna ją licytuje i użytkownik z najniższą ofertą wygrywa możliwość wykonania zlecenia. Towar ma podlegać kategoryzacji (np. poprzez wagę, rozmiar bądź bardziej indywidualne właściwości).

Pracę motywowałem chęcią zdobycia doświadczenia oraz umiejętności przy tworzeniu stron internetowych w najpopularniejszych technologiach, potencjalną możliwością zarobkową wynikającą z samej aplikacji ponieważ nie istnieją podobne rozwiązania na polskim rynku co można przeczytać w jej dalszej części.

1.2 Cel pracy

Cele pracy dyplomowej podzielone są na dwa trzy kategorie: główne, dodatkowe oraz ograniczenia. Pierwsza stanowi definicję zrealizowanego projektu. Druga jest uzupełniająca, natomiast trzecia ma za zadanie wyraźniej zarysować wcześniejsze dwie kategorie poprzez wykluczenie niektórych zadań z projektu, ponieważ stworzenie i zaprojektowanie całej aplikacji tego typu jest zbyt dużym przedsięwzięciem jak na jedną osobę i zakres czasu przeznaczony na realizację pracy dyplomowej.

Głównymi celami projektu są:

1. Porównanie istniejących rozwiązań.
2. Analiza dostępnych narzędzi.
3. Zaprojektowanie systemu realizującego funkcjonalność tematu pracy.
4. Zaprojektowanie bazy danych odpowiadającej potrzebom aplikacji.
5. Nauka nowych technologii.

Dodatkowymi celami projektu są:

1. Implementacja funkcjonalności wielojęzyczności.
2. Stworzenie oddzielnego interfejsu użytkownika na więcej niż jedną platformę (np. na telefony komórkowe).

Ograniczeniami projektu są:

1. Projekt systemu ma być realizowany z uwzględnieniem tylko indywidualnych użytkowników, pomijając firmy.
2. Funkcjonalność opłat za użycie systemu ma być pominięta.

2 Analiza rynku

Funkcjonalność i rozwiązanie tego typu aplikacji nie jest popularne w Europie. Jedynie w Ameryce północnej istnieje podobne rozwiązanie.

2.1 Podobne rozwiązanie - uShip

Aplikacją na której można się wzorować, jest strona internetowa skoncentrowana głównie na anglojęzycznych państwach z nastawieniem na Amerykę Północną - **uShip**. Strona ta jest dobrze zorganizowana, podział na kategorie jak i sama aplikacja przejrzysta oraz co najważniejsze, oparta jest na rozwiązaniu aukcji. Posiada także ofertę skierowaną dla firm, jednakże jest ona rozwinięta w odwrotny sposób do wymaganego tj. firma tworzy konto, które jest jedynie rozszerzeniem zwykłego konta. Nie ma funkcjonalności posiadania chociażby wielu, zarządzalnych pracowników.

Home > Find Deliveries

Refine Search

Categories [clear](#)

- ☐ Household Goods (493) [+](#)
- ☐ Moving (72) [+](#)
- ☐ Vehicles (742) [+](#)
- ☐ Motorcycles (406) [+](#)
- ☐ Boats (122) [+](#)
- ☐ Plant & Heavy Equipment (111) [+](#)
- ☐ Part Load (223) [+](#)
- ☐ FCL Freight (129) [+](#)
- ☐ Pets (124) [+](#)
- ☐ Horses & Livestock (27) [+](#)
- ☐ Special Care (72) [+](#)
- ☐ Bus. & Industrial (36) [+](#)
- ☐ Food & Agriculture (9) [+](#)
- ☐ Other (0) [+](#)

Collection [clear](#)

City **All**

[City or Postcode, Cntr](#) [📍](#) [🔍](#)
e.g. Berlin, DE or 75020, FR

Radius: **100** km

Delivery [clear](#)

City **All**

[City or Postcode, Cntr](#) [📍](#) [🔍](#)
e.g. Berlin, DE or 75020, FR

Radius: **100** km

Pricing

Price: **0** to **3646**

€0 [📊](#) Max

☐ Offer (260)

☐ Quote (2036)

Map View | ☐ Use map to filter results

Featured Listings [📌](#)

Results: 1 - 5 of 17 | [Next >](#) Sort by **-Select Sort-**

Shipment	Price	Collection	Delivery	Kilometres	Ending
Solcio 21 Open Power Boats Length: 6,50 m	3 quotes (3 active) Low: 1.335€ Quote	Bardolino, Italy Business (with loadin...	12527 Berlin Business (with loadin...	1011	2h 10m
Ufficio Da Cantiere-Baraccate Full Truckload Flatbed Weight: 2000 kg	Quote	Caorso, Italy Ritchie Bros. Auction ...	Montorio Al Vomano, Italy Residential	497	10h 40m
Atlanta Flipper 850 Power Boats Length: 9,00 m	2 quotes (0 active) Quote	34300 Cap D'agde, K, France Port	81700 Puylaurens, France Residential	183	17h 58m
9 Ft Billardtisch Pool Tables Weight: 500 kg	Quote	88662 Überlingen Residential	71686 Remseck Am Neckar Residential	198	21h 23m
Komatsu Pc18mr-2 Excavators Flatbed Weight: 1750 kg	Quote	Caorso, Italy Ritchie Bros. Auction ...	Zona 167, Italy Business (with loadin...	752	23h 46m

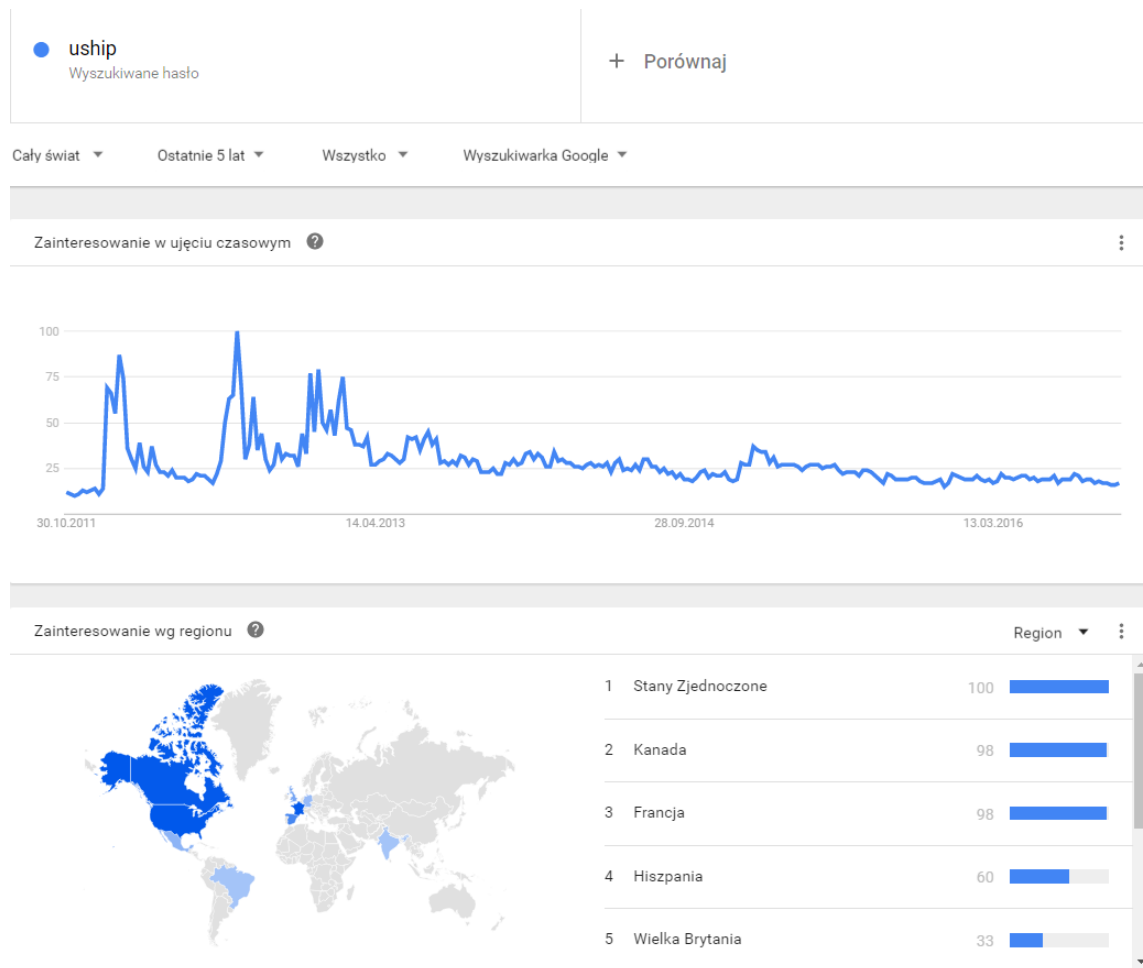
Results: 1 - 5 of 17, Page: 1 2 3 4 next> >> [View More Featured Listings >](#)

Results: 1 - 50 of 2296 | [Next >](#) Sort by **-Select Sort-**

Shipment	Price	Collection	Delivery	Kilometres	Ending
Renault Cars & Light Trucks	4 quotes (3 active) Low: 171€ Quote	Madrid, Spain Residential	Valencia, Spain Residential	359	2m ⚡
Irmier Klavier P 118 Von Leipzi... Pianos	3 quotes (2 active) Quote	04317 Leipzig	06268 Querfurt	89	8m ⚡

Rys. 1: uShip - widok dostępnych aukcji[14]

Powyżej widoczny jest widok dostępnych aukcji. Wyraźnie widać podział na konkretne kategorie i możliwości wyszukiwania (lewa strona) oraz widoki poszczególnych aukcji (prawa strona).

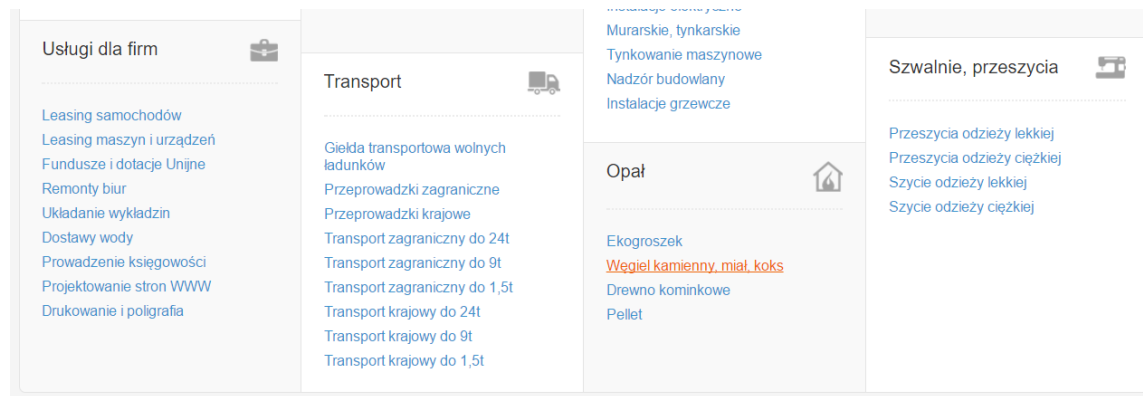


Rys. 2: uShip - statystyki zapytań o stronę w serwisie GoogleTrends[12]

Powyższa grafika pokazuje statystyki częstotliwości zapytań o stronę *uShip* w widocznych krajach. Wynika stąd, że aktualne rozwiązanie jest bardzo popularne w Ameryce północnej, jednakże powoli zaczyna rozprzestrzeniać się na Europę zachodnią. W Polsce to rozwiązanie nie istnieje, dlatego też jest to dobry rynek na rozpoczęcie działalności.

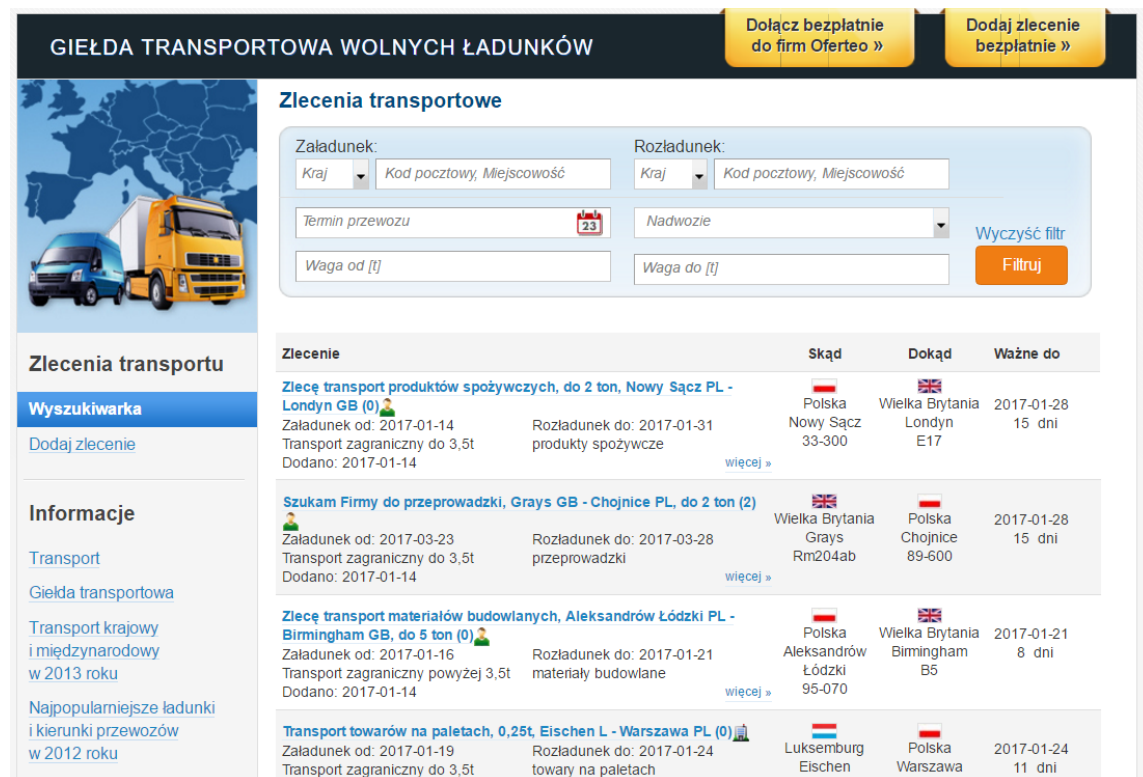
2.2 Podobne rozwiązanie - Oferteo

Portale internetowe o podobnej tematyce na polskim rynku to jedynie **Oferteo**. Funkcjonalność i rozwiązanie tej aplikacji nie jest popularne w kontekście usług przewozu towarów. Strona ta realizuje niewiele założeń naszego systemu, ponieważ pozwala tylko wystawiać oferty, bez większej interakcji z potencjalnymi klientami, które są mocno ograniczone ze względu na kategorie.



Rys. 3: Oferteo - widok dostępnych kategorii[13]

Jak widać, strona nie jest skoncentrowana na przewozach towarów lecz jest to jej jedna z funkcjonalności. Dodatkowo pokazana jest uboga ilość kategorii.



Rys. 4: Oferteo - widok dostępnych aukcji

Oferteo nie posiada zaawansowanej wyszukiwarki aukcji. Ponadto użytkownik nie ma interakcji z ofertą, ponieważ jedynie wystawiony jest (jak już wejdzie się w konkretną aukcję) numer kontaktowy do osoby zlecającej, oraz krótki opis.

2.3 Podobne rozwiązanie - Allegro

Profil tej popularnej strony, wskazywałby na bardzo podobną aplikację. Jednakże główna funkcjonalność, czyli aukcje, działają na odwrotnym założeniu do wymaganego - użytkownik który da największą sumę za towar wygrywa. Nasz system wymaga, aby to najmniej-sza oferta wygrała. Przez to te rozwiązanie nie nadaje się do wykorzystania. Niemniej można wzorować się na dobrze zorganizowanym systemie kategorii oraz wyszukiwania aukcji.

2.4 Projektowana aplikacja

Projektowana aplikacja będzie bardzo podobna do serwisu **UShip** ponieważ dzielą one wspólną ideę funkcjonalności czyli aukcyjne wspomaganie przewozu towarów. Rzeczą wyróżniającą moje rozwiązanie będzie dodatkowa funkcjonalność obsługi firm które mogłyby poprzez projektowaną aplikację rozszerzyć swoją działalność. Rozwiązanie wielojęzyczności w przypadku tej tematyki jest kluczowe ze względu na małe zapotrzebowanie na tego typu usługi w odrębnych państwach. Jak po wstępnej analizie widać w Europie nie ma jeszcze dużej konkurencji, a w Europie wschodniej jest ona zerowa.

2.5 Podsumowanie analizy

Aplikacje o podobnej bądź identycznej głównej funkcjonalności już istnieją, jednakże albo nie spełniają określonych wymagań, albo nie są widoczne i aktywne na polskim i wschodnio-europejskim rynku. Podsumowanie opisanych systemów najlepiej obrazuje poniższa tabela.

Tab. 1: Podsumowanie analizy rynku

Funkcjonalność	uShip	Oferteo	Projektowany system
Importowanie przedmiotów z portali aukcyjnych	●		
Ułatwiony opis przedmiotów (kreator wystawiania przedmiotów, wiele zdjęć)	●	◐	●
Zaznaczenie tras poprzez mapy	●		●
Zaawansowany profil użytkownika (zapisane metody płatności, dane, preferencje wyszukiwania)	●	◐	●
Integracja e-mail, SMS	●	●	●
Profile przewoźników mogą mieć wielu kierowców, którym można zlecać zadania			●
Wyszukiwanie/zlecanie zleceń po wielu różnych właściwościach	●	◐	●
Wyszukiwanie zleceń wokół trasy przejazdu oraz wyznaczenie nowych tras wraz z potencjalnymi nowymi zleceniami	◐		●
Możliwość założenia firmy(konta) przewoźniczej o większych możliwościach z dostępem wielu osób w jednym momencie	◐		●
Integracja z aplikacją mobilną	●		◐
Oszacowanie kosztów i czasu przejazdu	◐		●
Zapisywanie trasy przejazdu, oraz pokazanie jej na mapie	●		●
Możliwość licytacji	●		●
Obsługa wielu języków	◐		●

Oznaczenia

- Aplikacja posiada odpowiednią funkcjonalność w stopniu zadowalającym.
- ◐ Aplikacja posiada odpowiednią funkcjonalność, jednakże nie pasuje ona do profilu projektowanego systemu, bądź jest niepełna.
- ◑ Aplikacja posiada słabo zaimplementowaną funkcjonalność względem wymagań projektowanego systemu.

3 Omówienie koncepcji systemu

3.1 Ogólny opis projektu

Aplikacja ma za zadanie umożliwić użytkownikom wystawienia oferty wykonania usługi przewozu towaru (opisanego przez nich samych). System ma być szeroko i łatwo dostępny. Oferta po wystawieniu, ma być widoczna dla wszystkich osób, tych zalogowanych jak i niezalogowanych. Osoby które chciałyby podjąć się wykonania takiej usługi muszą wygrać przetarg, który jest w postaci aukcji, działającej na zasadzie kto zaproponuje za wykonanie usługi mniej, ten wygrywa. Wystawiona usługa musi być dokładnie opisana, tak aby zleceniobiorca był w stanie bez kontaktu z zleceniodawcą, używając tylko dostępnej charakterystyki, wykonać usługę. Program dodatkowo musi poza zwykłymi użytkownikami obsługiwać firmy. Firma może zarządzać przypisanymi do niej użytkownikami i rozdysponowywać wygranymi zleceniami. Konta użytkowników które podlegają firmie, działają w jej imieniu - każdy komentarz, wygrana aukcja ma być widoczne dla zarządzającego przedsiębiorstwa.

3.2 Docelowi użytkownicy

Aplikacja ma wyróżniać docelowo trzy rodzaje użytkowników:

1. Zwykły użytkownik - Jest dowolna osoba która może korzystać z całej podstawowej funkcjonalności aplikacji czyli wystawianie i licytacja aukcji, zarządzanie swoim kontem. Użytkownik nie może licytować własnych wystawionych aukcji.
2. Firma - jest to specjalny użytkownik, do jego założenia wymagany jest kontakt z administratorem, bądź moderatorem aplikacji. Konto tego typu może posiadać i zarządzać wieloma kontami zwykłych użytkowników, uprzednio przypisanych do niego. W momencie przypisania, każda licytacja, wygrana bądź komentarz jest wspólny wraz z firmą zarządzającą.
3. Moderator - jest to osoba posiadająca prawa do administrowania innymi kontami (zmiana haseł, zakończenie licytacji...) oraz jest pierwszą osobą kontaktu wszystkich rodzajów użytkowników z właścicielem aplikacji.

3.3 Cele funkcjonowania systemu

Z początku głównym celem rozwoju systemu będą osoby indywidualne, jednakże z czasem ciężar ten przesunąłby się na przedsiębiorstwa. W Polsce istnieje wiele różnych firm transportowych (od typowych dużych transportów do taksówek). Skoncentrowanie się z początku na osobach indywidualnych stworzyłoby bazę użytkowników która skłoniłaby małe i średnie przedsiębiorstwa do użycia aplikacji ze względu na to, że teraz sami będą mogli wybierać towary do transportu, a użytkownicy nie muszą się martwić o szukanie odpowiedniej firmy przewozowej, gdy chodzi o nietypowy towar.

3.4 Opis funkcjonalności

Jak korzystać z programu

Użytkownik powinien bez tworzenia konta móc przejrzeć wszystkie aktualne aukcje, jednakże licytacja oraz wystawienie byłoby możliwe po uprzedniej rejestracji konta oraz zalogowaniu się. Program ma być łatwo dostępny dla użytkowników, przejrzysty i zrozumiały. Wszystkie wiadomości skierowane do użytkownika mają być maksymalnie uproszczone i jednoznaczne. Licytacja aukcji ma działać na zasadzie wypisania oferowanej kwoty oraz potwierdzenia jej, a wystawienie poprzez uzupełnienie podstawowych opisów przedmiotu oraz charakterystyki umożliwiającej jego łatwe wyszukanie.

3.4.1 Wymagania funkcjonalne

1. System musi posiadać funkcjonalność aukcyjną, gdzie jeden użytkownik wystawia aukcję, a drugi ją licytuje.
2. System musi posiadać rozbudowaną funkcjonalność wyszukiwania aukcji uwzględniającą: wyszukiwaną frazę, charakterystykę przedmiotu jak i aukcji.
3. System ma powiadamiać użytkownika o wydarzeniach - wygrana aukcja, termin odbioru przesyłki - drogą e-mail bądź wiadomością tekstową SMS.
4. Punkt odbioru i dostarczenia towaru opisanego w aukcji, ma być widoczny na mapie z poziomu aplikacji.

5. System ma archiwizować dane, w szczególności aukcje.
6. System musi posiadać funkcjonalność obsługi firm, gdzie jedno przedsiębiorstwo posiada wielu użytkowników do zarządzania.
7. System ma posiadać możliwość wyboru języka z poziomu interfejsu użytkownika.
8. System musi posiadać narzędzia umożliwiające wysyłanie wiadomości SMS oraz e-mail.

3.4.2 Wymagania niefunkcjonalne

1. System ma być możliwie jak najbardziej modularny tj. funkcjonalność logiczna ma być oddzielona od wizualnej, aby łatwo można było dodać nowe interfejsy klienckie, używając zaimplementowanego już rozwiązania. Proponowanym rozwiązaniem jest implementacja aplikacji według wzorca *MVC - Model View Controller*.
2. System ma być dostępny za pomocą protokołu *HTTP*, oraz przez przeglądarkę internetową, dlatego też powinna być to aplikacja webowa.
3. System musi kodować hasła użytkowników, tak, aby nawet po ewentualnym wycieku danych nie były one możliwe do odczytania.
4. System ma być używalny na aktualnie trzech najpopularniejszych przeglądarkach tj. *Google Chrome*, *Microsoft Internet Explorer* oraz *Mozilla Firefox* w wersjach nie starszych niż dwa lata od aktualnej.
5. Każdorazowe zrestartowanie systemu nie może mieć wpływu na przechowywane dane oraz istniejące aukcje.
6. Kod aplikacji powinien być jasny i przejrzysty. Definicje klas i metod powinny, po przeczytaniu, sugerować swoją funkcjonalność w programie.
7. System musi być w pełni konfigurowalny.
8. Obsługa systemu powinna być nieskomplikowana oraz intuicyjna.
9. Wszystkie wymagane funkcjonalności muszą współpracować z sobą.
10. Słowniki systemu powinny być łatwo dostępne oraz modyfikowalne.
11. Użytkownik wystawiający aukcje powinien posiadać podstawowe narzędzia edycji tekstu (zmiana czcionki, kolor tekstu, listowanie...) przy tworzeniu opisu aukcji.
12. System należy zaimplementować używając możliwie tylko darmowych rozwiązań, preferowanymi narzędziami są:
 - (a) Baza danych: *Microsoft SQL Server*, *MariaDB*, *MySQL*, *PostgreSQL*.
 - (b) System logiczny: *Java*, *C#*, *C++* z uwzględnieniem *frameworków*, odpowiednio *Spring* bądź *.NET*
 - (c) Aplikacja kliencka: *AngularJS*, *Angular2*, *PHP*
13. System musi być odporny na ataki typu *SQL injection*.
14. System musi posiadać edytowalną możliwość ustawienia długości sesji poszczególnych użytkowników.
15. System powinien posiadać narzędzia monitorujące podstawowe czynności takie jak ilość zapytań na daną jednostkę czasu, ilość użytkownika czy historię akcji użytkownika.
16. Użytkownik powinien otrzymać oszacowane koszty przewozu towaru.

4 Analiza dostępnych narzędzi

Ze względu na upodobania i umiejętności, narzędzia programistyczne użyte do tworzenia serwera zostały zwężone do technologii związanych z Javą.

4.1 Baza danych

Pod uwagę zostały wzięte dwie technologie - *Microsoft SQL Server* oraz *MySQL* z uwagi na doświadczenie. Pierwsza z nich, technologia *Microsoftu* jest renomowaną i bardzo dobrze znaną bazą danych, jednakże faworyzowanymi przez nią narzędziami są pochodzące od tej samej firmy rozwiązania. Ograniczeniem tej bazy są środowiska na których można ją łatwo zainstalować oraz obsłużyć z których wykluczony jest system *Linux*.

Następną technologią jest *MySQL*. Popularna i dobrze znana każdemu baza danych. Jest łatwa w obsłudze oraz można ją zainstalować na każdym środowisku. Darmowa wersja zapewnia wszystkie niezbędne narzędzia, począwszy od partycji kończąc na statystykach. Finalnie jako technologia bazodanowa została wybrana baza *MySQL*, głównie ze względu na łatwą możliwość instalacji na większości używanych systemach.

4.2 Serwer

Serwer logiczny aplikacji będzie wykonany w technologii *Java* przy użyciu rekomendowanego *frameworku* - *Spring* ze względu na moje umiejętności. Jednakże system ma być wielomodułowy, dlatego też należy wybrać technologię, którą serwer będzie udostępniać informacje. Pod uwagę zostały wzięte dwie możliwości. Pierwszą z nich jest udostępnianie wiadomości za pomocą wbudowanych narzędzi *Spring* - *RESTful API* poprzez przekazywanie danych używając struktury typu *JSON* - *JavaScript Object Notation*. Drugim rozwiązaniem jest użycie także zaimplementowanego protokołu w *Spring'u* - *SOAP*.

Rozwiązania te różnią się głównie sposobem prezentacji danych. Pierwsze z nich wystawia dane, używając łatwego do przetworzenia obiektu poprzez języki używane przy tworzeniu stron internetowych. Drugie rozwiązanie oferuje pokazanie informacji poprzez język znaczników - *XML*. Ze względu, iż architektura *SOAP* jest już dość stara i nowsze technologie mogą jej nie uwzględniać w swojej implementacji, zdecydowałem się na nowszy i łatwy czytelny sposób wystawiania informacji poprzez technologię *RESTful API*.

4.3 Aplikacja kliencka

W wymaganiach aplikacji zostały uwzględnione trzy technologie *AngularJS*, *Angular2* oraz *PHP*.

PHP

Stare, jednakże ciągle rozwijane rozwiązanie. Ma one swoje problemy, głównie związane ze skalowalnością. Jednakże każdy większy problem ma swoje znane rozwiązanie bądź sposób ominięcia go, ale zazwyczaj są to bardzo skomplikowane procedury. Ze względu na możliwą wielkość aplikacji oraz ważny jej aspekt - skalowalność, język ten nie zostanie użyty.

AngularJS

Język stworzony przez korporację *Google*. Jest to bardzo dobra technologia do tworzenia dużych aplikacji internetowych. Szczególnie jest ona używana przez większe firmy które tworzą projekty wymagające dużego nakładu czasu oraz czytelnej struktury. Cechuje on się, w odróżnieniu do innych technologii *webowych*, obiektową hierarchią poszczególnych modułów aplikacji, oraz kontrolą typów poprzez wprowadzenie języka *TypeScript* (który kompiluje kod źródłowy do języka *JavaScript*). Istnieje bardzo dużo udostępnionych rozwiązań poprzez innych użytkowników którzy także aktywnie pomagają w rozwiązywaniu problemów.

Angular2

Jest to kontynuacja technologii *AngularJS*, jednakże ilość zmian jest znacząca na tyle, że nie da się przeprojektować aplikacji napisanej w *AngularJS* do *Angular2*. Wprowadzono w niej dużo nowości oraz ulepszono nowe rozwiązania. Całe skupienie firmy *Google* jest skoncentrowane właśnie na tej technologii. Jest ona relatywnie młoda, finalna wersja została wprowadzona dopiero na przełomie października i listopada, jednakże istnieje już spora grupa aktywnych użytkowników którzy rozwijają tę technologię poprzez udostępnianie swoich rozwiązań. Ze względu na to, iż projektowana aplikacja będzie dużym systemem zdecydowałem się na rozwiązanie firmy *Google*. Jednakże problemem było zdecydowanie się czy użyć nowszych czy starszych technologii. Za *AngularJS* stał jej wiek i stabilność, a za *Angular2* łatwiejsze i nowocześniejsze rozwiązania kosztem mniejszej stabilności. Finalnie wybrana

została technologia *Angular2* ze względu, iż w tej technologii *Google* pokłada największe nadzieje oraz pieniądze na rozwój, czego naturalnym efektem jest stopniowe porzucanie *AngularJS*.

4.4 Podsumowanie wyboru technologii

Wszystkie technologie są darmowe oraz bogato opisane przykładami na swoich stronach internetowych, które zostały załączone na poniższej liście. Każda z technologii posiada na stronie domowej własną dokumentację jak i można znaleźć dużo przykładów i opisów tworzonych przez samych, niezależnych, internautów. Dodatkowo wymienione, nieopisane wyżej narzędzia są standardowymi wyborami przy pracy z odpowiadającymi im *frameworkami* np. *Java - Spring + Hibernate*, dowolna technologia aplikacji klienckiej dostępnej poprzez przeglądarkę internetową - *Bootstrap*.

Wybrane technologie

1. Baza danych - *MySQL*[8]
2. Back-end - *Java* 8[10] wraz ze wsparciem *frameworków* - *Spring (RESTful Services, Security OAuth2, Springboot)*[6], *Hibernate*[11], oraz system zarządzania i budowy projektu *Apache Maven*[3]
3. Front-end - *Angular2*[9] pod wsparciem ulepszanego JavaScript - *TypeScript* - specjalnie pod tę technologię. Dodatkowo do upiększania strony *Bootstrap* 3[5].

Implementacja projektu będzie tworzona przy pomocy środowiska programistycznego *IntelliJ IDE*[1] na licencji studenckiej. Projekt będzie zapisywany na darmowym repozytorium *BitBucket*[4] umożliwiającym łatwą kontrolę wersji. Obydwa wybory są subiektywne, nie wpływają na realizację jak i utrzymanie projektu.

5 Projekt systemu

5.1 Architektura systemu

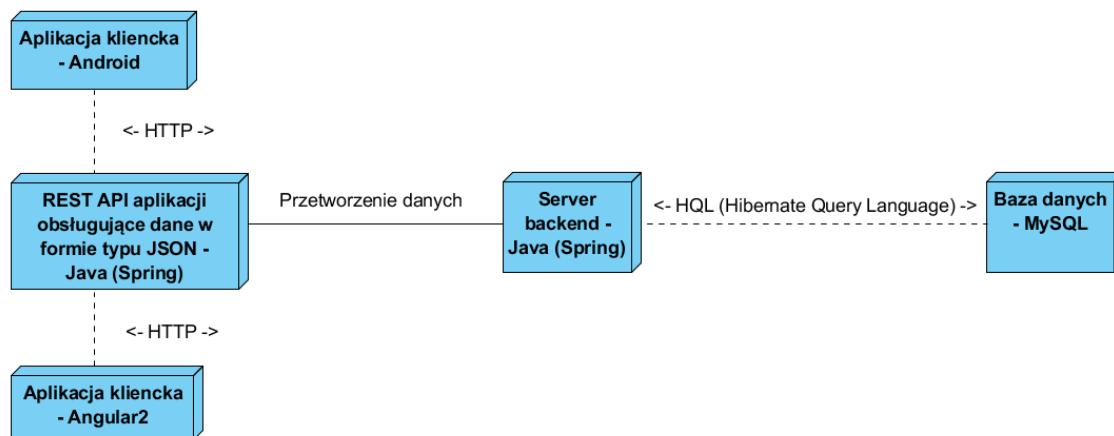
System będzie działał w oparciu o model *MVC* - *Model View Controller*[15]. Architektura ta wyróżnia następujące główne węzły aplikacji:

Model: Baza danych w technologii *MySQL* przechowująca dane.

Kontroler: Serwer napisany w technologii *Java* w oparciu o *frameworki Spring* oraz *Hibernate*, który ma za zadanie przetwarzać wszystkie informacje.

Widok: Podstawowym widokiem dla zwykłego użytkownika będą aplikacje klienckie napisane w technologii *Angular2* oraz na urządzenia mobilne *Android*. Jednakże te aplikacje będą korzystały i odpowiednio przystosowywały dane dla klienta które będą wystawiane na widoku podstawowym czyli *RESTful API* - interfejsie aplikacji do którego można będzie zwrócić się odpowiednimi zapytaniami *HTTP*, np. *GET*, *POST*..., aby pobrać bądź zmodyfikować dane znajdujące się w bazie danych.

Powyższe rozwiązanie graficznie przedstawia poniższy rysunek:



Rys. 5: Architektura systemu

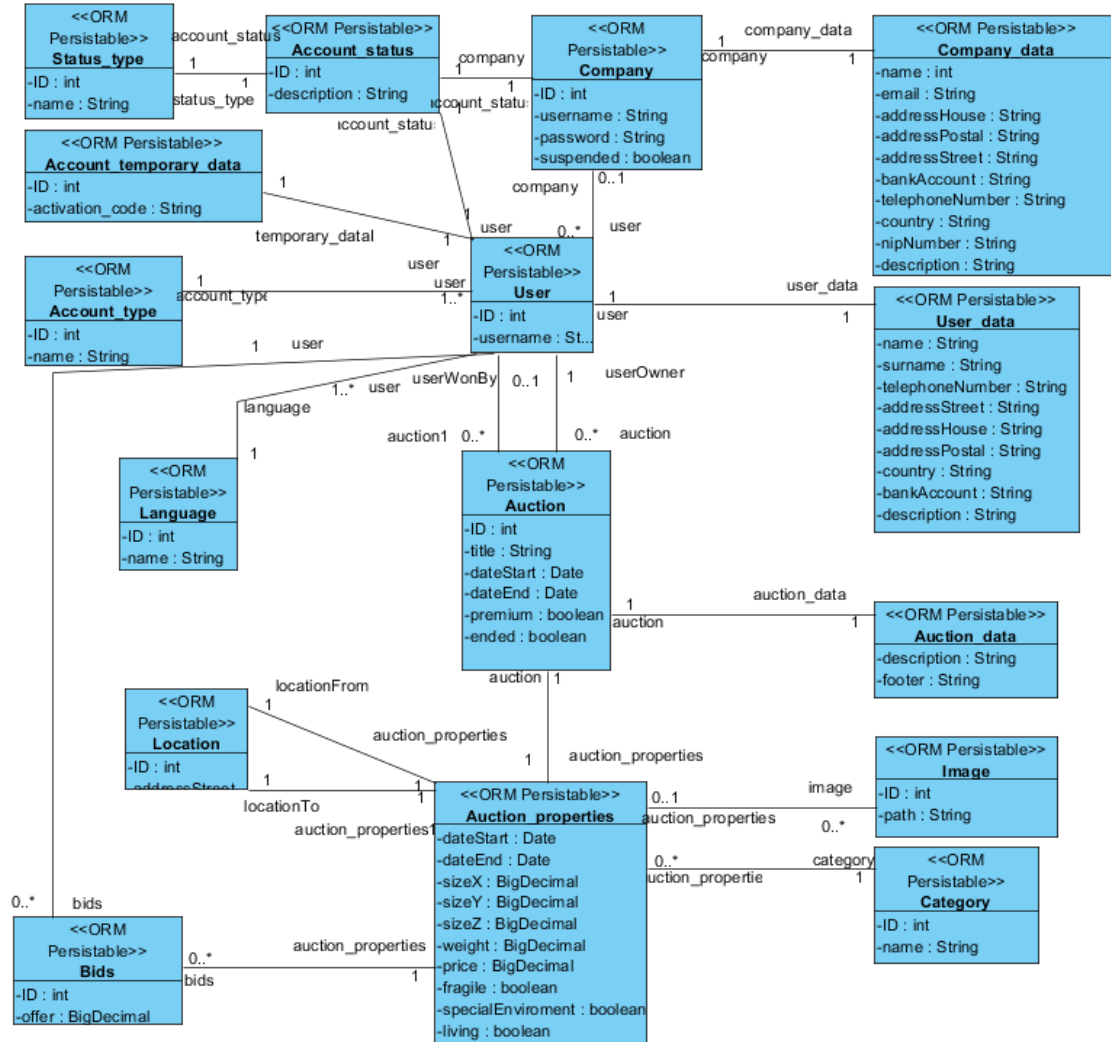
Optymalizacja i skalowalność aplikacji

Aplikacja ma za zadanie przetworzyć wiele względnie prostych zapytań w krótkiej jednostce czasu, dlatego też system należy przygotować na taką okoliczność. Poniżej znajdują się lista kluczowych dla aplikacji rozwiązań które umożliwią płynną pracę aplikacji przy dużym obciążeniu:

1. Baza danych jak i *serwer backendowy* powinny zostać uruchomione jak najbliżej siebie np. w jednej podsi sieci, albo najlepiej na jednej maszynie, ponieważ od reakcji między aplikacją serwerową, a bazą zależy czas odpowiedzi na zapytanie przychodzące z zewnątrz. Typowego przetwarzania danych nie jest dużo, więc można ten aspekt pominąć.
2. Niektóre zapytania będą częściej wykorzystywane niż inne dlatego należy stale monitorować obciążenie poszczególnych rodzajów zapytań jak i *endpointów* aplikacji, aby doraźnie optymalizować bazę danych jak i same zapytania. Proponowanym rozwiązaniem dla takiego problemu jest przeniesienie bazy danych na technologie *Oracle*. Bazy danych w tej technologii słyną z bardzo dobrej optymalizacji zapytań. Ostatecznym rozwiązaniem byłoby rozdzielenie bazy danych na kilka węzłów, jednakże wtedy skomplikowałaby się struktura samych zapytań, a na dodatek należałoby zmienić charakterystykę samej bazy na nierelacyjną (takie rozwiązanie stworzyłby podwaliny pod system rozproszony).

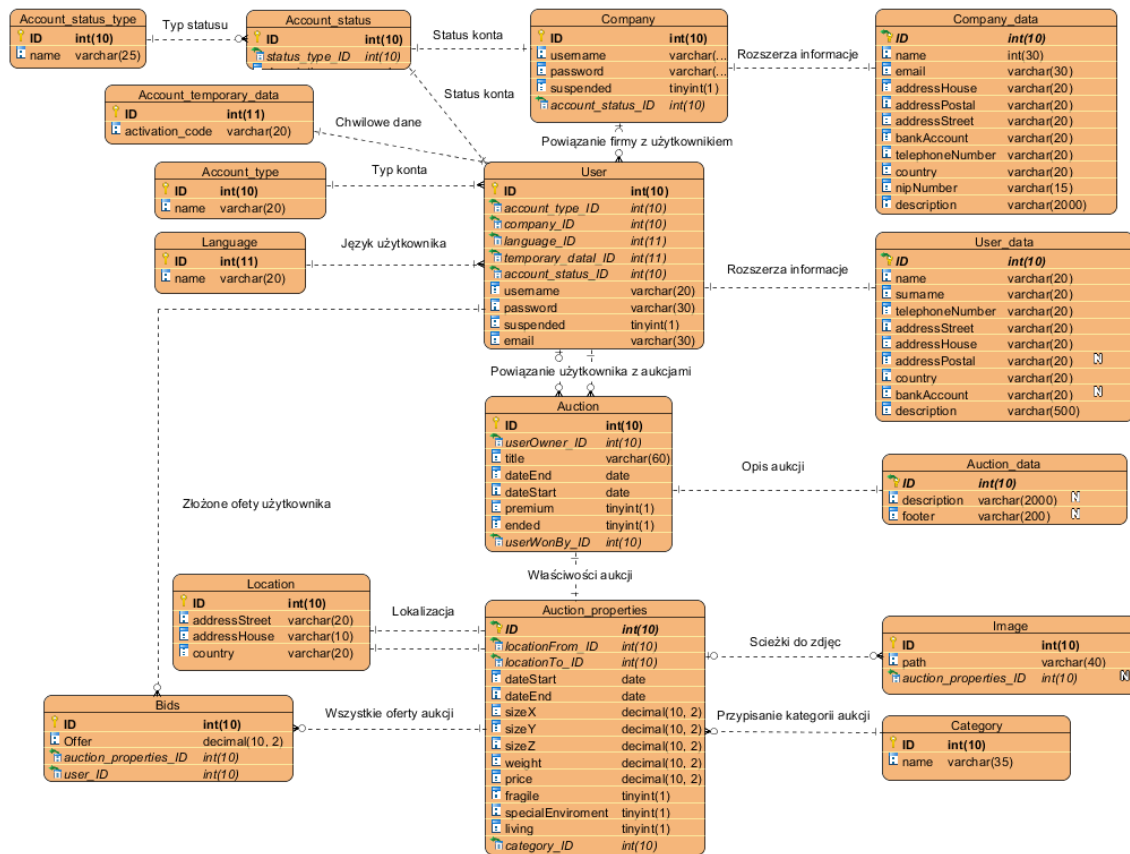
5.2 Diagramy klas

Diagram klas ściśle łączy się z diagramem związków encji, dlatego też szczegółowy opis widocznych rysunków znajduje się w sekcji diagramy związków encji.



Rys. 6: Diagram klas

5.3 Diagramy związków encji



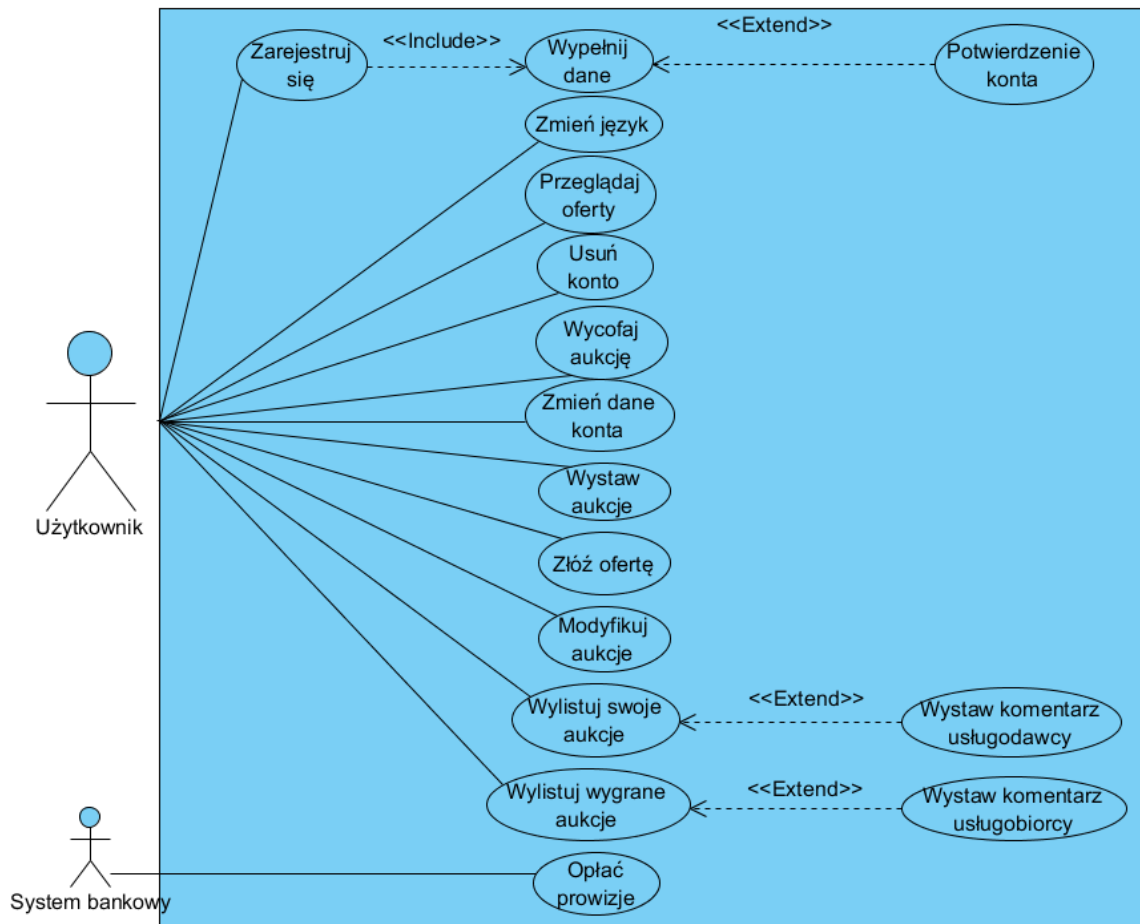
Rys. 7: Diagram związków encji

Powyższy diagram związków encji należy rozumieć następująco:

Firma (wraz ze swoimi danymi) może posiadać wielu użytkowników. Każdy użytkownik posiada swoje konto (wraz z danymi), któremu można przypisać język oraz specyficzny typ konta jak np. premium, albo zablokowane. Każde konto może mieć wiele aukcji swoich, jak i wiele aukcji które licytuje. Sama aukcja ma swoje specyficzne właściwości jak i sam opis oraz historię wszystkich zaliczowanych sum pieniędzy.

Diagram ten pokazuje główną część bazy danych, dodatkowo istnieje kilka encji związanych między innymi z autoryzacją (tokeny), wspomaganiem *frameworku Hibernate*.

5.4 Diagramy przypadków użycia



Rys. 8: Diagram przypadków użycia - użytkownik aplikacji

Szczegółowe definicje przypadków użycia dla diagramu użytkownika

1. Zarejestruj się

Cel użycia: Rejestracja nowego konta do użytkowania aplikacji

Warunek początkowy: Brak

- (a) Wejście na stronę rejestracji.
- (b) Wypełnienie wymaganych danych i potwierdzenie akcji - nazwa użytkownika, hasło, poczta e-mail, imię, nazwisko, numer telefonu, dane adresowe.
- (c) Przesłanie danych do *serweru backendowego* poprzez odpowiednie żądanie *HTTP*.
- (d) Walidacja danych.
- (e) zapisanie danych do bazy danych poprzez *serwer backendowy*.

- (f) Wysłanie maila z kodem aktywującym konto do użytkownika w celu weryfikacji konta e-mail.
- (g) (Opcjonalne) Wpisanie kodu aktywacyjnego i porównanie go z kodem, który został wysłany, w przypadku pozytywnym aktywacja konta, negatywnym – możliwość wysłania maila z kodem ponownie.

2. Zmień język

Cel użycia: Zmiana wyświetlanego języka

Warunek początkowy: Brak

- (a) Wybranie wyświetlanego języka z możliwych, widoczne informacje są zapisane po stronie klienta, więc nie ma potrzeby odwoływania się do bazy danych.

3. Przeglądaj oferty

Cel użycia: Wyszukanie aukcji przy użyciu dostępnych narzędzi

Warunek początkowy: Brak

- (a) Wybierz kategorie – zaznaczenie kategorii, z których aukcje mają zostać wyświetlone.
- (b) Wpisz frazę do wyszukiwarki – wpisanie słów kluczowych, po których zostaną wyszukane aukcje.
- (c) Przesłanie danych do *serweru backendowego* poprzez odpowiednie żądanie *HTTP*.
- (d) Odczytanie danych z bazy i wysłanie ich do klienta.

4. Usuń konto

Cel użycia: Usunięcie konta

Warunek początkowy: Zalogowanie się na istniejące konto

- (a) Ponowne zalogowanie się do konta w celach weryfikacji - otrzymanie dodatkowego tokena umożliwiającego wykonywanie dodatkowych operacji przez krótki okres czasu.
- (b) Potwierdzenie usunięcia konta.

5. Wycofaj aukcje

Cel użycia: Anulowanie istniejącej aukcji

Warunek początkowy: Zalogowanie się na istniejące konto

- (a) Wybieranie aukcji do anulowania
- (b) Ponowne zalogowanie się do konta w celach weryfikacji - otrzymanie dodatkowego tokena umożliwiającego wykonywanie dodatkowych operacji przez krótki okres czasu.
- (c) Potwierdzenie anulowania aukcji.

6. Zmień dane konta

Cel użycia: Zmiana danych konta użytkownika

Warunek początkowy: Zalogowanie się na istniejące konto

- (a) Edycja dostępnych danych.
- (b) Ponowne zalogowanie się do konta w celach weryfikacji - otrzymanie dodatkowego tokena umożliwiającego wykonywanie dodatkowych operacji przez krótki okres czasu.
- (c) Potwierdzenie zmiany danych.

7. Wystaw aukcję

Cel użycia: Wystawienie aukcji przez użytkownika

Warunek początkowy: Zalogowanie się na istniejące konto

- (a) Pobranie wymaganych danych do wystawienia aukcji .
- (b) Walidacja danych w kontekście poprawności oraz bezpieczeństwa.
- (c) Potwierdzenie wystawienia aukcji wraz z ekranem podglądu.

8. Złóż ofertę

Cel użycia: Złożenie oferty licytacyjnej

Warunek początkowy: Zalogowanie się na istniejące konto oraz wejście na stronę konkretnej aukcji

- (a) Wypełnienie pola z informacją o składanej ofercie pieniężnej.
- (b) Walidacja danych w kontekście poprawności (kwota musi być mniejsza od ostatniej najmniejszej oferty) oraz bezpieczeństwa.
- (c) Potwierdzenie.

9. Modyfikuj aukcję

Cel użycia: Modyfikacja istniejącej i aktywnej aukcji

Warunek początkowy: Zalogowanie się na istniejące konto

- (a) Wybranie aukcji z pośród własnych wystawionych, aktualnych aukcji.
- (b) Możliwość edycji danych aktualnej aukcji jedynie w zakresie dodanej stopki, inne dane nie mogą zostać zmienione.
- (c) Zapis do bazy – zapisanie zmian do bazy danych.

10. Wylistuj swoje/wygrane aukcje

Cel użycia: Wyświetlenie wszystkich własnych/wygranych aukcji

Warunek początkowy: Zalogowanie się na istniejące konto

- (a) Wejście na odpowiednią zakładkę.
- (b) Wyszukanie aukcji poprzez dostępne narzędzia.

11. Wystaw komentarz usługodawcy

Cel użycia: Wystawienie komentarza usługodawcy

Warunek początkowy: Zalogowanie się na istniejące konto oraz wybranie zakończonej aukcji w celu skomentowania

- (a) Skomentowanie realizacji zlecenia poprzez użytkownika.
- (b) Potwierdzenie.

12. Wystaw komentarz usługobiorcy

Cel użycia: Zalogowanie się na istniejące konto oraz wybranie zakończonej aukcji w celu skomentowania

Warunek początkowy: Zalogowanie się na istniejące konto.

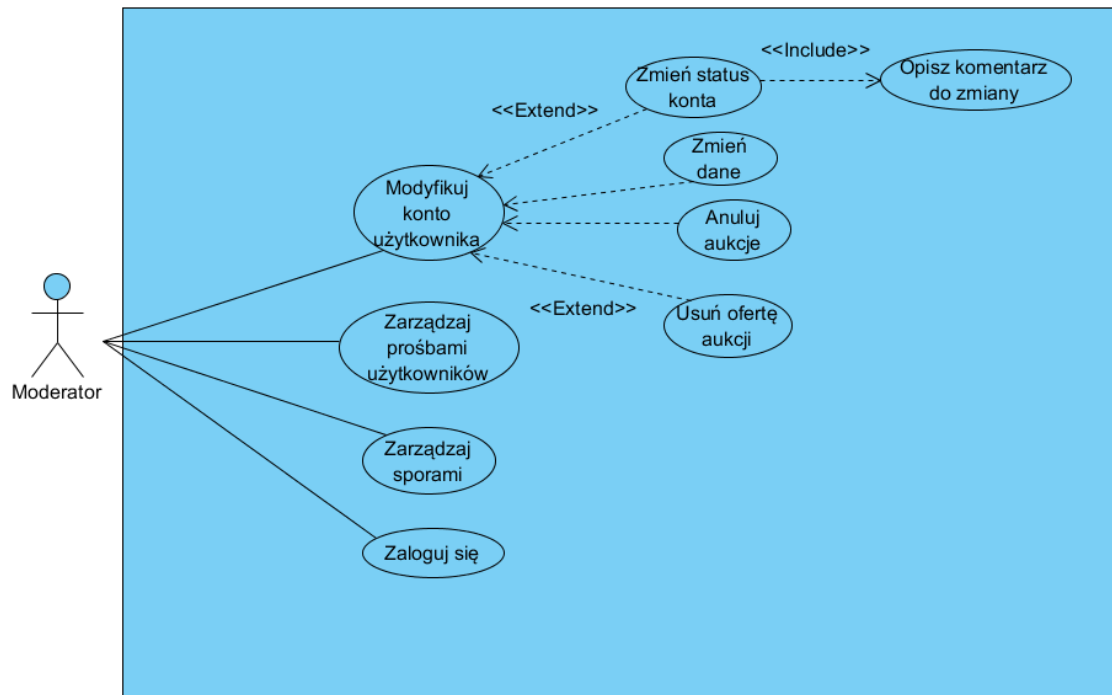
- (a) Skomentowanie realizacji zlecenia poprzez usługodawcę.
- (b) Potwierdzenie.

13. Opłać prowizję

Cel użycia: Opłacenie prowizji za korzystanie z usług aplikacji

Warunek początkowy: Zalogowanie się na istniejące konto

- (a) Wyświetlenie wszystkich niezapłaconych prowizji oraz ich podsumowanie.
- (b) Przekieruj na stronę banku – przekierowanie płatności do zewnętrznych aplikacji bankowych oferujących taką funkcjonalność.
- (c) Otrzymanie potwierdzenia zapłaty z aplikacji obsługującej płatności.



Rys. 9: Diagram przypadków użycia - moderator aplikacji

Szczegółowe definicje przypadków użycia dla diagramu moderatora**1. Zmień status konta**

Cel użycia: Usunięcie konta

Warunek początkowy: Zalogowanie się na istniejące konto oraz wybranie zakładki Modyfikuj konto użytkownika

- (a) Zmiana status konta użytkownika (np. z aktywnego na zablokowany).
- (b) Opisanie komentarza przyczyny zmiany statusu konta.

2. Zmień dane

Cel użycia: Zmiana danych konta użytkownika

Warunek początkowy: Zalogowanie się na istniejące konto oraz wybranie zakładki Modyfikuj konto użytkownika

- (a) Zmiana danych konta użytkownika (np. hasła bądź adresu e-mail).

3. Anuluj aukcje

Cel użycia: Anulowanie aktywnej aukcji

Warunek początkowy: Zalogowanie się na istniejące konto oraz wybranie zakładki Modyfikuj konto użytkownika

- (a) Zakończenie wybranej, aktywnej aukcji.

4. Zarządzaj prośbami użytkowników

Cel użycia: Zarządzanie prośbami użytkowników

Warunek początkowy: Zalogowanie się na istniejące konto

- (a) Wejście na odpowiednią zakładkę.
- (b) Przeczytanie kolejnej wolnej prośby oraz próba rozwiązania jej.
- (c) Przesłanie odpowiedzi ze statusem rozwiązania do użytkownika.
- (d) Zamknięcie prośby.

5. Zarządzaj sporami

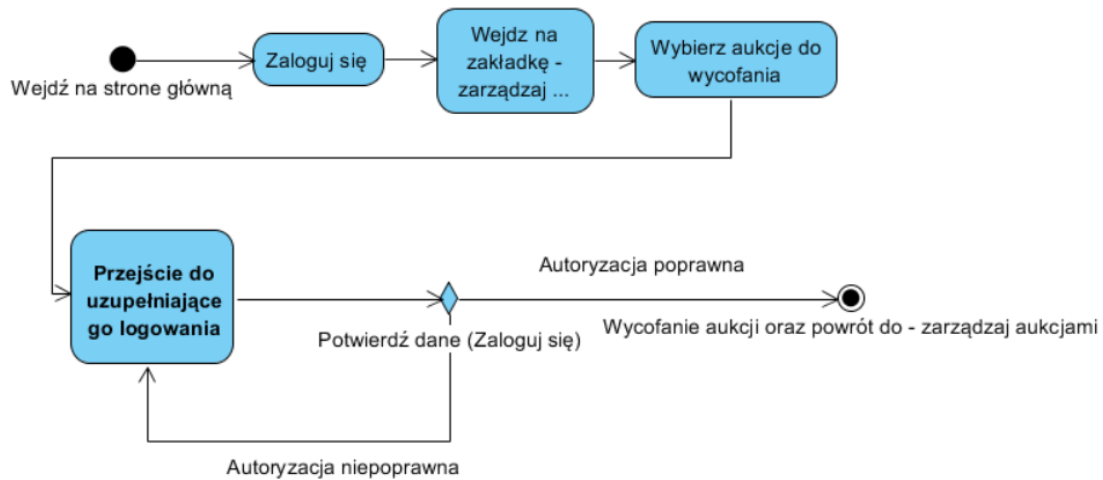
Cel użycia: Zarządzanie sporami użytkowników

Warunek początkowy: Zalogowanie się na istniejące konto

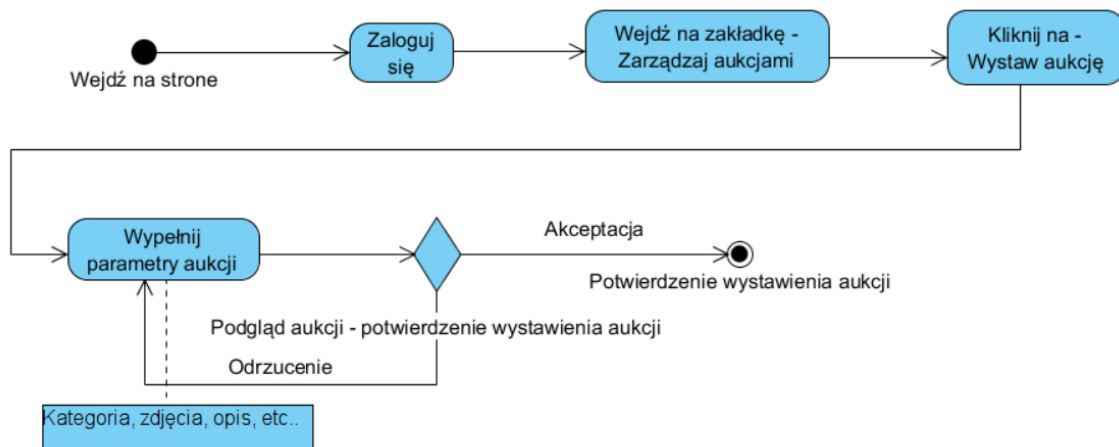
- (a) Wejście na odpowiednią zakładkę.
- (b) Przeczytanie kolejnej wolnej prośby oraz próba rozwiązania jej.
- (c) Przesłanie odpowiedzi ze statusem rozwiązania do użytkownika.
- (d) Zamknięcie prośby.

5.5 Diagramy aktywności

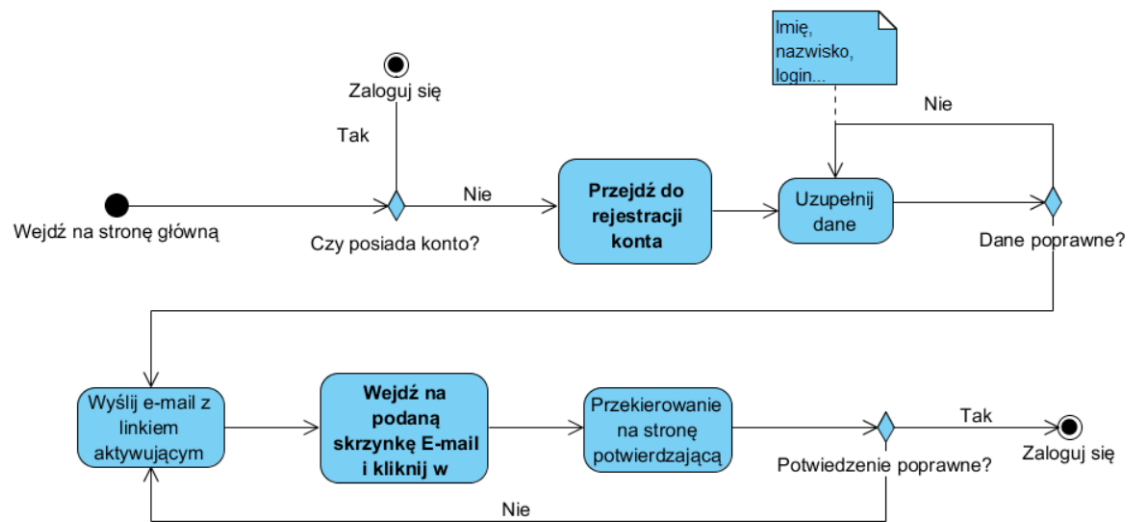
Są to tylko przykładowe przypadki, ponieważ każda akcja użytkownika jest trywialna i nie wymaga żadnych skomplikowanych działań (kilka kliknięć myszki oraz wypełnienie widocznego pola). Ze względu na ich trywialność i powtarzalność, nie ma sensu tworzyć logicznie takich samych kolejnych diagramów aktywności.



Rys. 10: Diagram aktywności - wycofanie aukcji



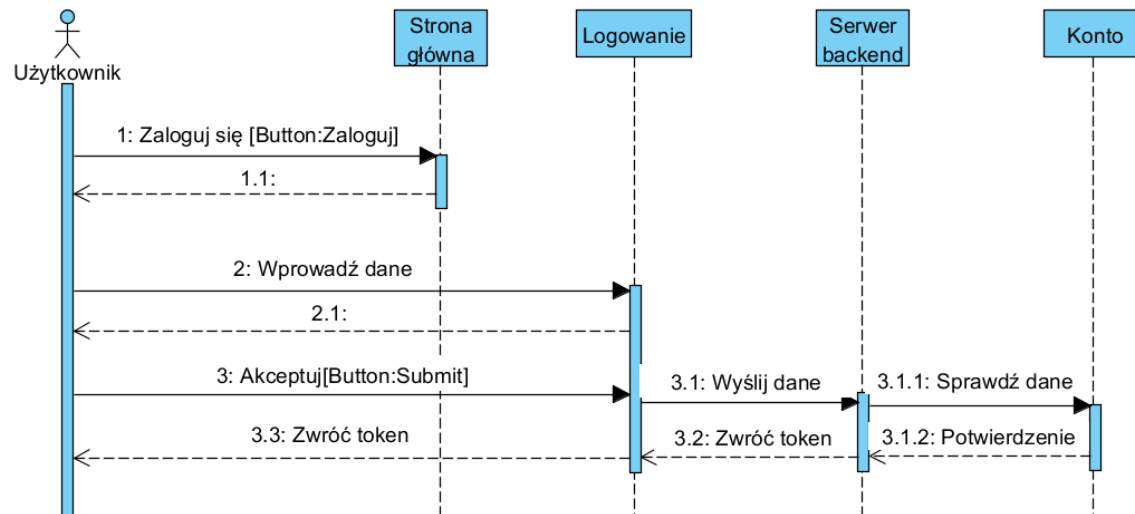
Rys. 11: Diagram aktywności - wystawienie aukcji



Rys. 12: Diagram aktywności - rejestracja konta

5.6 Diagramy sekwencji

Poniższe diagramy pokazują wszystkie moduły działania aplikacji poprzez cały swój zakres. Każda sekwencja w tej aplikacji jest bardzo podobna do siebie ponieważ wszystkie akcje działają w oparciu o model MVC (Model-View-Controller) co oznacza, że każde pytanie ze strony użytkownika, które wymaga jakichkolwiek danych musi przejść przez *serwer backendowy*, który przechowuje całą logikę aplikacji oraz walidacje. Aplikacja kliencka posiada walidacje takie, aby ułatwić klientowi przejście przez walidacje serwerowe, logiki decyzyjnej nie posiada.

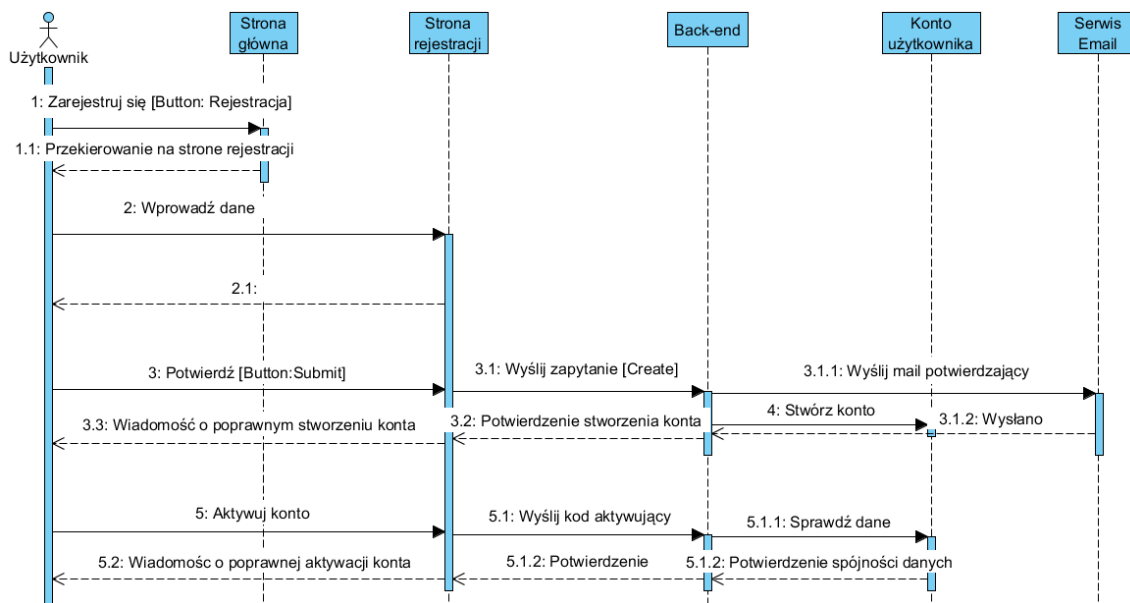


Rys. 13: Diagram sekwencji - logowanie

Powyższy diagram pokazuje sekwencje logowania klienta do aplikacji.

Oznaczenia tzw. linii życia

1. Strona główna - jest to *endpoint* strony głównej/reprezentacyjnej która ukazuje się użytkownikowi po wejściu na stronę
2. Logowanie - jest to *endpoint* logowania do aplikacji
3. Serwer backend - jest to serwer aplikacji napisany w języku *Java* który posiada całą logikę decyzyjną jak i wystawione *RESTful API*
4. Konto - oznacza konto klienckie w rozumieniu danych przechowywanych w bazie danych

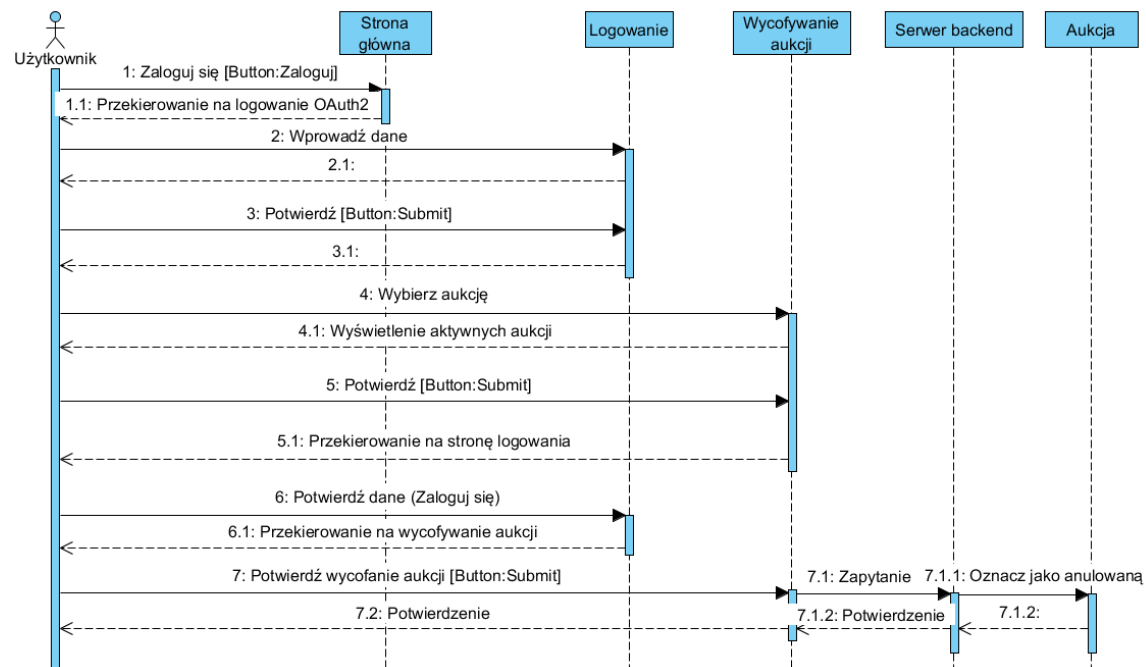


Rys. 14: Diagram sekwencji - rejestracja użytkownika

Powyższy diagram pokazuje sekwencje rejestracji nowego użytkownika aplikacji.

Oznaczenia tzw. linii życia

1. Strona główna - jest to *endpoint* strony głównej/reprezentacyjnej która ukazuje się użytkownikowi po wejściu na stronę
2. Strona rejestracji - jest to *endpoint* rejestracji nowego konta
3. Back-end - jest to serwer aplikacji napisany w języku *Java* który posiada całą logikę decyzyjną jak i wystawione *RESTful API*
4. Konto użytkownika- oznacza konto klienckie w rozumieniu danych przechowywanych w bazie danych
5. Serwis Email - jest to usługa za pośrednictwem której można wysłać pocztę elektroniczną na wskazany adres



Rys. 15: Diagram sekwencji - anulowanie aukcji

Oznaczenia tzw. linii życia Powyższy diagram pokazuje sekwencje anulowania aukcji klienta.

1. Strona główna - jest to *endpoint* strony głównej/reprezentacyjnej która ukazuje się użytkownikowi po wejściu na stronę
2. Logowanie - jest to *endpoint* logowania do aplikacji
3. Wycofywanie aukcji - jest to strona na której użytkownik może wybrać aukcje do anulowania
4. Serwer backend - jest to serwer aplikacji napisany w języku *Java* który posiada całą logikę decyzyjną jak i wystawione *RESTful API*
5. Aukcja - jest to reprezentacja aukcji danego użytkownika

5.7 Opis mechanizmów bezpieczeństwa

Mechanizmy zabezpieczające hasła

Docelowo wersja produkcyjna aplikacji posiadać kodowane hasła systemem SHA512. Hash ten ma za zadanie ukryć hasła użytkowników przed ewentualnym wyciekiem bazy danych. Dodatkowo wersja produkcyjna oprogramowania ukrywa wszelkie błędy komunikacji z baza danych w celu uniknięcia pokazania informacji poufnych.

Mechanizmy autoryzacyjne

Do autoryzacji użytkowników mają być używane tokeny, które będą czasowym identyfikatorem użytkownika. Użytkownik będzie musiał zalogować się raz na określony czas (dane logowania będą przesyłane z użyciem protokołu SSL) co zmniejsza ryzyko kradzieży danych. Aby dostać się do ważniejszych funkcji systemu, posiadacz tokena o podstawowych uprawnieniach będzie musiał zalogować się ponownie z użyciem tokena, hasła i loginu.

Mechanizmy przetwarzające zapytania po stronie serwera

Do walidacji zapytań zostaną użyte systemy wychytujące niedozwolone znaki (np. znaki ucieczki, kod *XML*), lub też kawałki kodu napisanego w *HTML*, *JavaScript*, *Java*. Przykładową biblioteką oferującą takie rozwiązanie jest OWASP AntiSamy Project[2].

5.8 Opis endpointów

5.8.1 Backend

Opisane poniżej *endpointy*, są to adresy *RESTful API* na które można wysłać odpowiednie żądania *HTTP*, aby otrzymać odpowiedź. Wszystkie parametry są przekazywane jako typ tekstowy *String*. Na *serwerze backendowym* zostają one parsowane. Struktura linku jest następująca:

`www.domena.com/{nazwa endpointa}?argument1=...&argument2=...`

gdzie:

1. {nazwa endpointa} - *opcje*
Opis słowny funkcjonalności endpointa
 - a. argument1 - opis słowny
 - b. argument2 - ...
 - c. ...
2. ...

Opcje:

Token - możliwość wykonania operacji tylko za pomocą tokena dostarczanego poprzez serwis logujący, umożliwiający uwierzytelnienie użytkownika. Token ten powinien zostać przekazany poprzez nagłówek żądania *HTTP*.

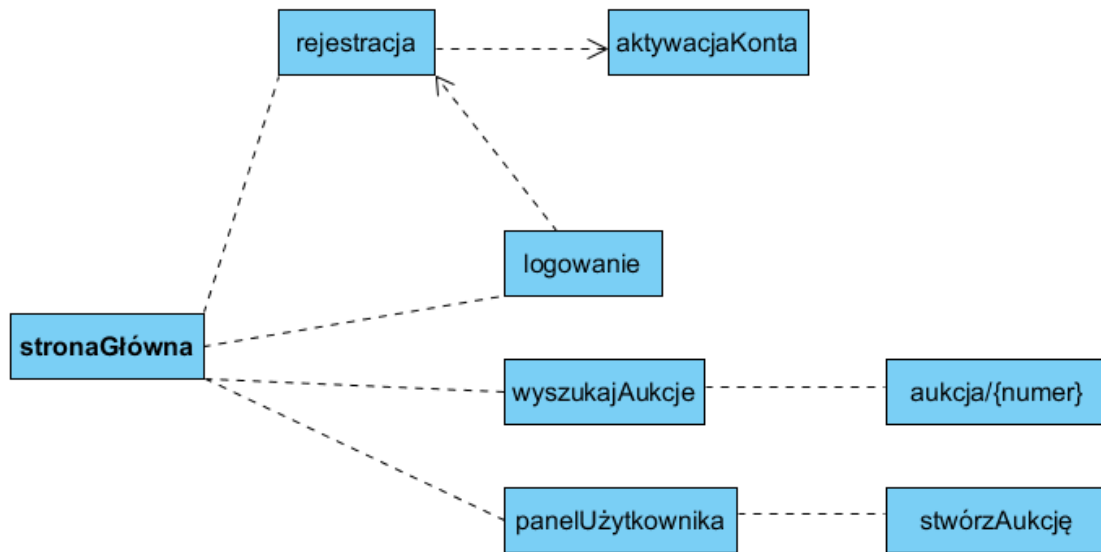
Endpointy

1. createUser
Zapisanie nowego użytkownika do bazy
 - a. username – nazwa użytkownika
 - b. password – hasło użytkownika
2. activateUser
Aktywacja nowego użytkownika za pomocą przysłanego linka na adres e-mail
 - a. activationID – unikatowy kod aktywacyjny wygenerowany dla konta
 - b. userID – unikatowy identyfikator użytkownika na bazie
3. listMyAuctions - Token
Spis aukcji użytkownika
 - a. categories – lista kategorii
 - b. userID – identyfikator użytkownika
4. listWonAuctions - Token
Spis aukcji które użytkownik wygrał
 - a. categories – lista kategorii
 - b. userID – identyfikator użytkownika
5. listActiveAuctions - Token
Spis aukcji które są aktywne i były licytowane przez użytkownika chociaż raz
 - a. categories – lista kategorii
 - b. userID – identyfikator użytkownika
6. listAuctions
Spis aukcji względem kategorii i wyszukanej frazy

- a. `searchPhrase` – słowa kluczowe, dla których mają zostać wyszukane rekordy
 - b. `categories` – lista kategorii
 - c. `parameters` – lista dodatkowych parametrów
7. `showAuction{auctionID}`
Wyświetlenie pojedynczej aukcji
- a. `{auctionID}` – identyfikator aukcji
8. `login`
Logowanie użytkownika
- a. `username` – nazwa użytkownika
 - b. `password` – hasło użytkownika
9. `bidAuction/${auctionID}` - Token
Złożenie oferty dla konkretnej aukcji
- a. `userID` – identyfikator użytkownika
 - b. `offer` – oferta
 - c. `${auctionID}` – identyfikator aukcji
10. `createAuction` - Token
- a. `userID` – identyfikator Użytkownika
 - b. `parametry` – tj. lokacja, daty etc.
11. `securedModifyUser` - Token
Modyfikuj dane użytkownika - zabezpieczone dodatkowym tokenem
- a. `userID` – identyfikator użytkownika
 - b. `username` – nazwa użytkownika
 - c. `password` – hasło użytkownika
 - d. `e-mail` – poczta użytkownika
12. `securedDeleteAuction` - Token
Wycofaj aukcję - zabezpieczone dodatkowym tokenem
- a. `userID` – identyfikator użytkownika
 - b. `auctionID` – identyfikator aukcji

5.8.2 Frontend

Aplikacja kliencka nie będzie zawierać wielu adresów, ponieważ służy ona do pomocy w obsłudze danych. Jeden *endpoint* może posiadać funkcjonalność obsługi wielu *endpointów serwera backendowego*. Dokładny opis tych punktów znajduje się w dziale szkice i funkcjonalność aplikacji klienckiej.

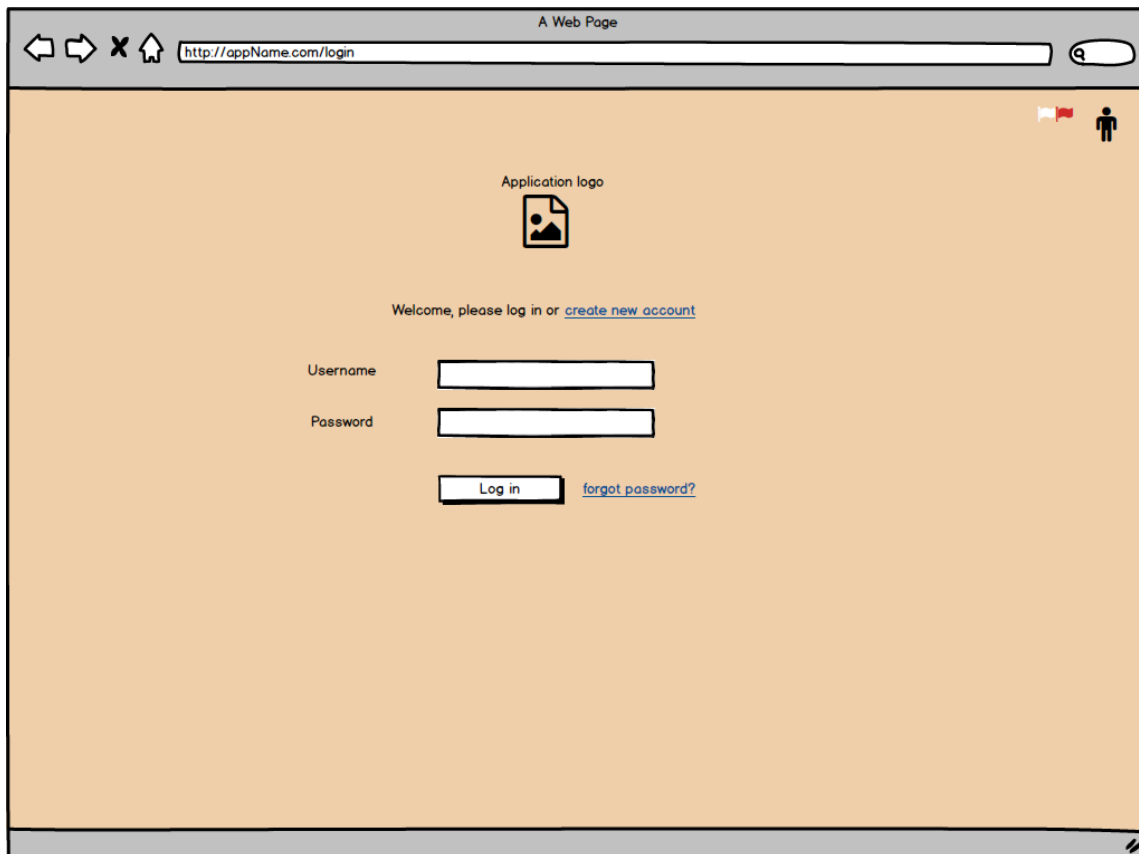
Rys. 16: Mapa *endpointów* aplikacji klienckiej**Objaśnienia**

1. Linia przerywana, bez strzałek: oznacza, iż istnieją odnośniki przekierowujące pomiędzy dwoma połączonymi daną linią adresami.
2. Linia przerywana, ze strzałką: oznacza, iż na konkretny adres strony, z danego punktu, można dostać się tylko jednostronnie (*np. z logowania można przejść do rejestracji, natomiast z rejestracji do logowania już nie*).

5.9 Szkice i funkcjonalność aplikacji klienckiej

Poniżej znajdują się wszystkie opisane *endpointy* stron dostępnych z poziomu aplikacji klienckiej. Jedyny adres który nie został uwzględniony czyli *stronaGłówna*, nie jest tutaj uwzględniony ponieważ ta strona ma zawierać tylko bieżące informacje więc jej wygląd będzie dopasowywany na bieżąco w zależności od wymagań klienta.

5.9.1 Logowanie



Rys. 17: Szkice aplikacji - logowanie użytkownika

Funkcjonalność

Strona ta służy jako interfejs logowania. Użytkownik ma za zadanie podać dane logowania. Następnie aplikacja wysyła zapytanie z danymi do *serwera backendowego*. W odpowiedzi pozytywnej otrzymuje token, a negatywnej nic. Token ma być przechowywany lokalnie, przy użyciu technologii *Local Storage* - nowocześniejszej wersji *Cookie*, tzw. *ciasteczek*.

Walidacja

1. Pole tekstowe z nazwą użytkownika lub adresem e-mail: brak
2. Pole tekstowe z hasłem: brak

5.9.2 Wyszukiwanie aukcji

The screenshot shows a web browser window with the URL `http://appName.com/listAuctions?param1=arg1¶m2=...`. The page layout includes:

- Search Bar:** A search input field with a magnifying glass icon.
- Categories:** A list of categories with checkboxes:
 - ☐ Living
 - ☐ Cars
 - ☐ Electronics & house appliances
 - ☐ Moving
 - ☐ Big objects (>1T)
 - ☐ Medium objects (>10<999kg)
 - ☐ Small objects (<9kg)
- Price Filter:** A section with input fields for price range, distance, weight, and maximum size, along with checkboxes for 'living', 'requires special environment', and 'fragile'.
- Auction List:** A table of auction items, each with a promotional image, title, current lowest bid (1234 \$), location (From: Warsaw To: Berlin), pickup/delivery times, and a 'To auction' button. The list is sorted by 'Time left' (21d 31m).
- Page Navigation:** A pagination bar at the bottom showing '1 2 3 ... 555'.

Rys. 18: Szkice aplikacji - wyszukiwanie aukcji

Funkcjonalność

Strona ta zapewnia funkcjonalność zaawansowanego wyszukiwania aukcji z uwzględnieniem wielu parametrów, pochodzących z różnych elementów: okienka wyszukiwania, okienka zaznaczenia kategorii specyficznych dla towarów (cena, waga itp.). Domyślnie wyświetlane będą aukcje w kolejności od najmniejszego pozostałego czasu do zakończenia. Po zaznaczeniu odpowiednich kategorii i wysłaniu zapytania, strona powinna otrzymać listę aukcji odpowiadającą zapytaniu. Lista ta powinna zostać wyświetlona w widoczny sposób (prostokąty pod elementem wyszukiwania stron). Te elementy powinny posiadać przycisk po którym użytkownik może przejść do interesującej go aukcji.

Walidacja

1. Pole tekstowe wyszukiwania danych: walidacja typu SQL i Java.
2. Pola tekstowe z zaawansowanym wyszukiwaniem (cena, dystans...): brak

5.9.3 Rejestracja nowego użytkownika

A Web Page

http://appName.com/createUser

Logo

Username

Password

Re-enter password

Email

Telephone

First name

Second name

Street address

Home number

City

Postal code

Create account

Your location, make sure if it's right

Street address

Rys. 19: Szkice aplikacji - rejestracja nowego użytkownika

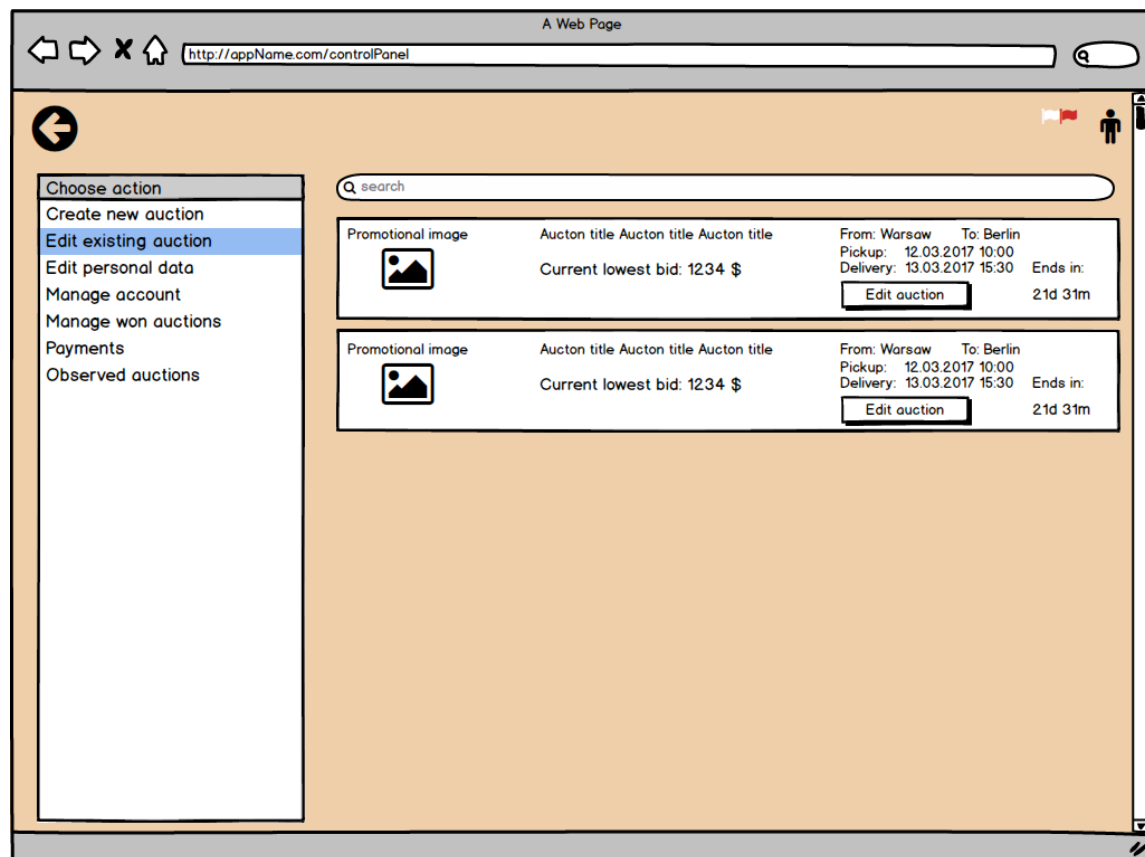
Funkcjonalność

Strona umożliwia zarejestrowanie konta nowego użytkownika w systemie. Aplikacja kliencka wysyła dane do serwera, a w odpowiedzi otrzymuje czy konto zostało założone poprawnie czy nie. Dodatkowo wyświetlana jest mapa która pokazuje zapisany adres. Użytkownika nie ma z niej innej interakcji niż poprzez uzupełnienie pól tekstowych.

Walidacja

1. Pole tekstowe nazwy użytkownika: walidacja słownikowa - zabronione hasła, niekulturalne wyrazy, minimum 6 znaków, maksimum 20.
2. Pole tekstowe z hasłem: wymagania - duża, mała litera, znak specjalny oraz cyfra, minimum 8 znaków. Brak nazwy konta bądź adresu e-mail w hasle, maksimum 30.
3. Pola tekstowe z adresem e-mail: Wymagany znak @ oraz domena i kraj, maksimum 30 znaków.
4. Pola tekstowe z danymi użytkownika (imię, nazwisko, miasto...): maksimum 30 znaków. Dodatkowym źródłem walidacji jest mapa, która w przypadku błędnie podanego adresu, zwróci błąd.

5.9.4 Panel użytkownika



Rys. 20: Szkice aplikacji - panel użytkownika

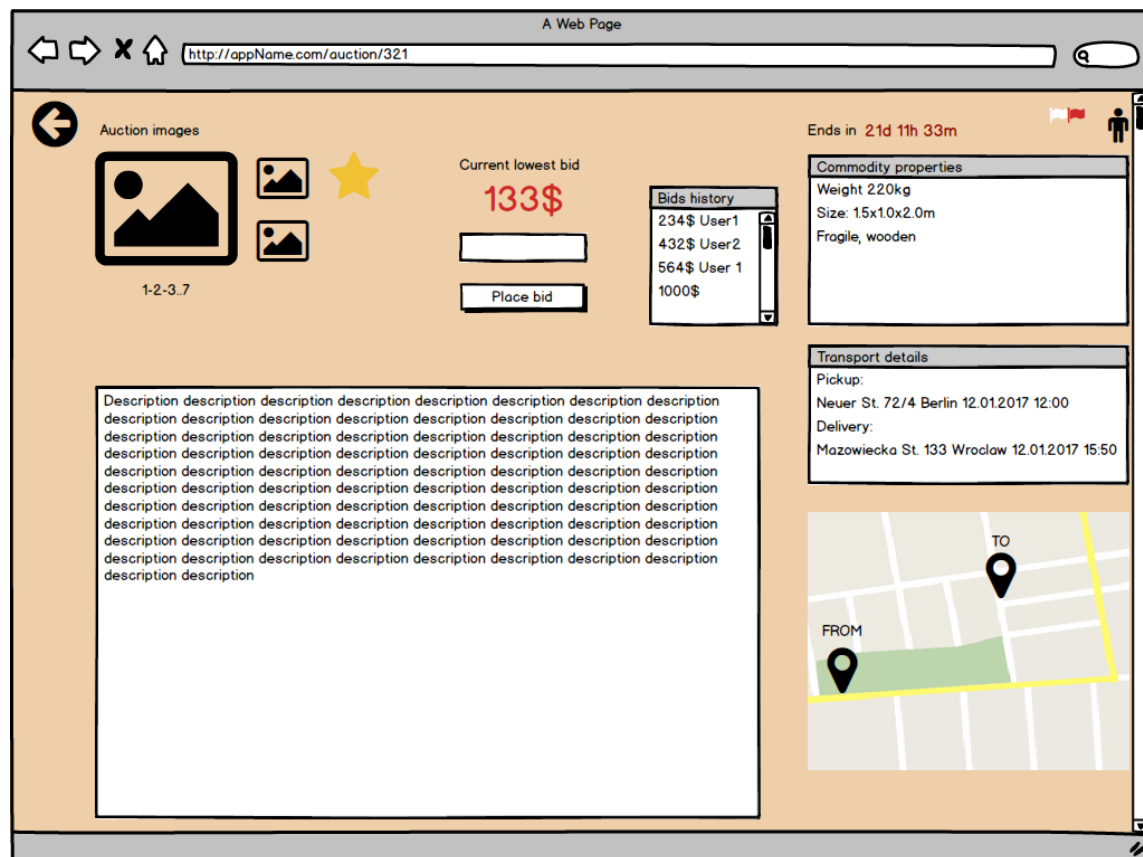
Funkcjonalność

Strona ta umożliwia zalogowanemu użytkownikowi zarządzanie swoim kontem oraz własnymi aukcjami.

Walidacja

1. do wejścia na stronę wymagane jest zalogowanie się (aktywny token).

5.9.5 Aukcja



Rys. 21: Szkice aplikacji - aukcja

Funkcjonalność

Strona ta pozwala zobaczyć treść konkretnej aukcji oraz złożyć swoją ofertę.

Walidacja

1. Pole tekstowe: Przyjmuje jedynie cyfry, oraz znak przecinka bądź kropki. Maksimum znaków 15.
2. Przycisk złoż ofertę: wymagane jest zalogowanie. Nowa oferta nie może być większa od aktualnej najmniejszej.

5.9.6 Stwórz nową aukcję

The screenshot shows a web browser window titled "A Web Page" with the address bar displaying "http://appName.com/createAuction". The page content is a form for creating a new auction. On the left, there's a section for "Auction images" with a large image placeholder and a "Add photography" button. Below this is a "Starting price (optional)" field with the value "250.00" and a "Last day of auction (max 21d)" field with the value "11/11/16 15:30". The main part of the form contains "Address from" (Warsaw, Mazowiecka 32/6) and "Address To" (Berlin, Stumpf st. 11) fields, each with a date and time picker. A map shows a route from "FROM" to "TO". To the right, "Commodity properties" include a "Category" dropdown (Choose category), and input fields for "Weight" (kg), "Width" (m), "Length" (0,32 m), and "Height" (5,11 m). There are checkboxes for "living", "requires special environment" (checked), and "fragile". Below these is the "Auction title" field (5/50 signs) and the "Auction description" field (87/2000 signs) with a rich text editor. At the bottom right is a "Create auction!" button.

Rys. 22: Szkice aplikacji - stwórz nową aukcję

Funkcjonalność

Strona ta pozwala na stworzenie nowej aukcji użytkownika. Pole tekstowe dotyczące opisu jest dynamicznie rozwijalne wraz z widocznym tekstem. Opis charakterystyki przedmiotu jest tylko przykładowy, dodatkowe parametry powinny zostać tutaj dodane. Po kliknięciu przycisku dodaj zdjęcie powinno wyskoczyć okienko z możliwością pobrania zdjęcia z dysku bądź z adresu internetowego.

Walidacja

1. Pole tekstowe adresów: Adres będzie sprawdzany informacją zwrotną z *Google Maps API*. Jeśli mapa poprawnie wykryje adres, to znaczy, że jest on prawidłowy. Maksimum znaków 50.
2. Kalendarz: Data do i data od. Kalendarz pierwszej daty nie może być mniejszy od aktualnego dnia, ani daty dostarczenia towaru.
3. Pole tekstowe ceny początkowej: przyjmuje tylko cyfry, przecinek bądź kropkę. Maksimum znaków 15.
4. Pole tekstowe tytułu aukcji: walidacja słownikowa - zabronione hasła, niekulturalne wyrazy. Maksimum 50 znaków, minimum 5.

5. Pole tekstowe opisu aukcji: walidacja HTML, JavaScript, słownikowa. Maksymalnie 2000 znaków, łącznie ze znakami dostępnego kodu HTML.
6. Pole tekstowe dotyczące wagi oraz metrażu: maksymalnie dwa znaki po przecinku oraz ogólnie 9 znaków. Przyjmuje przecinek bądź kropkę.
7. Przycisk stwórz aukcję: wypełnione wszystkie wymagane pola.

5.9.7 Aktywacja konta

A Web Page

http://appName.com/activateAccount

Application logo

Username or E-mail

Activation code

Activate

Did the code hasn't come yet?

Username or E-mail

Send again

Rys. 23: Szkice aplikacji - aktywacja konta

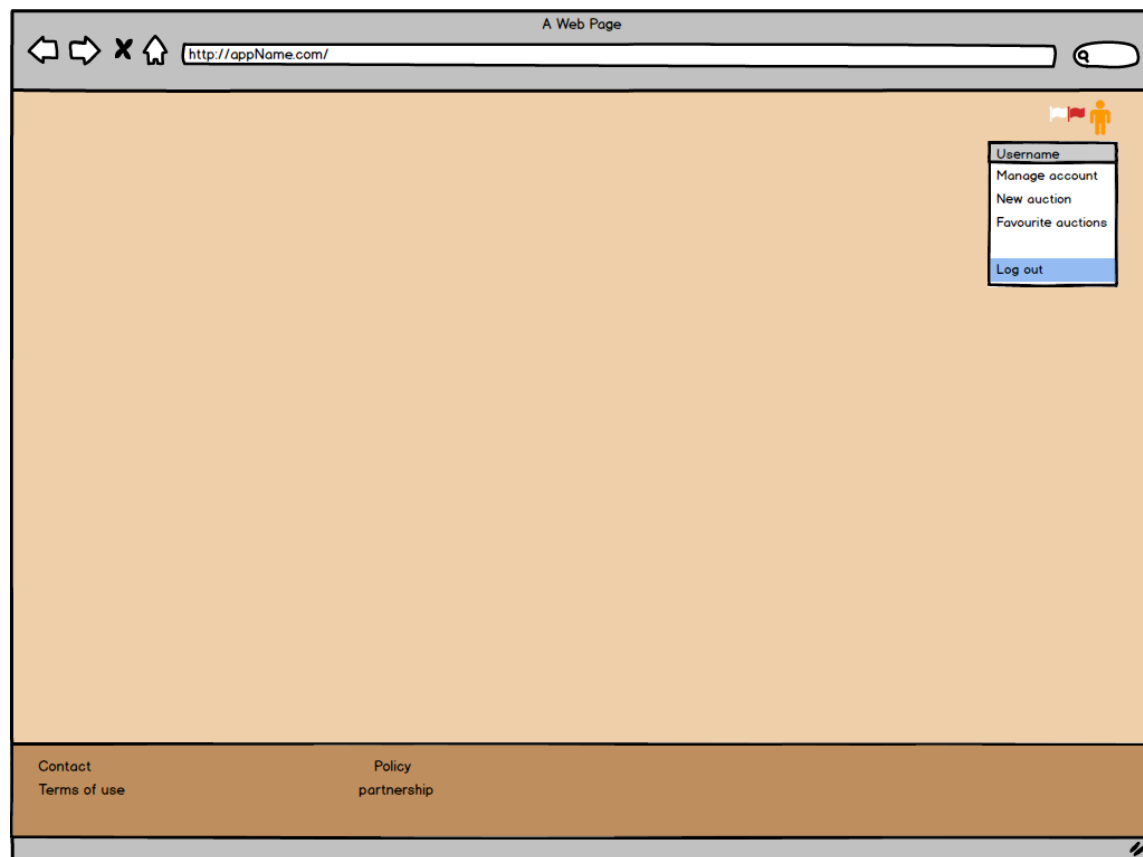
Funkcjonalność

Strona umożliwia aktywację nowego konta za pomocą kodu przysłanego na podany wcześniej adres e-mail.

Walidacja

1. Pola tekstowe: brak

5.9.8 Wspólne elementy



Rys. 24: Szkice aplikacji - wspólne elementy

Funkcjonalność

Na tym szkicu pokazane są wszystkie dodatkowe elementy wspólne, które pojawiają się na wszystkich innych podstronach.

Objaśnienia

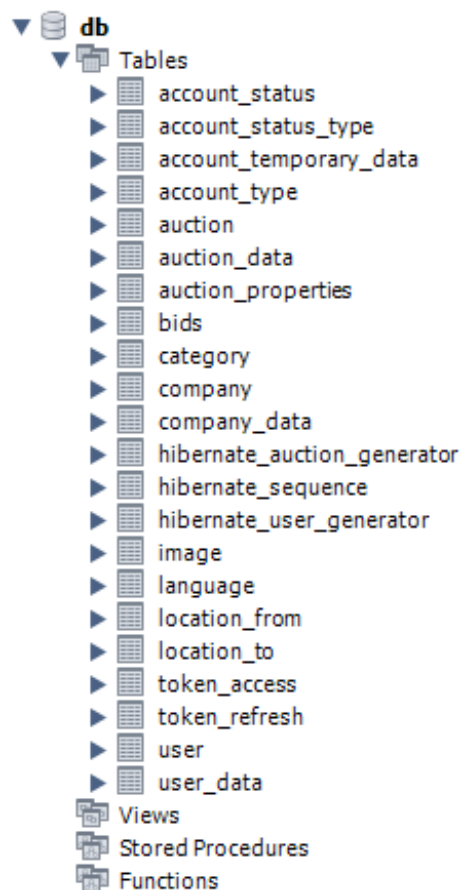
1. Ludzik: oznacza skrót do zarządzania kontem zalogowanego użytkownika. Dostępny jest jedynie po zalogowaniu. Przy kliknięciu ma pokazać listę szybkiego wyboru najczęściej używanych funkcjonalności.
2. Flagi: oznaczają aktualnie wyświetlany język. Po kliknięciu na flagę oznaczającą język konkretnego państwa, wszystkie wyświetlane elementy strony, zawierające tekst, mają się przeładować na kliknięty język.
3. Stopka: Stopka jest niewielkim, statycznym elementem strony na samym dole, który będzie zawierać informacje handlowe, kontakt, warunki użycia aplikacji itp.

6 Implementacja systemu

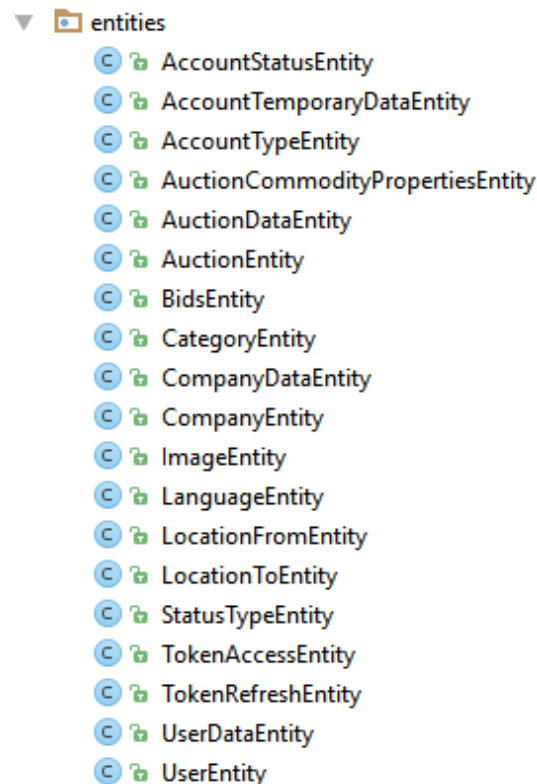
6.1 Implementacja bazy danych

Implementacja bazy danych jest mocno powiązana z serwerem napisanym w języku *Java*, dlatego też poniższe przykłady uwzględniają wycinki kodu z aplikacji. Zaprojektowany, a później wgrany na bazę danych, projekt modelu baz danych, za pomocą programu *IntelliJ* został przekonwertowany na odpowiednik klas *Java* przy wsparciu podstawowego standardu *Java - JPA*. Niestety charakterystyka relacyjności nie została zachowana, dlatego też trzeba było napisać ją ręcznie, przy pomocy *framework'a Hibernate*. Narzędzie to realizuje za programistę chociażby problem mapowania obiektów. Dodatkowo posiada on wsparcie wielu rozszerzeń, tzw. *toolboxów*, np. *Hibernate Search*, ułatwiające implementację systemu wyszukiującego interesujące nas dane przy uwzględnieniu różnych parametrów. Korzyści korzystania z *framework'a Hibernate* są dopiero widoczne dla średnich i większych aplikacji. Czas konfiguracji (encji i *framework'u*) dla małych aplikacji jest niewspółmierny do odniesionych korzyści.

W nawiasach podane zostały programy z których zostały zrobione powyższe zdjęcia.



(a) Widok SQL (MySQL Workbench)



(b) Widok Java (IntelliJ)

Rys. 25: Implementacja bazy danych - porównanie widoku bazy Java - MySQL

Powyższe rysunki pokazują różnicę w widoku bazy danych pomiędzy klasami *Javy*, a encjami wygenerowanymi w języku *SQL* poprzez te klasy. Różnica ilościowa widocznych encji wynika z tego, iż *Hibernate* tworzy dodatkowe, pomocnicze, encje do obsługi

danych, np. encja *"hibernate_auction_generator"* używana jest do przechowywania ostatniego stworzonego numeru identyfikacyjnego aukcji.

Column	Type	Default Value	Nullable	Character Set	Collation	Privileges
id	int(11)		NO			select,insert,update,references
email	varchar(30)		NO	latin1	latin1_swedish_d	select,insert,update,references
password	varchar(30)		NO	latin1	latin1_swedish_d	select,insert,update,references
suspended	bit(1)		NO			select,insert,update,references
username	varchar(20)		NO	latin1	latin1_swedish_d	select,insert,update,references
accountTypeEntity_id	int(11)		YES			select,insert,update,references
companyEntity_id	int(11)		YES			select,insert,update,references
languageEntity_id	int(11)		YES			select,insert,update,references

Name	Schema	Table	Column	Referenced Sch...	Referenced Table	Referenced Col...
FK7f4p1v7bawfry7ulkq9te8den	db	account_status	userEntity_id	db	user	id
FKp0sadohow6nkp4rtq90ljhyfj	db	auction	ownerUserEntity...	db	user	id
FKc9cxerfvm0kvhb8iayv7cie	db	auction	wonUserEntity_id	db	user	id
FK1p09lmx6qpgs043qc3mx5coxj	db	bids	userEntity_id	db	user	id
FK3gckd730v6aj7jifpc345ru7w	db	user	accountTypeEnti...	db	account_type	id
FK2hfd5ddgaxx7ho3k9f44hcugug	db	user	companyEntity_id	db	company	id
FK6i4q5gsxhow9ff4fjgbsdqdx	db	user	languageEntity_id	db	language	id

(a) Widok SQL (MySQL Workbench)

```

@Table(name = "user", schema = "db", catalog = "")
public class UserEntity {
    public static final String GET_USER_BY_ID = "GET_USER_BY_ID";
    public static final String GET_USER_BY_PASSWORD_AND_LOGIN = "GET_USER_BY_PASSWORD_AND_LOGIN";
    public static final String GET_USER_BY_TOKEN_REFRESH = "GET_USER_BY_TOKEN_REFRESH";
    public static final String GET_USER_BY_TOKEN_ACCESS = "GET_USER_BY_TOKEN_ACCESS";
    private int id;
    private String username;
    private String password;
    private boolean suspended;
    private String email;
    private UserDataEntity userDataEntity;
    private CompanyEntity companyEntity;
    private AccountTypeEntity accountTypeEntity;
    private LanguageEntity languageEntity;
    private TokenRefreshEntity refreshTokenEntity;
    private TokenAccessEntity accessTokenEntity;
    private AccountTemporaryDataEntity accountTemporaryDataEntity;
    private AccountStatusEntity accountStatusEntity;
    private Set<AuctionEntity> wonAuctionEntities;
    private Set<AuctionEntity> offeredAuctionEntities;
    private Set<BidsEntity> bidsEntities;

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO, generator = "user_generator")
    @SequenceGenerator(name = "user_generator", sequenceName = "hibernate_user_generator", allocationSize = 1)
    @Column(name = "id", nullable = false)
    public int getId() { return id; }

    public void setId(int id) { this.id = id; }

    @Basic
    @Column(name = "username", nullable = false, length = 20)
    public String getUsername() { return username; }

    public void setUsername(String username) { this.username = username; }

    @OneToOne(fetch = FetchType.LAZY, mappedBy = "userEntity", cascade = {CascadeType.REFRESH, CascadeType.PERSIST})
    public AccountTemporaryDataEntity getAccountTemporaryDataEntity() { return accountTemporaryDataEntity; }

```

(b) Widok Java (IntelliJ)

Rys. 26: Implementacja bazy danych - porównanie widoku encji Java - MySQL

Powyższe rysunki pokazują różnice widoku i implementacji danych pomiędzy encją napisaną w języku *Java*, a *SQL*. Szczegółowe objaśnienie zastosowanych rozwiązań encji napisanej w języku *Java* zostało opisane w dziale Implementacja serwera - Encje.

Listing 1: Konfiguracja dostępu do bazy danych przy użyciu *Hibernate* - XML

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <!-- Database connection properties - Driver, URL, user, password -->
    <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
    <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
    <property name="hibernate.connection.url">jdbc:mysql://127.0.0.1:3306/db</property>
    <property name="hibernate.connection.username">root</property>
    <property name="hibernate.connection.password">root</property>
    <property name="hibernate.generate_statistics">>false</property>
    <property name="hibernate">>false</property>
    <!-- org.hibernate.HibernateException: No CurrentSessionContext configured! -->
    <property name="hibernate.current_session_context_class">thread</property>
    <!-- Echo all executed SQL to stdout -->
    <property name="show_sql">>true</property>
    <!-- Drop and re-create the database schema on startup -->
    <property name="hbm2ddl.auto">create-drop</property>

    <!-- Mapping with model class containing annotations -->
    <mapping class="application.database.entities.AccountStatusEntity"/>
    <mapping class="application.database.entities.AccountTypeEntity"/>
    <mapping class="application.database.entities.AuctionEntity"/>
    <mapping class="application.database.entities.AuctionDataEntity"/>
    <mapping class="application.database.entities.AccountTemporaryDataEntity"/>
    <mapping class="application.database.entities.AuctionCommodityPropertiesEntity"/>
    <mapping class="application.database.entities.BidsEntity"/>
    <mapping class="application.database.entities.CategoryEntity"/>
    <mapping class="application.database.entities.CompanyEntity"/>
    <mapping class="application.database.entities.CompanyDataEntity"/>
    <mapping class="application.database.entities.ImageEntity"/>
    <mapping class="application.database.entities.LanguageEntity"/>
    <mapping class="application.database.entities.LocationToEntity"/>
    <mapping class="application.database.entities.LocationFromEntity"/>
    <mapping class="application.database.entities.StatusTypeEntity"/>
    <mapping class="application.database.entities.TokenAccessEntity"/>
    <mapping class="application.database.entities.TokenRefreshEntity"/>
    <mapping class="application.database.entities.UserEntity"/>
    <mapping class="application.database.entities.UserDataEntity"/>
  </session-factory>
</hibernate-configuration>
```

Listing 2: Konfiguracja dostępu do bazy danych przy użyciu *Hibernate* - Java

```

public class HibernateUtil {
    //Annotation based configuration
    private static SessionFactory sessionAnnotationFactory;

    private static SessionFactory buildSessionAnnotationFactory() {
        try {
            // Create the SessionFactory from hibernate.cfg.xml
            Configuration configuration = new Configuration();
            configuration.configure("hibernate-annotation.cfg.xml");

            // Add all annotated entities
            configuration.addAnnotatedClass(AccountStatusEntity.class);
            configuration.addAnnotatedClass(AccountTypeEntity.class);
            configuration.addAnnotatedClass(AccountTemporaryDataEntity.class);
            configuration.addAnnotatedClass(AuctionDataEntity.class);
            configuration.addAnnotatedClass(AuctionEntity.class);
            configuration.addAnnotatedClass(AuctionCommodityPropertiesEntity.class);
            configuration.addAnnotatedClass(BidsEntity.class);
            configuration.addAnnotatedClass(CategoryEntity.class);
            configuration.addAnnotatedClass(CompanyDataEntity.class);
            configuration.addAnnotatedClass(CompanyEntity.class);
            configuration.addAnnotatedClass(ImageEntity.class);
            configuration.addAnnotatedClass(LanguageEntity.class);
            configuration.addAnnotatedClass(LocationToEntity.class);
            configuration.addAnnotatedClass(LocationFromEntity.class);
            configuration.addAnnotatedClass(StatusTypeEntity.class);
            configuration.addAnnotatedClass(UserDataEntity.class);
            configuration.addAnnotatedClass(UserEntity.class);
            configuration.addAnnotatedClass(TokenAccessEntity.class);
            configuration.addAnnotatedClass(TokenRefreshEntity.class);

            System.out.println("Hibernate Annotation Configuration loaded");
            ServiceRegistry serviceRegistry = new StandardServiceRegistryBuilder()
                .applySettings(configuration.getProperties()).build();
            System.out.println("Hibernate Annotation serviceRegistry created");

            SessionFactory sessionFactory = configuration.buildSessionFactory(serviceRegistry);

            return sessionFactory;
        }
    }
}

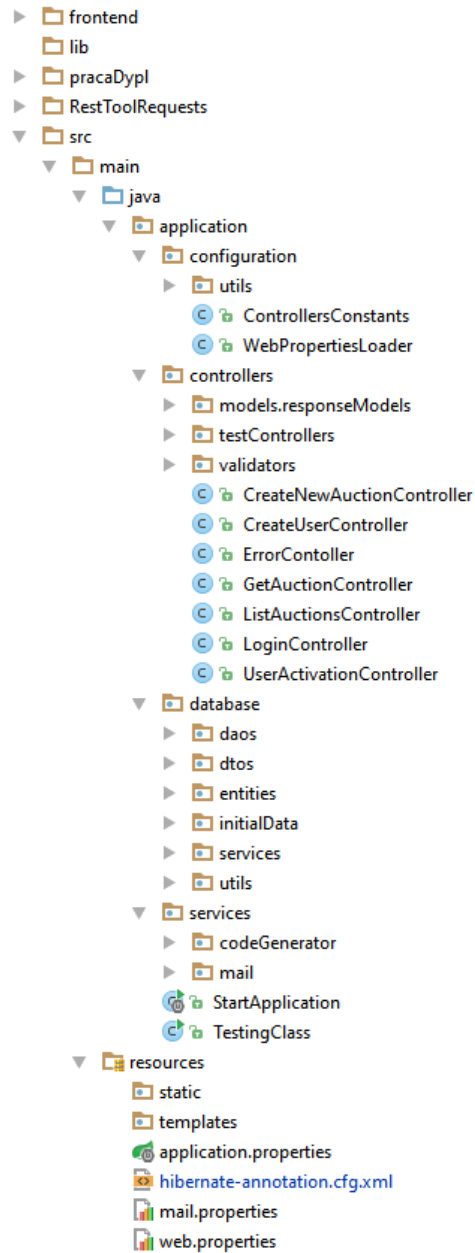
```

Główna konfiguracja ukryta jest w pliku *XML*, która jest ładowana do konfiguracji napisanej w *Java*. Klasa ta tworzy instancję obiektu *SessionFactory*, która odpowiedzialna jest za wszystkie czynności związane z komunikacją pomiędzy bazą danych, a *Java* (np. zawieranie transakcji i dodawanie nowego obiektu).

Wykorzystanie klasy *SessionFactory* jest bardziej szczegółowo opisane w dziale obiekty dostępu danych - *DAO*.

6.2 Implementacja serwera

6.2.1 Struktura projektu



Rys. 27: Implementacja serwera - struktura projektu

Powyżej została przedstawiona struktura widoczna jest z poziomu projektu serwera aplikacji. Widać na niej hierarchię wszystkich klas oraz wyraźny podział funkcjonalności pomiędzy bazę danych, a logikę biznesową.

6.2.2 Konfiguracja projektu - Maven

Konfiguracja zależności (używanych bibliotek i ich wersji) jest utrzymywana przy pomocy *framework'u Maven* w pliku *pom.xml*. Jest to bardzo prosty sposób utrzymywania projektu, ponieważ, aby zainstalować projekt na innym komputerze, należy jedynie użyć komendy *mvn clean install* w terminalu konsoli, będąc w katalogu z projektem.

Listing 3: Implementacja serwera - konfiguracja Maven'a

```

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://maven.apache.org/POM/4.0.0"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>grupaGlowna</groupId>
  <artifactId>artefaktGlowny</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>ApplicationNameGamma</name>
  <description>Thesis</description>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.4.0.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>

  <properties>
    <start-class>application.StartApplication</start-class>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
    <java.version>1.8</java.version>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-jdbc</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-jersey</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web-services</artifactId>

```

6.2.3 Encje

Listing 4: Implementacja serwera - encje

```

12 @Entity
13 @NamedQueries({
14     @NamedQuery(name = UserEntity.GET_USER_BY_ID, query = "FROM UserEntity user WHERE user.id = :id"),
15     @NamedQuery(name = UserEntity.GET_USER_BY_PASSWORD_AND_LOGIN,
16         query = "FROM UserEntity user WHERE user.username = :username AND user.password = :password"),
17 })
18 @Table(name = "user", schema = "db", catalog = "")
19 public class UserEntity {
20     public static final String GET_USER_BY_ID = "GET_USER_BY_ID";
21     public static final String GET_USER_BY_PASSWORD_AND_LOGIN = "GET_USER_BY_PASSWORD_AND_LOGIN";
22     private int id;
23     private String username;
24     private String password;
25     private boolean suspended;
26     private String email;
27     private UserDataEntity userDataEntity;
28     private CompanyEntity companyEntity;
29     private AccountTypeEntity accountTypeEntity;
30     private LanguageEntity languageEntity;
31     private TokenRefreshEntity refreshTokenEntity;
32     private TokenAccessEntity accessTokenEntity;
33     private AccountTemporaryDataEntity accountTemporaryDataEntity;
34     private AccountStatusEntity accountStatusEntity;
35     private Set<AuctionEntity> wonAuctionEntities;
36     private Set<AuctionEntity> offeredAuctionEntities;
37     private Set<BidsEntity> bidsEntities;
38
39     @Id
40     @GeneratedValue(strategy = GenerationType.AUTO, generator = "user_generator")
41     @SequenceGenerator(name = "user_generator", sequenceName = "hibernate_user_generator", allocationSize = 1)
42     @Column(name = "id", nullable = false)
43     public int getId() { return id; }
44
45
46
47     @OneToMany(mappedBy = "wonUserEntity", fetch = FetchType.LAZY, cascade = CascadeType.REFRESH)
48     public Set<AuctionEntity> getWonAuctionEntities() { return wonAuctionEntities; }
49
50
51
52     @Basic
53     @Column(name = "username", nullable = false, length = 20)
54     public String getUsername() { return username; }

```

Powyższy przykład pokazuje przykładową encję napisaną w języku Java (została uproszczona względem aktualnej), używającej rozwiązania z *framework'u Hibernate* oraz standardowej biblioteki *JPA* służące do modelowania obiektów oraz rozwiązujących za nas

problem mapowania obiektów. *Hibernate* umożliwia nam także korzystanie z definiowanych zapytań które są napisane w specjalnym języku wspieranym przez ten *framework* - *HQL* - *Hibernate Query Language* (są one widoczne przy adnotacji `@NamedQueries` - 13 linia). Zapytania te nie różnią się specjalnie od zwykłych zapytań *SQL*. Do oznaczenia tabeli używana jest adnotacja `@Table` - 18 linia wraz z odpowiednimi argumentami. Do generowania sekwencji identyfikatora rekordu, używany jest modyfikowalny `@SequenceGenerator` - 41 linia. Dla każdej encji zalecane jest używanie własnego generatora (bądź, jeśli mamy encje jeden-do-jednego, wspólnego z encją rodzicem). W przypadku jednego generatora, każdy rekord będzie miał kolejny identyfikator, dzielony ze wszystkimi encjami. W 47 linii mamy przykład użycia relacji jeden-do-wielu między encją użytkownika (widoczną), a aukcji, gdzie encja użytkownika jest właścicielem relacji, (zawiera w sobie kolekcję obiektów `Set<AuctionEntity>` - linia 35 encji aukcji) co oznacza, że jeden użytkownik może posiadać 0 bądź wiele aukcji. Na samym dole mamy oznaczenie pola przy użyciu adnotacji `@Basic` oraz `@Column`. Pierwsza z nich oznacza domyślną konfigurację pola, czyli czy pole jest opcjonalne oraz tryb wyciągania danych (*fetch*) - gorliwy (*eager*) czy leniwy (*lazy*).

6.2.4 Obiekty dostępu danych

Listing 5: Implementacja serwera - wspólne metody

```
14 public abstract class DaoSupportImpl implements DaoSupport {
15
16     protected SessionFactory sessionFactory;
17
18     @PostConstruct
19     private void setDefaultSessionFactory() { this.sessionFactory = HibernateUtil.getSessionAnnotationFactory(); }
20
21
22
23     @Override
24     public void save(Object object) {
25         Session session = this.sessionFactory.openSession();
26         Transaction transaction = session.beginTransaction();
27         session.persist(object);
28         transaction.commit();
29         session.close();
30     }
31
32     @Override
33     public void update(Object object) {
34         Session session = this.sessionFactory.openSession();
35         Transaction transaction = session.beginTransaction();
36         session.update(object);
37         transaction.commit();
38         session.close();
39     }
40
41     @Override
42     public void batchInsert(Set<?> objects) {
43         Session session = this.sessionFactory.openSession();
44         Transaction transaction = session.beginTransaction();
45         objects.forEach(obj -> session.persist(obj));
46         transaction.commit();
47         session.close();
48     }
49 }
```

Widoczne metody, są podstawowymi operacjami dostępnymi dla wszystkich obiektów dostępu danych (DAO - Data Access Object). Każda klasa typu *DAO* dziedziczy po wyżej widocznej klasie.

Listing 6: Implementacja serwera - obiekt dostępu danych użytkownika

```

14  @Repository
15  public class UserEntityDaoImpl extends DaoSupportImpl implements UserEntityDao {
16
17      @Override
18      public UserEntity getById(int id) {
19          Session session = this.sessionFactory.openSession();
20          UserEntity userEntity = (UserEntity) session.getNamedQuery(UserEntity.GET_USER_BY_ID)
21              .setParameter("id", id).uniqueResult();
22          session.close();
23          return userEntity;
24      }
25
26      @Override
27      public UserEntity getByUsernameAndPassword(String username, String password) {
28          Session session = this.sessionFactory.openSession();
29          UserEntity userEntity = (UserEntity) session
30              .getNamedQuery(UserEntity.GET_USER_BY_PASSWORD_AND_LOGIN)
31              .setParameter("username", username)
32              .setParameter("password", password)
33              .uniqueResult();
34          session.close();
35          return userEntity;
36      }
37
38      //// TODO: 06.01.2017 One query to get UserToken!
39      @Override
40      public UserEntity getUserByTokenRefresh(String token) {
41          Session session = this.sessionFactory.openSession();
42          TokenRefreshEntity tokenRefreshEntity = (TokenRefreshEntity) session
43              .getNamedQuery(UserEntity.GET_USER_BY_TOKEN_REFRESH)
44              .setParameter("token", token)
45              .uniqueResult();
46          UserEntity userEntity = tokenRefreshEntity.getUserEntity();
47          session.close();
48          return userEntity;
49      }

```

Powyższy przykład pokazuje przykładowy obiekt dostępu danych do bazy (został on uproszczony względem aktualnego). Używa on uprzednio skonfigurowanych obiektów typu *org.hibernate.SessionFactory* które przechowują w sobie dane potrzebne do połączenia oraz narzędzia umożliwiające przesył informacji pomiędzy bazą danych, a aplikacją Javy. Kod jest napisany bardzo schematycznie, na początku tworzymy sesję i w zależności czy potrzebujemy pobrać dane czy je zapisać zawieramy transakcję i potwierdzamy zmiany. Na samym końcu należy pamiętać o zamknięciu połączenia.

6.2.5 Serwisy danych

Serwisy danych są odpowiedzialne za przetworzenie danych oraz udostępnienie jej aplikacji (serwisy danych), bądź wystawienie na zewnątrz (np. serwis poczty elektronicznej) - posiadają większość logiki biznesowej.

Listing 7: Implementacja serwera - serwis użytkownika

```

25  @Service
26  public class UserService {
27      private final Logger logger = LoggerFactory.getLogger(this.getClass());
28      @Autowired
29      UserEntityDao userEntityDao;
30      @Autowired
31      MailService mailService;
32      @Autowired
33      RandomCodeGenerator randomCodeGenerator;
34      @Autowired
35      WebPropertiesLoader webPropertiesLoader;
36
37      public UserEntity getUserEntityById(String id) throws NumberFormatException, NullPointerException {
38          UserEntity userEntity = userEntityDao.getById(Integer.parseInt(id));
39          if (userEntity instanceof UserEntity) {
40              return userEntity;
41          } else {
42              throw new NullPointerException("No matching user");
43          }
44      }
45
46      public UserEntity createNewUserAndSendActivationMail(
47          String accountType, String company, String username, String password, String language, String email)
48          throws CustomParseException, MailException {
49
50          UserEntity userEntity = UserValidator.parseAndCreateUser(
51              accountType, company, username, password, "true", language);
52          userEntity.setEmail(email);
53
54          AccountTemporaryDataEntity accountTemporaryData = new AccountTemporaryDataEntity();
55          accountTemporaryData.setActivationCode(randomCodeGenerator.nextRandomString(ACTIVATION_STRING_LENGTH));
56
57          userEntity.setAccountTemporaryDataEntity(accountTemporaryData);
58          accountTemporaryData.setUserEntity(userEntity);
59
60          createRefreshTokenEntityForUser(userEntity);
61          createAccessTokenEntityForUser(userEntity);
62
63          userEntityDao.save(userEntity);
64
65          mailService.sendMail(userEntity.getEmail(), NEW_USER_MAIL_SUBJECT_TEMPLATE, String.format(
66              NEW_USER_MAIL_TEXT_TEMPLATE, userEntity.getUsername(), userEntity.getPassword(),
67              String.format(ACTIVATION_URL_ADDRESS_LINK_TEMPLATE, webPropertiesLoader.getCONTEXT_PATH(),
68                  userEntity.getId(), accountTemporaryData.getActivationCode())));
69          //:TODO cofnięcie stworzenia konta jeśli nie wysłano maila
70          return userEntity;
71      }

```

Jest to przykład serwisu danych używanego do wszystkich akcji związanych z użytkownikiem - większości w tej aplikacji. Przy 37 oraz 45 widać dwie przykładowe metody, *getUserEntityById* oraz *createNewUserAndSendActivationMail*. Pierwsza z nich odpowiada za pobranie istniejącego użytkownika z bazy danych po unikatowym identyfikatorze, oraz przekazanie go. Druga tworzy nowego

użytkownika oraz wysyła e-mail z kluczem i linkiem aktywacyjnym. Widać tutaj wcześniej opisane obiekty *DAO* - *Data Access Object* np. klasę *UserEntityDao* - linia 29. Obiekty te są zarządzane przez implementację kontenera *IoC* - *Inversion of Control* przez framework *Spring*. Kontener ten zarządza cyklem życia wszystkich oznaczonych dla niego obiektów. Szczegółowy opis tej implementacji można znaleźć na oficjalnej stronie dokumentacji framework'u *Spring*[7]. Przykładem takiego oznaczenia jest adnotacja *@Service* - 25 linia, oznaczająca serwis danych. Wstrzykiwanie takiego obiektu oznacza się adnotacją *@Autowired* - 28, 30, 32, 34 linia, z ewentualnym argumentem nazwy ziarenka (*Bean*), jeśli jest ono inne od nazwy domyślnej (nazwy klasy).

6.2.6 Serwis mailowy

Listing 8: Implementacja serwera - serwis e-mail

```
14  @Service
15  public class MailService {
16      @Autowired
17      private JavaMailSender javaMailSender;
18
19      private final Logger logger = LoggerFactory.getLogger(this.getClass());
20
21      public void sendMail(String email, String subject, String message) throws MailException{
22          SimpleMailMessage mailMessage = new SimpleMailMessage();
23          mailMessage.setSubject(subject);
24          mailMessage.setText(message);
25          mailMessage.setTo(email);
26          this.javaMailSender.send(mailMessage);
27      }
28  }
29  }
```

Implementacja serwisu mailowego używa już dostępnej klasy z pakietu *Spring* i w związku z tym ogranicza się jedynie do wykorzystania już dostępnych funkcji widocznych na powyższym obrazku. Jedynym wymaganiem jest przekazanie skonfigurowanego obiektu który by e-maile wysyłał. Ten obiekt widoczny jest w 17 linii - *JavaMailSender*.

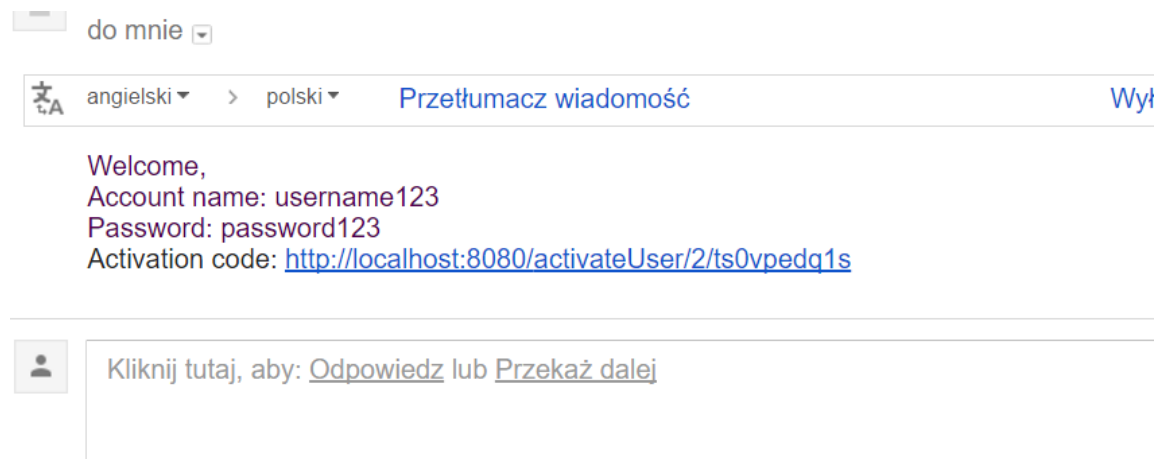
Listing 9: Implementacja serwera - konfiguracja serwisu e-mail

```

15  @Configuration
16  @PropertySource("classpath:mail.properties")
17  public class MailConfiguration {
18
19      @Value("${mail.protocol}")
20      private String protocol;
21      @Value("${mail.host}")
22      private String host;
23      @Value("${mail.port}")
24      private int port;
25      @Value("${mail.smtp.socketFactory.port}")
26      private int socketPort;
27      @Value("${mail.smtp.auth}")
28      private boolean auth;
29      @Value("${mail.smtp.starttls.enable}")
30      private boolean starttls;
31      @Value("${mail.smtp.starttls.required}")
32      private boolean starttls_required;
33      @Value("${mail.smtp.debug}")
34      private boolean debug;
35      @Value("${mail.smtp.socketFactory.fallback}")
36      private boolean fallback;
37      @Value("${mail.username}")
38      private String username;
39      @Value("${mail.password}")
40      private String password;
41
42      @Bean
43      public JavaMailSender javaMailSender() {
44          JavaMailSenderImpl mailSender = new JavaMailSenderImpl();
45
46          Properties mailProperties = new Properties();
47          mailProperties.put("mail.smtp.auth", auth);
48          mailProperties.put("mail.smtp.starttls.enable", starttls);
49          mailProperties.put("mail.smtp.starttls.required", starttls_required);
50          mailProperties.put("mail.smtp.socketFactory.port", socketPort);
51          mailProperties.put("mail.smtp.debug", debug);
52          mailProperties.put("mail.smtp.socketFactory.class", "javax.net.ssl.SSLSocketFactory");
53          mailProperties.put("mail.smtp.socketFactory.fallback", fallback);
54
55          mailSender.setJavaMailProperties(mailProperties);
56          mailSender.setHost(host);

```

Plik konfiguracyjny zarządzany przez kontener *Spring* musi być widoczny dla niego. Najłatwiej osiągnąć można to używając adnotacji *@Configuration* - linia 15. Adnotacja poniżej *@PropertySource* - linia 16 oznacza, iż dane będą wczytywane z zewnętrznego pliku (w tym przypadku pliku znajdującego się w katalogu z lokalnymi zasobami projektu (*resources*)). Aby wczytać poszczególne dane, używana jest adnotacja *@Value*(\$szukany ciąg znaków), która po szukanym ciągu znaków wczytuje wiersz danych. Przy użyciu wspólnie adnotacji *@Configuration* - linia 15 oraz *@Bean* - linia 42 można innym sposobem stworzyć obiekt (*ziarenko* - *Bean*) zarządzane przez kontener *Spring*. Widoczny skonfigurowany obiekt klasy *JavaMailSender* jest wstrzykiwany w serwisie mailowym opisanym wyżej.



Rys. 28: Implementacja serwera - przykładowy e-mail

Powyżej widoczna jest przykładowa wiadomość aktywacyjna użytkownika wysłana przez serwis e-mail.

6.2.7 Udostępnianie danych za pomocą kontrolera

Kontrolery mają za zadanie udostępnić dane na zewnątrz które są transportowane przy pomocy protokołów *HTTP*, wykonując zapytanie z odpowiednimi argumentami oraz adresem strony.

Listing 10: Implementacja serwera - kontroler tworzący nowego użytkownika

```

18 @CrossOrigin(origins = "http://localhost:3000")
19 @RestController
20 public class CreateUserController {
21
22     private final Logger logger = LoggerFactory.getLogger(this.getClass());
23     @Autowired
24     UserService userService;
25
26     @RequestMapping(value = CREATE_USER_ENDPOINT, method = RequestMethod.POST)
27     public BasicResponse createNewUser(
28         @RequestParam(value = "accountType", required = false, defaultValue = "0") String accountType,
29         @RequestParam(value = "company", required = false, defaultValue = "0") String company,
30         @RequestParam(value = "username", required = true) String username,
31         @RequestParam(value = "password", required = true) String password,
32         @RequestParam(value = "language", required = false, defaultValue = "0") String language,
33         @RequestParam(value = "email", required = true) String email) {
34         try {
35             userService.createNewUserAndSendActivationMail(accountType, company,
36                 username, password, language, email);
37         } catch (CustomParseException e) {
38             logger.warn("Parsing error");
39             return new BasicResponse(false, "Parsing error", 0);
40         } catch (MailException e) {
41             logger.warn("Mail error");
42             return new BasicResponse(false, "Couldn't send mail", 0);
43         }
44         return new BasicResponse(true, "Account created, mail sent", 0);
45     }
46 }

```

Powyższy przykład posiada wiele adnotacji, jednakże najważniejszymi z nich są `@RequestMapping` - 26 linia którego argumentami jest nazwa *endpointu* oraz metoda którą należy się odwoływać do niego, aby otrzymać prawidłową odpowiedź. Poniżej widoczna jest adnotacja `@RequestParam` - 27 do 32 linii która odpowiada za przyjmowane argumenty przez *endpoint*. Dane zwracane poprzez ten *endpoint* są automatycznie parsowane przez parser *JSON* dzięki adnotacji `@RestController` - 19 linia.

6.2.8 Przykładowe obiekty transferu danych - DTO

Obiekty *DTO* - *Data Transfer Object* są zwyczajnymi kontenerami na dane. Jednakże są one potrzebne, aby wystawiać informacje tylko te które chcemy. Jeśli udostępnialibyśmy obiekty reprezentujące encje bazodanowe mogłoby to zrodzić sytuację kiedy to parser *JSON*’owy wystawiłby niepożądane informacje takie jak w przypadku aukcji - hasło, login, oraz wszystkie szczegółowe informacje użytkowników mających zależności z daną aukcją (licytujący, właściciel aukcji).

Listing 11: Implementacja serwera - przykładowy obiekt transferu danych - aukcja

```

12 public class Auction {
13     private int id;
14     private String title;
15     @JsonSerialize(using = CustomJSONDateSerializer.class)
16     private Date dateStart;
17     @JsonSerialize(using = CustomJSONDateSerializer.class)
18     private Date dateEnd;
19     private boolean premium;
20     private boolean ended;
21
22     public Auction(AuctionEntity auctionEntity) {
23         this.id = auctionEntity.getId();
24         this.title = auctionEntity.getTitle();
25         this.dateStart = auctionEntity.getAuctionStartDate();
26         this.dateEnd = auctionEntity.getAuctionEndDate();
27         this.premium = auctionEntity.getPremium();
28         this.ended = auctionEntity.getEnded();
29     }
30
31     public int getId() { return id; }
32
33     public void setId(int id) { this.id = id; }
34
35     public String getTitle() { return title; }
36
37     public void setTitle(String title) { this.title = title; }
38
39     public Date getDateEnd() { return dateEnd; }
40
41     public void setDateEnd(Date dateEnd) { this.dateEnd = dateEnd; }
42
43     public boolean isPremium() { return premium; }
44
45     public void setPremium(boolean premium) { this.premium = premium; }
46
47     public boolean isEnded() { return ended; }
48
49     public void setEnded(boolean ended) { this.ended = ended; }
50 }

```

Na rysunku powyżej widoczny jest klasyczny przykład obiektu *DTO*. Posiada on konstruktor używający encji *AuctionEntity* - linia 22 do wypełnienia pól. Takie rozwiązanie umożliwia pokazanie wybranych danych. Dodatkowo adnotacja *JsonSerialize* - linia 15 i 17 oznacza, iż używamy własnej implementacji serializatora *JSON* zamiast domyślnej do parsowania obiektu. *Gettery* i *Settery* są wymagane przez każdy serializator *JSON*, aby dostać się do konkretnych pól, ponieważ używa on mechanizmu refleksji klas, aby dowiedzieć się jakie pola posiada serializowany obiekt i na ich podstawie próbuje dostać się do danych.

6.2.9 Dane inicjujące

Dane inicjujące są zapisane w formie kodu *Java* za pomocą encji (zagnieżdżonych jeśli występują asocjacje).

Listing 12: Implementacja serwera - przykładowy dane inicjujące

```

9  public enum TestingUsers {
10     USER1("username", "password", "olkiewiczhubert@gmail.com", false, AccessTokens.TOKEN1, RefreshTokens.TOKEN1);
11
12     private String username;
13     private String password;
14     private String email;
15     private boolean suspended;
16     private AccessTokens accessToken;
17     private RefreshTokens refreshToken;
18
19     TestingUsers(String username, String password, String email,
20         boolean suspended, AccessTokens accessToken, RefreshTokens refreshToken) {
21         this.username = username;
22         this.password = password;
23         this.email = email;
24         this.suspended = suspended;
25         this.accessToken = accessToken;
26         this.refreshToken = refreshToken;
27     }
28
29     public String getUsername() { return username; }
32
33     public String getPassword() { return password; }
36
37     public String getEmail() { return email; }
40
41     public boolean isSuspended() { return suspended; }
44
45     public AccessTokens getAccessToken() { return accessToken; }
48
49     public RefreshTokens getRefreshToken() { return refreshToken; }
52
53     public enum AccessTokens {
54         TOKEN1("AccessToken1", new Date(Calendar.getInstance().getTimeInMillis()));
55         private String token;
56         private Date expireDate;
57
58         AccessTokens(String token, Date expireDate) {
59             this.token = token;
60             this.expireDate = expireDate;
61         }
62

```

Powyższy przykład pokazuje jedną z takich klas - *TestingUsers* - użytkownicy testowi. Widoczna jest dodatkowo zagnieżdżona encja posiadające dane o tokenach, potrzebna do testowania *endpointów* serwera. Raz napisany mechanizm, pozwala bardzo łatwo i schematycznie, z poziomu kodu, dodawać nowe rekordy do testowania aplikacji. Wystarczy według wzoru wstawiać odpowiednie dane tak jak w 10 linii kodu.

6.2.10 Przykład uruchomienia serwera

Spring za nas zajmuje się całą konfiguracją sieciową oraz serwerem aplikacji internetowych (*Tomcat*) na którym instalowany jest nasz napisany program.

Listing 13: Implementacja serwera - klasa uruchamiająca

```

8  @SpringBootApplication
9  public class StartApplication extends SpringBootServletInitializer {
10
11      @Override
12      protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {
13          return application.sources(StartApplication.class);
14      }
15
16      public static void main(String[] args) {
17          SpringApplication.run(StartApplication.class, args);
18      }

```

Aby uruchomić naszą aplikację należy zwyczajnie uruchomić ją z klasy *StartApplication* - 9 linia, metoda *main(String[] args)* - 15 linia. Klasa ta jest bardzo schematycznie napisana, wymagana jest adnotacja *SpringBootApplication* - 8 linia oraz rozszerzenie o klasę *SpringBootServletInitializer* - 9 linia, która posiada metodę *configure* - 12 konfiguracyjną, którą należy nadpisać.

```

Hibernate: insert into token_refresh (expirationDate, token, id) values (?, ?, ?)
2017-01-08 20:44:06.906 WARN 3740 --- [main] a.d.initialData.InitialDatabaseInserts : Initial insert: Inserted TESTING users
Hibernate: select next_val as id_val from hibernate_auction_generator for update
Hibernate: update hibernate_auction_generator set next_val= ? where next_val=?
Hibernate: insert into auction (auction_end_date, auction_start_date, delivery_date, ended, load_date, ownerUserEntity_id, premium, title, wonUserEntity_id, id) value
Hibernate: insert into auction_properties (auctionEntity_id, categoryEntity_id, fragile, living, sizeX, sizeY, sizeZ, specialEnviroment, weight, id) values (?, ?, ?,
Hibernate: insert into auction_data (description, footer, id) values (?, ?, ?)
Hibernate: insert into location_from (country, full_address, id) values (?, ?, ?)
Hibernate: insert into location_to (country, full_address, id) values (?, ?, ?)
Hibernate: select next_val as id_val from hibernate_auction_generator for update
Hibernate: update hibernate_auction_generator set next_val= ? where next_val=?
Hibernate: insert into auction (auction_end_date, auction_start_date, delivery_date, ended, load_date, ownerUserEntity_id, premium, title, wonUserEntity_id, id) value
Hibernate: insert into auction_properties (auctionEntity_id, categoryEntity_id, fragile, living, sizeX, sizeY, sizeZ, specialEnviroment, weight, id) values (?, ?, ?,
Hibernate: insert into auction_data (description, footer, id) values (?, ?, ?)
Hibernate: insert into location_from (country, full_address, id) values (?, ?, ?)
Hibernate: insert into location_to (country, full_address, id) values (?, ?, ?)
2017-01-08 20:44:06.989 WARN 3740 --- [main] a.d.initialData.InitialDatabaseInserts : Initial insert: Inserted TESTING auctions
2017-01-08 20:44:07.287 INFO 3740 --- [main] s.w.s.m.m.a.RequestMappingHandlerAdapter : Looking for @ControllerAdvice: org.springframework.boot.context.en
2017-01-08 20:44:07.586 INFO 3740 --- [main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "[[/createAuction],methods=[POST]]" onto public application
2017-01-08 20:44:07.590 INFO 3740 --- [main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "[[/createUser],methods=[POST]]" onto public application.cc
2017-01-08 20:44:07.591 INFO 3740 --- [main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "[[/error]]" onto public java.lang.String application.contr
2017-01-08 20:44:07.591 INFO 3740 --- [main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "[[/getAuction],methods=[GET]]" onto public application.dat
2017-01-08 20:44:07.593 INFO 3740 --- [main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "[[/listAuctions],methods=[GET]]" onto public java.util.Lis
2017-01-08 20:44:07.593 INFO 3740 --- [main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "[[/login],methods=[POST]]" onto public application.control
2017-01-08 20:44:07.601 INFO 3740 --- [main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "[[/greeting]]" onto public java.util.Map<java.lang.String,
2017-01-08 20:44:07.602 INFO 3740 --- [main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "[[/activateUser/{user}/{activationCode}],methods=[GET]]" c
2017-01-08 20:44:07.772 INFO 3740 --- [main] o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path [/webjars/**] onto handler of type [class org.spr
2017-01-08 20:44:07.772 INFO 3740 --- [main] o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path [/**] onto handler of type [class org.springframew
2017-01-08 20:44:07.837 INFO 3740 --- [main] o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path [/**/favicon.ico] onto handler of type [class org
2017-01-08 20:44:08.432 INFO 3740 --- [main] o.s.j.e.a.AnnotationBeanExporter : Registering beans for JMX exposure on startup
2017-01-08 20:44:08.485 INFO 3740 --- [main] s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat started on port(s): 8080 (http)
2017-01-08 20:44:08.489 INFO 3740 --- [main] application.StartApplication : Started StartApplication in 8.675 seconds (JVM running for 9.414)

```

Rys. 29: Wyciąg z konsoli po uruchomieniu serwera

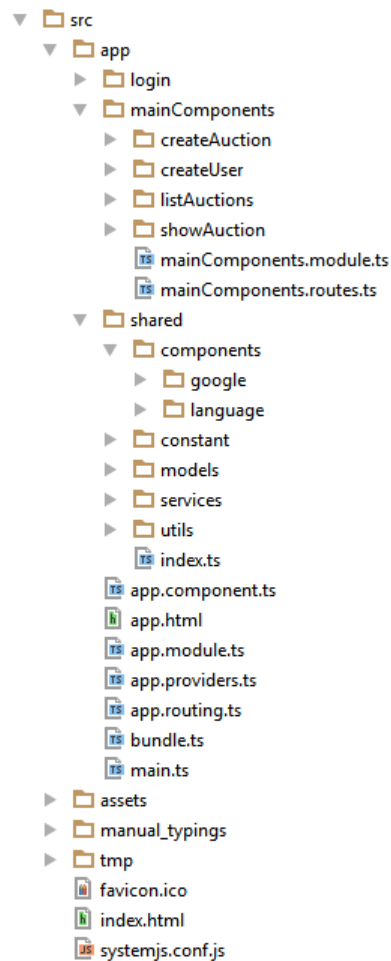
Powyższe zdjęcie pokazuje kawałek wypisanych informacji przy uruchomieniu aplikacji (z klasy opisanej zdjęcie wyżej). Widać

na nim fragmenty kodu *HQL* - *Hibernate Query Language* inicjujące bazę danych oraz konfigurację *endpointów* naszej aplikacji i port na który można się do nich odwoływać.

6.3 Implementacja aplikacji klienckiej

Aplikacja kliencka jest zintegrowana z *serwerem backendowym*, jednakże nie jest ona jeszcze dopracowana. Większość szablonów projektowych zostało zaimplementowanych. Niestety nie starczyło czasu na upiększenie jej za pomocą odpowiednich znaczników języka *HTML* oraz *CSS*.

6.3.1 Struktura projektu



Rys. 30: Implementacja aplikacji klienckiej - struktura projektu

Powyżej widoczna jest struktura projektu (z poziomu programu *WebStorm*). *Angular2* stawia na modułowość, przez co jest podobny do języków obiektowych takich jak *Java* czy *C#*. Głównym modulem z aplikacji są klasy zaczynające się od nazwy *app*.

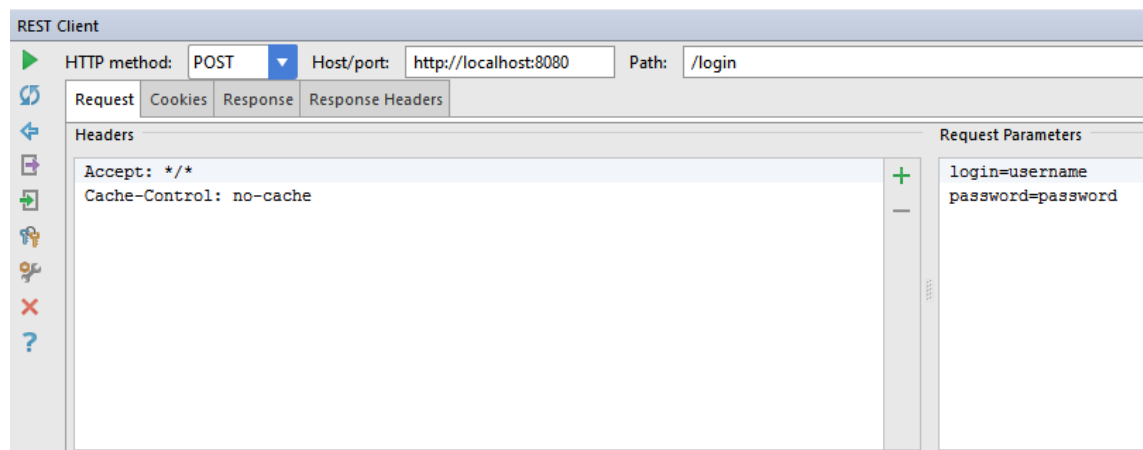
(szczegółowo opisane w rozdziale główny moduł aplikacji). Od nich zaczyna się główny program, który ładuje pozostałe moduły (np. *mainComponents* - główne komponenty odpowiedzialne za pokazanie aukcji czy stworzenie użytkownika).

6.3.2 Integracja z serwerem

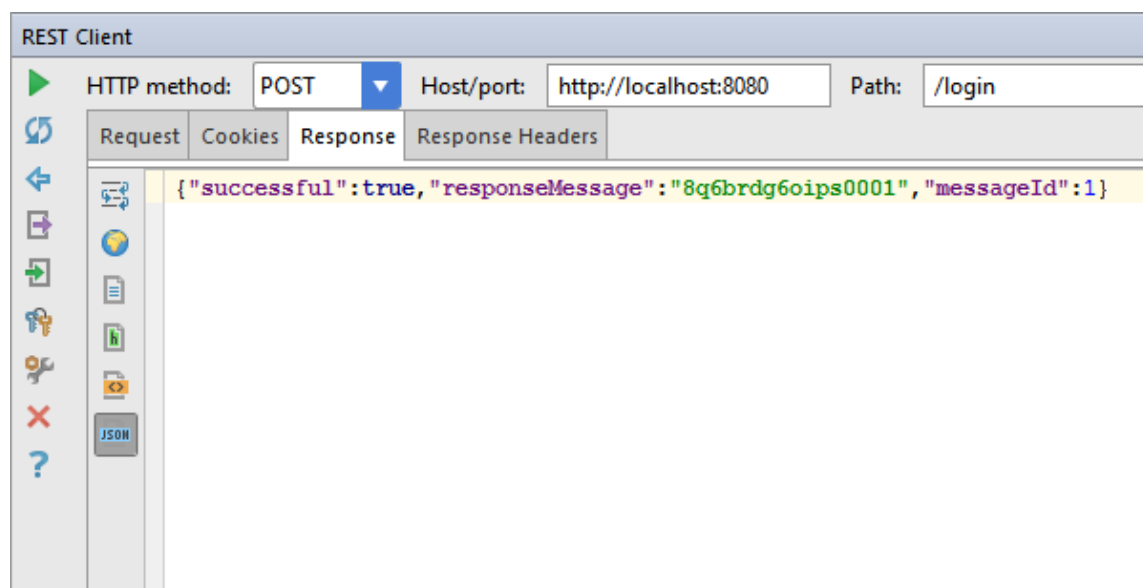
Listing 14: Implementacja aplikacji klienckiej - integracja z serwerem

```
15  @Injectable()
16  export class HTTPService{
17      constructor (private _http: Http) {}
18
19      login(login:string, password:string):Observable<BasicResponse> {
20          var urlSearchParams = new URLSearchParams();
21          urlSearchParams.append('login', login);
22          urlSearchParams.append('password', password);
23          var params = urlSearchParams.toString();
24          var headers = new Headers();
25          headers.append('Content-Type', 'application/x-www-form-urlencoded');
26          console.log(SERVER_PATH + ENDPOINT_LOGIN + "?" + params);
27          return this._http.post(SERVER_PATH + ENDPOINT_LOGIN, params, {
28              headers: headers
29          }).map((response:Response) => response.json());
30      }
```

Powyżej widać przykład metody napisanej w języku *TypeScript* przy użyciu rozwiązań z *framework'a Angular2* - moduł obsługujący zapytania *HTTP*. Widoczna jest także zaimplementowana metoda logowania *login* - 19 linia. Zwraca ona charakterystyczny dla tego języka typ danych - *Observable* który pozwala "leniwie" załadować dane (np. ustawić czas oczekiwania na dane na 10 sekund i dopiero po nim pokazać błąd). Obiekt ten działa asynchronicznie względem reszty aplikacji - kiedy wywołany, uruchamia się oddzielny wątek, wywołując go i realizujący operacje na nim. Parametrami przy logowaniu są hasło oraz nazwa użytkownika, oraz opcjonalnie token. Jeśli token znajdzie się w zapytaniu do *serwera backendowego*, wtedy pobierany jest, przy pozytywnym sprawdzeniu danych i tokena, token o podwyższonych uprawnieniach.



(a) Zapytanie HTTP



(b) Odpowiedź na powyższe zapytanie

Rys. 31: Implementacja aplikacji klienckiej - przykładowe zapytanie HTTP

Powyżej widać przykład zapytania generowanego poprzez kod widoczny w poprzednim rysunku, oraz odpowiedź z *serweru backendowego*. Pod polem *"successful"* zapisana jest informacja o powodzeniu zapytania (*true* - powodzenie, *false* - niepowodzenie, a pod *"responseMessage"* widać wygenerowany token dla użytkownika, którym można się teraz posługiwać przy operacjach na indywidualnym koncie, zamiast danych logowania.

6.3.3 Główny moduł aplikacji

Listing 15: Implementacja aplikacji klienckiej - główny moduł aplikacji

```
11 @NgModule({
12   declarations: [
13     AppComponent,
14     LanguageChoiceComponent,
15   ],
16   imports: [
17     BrowserModule,
18     LoginModule,
19     MainComponentsModule,
20     HttpClientModule,
21     JsonpModule,
22     routing,
23   ],
24   providers: [APP_PROVIDERS],
25   bootstrap: [AppComponent]
26 })
27 export class AppModule {
28   constructor() {
29     localStorage.setItem('logged', "false");
30   }
31 }
32
```

Główny moduł aplikacji zawiera wszystkie globalne importowane klasy, oraz inne, zewnętrzne moduły. Poziom importowania (można także importować moduły na niższym poziomie aplikacji), są związane z cyklem życia danego obiektu. Jeśli import zadeklarowany jest na głównym module, to cała aplikacja będzie posiadała jeden obiekt realizujący funkcjonalność importowanej klasy. Ważną deklaracją jest *bootstrap* - 25 linia. Oznacza ona komponent główny, z której rozpocznie się wyświetlanie aplikacji.

Listing 16: Implementacja aplikacji klienckiej - router

```
5 const appRoutes: Routes = [
6   {path: '', redirectTo: '/login', pathMatch: 'full'},
7   ...LoginRoutes,
8   ...MainComponentsRoutes
9 ];
10
11 export const appRoutingProviders: any[] = [];
12
13 export const routing = RouterModule.forRoot(appRoutes);
14
```

"Router" jest bardzo ważnym elementem każdej aplikacji internetowej. To on oznacza wszystkie widoczne dla użytkownika *end-pointy*, oraz automatyczne przekierowania między nimi. Przykład takiego przekierowania widoczny jest w 6 linii kodu. Każda próba wejścia na domyślną nazwę strony (*www.nazwastrony.com/*) będzie kończyło się przekierowaniem na podstronę logowania. Poniżej tej linii dodatkowo są importowane definicje podstron z innych modułów aplikacji.

6.3.4 Logowanie

Poniżej zaprezentowana jest struktura i kod odpowiedzialny za funkcjonalność logowania i wylogowywania.

Listing 17: Implementacja aplikacji klienckiej - logowanie *HTML*

```

1  <div *ngIf="logged == false">
2    <div class="container">
3      <div class="title">
4        {{messages?.loginInfo}}
5      </div>
6      <div class="panel-body">
7        <div class="row">
8          <div class="input-field col s12">
9            <label for="email">{{messages?.inputLogin}}</label>
10           <input [(ngModel)]="username" id="email"
11             type="email" class="validate">
12         </div>
13       </div>
14
15       <div class="row">
16         <div class="input-field col s12">
17           <label for="password">{{messages?.inputPassword}}</label>
18           <input [(ngModel)]="password" id="password"
19             type="password" class="validate">
20         </div>
21       </div>
22
23       <div class="row">
24         <label for="login"></label>
25         <button (click)="login()"
26           class="btn waves-effect waves-light" id="login"
27           type="submit" name="action">{{messages?.buttonLogin}}
28         </button>
29       </div>
30       <span>{{errorMsg}}</span>
31     </div>
32   </div>
33 </div>
34 <div *ngIf="logged == true">
35   <div class="container">
36     <div class="row">
37       Zalogowany
38       <button (click)="logout()"
39         class="btn waves-effect waves-light"
40         type="submit" name="action">Logout
41     </div>
42   </div>
43 </div>
44 </div>

```

Charakterystycznym rozwiązaniem dla języka *Angular2* jest użycie specjalnych znaczników takich jak **ngIf="warunek"* - 1 i 34 linia. Ten znacznik odpowiada za wyświetlanie danego fragmentu kodu dla użytkownika, przy spełnieniu warunków. Dodatkowo wyświetlane treści przechowywane są w strukturach. Przykład użycia takiej struktury widoczny jest w 4, 9, 7 i 27 linii. Dzięki takiemu rozwiązaniu, bardzo łatwo został zaimplementowany słownik obsługujący wiele języków.

Listing 18: Implementacja aplikacji klienckiej - logowanie CSS

```
1  div.container {  
2      margin-top: 200px;  
3      width: 50%;  
4      padding-left: 10%;  
5  }  
6  
7  label {  
8      width: 100px;  
9      display: inline-block;  
10 }  
11  
12 button#login {  
13     width: 150px;  
14 }  
15  
16 div.row {  
17     margin-top: 10px;  
18     margin-bottom: 10px;  
19 }  
20  
21 input {  
22     size: 50px;  
23 }  
24
```

Powyżej widoczny jest fragment kodu odpowiadający za stylistykę komponentu logowania.

Listing 19: Implementacja aplikacji klienckiej - logowanie *Component*

```

8  @Component({
9      moduleId: module.id,
10     selector: 'as-login-form',
11     templateUrl: 'login.html',
12     styleUrls: ['login.css']
13 })
14 export class LoginComponent {
15     username: string;
16     password: string;
17     logged: boolean;
18     basicResponse: BasicResponse;
19
20     messages: LoginMessages;
21
22     constructor(private _HTTPService: HTTPService, private _JSONReader: LanguageMessagesReader) {
23     };
24
25     ngOnInit() {
26         this.logged = JSON.parse(localStorage.getItem('logged'));
27         this._JSONReader.readLanguageFile(CONSTANTS.LANGUAGES.MESSAGE_FILE_NAMES.LOGIN)
28             .subscribe(response => {
29                 this.messages = response;
30             });
31     }
32
33     ngOnChanges(changes: SimpleChanges) {
34         // changes.prop contains the old and the new value...
35         console.log(changes.toString());
36     }
37
38     login() {
39         this._HTTPService.login(this.username, this.password)
40             .subscribe(response => {
41                 this.basicResponse = response;
42                 this.storeCredentials();
43             });
44     }
45
46     storeCredentials() {
47         if (this.basicResponse.successful) {
48             localStorage.setItem("refreshToken", this.basicResponse.responseMessage);
49             this.logged = true;
50         }
51     }

```

Powyższy komponent realizuje całą logikę funkcjonalności logowania. Widać na nim charakterystyczną strukturę kodu napisanego w języku *Angular2*. Adnotacja *@Component* oznacza, iż dany element ma być rozpoznawalny jako komponent oraz można go użyć jako znacznik napisany w języku *HTML* - "*selector: 'as-login-form'*" - 10 linia. Poniżej (linia 11 oraz 12) deklarowany jest widok kodu *HTML* wraz z plikiem stylizacji *CSS*, który jest częścią implementacji tego komponentu. Przykładem charakterystycznej funkcji dla *Angular2* jest *ngOnInit* - 25 linia - metoda automatycznie włączająca się za każdym razem gdy użytkownik uruchamia dany komponent (np. poprzez wejście na używającą go podstronę aplikacji).

6.3.5 Widok dostępnych aukcji

Listing 20: Implementacja aplikacji klienckiej - lista aukcji *HTML*

```
1 <div class="container">
2   <table class="table table-hover">
3     <tr>
4       <th>ID</th>
5       <th>Title</th>
6       <th>dateStart</th>
7       <th>dateEnd</th>
8     </tr>
9     <tr *ngFor="let auction of auctions" (click)="navigateToAuction(auction.id)">
10      <td>{{auction.id}}</td>
11      <td>{{auction.title}}</td>
12      <td>{{auction.dateStart}}</td>
13      <td>{{auction.dateEnd}}</td>
14    </tr>
15  </table>
16 </div>
```

Zdjęcie powyżej pokazuje rozwiązanie języka *Angular2* do pokazywania kolekcji. Wystarczy użyć w definicji elementu drzewa *HTML* znacznika **ngFor="..."* - 9 linia kodu z odpowiednimi argumentami, aby ten element, w zależności od wielkości kolekcji, został wyświetlony. Dodatkowo widać tutaj użycie funkcji, która przekierowuje nas do widoku aukcji po naciśnięciu na odpowiadający jej element.

Listing 21: Implementacja aplikacji klienckiej - lista aukcji *Component*

```

7  @Component({
8      moduleId: module.id,
9      selector: 'as-list-auctions',
10     templateUrl: 'listAuctions.html'
11 })
12 export class ListAuctionsComponent {
13     auctions: Auction[];
14
15     constructor(private _HTTPService: HTTPService, private _router: Router) {
16     };
17
18     getAuctions() {
19         this._HTTPService.getAuctions(50)
20             .subscribe(response => this.auctions = response);
21     }
22
23     ngOnInit() {
24         this.getAuctions();
25     }
26
27     navigateToAuction(id) {
28         console.log(id);
29         this._router.navigate([CONSTANTS.LOCAL_ENDPOINTS.SHOW_AUCTION, id]);
30     }
31 }
32
33

```

Widoczny powyżej rysunek przedstawia implementację funkcjonalności i logiki pokazywania aukcji.

6.3.6 Implementacja wielojęzyczności

Implementacja obsługi wielu języków działa na zasadzie zaczytywania gotowych plików z definicjami wyświetlanego tekstu w schematyczny sposób. Samym zaczytywaniem plików zajmuje się serwis danych, a funkcjonalnością dostępną dla użytkownika odpowiedni komponent.

Listing 22: Implementacja aplikacji klienckiej - implementacja wielojęzyczności - *HTML*

```

1  <div class="container languages">
2      
5      
8  </div>

```

Zdjęcie pokazuje napisany kod źródłowy *HTML*. Użytkownik widzi flagi, na które może nacisnąć, aby zmienić wyświetlany język.

Listing 23: Implementacja aplikacji klienckiej - implementacja wielojęzyczności - CSS

```
1  div.container.languages {
2      text-align: right;
3      float: right;
4      width: 100px;
5  }
6
7  .image {
8      height: 18px;
9      width: 23px;
10     border: 1px solid lightgray;
11     margin: 1px;
12 }
13
14 .image:hover {
15     -moz-box-shadow: 0 0 10px #ccc;
16     -webkit-box-shadow: 0 0 10px #ccc;
17     box-shadow: 0 0 10px #ccc;
18 }
19
20 .image:active {
21     -moz-box-shadow: 0 0 10px #ccc;
22     -webkit-box-shadow: 0 0 10px #ccc;
23     box-shadow: 0 0 10px #ccc;
24     border: 1px solid black;
25 }
```

Powyżej widoczna jest implementacja stylów flag. Po najechnaniu myszką na odpowiedni element, jest on otaczany szarym tłem. Kiedy kliknie się na element, zostaje zmieniony kolor obramowania.

Listing 24: Implementacja aplikacji klienckiej - implementacja wielojęzyczności - *Component*

```
4  @Component({
5      moduleId: module.id,
6      selector: 'as-language-choice',
7      templateUrl: 'languageChoice.html',
8      styleUrls: ['languageChoice.css']
9  })
10
11  export class LanguageChoiceComponent {
12
13      english: string = CONSTANTS.LANGUAGES.ENGLISH;
14      polish: string = CONSTANTS.LANGUAGES.POLISH;
15
16      constructor(private __ngZone: NgZone) {
17      };
18
19      setLanguage(language: string) {
20          localStorage.setItem(CONSTANTS.LOCAL_STORAGE.LANGUAGE, language);
21          this.__ngZone.runOutsideAngular(() => {
22              location.reload();
23          });
24      }
25  }
26
```

Serwis *LanguageMessagesReader* odpowiedzialny jest za zaczytywanie odpowiedniego pliku z danymi językowymi. Informacja o aktualnie wyświetlanym języku przechowywana jest w pamięci lokalnej - *localStorage*, za której zmianę odpowiedzialny jest wyżej opisany komponent. Jako argument do głównej metody *readLanguageFile* - 13 linia przyjmuje się nazwę pliku zaczytywanego. Każdy komponent posiada własny plik z definicjami słownika. Następnie zwracany obiekt jest już indywidualnie mapowany na strukturę danych.

Listing 25: Implementacja aplikacji klienckiej - implementacja wielojęzyczności - *Serwis*

```
6 @Injectable()
7 export class LanguageMessagesReader {
8     private readonly relativePathToSources: string = "src/app/shared/constant/languagesData/";
9
10    constructor(private _http: Http) {
11    }
12
13    readLanguageFile(fileName: string) {
14        let relativePathToData = this.relativePathToSources;
15        relativePathToData += localStorage.getItem(CONSTANTS.LOCAL_STORAGE.LANGUAGE) + "/" + fileName + ".json";
16        return this._http.request(relativePathToData)
17            .map(res => res.json());
18    }
19 }
20
```

6.3.7 Uruchomienie aplikacji klienckiej

Rys. 32: Wyciąg z konsoli po uruchomieniu aplikacji klienckiej

```
C:\Users\DZONI\IdeaProjects\ApplicationNameGamma\frontend>npm start

> angular2-starter@1.0.1 start C:\Users\DZONI\IdeaProjects\ApplicationNameGamma\frontend
> gulp serve-dev

===== Angular 2 Starter =====
Current environment: development
- Change environment via --env or NODE_ENV
- env.json is detected. 1 values loaded.
=====
[23:02:49] Using gulpfile ~\IdeaProjects\ApplicationNameGamma\frontend\gulpfile.js
[23:02:49] Starting 'serve-dev'...
[23:02:49] Starting 'sass'...
[23:02:49] Starting 'env'...
[23:02:49] Finished 'serve-dev' after 20 ms
[23:02:49] src/app/shared/constant/env.ts is generated successfully
[23:02:49] Finished 'env' after 35 ms
[23:02:49] Starting 'tsc-app'...
[23:02:50] Finished 'sass' after 823 ms
[23:02:56] Finished 'tsc-app' after 7.18 s
[23:02:56] Starting 'html'...
[23:02:56] Starting 'css'...
[23:02:57] Finished 'css' after 30 ms
[23:02:57] Finished 'html' after 41 ms
[23:02:57] Starting 'watch-sass'...
[23:02:57] Finished 'watch-sass' after 17 ms
[23:02:57] Starting 'watch-ts'...
[23:02:57] Finished 'watch-ts' after 31 ms
[23:02:57] Starting 'watch-html'...
[23:02:57] Finished 'watch-html' after 8.33 ms
[23:02:57] Starting 'watch-css'...
[23:02:57] Finished 'watch-css' after 9.75 ms
[BS] Reloading Browsers...
[BS] Access URLs:
-----
    Local: http://localhost:3000
    External: http://192.168.56.1:3000
-----
    UI: http://localhost:3001
    UI External: http://192.168.56.1:3001
-----
[BS] Serving files from: ./src/
[BS] Watching files...
```

Powyżej pokazany jest przykład uruchomienia serwera aplikacji klienckiej. Jest on uruchamiany z poziomu linii komend (z katalogu w którym znajduje się plik *index.html*) komendą *npm start*. Kiedy wszystko się załaduje, domyślna przeglądarka internetowa automatycznie włączy się na wyznaczonym adresie aplikacji klienckiej.

7 Użytkowanie aplikacji

Aktualnie aplikacja, dla wygody testowania, posiada widoczny na górze spis wszystkich użytecznych *endpointów*. W przyszłości oczywiście zostanie on wyłączony.

7.0.1 Logowanie i wylogowywanie

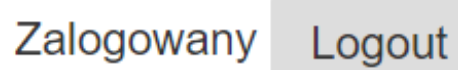
Zaloguj się, aby kontynuować lub stwórz nowe konto

**E-mail lub
login**

Hasło

Zaloguj się

Rys. 33: Użytkowanie aplikacji - logowanie

The image shows a user interface element. On the left, the word "Zalogowany" is displayed in a dark blue font. To its right is a light gray rectangular button with rounded corners, containing the word "Logout" in a dark blue font.

Rys. 34: Użytkowanie aplikacji - wylogowanie

Powyższe dwa rysunki przedstawiają funkcjonalność logowania i wylogowywania. Aby zalogować się, użytkownik musi podać nazwę użytkownika, bądź adres e-mail, użyty przy rejestracji konta.

7.0.2 Stworzenie nowej aukcji

Create new auction for given user

Title

End of auction

Load commodity

Location from

Delivery time

Location to

Size X

Size Y

Size Z

Weight

Fragile ☐

Living ☐

Special environment ☐

Description

Rys. 35: Użytkowanie aplikacji - stworzenie nowej aukcji

Wszystkie pola aukcji muszą być wypełnione. W innym przypadku nowa aukcja nie zostanie zapisana do bazy danych. Dodatkowo, aby można było stworzyć aukcję, użytkownik musi być zalogowany.

7.0.3 Stworzenie nowego użytkownika

Create new account

Username


Password


Email

Create account


Rys. 36: Użytkowanie aplikacji - stworzenie nowego użytkownika

Użytkownik musi wprowadzić prawidłowy adres e-mail, aby stworzenie konta powiodło się.

**applicationnamegamma@gmail.com**
do mnie ▾

 angielski ▾ > polski ▾ [Przetłumacz wiadomość](#)

Welcome,
Account name: username123
Password: password123
Activation code: <http://localhost:8080/activateUser/2/uaiantel00>




Kliknij tutaj, aby użyć polecenia [Odpowiedz](#) lub [Prześlęż dalej](#)

Rys. 37: Użytkowanie aplikacji - aktywacja nowego użytkownika

Jest to przykładowy e-mail wysłany po rejestracji konta. Użytkownik musi jedynie kliknąć w link aktywacyjny.

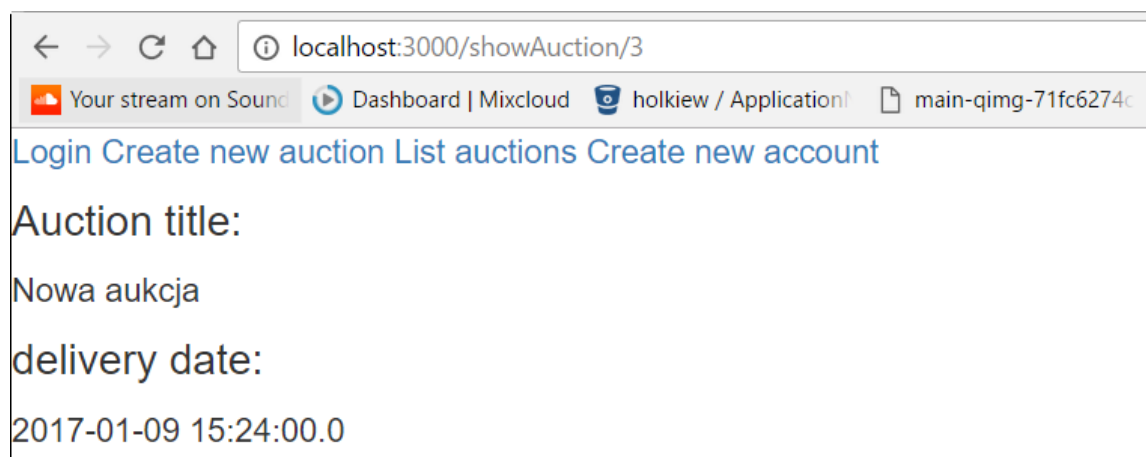
7.0.4 Wyświetlenie aukcji



ID	Title	dateStart	dateEnd
1	Title1	2016-01-01 12:00:00.0	2016-01-07 12:00:00.0
2	Title2	2016-01-01 12:00:00.0	2016-01-07 12:00:00.0
3	Nowa aukcja	2017-01-09 16:27:23.0	2017-01-09 15:24:00.0

Rys. 38: Użytkowanie aplikacji - wyświetlanie aukcji

Podstawowe wyświetlanie aukcji (na razie bez funkcjonalności wyszukiwania interesujących nas rekordów przy użyciu parametrów). Użytkownik może nacisnąć na wybrany wiersz, reprezentujący interesującą go aukcję, aby przejść do widoku poszczególnej aukcji. Dodatkowo po prawej stronie widoczne są flagi umożliwiające zmianę wyświetlanego języka.



Rys. 39: Użytkowanie aplikacji - wyświetlenie konkretnej aukcji

Jedynie funkcjonalność przejścia na konkretną podstronę została zaimplementowana, widok ma za zadanie potwierdzić poprawną integrację środowiska. Na samej górze widać pasek adresu, cyfra jest dynamicznie przetwarzana i odpowiadająca jej aukcja wyświetlana.

8 Wdrożenie i ocena jakości

8.1 Podstawowe wymagania instalacji systemu

8.1.1 Instalacja Javy

Instalacja dla systemu Windows Należy pobrać najnowszy pakiet Javy z oficjalnej dystrybucji Oracle (min. JDK/JRE 1.8.97) i zainstalować go: <http://www.oracle.com/technetwork/java/javase/downloads/index.html>

Instalacja dla Linuxa Ubuntu Domyślnie do każdego systemu Ubuntu dołączany jest pakiet JDK javy. Do uruchomienia aplikacji potrzebna jest pakiet JDK/JRE 1.8.97 lub wyższy. Te wymagania spełnia chociażby najnowsza wersja Ubuntu 16.06

8.1.2 Instalacja Maven'a

Dla obydwu systemów: Należy postępować zgodnie z instrukcją, która znajduje się na stronie głównej po kliknięciu na "Download", a następnie "Install". Instrukcje "Run" należy pominąć. <https://maven.apache.org/>

8.1.3 Instalacja nodeJS

Instalacja dla systemu Windows Ściągamy odpowiednią wersję pliku instalacyjnego ze strony: <https://nodejs.org/en/download/> i postępujemy wg. instrukcji. Po instalacji wykonanie komendy `npm -v` w wierszu poleceń powinno zwrócić wersję.

Instalacja dla Linuxa Ubuntu Wykonujemy serię poleceń z linii komend

1. `sudo apt-get update`
2. `sudo apt-get install nodejs`
3. `sudo apt-get install npm`
4. Jeśli wszystko przebiegło pomyślnie to komenda: `npm --version` powinna zwrócić aktualną wersję

8.1.4 Instalacja i konfiguracja bazy danych

Instalacja dla systemu Windows Na tym systemie instalacja sprowadza się do ściągnięcia najnowszej paczki instalacyjnej `.msi` ze strony: <https://dev.mysql.com/downloads/mysql/> Następnie należy przejść do instalacji. Nie należy zmieniać domyślnych ustawień podczas instalacji.

Za każdym uruchomieniem serwera należy uruchomić serwis bazodanowy:

- a. Włączamy wyszukiwarkę *Windows*.
- b. W wyszukiwarce wpisujemy: *usługi* bądź *services*.
- c. Wyszukujemy serwis MySQL i klikamy na niego prawym klawiszem myszki.
- d. Wybieramy opcję *uruchom*.

Instalacja dla Linuxa Ubuntu W systemie *Ubuntu* instalacja *MySQL* sprowadza się do wpisania jednej komendy. Należy wpisać polecenie: `sudo apt-get install mysql-server` Następnie należy postępować zgodnie z instrukcją.

Aby uruchomić serwis bazodanowy należy:

1. Włączyć konsolę oraz wpisać komendę: `service --status-all`
2. Następnie znaleźć nasz serwis bazodanowy, domyślnie *MySQL*
3. Wyszukaną nazwę należy podać w komendzie `service <service name> start` przykład: `service MySQL start`

(Linux i Windows) Konfiguracja bazy danych

1. Za pomocą polecenia należy zalogować się do bazy *MySQL* poleceniem: *mysql -u root -p*
2. Następnie należy wpisać hasło i zaakceptować
3. Stwórz bazę danych poleceniem: *create database <nazwa_bazy_danych >;*
4. Wybierz utworzoną bazę danych za pomocą komendy: *use <nazwa_bazy_danych >;*
5. Załaduj plik tworzący bazę danych: *source <ścieżka_do_pliku >;*
 <nazwa_bazy_danych >, przykład: db
 <ścieżka_do_pliku >, przykład: C:/user/Adam/Desktop/bazadanych.sql

8.2 Instalacja i budowa aplikacji

Aby przystąpić do uruchomienia aplikacji, należy najpierw przejść przez etap instalacji narzędzi - 8.1.

1. Należy ściągnąć ze strony i rozpakować projekt do folderu np. C:/projekt

Ręczne ściągnięcie

Należy wejść na stronę oraz ściągnąć projekt w formacie .zip <https://bitbucket.org/holkiew/applicationnamegamma>

Ściągnięcie za pomocą systemu *Git* Link który należy podać przy tworzeniu nowego lokalnego repozytorium <https://holkiew@bitbucket.org/holkiew/applicationnamegamma>

2. Wejść do katalogu w którym rozpakowaliśmy projekt (upewnić się, że jest w nim plik pom.xml, jeśli go nie ma, wyszukać go i przejść do tego katalogu)
3. Uruchomić linię komend (np. wpisując w domyślnym eksploratorze Windows, w miejscu gdzie jest widoczna ścieżka do katalogu, *cmd*)
4. W okienku wiersza poleceń wpisujemy: *mvn clean install*
5. Po pomyślnej instalacji wpisujemy *mvn build*
6. Następnie wchodzimy do katalogu "target", np. C:/projekt/target i wyszukujemy plik *projekt-numerWersji-SNAPSHOT.jar* np. *projekt-0.0.1-SNAPSHOT.jar*
7. Upewniamy się czy plik jest aktualny (Data modyfikacji pliku)
8. Wchodzimy do katalogu *frontend* np. C:/projekt/frontend
9. Uruchamiamy linię komend w tym katalogu
10. Wpisujemy komendę *npm install* (instalacja trwa około 2 minuty)

8.3 Uruchomienie aplikacji

Jeśli dostarczony został kod źródłowy aplikacji, należy uprzednio wykonać kroki z działu 8.2.

1. Uruchamiamy serwis bazy danych
2. Wyszukujemy plik aplikacji (np. *projekt-0.0.1-SNAPSHOT.jar*)
3. Wchodzimy do tego pliku np. za pomocą programu *WinRAR* (otwórz za pomocą *WinRAR*) bądź innego programu do archiwizacji danych
4. Przechodzimy do katalogu *BOOT-INF/classes/*
5. Konfigurujemy odpowiednie pliki, gdzie:
 - A. *application.properties* - zmieniamy tylko namiary na bazę danych (port oraz dane logowania)
 - B. *hibernate-annotation.cfg.xml* - robimy to samo co powyżej
 - C. *mail.properties* zmieniamy konfigurację serwisu poczty elektronicznej z której będzie korzystać aplikacja
 - D. *web.properties* zmieniamy właściwości aplikacji
6. Uruchamiamy aplikację (podwójne kliknięcie na plik *.jar* np. *projekt-0.0.1-SNAPSHOT.jar*)
7. Wchodzimy do katalogu gdzie znajduje się aplikacja kliencka (domyślnie katalog *frontend*)
8. Plik w katalogu *frontend/src/config/gulp/config.js* zawiera konfigurację portów wyjściowych
9. Uruchamiamy linię komend i wpisujemy komendę: *npm run serve-build* co uruchamia serwer aplikacji
10. *Jeśli chcemy wygenerować tylko pliki statyczne, aby uruchomić stronę na własnym serwerze, należy użyć komendy *npm run build* zamiast *npm run serve-build*. Będą one znajdować się w katalogu *frontend/build*

8.4 Testy aplikacji

Rozdział ten został podzielony na dwa podrozdziały. W pierwszym z nich zostały omówione testy które już istnieją w systemie. W drugim natomiast są opisane testy które należy stworzyć. Testy automatyczne są napisane jedynie na poziomie serwera aplikacji. Strona kliencka była testowana ręcznie.

8.4.1 Istniejące testy

Zostały stworzone testy jednostkowe polegające na sprawdzaniu przy każdym uruchomieniu serwera czy żaden z kontrolerów nie zmienił swojej definicji (przyjmowanych parametrów, rodzaju zapytania *HTTP*) oraz zwracanego obiektu. Do stworzenia tych testów została użyta biblioteka *Mockito* umożliwiająca zaślepienie funkcjonalności pobierania informacji z bazy danych. Są to bardzo ważne testy, ponieważ każda zmiana definicji *endpointa* może popsuć wymianę informacji z aplikacjami klienckimi.

8.4.2 Testy które powinny zostać zaimplementowane

1. Testy jednostkowe funkcji logicznych serwera jak i aplikacji klienckiej.
2. Testy *endpointów* aplikacji klienckiej.
3. Testy tzw. *End-to-end*, polegające na przetestowaniu całej ścieżki konkretnej funkcjonalności aplikacji, np. tworzenia aukcji.
4. Testy integracyjne sprawdzające połączenia pomiędzy poszczególnymi modułami systemu.
5. Testy sprawdzające "wytrzymałość" aplikacji polegające na generowaniu losowych akcji przez określony czas na aplikacji klienckiej. Takie testy pomogłyby znaleźć nietypowe, trudne do wykrycia błędy.

8.5 Weryfikacja

System napisany jest bez wewnętrznej dokumentacji, jednakże wystarczającymi opisami są nazwy klas oraz ich funkcjonalność. Jest to spełnienie jednego z wymagań niefunkcjonalnych. Rozwiązania podzielone są na jak największą ilość metod, które opisują słownie, krok po kroku, implementacje funkcjonalności. Każdy powtórzony kod (np. jedna pętla zmieniająca dane, używana w ten sam sposób przez wiele innych metod) jest opakowany w funkcję. Takie rozwiązanie umożliwia łatwe wykrycie oraz naprawę błędów (które w innym przypadku, nawet po wykryciu, byłyby bardzo czasochłonne do wykrycia). Dodatkowo system został oparty na dobrze znanym wzorcu projektowym *MVC - Model-View-Controller*. Każda oddzielna funkcjonalność aplikacji posiada swój własny pakiet w projekcie. Podejście takie umożliwia łatwą segregację kodu oraz utrzymuje porządek i ułatwia poruszanie się po aplikacji. Stworzone i zaprojektowane testy pomagają nie tylko szybko wychwytywać błędy w oprogramowaniu jak i komunikacji pomiędzy poszczególnymi modułami, ale także służą do informowania osób modyfikujących kod, że ich zmiana może istotnie wpłynąć na inne funkcjonalności systemu. Dodatkowo zostały wykonane testy sprawdzające działanie aplikacji klienckiej na najpopularniejszych przeglądarkach internetowych - *Mozilla Firefox*, *Google Chrome*, *Microsoft Internet Explorer*. Na wszystkich z nich aplikacja działała poprawnie.

8.6 Walidacja

Na tym poziomie projektu bardzo trudno jest ocenić czy system jest prawidłowy ponieważ do jego funkcjonowania potrzebna jest dobrze zrealizowana aplikacja kliencka, współdziałająca z serwerem oraz bazą danych. O ile baza danych jest dobrze zaprojektowana i spełnia swoje zadania o tyle serwer jak i aplikacja kliencka jeszcze tego nie robią. Należałoby poświęcić jeszcze dużo czasu nad implementacją tych dwóch modułów systemu, aby doprowadzić je do pełnej używalności. Wtedy można byłoby ocenić poprawność systemu, jednakże na ten moment stworzony system nie spełnia wszystkich założeń.

9 Podsumowanie

9.1 Technologie

Jednym z celów pracy była nauka nowych technologii. Poniżej streściłem swoją subiektywną opinię nad głównie używanymi narzędziami przy tworzeniu pracy dyplomowej. Wszystkie użyte technologie, stojące po stronie *serwera backendowego*, czyli *Spring* oraz *Hibernate* okazały się bardzo dobrym wyborem. Są one dobrze udokumentowane, oficjalnie jak i nieoficjalnie, poprzez wsparcie przez innych użytkowników na forach. Nie doszukałem się żadnych błędów wymagających ode mnie zmiany podejścia do implementacji kodu. Baza danych postawiona przy użyciu *MySQL*, także nie sprawiała żadnych problemów. Jednakże dużo czasu spędziłem na konfiguracji aplikacji. Pomimo tego uważam, że czas poświęcony na to był optymalny do rozmiaru mojego projektu, ponieważ kiedy miałem napisane już wszystkie "szablony", tworzenie widocznej i użytkowej funkcjonalności zajmowało porównywalnie mało czasu do implementacji bez użycia tych dwóch technologii.

Niestety zawiodłem się na wybranej przeze mnie technologii realizującej widok u klienta czyli *Angular2*. Kiedy zaczynałem konfigurować projekt, technologia była jeszcze w fazie *Alfa*, jednakże z wyczytanych opinii innych użytkowników wywnioskowałem, iż jest na tyle stabilna, aby spróbować przy jej użyciu napisać aplikację kliencką. Niestety po miesiącu do użytku trafiła wersja finalna, która znacznie zmieniła podejście do deklaracji poszczególnych komponentów tworzących aplikację, jak i same krytyczne komponenty użytkowe (np. router) czy sposób uruchomienia głównego modułu. Wraz z tymi zmianami, większość bibliotek stworzonych przez indywidualnych użytkowników, jak i wszelakich wskazówek i rozwiązań popularnych problemów, dostępnych na forach, przestały być kompatybilne z aktualną wersją języka. Z początku było to dla mnie, początkującego użytkownika tego języka jak i rejonu technologii aplikacji internetowych, bardzo uciążliwe i mylące. Sama ta zmiana wymusiła na mnie przepisanie mojego kodu, co skutkowało straceniem około jednego dnia, na zrozumienie nowego podejścia i doprowadzenie mojej aplikacji do stanu użytkowego. Dodatkowo kiedy chciałem użyć biblioteki napisanej przez innego użytkownika, implementującą funkcjonalność *Google Maps*, okazało się, że została ona napisana właśnie za czasów wersji *Angular2 Alfa* i nie jest kompatybilna. Kolejnym problemem jest słabo opisana oficjalna dokumentacja oraz system *deploymentu* aplikacji. Dopiero od wersji finalnej *Angular2*, twórcy zaczęli wspierać oficjalne narzędzie kontroli i konfiguracji projektu - *Angular CLI*. Niestety nie jest ono perfekcyjne, bardzo dużo problemów jest z stworzeniem wersji produkcyjnej. Mimo wszystko uważam, iż jest to przyszłościowa technologia i jeśli jej twórcy rozwiążą większość aktualnych problemów, nowi użytkownicy chętniej sami zaczną ją wspierać poprzez własne rozszerzenia jak i obszerne informacje zwrotne.

9.2 Realizacja pracy

Realizacja pracy niestety nie przebiegła według założonego harmonogramu. Powodem tego było przecenienie własnej wytrzymałości. Założyłem, iż po każdym pełnym dniu pracy, przez 2 do 4 godzin będę pracować na pracę dyplomową. Pisanie aplikacji w taki sposób było bardzo nieefektywne, ponieważ byłem już zazwyczaj zmęczony całonocnym siedzeniem przed monitorem. Finalnie skończyło się na pisaniu pracy dyplomowej tylko w weekendy i to nie każde ze względu na liczne wyjazdy.

Implementacja aplikacji pokazała, że nie wolno lekceważyć nowych technologii. Próg wejściowy do *Angulara2* dla człowieka który nic nie miał wspólnego z technologiami webowymi jest na tyle duży, że musiałem kompletnie pominąć realizację samego wyglądu dla użytkownika. *Framework Hibernate* także przysporzył wiele problemów związanych z niezrozumieniem działania tej technologii, co znacznie opóźniło skonfigurowanie bazy danych w kodzie *Java*. Do ostatniego założonego dnia harmonogramu, udało mi się zrealizować projekt systemu, analizę rynku i technologii oraz część implementacji aplikacji. Po tym terminie dodatkowo udało mi się zrealizować funkcjonalność wielojęzyczności, testy po stronie serwera jak i dokończenie pisania pracy dyplomowej.

Podsumowując, wszystkie założone, główne cele projektu czyli *porównanie istniejących rozwiązań, analiza dostępnych narzędzi, zaprojektowanie systemu realizującego funkcjonalność tematu pracy dyplomowej i textitzaprojektowanie bazy danych* zostały spełnione. Cel opisany jako *nauka nowych technologii* został podsumowany w rozdziale - Technologie.

Z dodatkowych celów jedynie *implementacja wielojęzyczności aplikacji* została stworzona. Drugi z dodatkowych celów czyli *stworzenie dodatkowego interfejsu użytkownika na telefony komórkowe* okazał się zbyt czasochłonny, aby nawet rozpocząć jego projektowanie.

9.3 Wnioski

Dzięki projektowi nauczyłem się nowego podejścia do programowania - webowego oraz najpopularniejszego wzorca projektowego aplikacji internetowych *MVC - Model View Controller* jak i nowoczesnych, cenionych na rynku pracy, technologii i narzędzi umożliwiających szybką pracę w tej dziedzinie informatyki. Jednakże nie tylko miałem zaimplementować aplikację, ale też ją zaprojektować, łącznie z analizą rynku jak i technologiczną. Faza ta jest bardzo wymagająca i czasochłonna. Dobry projekt znacznie skraca czas implementacji oraz ucina wiele niepotrzebnych dyskusji nad funkcjonalnością do zaimplementowania, która może być postrzegana przez wielu ludzi inaczej, jeśli nie jest opisana, bądź jest opisana, ale niejasno i niejednoznacznie. Rozwój aplikacji będzie kontynuowany.

9.4 Dalsze możliwości rozwoju systemu

Projektowana aplikacja jest za duża, aby w proponowanym czasie ją kompletnie rozwiązać. Jest to zadanie na więcej osób, co wiąże się z kosztami, jak i wymaganymi później dochodami, aby ją utrzymać. Możliwości i cele przyszłego rozwoju aplikacji są następujące

1. Reklama aplikacji w serwisach internetowych pokrewnych z tematyką transportu towarów oraz w popularnych serwisach aukcyjnych np. *Allegro*
2. Zachęcenie dużych firm poprzez wprowadzenie dodatkowej funkcjonalności wspomagającej obsługę wielu zleceń i osób do zarządzania.
3. Pozyskanie funduszy na dalszy rozwój aplikacji np. unijnych bądź prywatnych inwestorów.
4. Stworzenie tzw. *Start-up'u*.

Literatura

- [1] Oficjalna strona oprogramowania intelliJ. <https://www.jetbrains.com/idea/>. Data ostatniej wizyty: 2016-10-01.
- [2] Oficjalna strona projektu owasp antisamy project. https://www.owasp.org/index.php/Category:OWASP_AntiSamy_Project#tab=Main. Data ostatniej wizyty: 2016-12-20.
- [3] Oficjalna strona systemu budowania projektu apache maven. <https://maven.apache.org/>. Data ostatniej wizyty: 2016-12-02.
- [4] Oficjalna strona systemu wersjonowania bitbucket. <https://bitbucket.org/>. Data ostatniej wizyty: 2016-10-01.
- [5] Oficjalna strona szablonów bootstrap. <http://getbootstrap.com/>. Data ostatniej wizyty: 2016-12-02.
- [6] Oficjalna strona *frameworku* spring na której można znaleźć informacje oraz dokumentacje wszystkich technologii wchodzących w ten pakiet. <https://spring.io/>. Data ostatniej wizyty: 2017-01-01.
- [7] Oficjalna strona *frameworku* spring zawierająca informacje na temat cyklu życia implementacji kontenera *IoC*. <https://docs.spring.io/spring/docs/current/spring-framework-reference/html/beans.html>. Data ostatniej wizyty: 2016-11-20.
- [8] Oficjalna strona z dokumentacją bazy danych mysql. <https://dev.mysql.com/doc/>. Data ostatniej wizyty: 2016-12-08.
- [9] Oficjalna strona z dokumentacją technologii angular2 przy użyciu języka typescript. <https://angular.io/docs/ts/latest/>. Data ostatniej wizyty: 2016-12-02.
- [10] Oficjalna strona z dokumentacją technologii java w wersji 8. <https://docs.oracle.com/javase/8/docs/>. Data ostatniej wizyty: 2017-01-29.
- [11] Oficjalna strona z dokumentacją *frameworku* hibernate. <http://hibernate.org/orm/documentation/>. Data ostatniej wizyty: 2016-12-02.
- [12] Strona internetowa portalu google trends. <https://www.google.com/trends/explore?q=uship>. Data ostatniej wizyty: 2016-11-17.
- [13] Strona internetowa portalu oferteo. <http://www.oferteo.pl/transport>. Data ostatniej wizyty: 2016-11-17.
- [14] Strona internetowa portalu uship. <http://www.uship.com/find>. Data ostatniej wizyty: 2016-11-16.
- [15] Strona na portalu wikipedia z podstawowymi informacjami użytego w projekcie systemu modelu mvc. <https://pl.wikipedia.org/wiki/Model-View-Controller>. Data ostatniej wizyty: 2016-10-01.

A Opis załączonej płyty CD/DVD

- Katalog *projekt* - zawiera cały projekt (aplikacja serwerowa jak i kliencka). Można go zainstalować na każdym środowisku, pamiętając o uprzedniej instalacji *Maven'a* który zarządza zależnościami. Użyte *IDE* do pisaniu projektu: *IntelliJ IDEA 15.0.4* - program ten nie powinien mieć żadnych kłopotów z instalacją projektu.
- Katalog *bin* - zawiera, skompilowany, plik wykonywalny samego serwera *backendowego* (należy odpowiednio skonfigurować bazę danych) oraz zbudowane pliki aplikacji klienckiej.
- Katalog *app_src* - zawiera pełne kody źródłowe plików pokazanych w pracy dyplomowej (listingi).
- Katalog *pdyp_src* - zawiera wszystkie pliki źródłowe pisanej pracy dyplomowej (wraz z samą pracą).
- Katalog *img* - zawiera widoki obrazujące działanie systemu.