

POLITECHNIKA ŚWIĘTOKRZYSKA
Wydział Elektrotechniki, Automatyki i Informatyki

KRZYSZTOF DRAGAN

Numer albumu: 083524

Aplikacja internetowa do wyszukiwania połączeń lotniczych

Praca dyplomowa inżynierska
na kierunku Informatyka

Opiekun pracy dyplomowej:
dr inż. Arkadiusz CHROBOT

Katedra Systemów Informatycznych

Kielce 2019

POLITECHNIKA ŚWIĘTOKRZYSKA
WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI I INFORMATYKI

Zatwierdzam:

PRZEDZIĘKAN
ds. Kształcenia i Służb Studenckich
na Studiach Specjalistycznych
Wydziału Elektrotechniki, Automatyki i Informatyki

Rok akademicki: 2018/19

Temat nr. 10/1539/2018/19

Dnia: 21 marca 2018

dr inż. Andrzej Stobiecki

ZADANIE NA PRACĘ DYPLOMOWĄ

Studiów pierwszego stopnia na kierunku
INFORMATYKA

Wydano dla studenta: **Krzysztof Dragan**

I. Temat pracy:

Aplikacja internetowa do wyszukiwania połączeń lotniczych

II. Plan pracy:

1. Wstęp
2. Opis rozwiązywanego zagadnienia
3. Przegląd istniejących rozwiązań
4. Projekt aplikacji
5. Implementacja
6. Testy
7. Uwagi i wnioski

III. Cel pracy:

Celem pracy jest stworzenie aplikacji internetowej, która będzie zbierała informacje o dostępnych połączeniach lotniczych i umożliwiała użytkownikowi zaplanowanie na ich podstawie podróży. Źródłem informacji o dostępnych połączeniach lotniczych, które aplikacja będzie pobierała automatycznie, powinny być strony internetowe agregujące dane o lotach lub serwisy przewoźników. Opracowane oprogramowanie powinno pozwalać na określenie takich parametrów wyszukiwania lotów, jak: czas podróży, termin i miejsce jej rozpoczęcia i zakończenia, liczba przesiadek, cena przelotów i gabaryty oraz waga zabieranego bagażu. Aplikacja powinna się składać z części serwerowej oraz klienckiej i być zaimplementowana w języku Java oraz przy użyciu technologii związanych z tym językiem.

IV. Uwagi dotyczące pracy:

V. Termin oddania pracy: **30 stycznia 2019**

VI. Konsultant:.....

Kierownik Zakładu

Zakładu Informatyki

Katedry Systemów Informatycznych
Wydziału Elektrotechniki, Automatyki i Informatyki

dr hab. inż. Roman Skórka, prof. PŁ
(pieczęć i podpis)

Opiekun pracy dyplomowej

Andrzej Stobiecki
(podpis)

.....
(imię i nazwisko)

Temat pracy dyplomowej celem jej wykonania otrzymałem:

Kielce, dnia 21.03.2018 r. Dragan Krzysztof
czytelny podpis studenta



Kielce, dnia 12.10.2018

.....Krzesław Dragan.....
Imię i nazwisko studenta

.....083524.....
nr albumu

.....ul. Dembowska 83, Ostrowiec Świętokrzyski.....
Adres zamieszkania

.....Informatyka, Systemy Informacyjne, IV rok, stacjonarne.....
Kierunek, specjalność, rok studiów, rodzaj studiów (stacjonarne, niestacjonarne)

.....dr inż. Arkadiusz Chrobot.....
Opiekun pracy dyplomowej inżynierskiej/magisterskiej*

OŚWIADCZENIE

Przedkładając w roku akademickim 20.18./19. opiekunowi pracy dyplomowej inżynierskiej/magisterskiej*, powołanemu przez Dziekana Wydziału Elektrotechniki Automatyki i Informatyki Politechniki Świętokrzyskiej, pracę dyplomową inżynierską/magisterską* pod tytułem:

.....Aplikacja internetowa do wyszukiwania połazem latniczych.....

oświadczam, że:

- 1) przedstawiona praca dyplomowa inżynierska/magisterska* została opracowana przeze mnie samodzielnie, stosownie do wskazówek merytorycznych opiekuna pracy,
- 2) przy wykonywaniu pracy dyplomowej inżynierskiej/magisterskiej* wykorzystano materiały źródłowe, w granicach dozwolonego użytku wymieniając autora, tytuł pozycji i miejsce jej publikacji,
- 3) praca dyplomowa inżynierska/magisterska* nie zawiera żadnych danych, informacji i materiałów, których publikacja nie jest prawnie dozwolona,
- 4) przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem stopnia zawodowego/naukowego w wyższej uczelni,
- 5) niniejsza wersja pracy jest identyczna z załączoną treścią elektroniczną (na CD i w systemie Archiwum Prac Dyplomowych).

Przymuję do wiadomości, iż w przypadku ujawnienia naruszenia przepisów ustawy o prawie autorskim i prawach pokrewnych, praca dyplomowa inżynierska/magisterska* może być unieważniona przez Uczelnię, nawet po przeprowadzeniu obrony pracy.

Zostałem uprzedzony:

- 1) o odpowiedzialności karnej wynikającej z art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t. j. Dz. U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystyczne wykonanie albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, videogram lub nadanie.”,
- 2) odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t. j. Dz. U. z 2012 r. poz. 572, z późn. zm.) „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwany dalej sądem koleżeńskim”

Prawdziwość powyższego oświadczenia potwierdzam własnoręcznym podpisem.

.....Dragan Krzesław.....
czytelny podpis studenta

* niepotrzebne skreślić

Aplikacja internetowa do wyszukiwania połączeń lotniczych

Streszczenie

Celem niniejszej pracy było stworzenie aplikacji internetowej, która pozwala na wyszukiwanie połączeń lotniczych korzystając z danych zawartych na stronach internetowych przewoźników bądź z innych centr danych. Aplikacja została podzielona na część kliencką oraz serwerową. Klient został napisany przy użyciu technologii Angular 6, natomiast serwer w technologii Java 10. W pracy znajduje się opis architektury stworzonej aplikacji, modułu wyszukiwania połączeń lotniczych a także zagadnień teoretycznych związanych z projektowaniem interfejsu użytkownika dla przeglądarki internetowej.

Słowa kluczowe: Java, Angular 6, REST, programowanie obiektowe, protokół HTTP, programowanie funkcyjne

A web application to search for flight connections

Summary

The purpose of the thesis was to build a web application, which will be able to search flight connections using data from websites of airlines or other data sources. The application is split into two parts: a client and a server. The client was implemented using technology of Angular 6, whereas server in Java 10 technology. Description of architecture built application, module of air connections searching and theoretical issues related to building user interface for web application are included in this thesis.

Keywords: - Java, Angular 6, REST, Object Oriented Programming, HTTP Protocol, Functional Programming

Spis treści

| | |
|--|-----------|
| Wstęp | 13 |
| 1. Opis zagadnienia | 14 |
| 1.1. Źródła danych o połączeniach lotniczych | 14 |
| 1.2. Parsowanie danych | 15 |
| 1.2.1. Język znaczników XML | 15 |
| 1.2.2. Notacja JSON | 17 |
| 1.2.3. Język HTML | 18 |
| 1.3. Wydajność wyszukiwania | 18 |
| 2. Przegląd istniejących rozwiązań | 20 |
| 2.1. Wyszukiwarka lotów Skyscanner | 20 |
| 2.2. Wyszukiwarka lotów Google Flights | 22 |
| 2.3. Porównanie aplikacji | 23 |
| 3. Projekt aplikacji | 25 |
| 3.1. Diagram przypadków użycia | 25 |
| 3.2. Diagram aktywności | 26 |
| 3.3. Architektura aplikacji | 27 |
| 3.4. Projekt bazy danych | 28 |
| 3.5. Projekt interfejsu użytkownika | 29 |
| 3.5.1. Projekt strony głównej | 29 |
| 3.5.2. Projekt panelu wyszukiwania lotów | 29 |
| 3.5.3. Projekt strony z wynikami wyszukiwania | 31 |
| 3.6. Moduł wyszukiwania połączeń lotniczych | 31 |
| 3.7. Komunikacja między komponentami aplikacji | 33 |
| 3.7.1. Protokół HTTP | 33 |
| 3.7.2. JDBC | 33 |
| 3.7.3. I/O | 33 |
| 3.8. Podsumowanie | 34 |
| 4. Implementacja | 35 |
| 4.1. Oprogramowanie po stronie klienta | 35 |

| | |
|---|-----------|
| 4.1.1. Podział strukturalny | 35 |
| 4.1.2. Moduły pobierania danych | 35 |
| 4.1.3. Interfejs graficzny | 37 |
| 4.2. Oprogramowanie po stronie serwera | 40 |
| 4.2.1. Podział na pakiety | 41 |
| 4.2.2. Oprogramowanie wyszukujące połączenia lotnicze | 42 |
| 4.2.3. Parsowanie danych | 43 |
| 4.2.4. Zabezpieczenia | 44 |
| 4.2.5. Kontrolery serwera | 45 |
| 5. Testy | 47 |
| 5.1. Testy jednostkowe | 48 |
| 5.2. Testy integracji | 49 |
| 6. Uwagi i wnioski | 51 |

Wstęp

Branża lotnicza to jedna z głównych gałęzi współczesnego transportu. Za jej początek uznaje się pierwszy pomyślny lot braci Wright 17 grudnia 1903 roku na polach Kitty Hawk. To wydarzenie zapoczątkowało proces tworzenia się przemysłu lotniczego. W dzisiejszych czasach transport lotniczy uznaje się za najszybszy i najbardziej bezpieczny. Wykorzystuje się go między innymi w transporcie osób, towarów lub też w celach militarnych. Warto wspomnieć też o jego roli w rozwoju lotów kosmicznych. Dzięki niemu powstała nieosiągalna wcześniej możliwość podróżowania po całym świecie w względnie krótkim czasie.

W ramach niniejszej pracy stworzono aplikację internetową umożliwiającą wyszukiwanie realnych połączeń lotniczych. Aplikacja ta składa się z części klienckiej oraz serwerowej. Celem zwiększenia jej wydajności zastosowano mechanizmy pamięci podręcznej. Jej głównym zadaniem jest zbieranie danych o połączeniach lotniczych zewnętrznych serwisów oraz zasobów internetowych. Pozwala ona na uzyskanie informacji o lotach z uwzględnieniem danych o ich cenie, liniach lotniczych je obsługujących, wymiarach i wagach dozwolonego bagażu, czasie podróży lub też liczbie przesiadek.

Praca została podzielona na 6 rozdziałów. Rozdział pierwszy przedstawia zagadnienie wyszukiwania połączeń lotniczych wraz z opisem najważniejszych problemów, które starano się rozwiązać w opracowywanej aplikacji. Drugi rozdział opisuje istniejące rozwiązania w dziedzinie wyszukiwania połączeń lotniczych, które dostępne są na rynku. Jest to zestawienie dwóch komercyjnych aplikacji które zyskały duże uznanie swoich użytkowników. Rozdział został zakończony podsumowującym porównaniem obu aplikacji. Rozdział trzeci przedstawia projekt aplikacji. Znajdują się w nim schematy z opisem, które przedstawiają architekturę powstałej aplikacji. Obrazują one podział funkcjonalności między bazę danych, warstwę logiki biznesowej oraz warstw prezentacji. Opisane zostały również ścieżki komunikacji pomiędzy tymi warstwami. Rozdział czwarty objaśnia implementację aplikacji. W rozdziale tym zostaną przedstawione najważniejsze fragmenty oprogramowania tworzące funkcjonalność aplikacji. Znajdują się w nim informacje o użytych technologiach jak i zewnętrznych bibliotekach którymi się posłużono. Piąty rozdział poświęcony jest testom oprogramowania, które potwierdzają zgodne z wymaganiami działanie aplikacji. Ostatni rozdział przedstawia uwagi i wnioski dotyczące tematu pracy oraz stworzonej aplikacji.

1. Opis zagadnienia

Głównym zagadnieniem podjętym w jest pozyskanie realnych danych o połączeniach lotniczych i zaprezentowanie ich w kompleksowym interfejsie oraz we względnie krótkim czasie dla użytkownika powstałej aplikacji.

Zagadnienie to można podzielić na 3 części:

- znalezienie źródeł danych o połączeniach lotniczych,
- parsowanie różnych typów danych,
- zapewnienie dobrej wydajności podczas wyszukiwania połączeń lotniczych.

Części te zostaną opisane w podrozdziałach bieżącego rozdziału.

1.1. Źródła danych o połączeniach lotniczych

Największą trudnością podczas pisania pracy było znalezienie odpowiednich zasobów danych, które nie byłyby płatne oraz zapewniałyby rzetelne i sprawdzone dane lotnicze. Poszukiwania zaczęto od złożenia podań do centrów danych o dostęp do ich zasobów. Większość z nich wymaga opłaty za swoje usługi, które sięgają nawet 10 000\$. Niektóre z nich oferują jednak darmowy, choć limitowany, dostęp do ich zasobów. Na potrzeby pracy wybrano serwis FlightLookup jako głównego dostawcę danych. Informacje przez niego dostarczane stanowią większą część odpowiedzi serwera. Darmowy dostęp jest do 500 zapytań w trakcie miesiąca. Dodatkowymi źródłami danych są:

- Skyscanner - serwis udostępniający średnie ceny przelotów w określonym przedziale czasowym oraz informacje dotyczące dozwolonego bagażu,
- Aviation Edge - usługa chmurowa udostępniająca dane o liniach lotniczych,
- ourairports.com - strona internetowa umożliwiająca pobranie danych o większości lotnisk na świecie.

Skyscanner jest komercyjną aplikacją zbierającą wiele rodzajów danych lotniczych. Jej szczegółowo działanie zostanie opisane w następnym rozdziale. W tej części pracy zostanie przedstawione użycie zasobów tego produktu w stworzonym oprogramowaniu. Zasoby Skyscanner'a dostarczyły danych dotyczących dozwolonego bagażu podczas podróży. Aktualne ceny lotów zostały pobrane z serwisu RapidApi korzystającego wewnętrznie z zasobów aplikacji Skyscanner. Jest on dostępny pod adresem sieciowym: <https://rapidapi.com/skyscanner/api/skyscanner-flight-search>.

Użycie jego usług wymaga podania miejsca jak i daty wylotu oraz przylotu. Dodatkowym obowiązkowym parametrem jest unikalny kod walutowy który umożliwiał zwrócenie

poprawnych wyników. Oprócz cen przelotów Skyscanner posiada też stronę internetową z tabelą opisującą dozwolone wymiary oraz wagę bagażu podczas przelotu. Adres tej strony to: www.skyscanner.net/news/tips/check-in-luggage-size-and-weight-restrictions. Zawartość tej strony jest parsowana przez część serwerową, jednak to zagadnienie zostanie opisane bardziej szczegółowo w następnym podrozdziale.

Kolejnym ważnym źródłem danych jest usługa chmurowa Aviation Edge. Serwis ten udostępnia czytelne dane o liniach lotniczych wykorzystując do tego interfejs REST¹. Do użycia tej usługi wymagane jest podanie unikalnego kodu linii lotniczej w celu jej zidentyfikowania oraz pobrania danych w postaci JSON. Aviation Edge jest bezpłatnym serwisem. Do korzystania z jego zasobów wymagane jest tylko założenie konta w celu uzyskania klucza identyfikującego użytkownika.

Ostatnim źródłem danych jest strona internetowa ourairport.com znajdująca się pod adresem internetowym: <http://ourairports.com/>. Jej zasobem, który został wykorzystany jako źródło danych jest plik CSV zawierający ponad 50000 rekordów dotyczących lotnisk. Przechowuje on dane z szerokiego spektrum typów lotnisk począwszy od tych małych dedykowanych dla awionetek, aż po największe lotniska świata takie jak London Heathrow Airport. Dane te gromadzone są w pliku CSV, który jest pobierany przez część serwerową. Następnie informacje te są filtrowane w celu wyeliminowania lotnisk obsługujących poniżej 4 tysięcy pasażerów rocznie.

1.2. Parsowanie danych

Dane dostarczone przez zewnętrzne usługi są w różnych formatach. Aby zebrać pełną odpowiedź serwera należy sparsować pojedyncze elementy a następnie zbudować z nich obiekty języka Java. (**To co Pan tu opisuje dzieje się cały czas w aplikacji, nie tylko wtedy gdy Pan ją tworzył. Należy zatem użyć czasu teraźniejszego. Proszę przerobić w ten sposób resztę rozdziału. Oczywiście tam, gdzie faktycznie coś było wykonane tylko raz proszę zostawić czas przeszły.**)

1.2.1. Język znaczników XML

XML² to standard formatowania treści wspierany przez organizację W3C. Określa on uniwersalną składnię, używaną przy oznaczaniu danych za pomocą znaczników. Ponadto oferuje standardowy format dokumentów komputerowych. Format ten można dostosowywać do dziedzin tak odmiennych jak witryny WWW, grafika wektorowa, serializacja obiektów, wymiana danych

¹Representational State Transfer

²eXtensible Markup Language

elektronicznych czy systemy poczty głosowej. Oznakowanie XML jest udokumentowane za pomocą specjalnych znaczników. Są to konstrukcje z ogranicznikami w nawiasach trójkątnych. Specyfikacja XML określa, jakie wymagania składniowe musi spełniać takie oznaczenie: w jaki sposób elementy są rozpoznawane przez parsery, jak przedstawiony jest znacznik, jakie nazwy elementów są akceptowalne oraz gdzie elementy powinny być rozmieszczone. Dokumenty słownika XML są podobne do dokumentów HTML³, jednak istnieją pomiędzy nimi zasadnicze różnice. Kluczową różnicą między XML a HTML jest to, że XML umożliwia tworzenie własnych znaczników, podczas gdy w HTML nie ma takiej możliwości[1].

```
1 <FlightDetails TotalFlightTime="PT3H35M"
2     TotalMiles="931"
3     TotalTripTime="PT4H25M"
4     FLSDepartureDateTime="2018-11-15T06:40:00"
5     FLSDepartureTimeOffset="+0100"
6     FLSDepartureCode="WAW"
7     FLSDepartureName="Warsaw"
8     FLSArrivalDateTime="2018-11-15T10:05:00"
9     FLSArrivalTimeOffset="+0000"
10    FLSArrivalCode="LHR"
11    FLSArrivalName="London Heathrow"
12    FLSFlightType="Connect"
13    FLSFlightLegs="2"
14    FLSFlightDays="...4..."
15 >
```

Listing 1.2.1: Fragment danych w formacie XML

XML jest językiem znaczników, nie jest on językiem programowania, magazynem danych czy też protokołem transportu sieciowego. XML oferuje możliwość formatowania danych. Zapewnia to im dostępność na wielu platformach i odporność na upływ czasu. W starszych wersjach standardu XML dokument zapisany za pomocą dowolnego oprogramowania na jednej platformie nie dawał się odczytywać na innej. Wersja 3.2 usunęła tę wadę. Zapewnia ona obsługę Unicode, co oznacza, że dokumenty XML składają się w całości ze znaków należących do zestawu Unicode.

Wyszukiwanie informacji o połączeniach lotniczych rozpoczęło się od odebrania danych z serwisu FlightLookup w postaci XML. Jest to najważniejsza operacja podczas wy-

³Hypertext Markup Language

szukiwania połączeń lotniczych. Dane zebrane na tym etapie są parametrami wyszukiwania podczas korzystania ze źródeł danych wykorzystanych w późniejszej fazie tego procesu. Przykład odebranej treści znajduje się na listingu 1.2.1. (Skąd jest Pan pewien, że ten listing będzie na poprzedniej stronie? Być może w ostatecznej wersji znajdzie się na następnej. L^AT_EX posiada system odwołań do listingów, tabel, rozdziałów, rysunków itp., który trzeba zastosować. Wymaga on użycia poleceń `label` i `ref`. Przykład dla listingów ma Pan w tym akapicie. Resztę proszę uzupełnić samodzielnie.)

Operacja parsowania obiektów XML na obiekty języka Java zostanie dokładniej opisana w rozdziale o implementacji. Załączony listing 1.2.1 przedstawia jeden z elementów danych XML o nazwie *FlightDetails*. Zawiera on przykładowe pola takie jak: *TotalMiles* i *FLSFlightDays*. Elementy te zostały odwzorowane w postaci klas języka Java o takich samych nazwach jak nazwa elementu. Powstała klasa posiada też takie same pola jak obiekt w XML.

1.2.2. Notacja JSON

Notacja JSON⁴ jest nowoczesnym sposobem prezentacji danych. Wywodzi się ona z języka JavaScript, w którym została głównym formatem prezentacji obiektów tej technologii. Notacja JSON jest zbudowana na dwóch strukturach:

- kolekcji par nazwa/wartość, w zależności od języka programowania zrealizowana jako obiekt, słownik bądź kolekcja danych,
- uporządkowanej listy wartości, w większości języków zrealizowanej jako tablica, wektor, lista lub sekwencja.

Są to uniwersalne struktury danych. Prawie wszystkie nowoczesne języki programowania wspierają je w odpowiedniej dla siebie formie. Notacja JSON jest to format danych, który jest wymienny między językami programowania. Ta właściwość czyni ją najpopularniejszym formatem wymiany informacji między aplikacjami oraz mikroserwisami[2]. W stworzonej aplikacji dane w tym formacie dotyczyły linii lotniczych oraz cen przelotów. W pierwszym przypadku obiekt JSON można było w prosty sposób skonwertować na obiekt języka Java. Przykładową strukturę zaprezentowano na listingu 1.2.2. Kod źródłowy odpowiedzialny za konwersję tego obiektu zostanie przedstawiony w rozdziale czwartym.

```
1  {
2      "airlineId": "1",
3      "nameAirline": "American Airlines",
4      "codeIataAirline": "AA",
5      "iataPrefixAccounting": "1",
```

⁴JavaScript Object Notation

```

6     "codeIcaoAirline": "AAL",
7     "callsign": "AMERICAN",
8     "type": "scheduled",
9     "statusAirline": "active",
10    "founding": "1934",
11    "nameCountry": "United States",
12    "codeIso2Country": "US"
13

```

Listing 1.2.2: Przykładowy obiekt w notacji JSON

1.2.3. Język HTML

HTML jest standardowym językiem znaczników do tworzenia stron WWW. Dokumenty HTML są podstawową treścią jaką generują przeglądarki internetowe. W pracy dyplomowej źródłem danych o wymiarach i wadze dozwolonych bagażów była strona internetowa aplikacji Skyscanner[3]. Listing 1.2.3 przedstawia fragment takich danych zapisanych w formacie HTML.

```

1 <table class="tftable" style="height: 3990px" border="1" width="578">
2   <tbody>
3     <tr><a href="https://www.skyscanner.net/news/tips/
4       aer-lingus-baggage-allowance-explained/">Aer Lingus</a></td>
5     <td>No free allowance</td>
6     <td>No size restriction
7       <p>&nbsp;</p>
8       <p><strong>32kg</strong> per bag, or<br>40kg across 2 bags</p>
9     </td>
10   </tr>
11   <tr>

```

Listing 1.2.3: Fragment dokumentu HTML

1.3. Wydajność wyszukiwania

Wyszukiwanie tak złożonych danych jak informacje o połączeniach lotniczych, a następnie parsowanie ich, niesie za sobą pewne konsekwencje. Są one związane z czasem. Użytkownik powinien otrzymać interesującą go treść w jak najkrótszym czasie. W celu poprawy wydajności aplikacji wprowadzono mechanizmy skracające czas odpowiedzi części serwerowej. Dla zbierania danych dotyczących lotnisk oraz bagażów zaimplementowano rozwiązania polegające na pobieraniu pełnych zbiorów tych informacji do bazy danych lub do pliku tekstowego znajdującego się na serwerze. Pozwoliło to na ograniczenie częstotliwości pobierania danych ze stron lub zewnętrznych baz danych.

Kolejnym rozwiązaniem było wprowadzenie funkcyjnego stylu programowania w kluczowych elementach części serwerowej, które odpowiadały za wyszukiwanie połączeń lotniczych. Programowanie funkcyjne wprowadzone w Javie 8 pozwala skrócić czas trwania niektórych operacji wykonywanych przez maszynę wirtualną Javy, a więc też czas wyszukiwania lotów. Implementacje tych rozwiązań są opisane w rozdziale czwartym.

Ostatnim zastosowanym mechanizmem są pamięci podręczne (ang. *cache*) danych. Część danych, która jest gromadzona przez aplikację w zasobach o długich czasach dostępu i względnie niskiej przepustowości jest dodatkowo przechowywana w pamięci RAM o lepszych parametrach[4]. To rozwiązanie jest często stosowane w nowoczesnych systemach informatycznych. Odczytywanie danych z tej pamięci jest szybsze w porównaniu na przykład z odczytywaniem dokumentów XML i JSON z pamięci masowej, zewnętrznej bazy danych czy też zasobów sieciowych. Przeznaczeniem pamięci masowych jest skracanie czasu odpowiedzi dla określonych zapytań do części serwerowej, które są zwielokrotnione i wykonywane przez wielu użytkowników. Wykonując operacje wyszukiwania lotów, moduł wyszukiwania sprawdza czy w pamięci podręcznej znajdują się poszukiwane wyniki. Jeśli tak, zwraca je użytkownikowi. W przeciwnym wypadku wyszukuje loty standardowym sposobem a wynik zapisuje do pamięci podręcznej. Dla danych o liniach lotniczych, bagażach oraz dla całego obiektu lotu w części serwerowej stworzono 3 kontenery pamięci podręcznych. Czas przetrzymywania danych w tych kontenerach wynosi odpowiednio: 7 dni, 7 dni oraz 2 godziny. Definiując czasy dla informacji o lotniskach oraz bagażach sugerowano się częstotliwością aktualizowania tych danych przez dostawców. Na ich stronach zmiany są dokonywane średnio raz na kwartał. Ponadto, opracowany serwer zawiera oprogramowanie, które codziennie sprawdza czy nastąpiła jakaś zmiana we wspomnianych zasobach. Jeśli taka sytuacja by wystąpiła, odpowiednie kontenery pamięci podręcznej zostaną wyczyszczone. Dane o połączeniach lotniczych charakteryzują się znacznie większą wrażliwością na zmiany w czasie rzeczywistym. Parametry lotu takie jak data wylotu bądź czas przelotu mogą okazać się dla użytkownika kluczowe i zadecydować o jego wyborze. Z tego powodu czas przetrzymywania tych danych w pamięci podręcznej został ograniczony do 2 godzin. Wybór informacji, które powinny znaleźć się w pamięci podręcznej bazuje na tzw. temperaturze danych. Trzy wymienione wcześniej zasoby są najczęściej wykorzystywane przez serwer, dlatego też zdecydowano się na stworzenie dla nich odpowiednich kontenerów pamięci podręcznej.

2. Przegląd istniejących rozwiązań

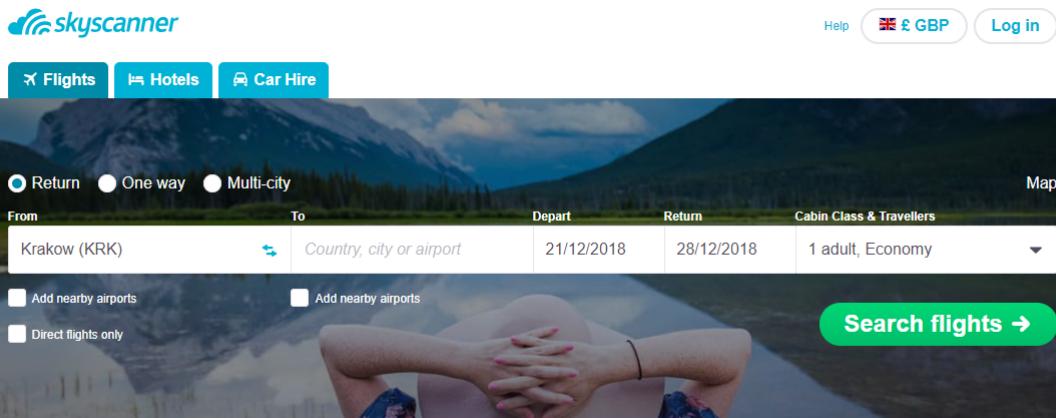
Analiza istniejących rozwiązań, czyli aplikacji wyszukujących połączenia lotnicze pozwoliła sprecyzować wymagania względem budowanego oprogramowania. W Internecie można znaleźć wiele aplikacji o podobnej lub takiej samej dziedzinie zastosowania, jak opracowana aplikacja. W tym rozdziale opisane są najbardziej znane wyszukiwarki lotów dostępnych na rynku.

2.1. Wyszukiwarka lotów Skyscanner

Pierwszym przykładem jest aplikacja internetowa Skyscanner. To wyszukiwarka lotów, która umożliwia użytkownikom szukanie według ceny i lokalizacji. Oprócz wyszukiwania lotów, Skyscanner oferuje opcje szukania hoteli blisko lotnisk oraz wypożyczania auta w pobliżu lotniska docelowego. Warto wspomnieć o innej jej usłudze, czyli udostępnianiu danych o połączeniach lotniczych zewnętrznym firmom i programistom. W związku z tym aplikacja Skyscanner była brana pod uwagę, jako potencjalne źródło danych, ale jej właściciel wymaga dużych opłat za swoje usługi, w warunkach akademickich niemożliwe było skorzystanie z nich. Aplikacja ta jest dostępna w ponad 30 językach oraz używana przez 60 milionów użytkowników miesięcznie. Skyscanner znajduje się pod adresem sieciowym: <https://www.skyscanner.net/>.

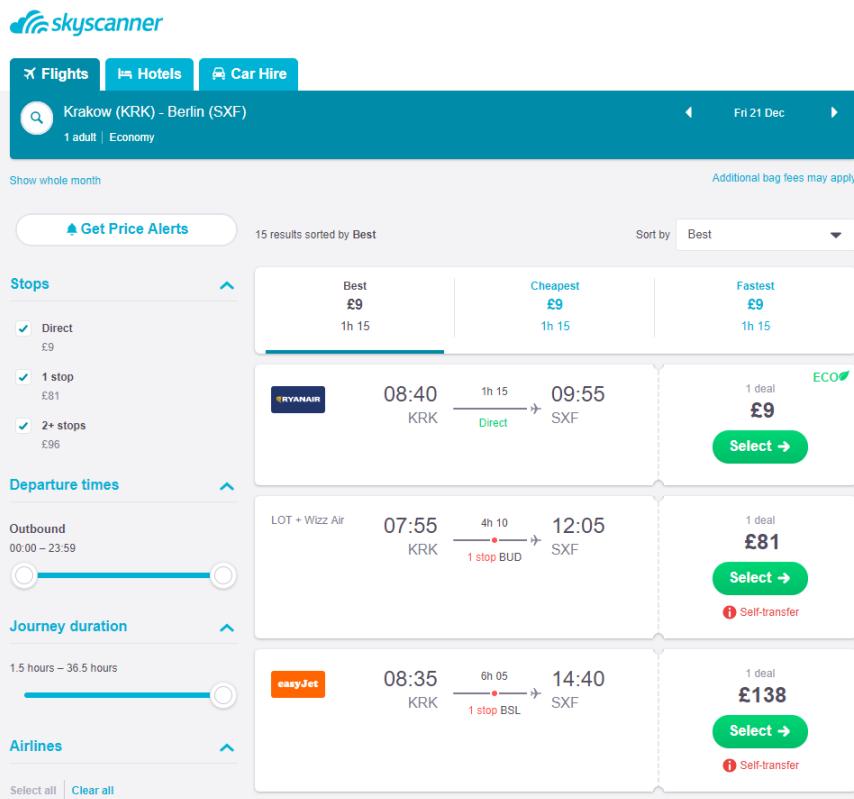
Opisywana aplikacja oferuje swoim użytkownikom szereg przydatnych funkcji. Korzystające z jej osoby mogą posługiwać się kompleksowym modułem wyszukiwania połączeń lotniczych. W panelu wyboru parametrów lotów można określić lotnisko wylotu, lotnisko docelowe jak i daty lotu. Skyscanner oferuje trzy opcje wyszukiwania lotów: z lotem powrotnym, lot w jedną stronę oraz podróż wieloetapowa. Dodatkowo użytkownik może określić liczbę osób biorących udział w podróży jak i klasę biletu lotniczego. Interesującą funkcją jest możliwość dodania pobliskich lotnisk do tych wybranych przez użytkownika. Znajduje ona zastosowanie, gdy na przykład aplikacja nie może znaleźć lotów z wybranego lotniska. Następuje wtedy wyszukiwanie lotów z pobliskich lotnisk znajdujących się w zasięgu o kształcie koła o zadanej długości promienia. Oprócz rozbudowanego interfejsu do wyboru parametrów lotów Skyscanner oferuje bardzo dobrą wydajność podczas wyszukiwania. Loty bezpośrednie wyszukiwane są natychmiastowo. Na loty z zaznaczoną opcją możliwych przesiadek trzeba poczekać zwykle kilka sekund. Interfejs aplikacji Skyscanner jest zaprezentowany na ilustracji 2.1.1.

Po wybraniu odpowiednich parametrów w widocznym na rysunku 2.1.1 menu należy nacisnąć zielony przycisk po prawej, aby rozpocząć proces wyszukiwania lotów. Przykładowe



Rysunek 2.1.1: Panel wyszukiwania lotów aplikacji Skyscanner

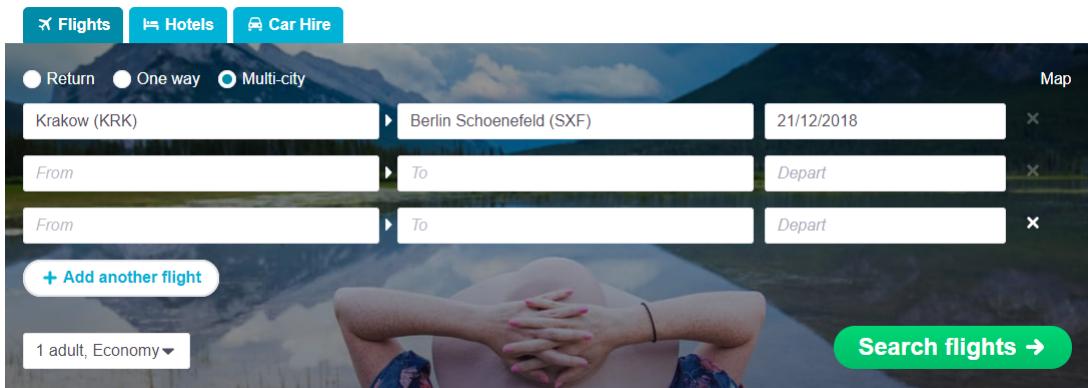
wyniki zaprezentowano na rysunku 2.1.2. Można na nim zauważyć wyszukane loty z dokładnymi informacjami o linii lotniczej obsługującej lot, miejsca wylotu oraz przylotu, czasie podróży jak i liczbie przesiadek. Z lewej strony dostępny jest panel dzięki któremu można dokonać filtrowania otrzymanych wyników.



Rysunek 2.1.2: Wyniki wyszukiwania lotów aplikacji Skyscanner

Warto zwrócić uwagę na panel podróży wieloetapowej, który jest dostępny po kliknięciu odpowiedniego przycisku powyżej pola wyboru lotniska wylotowego. Można w nim wybrać do siedmiu osobnych połączeń lotniczych i na ich podstawie zaplanować podróż. Każde z tych

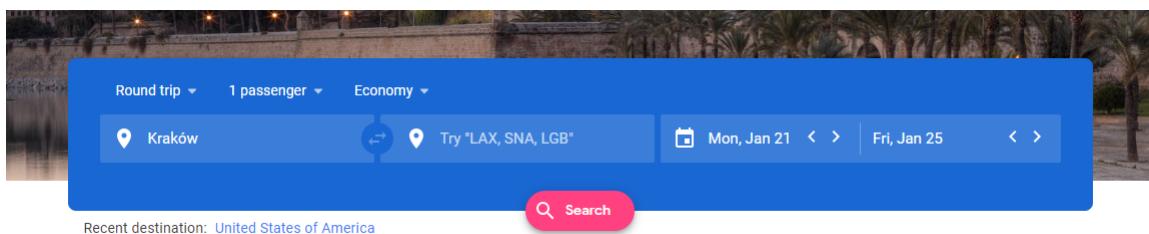
połączeń jest domyślnie lotem w jedną stronę.



Rysunek 2.1.3: Komponent podróży wieloetapowej

2.2. Wyszukiwarka lotów Google Flights

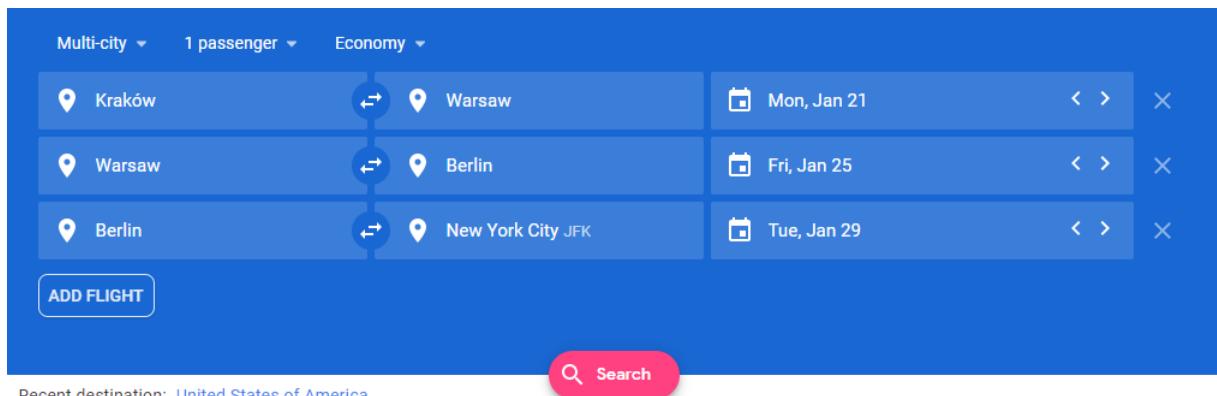
Kolejnym przedstawianym rozwiązaniem jest aplikacja internetowa Google Flights firmy Google. Jest to jeden z głównych internetowych produktów firmy z Kalifornii, zintegrowany z innymi usługami tej firmy, co czyni go bardzo praktycznym systemem. Google Flights zostało wdrożone 13 października 2011 roku. Ma zbliżone funkcje do wcześniej opisywanego konkurenta, Skyscanner. Oferuje jednak pewne dodatkowe udoskonalenia. Pozwala on użytkownikowi na wyszukiwanie lotów według takich kryteriów, jak czas trwania podróży oraz koszt. Największą zaletą tej aplikacji jest jej efektywność. Silnik tej wyszukiwarki jest podobny do tego stosowanego w Google Search. Wydajność tego oprogramowania została uzyskana przez użycie wyspecjalizowanych algorytmów szukających firmy Google oraz specjalnemu indeksowaniu danych. Oprócz tej zalety aplikacja prezentuje również czytelny i dostosowujący się do używanego medium (ang. *responsive*) interfejs, który jest dobrze oceniany przez użytkowników.



Rysunek 2.2.1: Panel wyszukiwania lotów aplikacji Google Flights

Rysunek 2.2.1 przedstawia panel wyszukiwania lotów oferowany przez opisywaną aplikację. Dostępny jest też komponent podróży wieloetapowej który znalazł się też w poprzednio opisywanej aplikacji oraz w aplikacji powstałej w ramach tej pracy.

Wszystkie produkty firmy Google, które dostępne są za pomocą przeglądarki WWW



Rysunek 2.2.2: Komponent podróży wieloetapowej aplikacji Google Flights

charakteryzują się specyfcznym stylem interfejsu. Jest to styl Material Design, który określa specyfikację pojedynczych elementów aplikacji Google'a. Rysunek 2.1.2 przedstawia przykładowe wyniki wyszukiwania lotów. Znajdują się one na rozwijanej liście. Po kliknięciu w wybrany element listy rozwija się panel z dokładniejszymi informacjami o locie. Każdy element zaprezentowany na wspomnianym rysunku jest zgodny z Material Design. Widać na przykład zaokrąglenia przycisków charakterystyczne dla produktów marki Google.

(Informacje w akapicie dotyczącym stylu są interesujące, ale czy na pewno wnoszą coś do pracy? Proszę się zastanowić nad jego usunięciem.)

Flight insights

- Dates**: Selected dates are among the cheapest
- Price graph**: Explore price trends for trips to Warsaw
- Airports**: Compare prices for airports near Warsaw
- Tips**: Plan your visit to Warsaw

Best flights ⓘ

Total price includes taxes + fees for 1 adult. [Additional bag fees](#) and other fees may apply.

| LOT | 11:10 AM – 12:05 PM | 55m KRK-WAW | Nonstop | PLN 150 | ▼ |
|-----|---------------------|----------------|---------|---------|---|
| LOT | 12:10 PM – 1:00 PM | 50m KRK-WAW | Nonstop | PLN 150 | ▼ |
| LOT | 9:05 PM – 10:00 PM | 55m KRK-WAW | Nonstop | PLN 150 | ▼ |

Track prices · Monitor the lowest price for this trip, and receive price alerts and travel tips by email

Rysunek 2.2.3: Wyniki wyszukiwania lotów

2.3. Porównanie aplikacji

Obie opisane aplikacje są rozwiązaniami komercyjnymi, stworzonymi przez wieloosobowe zespoły programistów. Są także systemami sprawnie działającymi, z czytelnym interfejsem

graficznym. Dostarczają one niemal identycznych usług - posiadają tę samą ofertę dla użytkownika, jednak różnią się jakością jej wykonania.

Zaletą aplikacji Skyscanner jest z pewnością rzetelność jej danych, która jest potwierdzona licznymi nagrodami od linii lotniczych. Warto też wspomnieć o jej systemie wyszukiwania lotów. Liczba wyszukanych lotów z przesiadkami często przekracza sto wystąpień. Świadczy to o zaawansowanym silniku wyszukującym i analizującym różne przypadki połączeń. Za wady można uznać dosyć prosty i niedostosowany do potrzeb urządzeń mobilnych interfejs użytkownika. Nie reprezentuje on poziomu nowoczesnych aplikacji.

Druga wyszukiwarka jest dopracowanym produktem firmy Google. Wyszkuje ona loty bardziej efektywnie niż rozwiązania konkurencji, między innymi Skyscanner. Google Flights współpracuje z systemami sprzedawczymi większości przewoźników lotniczych. Daje to możliwość kupna biletu bezpośrednio z jej strony WWW bez konieczności przechodzenia na zewnętrzną stronę linii lotniczej. Zaletą wyszukiwarki z pewnością jest dopracowany interfejs zgodny ze standardami firmy Google. Gwarantuje on dostosowanie do używanego medium oraz czytelność prezentowanych danych. Wyszukiwarka Google Flights wydaje się być lepszym rozwiązaniem od aplikacji Skyscanner jednak różnica jakości je dzieląca nie jest duża.

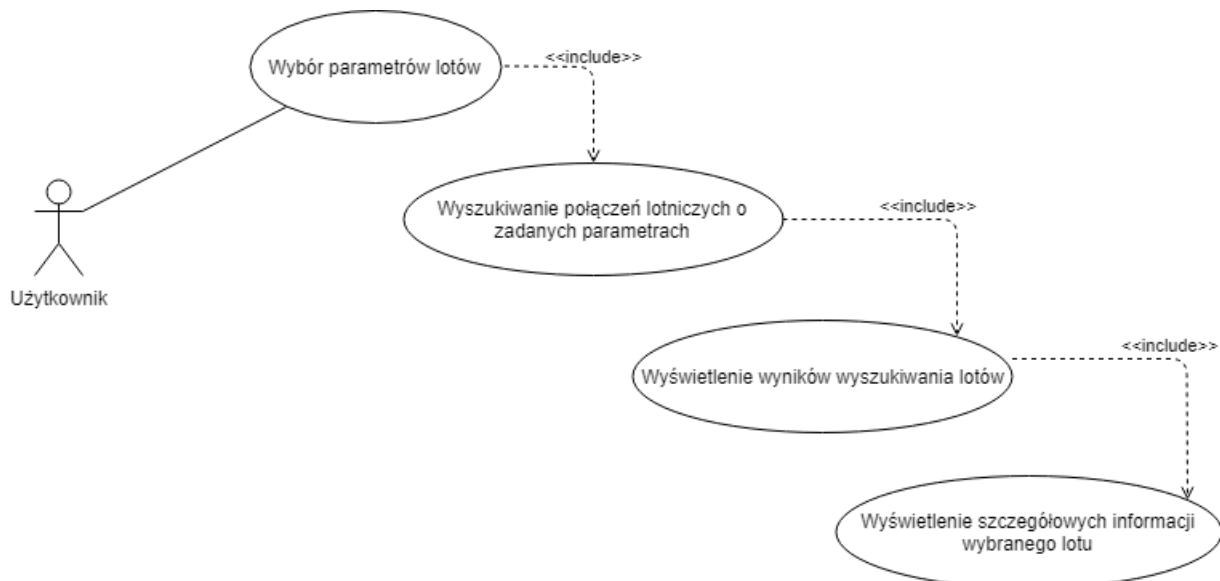
~~W opracowywanej aplikacji starano się w rzetelny sposób zrealizować podstawowe funkcje obu opisywanych wcześniej wyszukiwarek. Komercyjne aplikacje mają dużą przewagę biorąc pod uwagę szybkość wyszukiwania. W powstałej aplikacji czas odpowiedzi na żądania użytkownika jestauważalnie dłuższy. W warunkach akademickich trudno było o uzyskanie konkurencyjnych rezultatów. (Ten wykreślony akapit proszę przenieść do podsumowania pracy. Tutaj powinny się pojawić wymagania odnośnie tworzonej przez Pana aplikacji, które wynikają z analizy istniejących rozwiązań.)~~

3. Projekt aplikacji

W tym rozdziale jest opisany projekt aplikacji. Zrozumienie podejmowanego zagadnienia w pracy oraz przegląd istniejących rozwiązań pozwolił określić sposoby realizacji wymaganych funkcji w stworzonej aplikacji. Kluczową sprawą było też ustalenie odbiorców aplikacji oraz przebiegu procesu wyszukiwania połączeń lotniczych. W tym celu stworzono diagramy aktywności oraz przypadków użycia. Poświęcono osobne podrozdziały dla architektury aplikacji, projektu jej interfejsu użytkownika, strukturze serwera oraz modułowi wyszukiwania lotów. Na końcu rozdziału znajduje część podsumowująca opracowane rozwiązania i diagramy.

3.1. Diagram przypadków użycia

Diagram przypadków użycia przedstawia funkcje systemu wraz z jego otoczeniem. Jest on specjalnym rodzajem ogólniejszego zbioru diagramów UML. Na stronie WWW opisującej te diagramy, Michał Wolski przedstawia ich funkcje następująco: *"Diagramy przypadków użycia pozwalają na graficzne zaprezentowanie własności systemu tak, jak są one widziane po stronie użytkownika."*.[6]



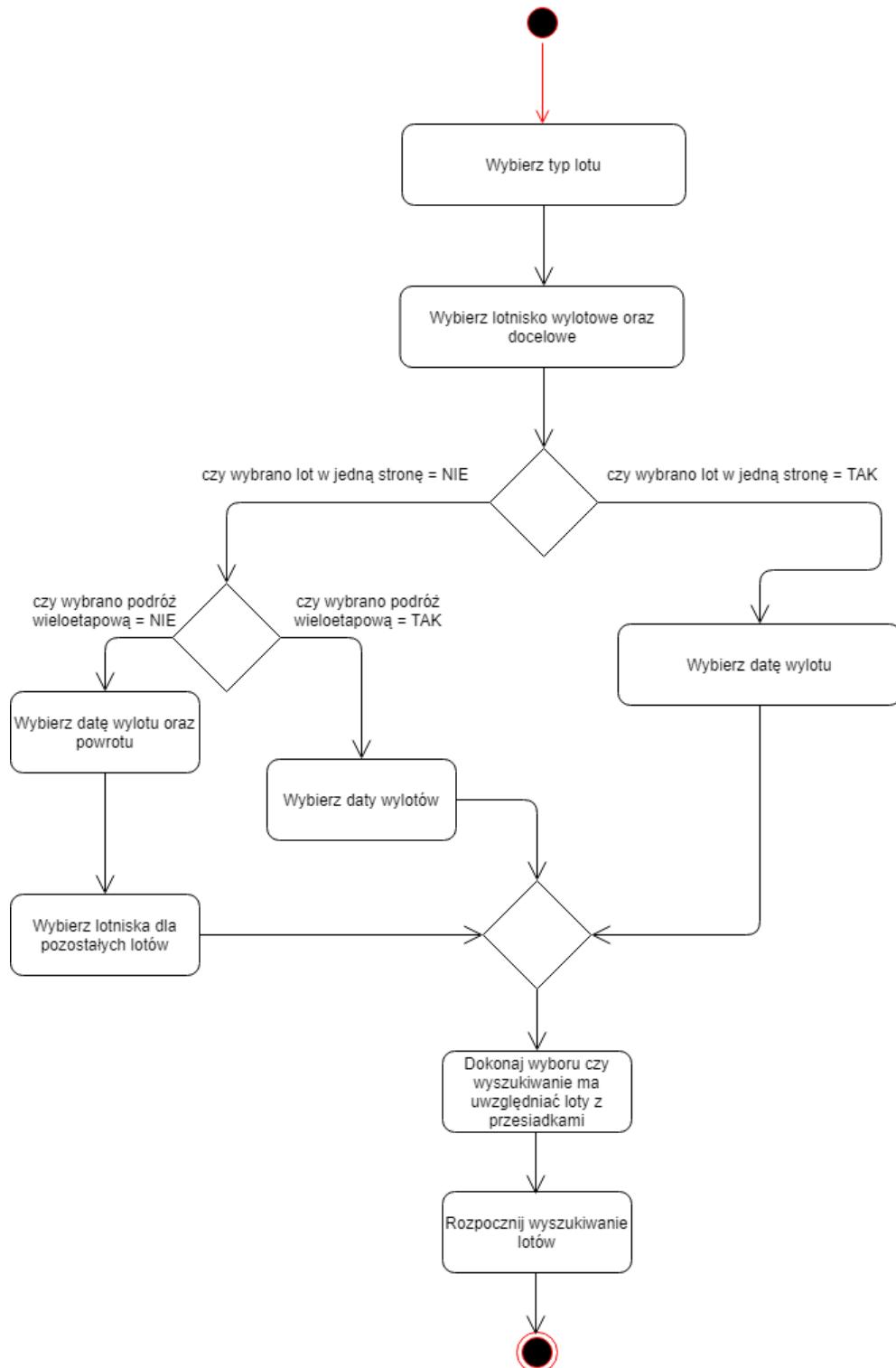
Rysunek 3.1.1: Diagram przypadków użycia aplikacji

Zaprezentowany diagram na rysunku 3.1.1 obrazuje aktorów i przypadki użycia w stworzonej aplikacji. Uwzględniono w nim tylko jedno aktora, którym jest *Użytkownik*. Reprezentuje on każdą osobę, która korzysta z usług aplikacji. Użytkownik ma możliwość wyboru parametrów lotów, który zawiera się w kolejnym przypadku użycia reprezentującym cały proces wyszukiwania połączeń lotniczych. Jest to oznaczona adnotacją *include*. Kolejnym przypadkiem użycia jest wyświetlenie wyników wyszukiwania lotów. Użytkownik mając te wyniki może wybrać

interesujący go przelot i uzyskać o nim dokładniejsze informacje.

3.2. Diagram aktywności

Diagram aktywności jest diagramem przedstawiającym interakcję dynamicznych aspektów systemu.[6]

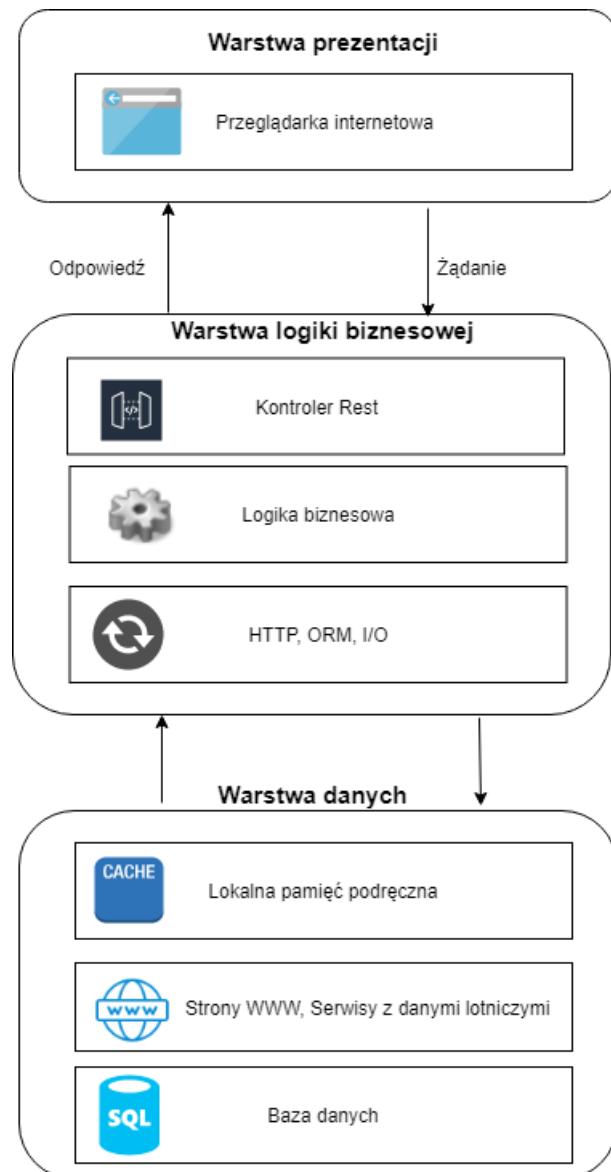


Rysunek 3.2.1: Diagram aktywności dla procesu wyszukiwania połączeń lotniczych

Dla opracowanej aplikacji przygotowano diagram aktywności widoczny na rysunku 3.2.1. Prezentuje on przebieg kroków, które wykonywane są przez stworzone oprogramowanie. Etap początkowy oraz końcowy został oznaczony czarnym kółkiem. Między nimi znajdują się poszczególne etapy procesu wyszukiwania lotów. Miejsca połączenia różnych ścieżek przebiegu tego procesu lub elementy warunkowe zostały oznaczone rombem.

3.3. Architektura aplikacji

Projektowanie aplikacji rozpoczęto od stworzenia schematu obrazującego poszczególne części systemu oraz schematy komunikacji między nimi. Opracowane oprogramowanie zostało stworzone według architektury trójwarstwowej. Jest ona rodzajem architektury oprogramowania, która składa się z trzech warstw logicznego przetwarzania danych.



Rysunek 3.3.1: Schemat architektury stworzonej aplikacji

Rysunek 3.3.1 zawiera schemat architektury opracowanej aplikacji. Przedstawia on jej trzy warstwy. Pierwszą z nich jest warstwa prezentacji. Renderuje ona treści odpowiednie treści w przeglądarce internetowej. Komunikacja między warstwą prezentacji a logiki biznesowej biznesowej odbywa się z użyciem protokołu HTTP. Warstwa logiki biznesowej obsługuje żądanie tego protokołu przekazując zawarte w nim parametry do modułu wyszukiwania połączeń lotniczych. Z uwagi na jego złożoność zostanie mu poświęcony osobny podrozdział.

3.4. Projekt bazy danych

Baza danych jest używana w aplikacji jako miejsce dające szybki dostęp do dużego zbioru jakim są dane o lotniskach. Na jej użycie zdecydowano się ze względu na duży rozmiar zasobu informacji reprezentowanych danych oraz sprawdzonych sposobów dostępu do niego.

| airport | |
|-------------------|---------|
| id | int(11) |
| ident | text |
| type | text |
| name | text |
| latitude_deg | double |
| longitude_deg | double |
| elevation_ft | int(11) |
| continent | text |
| iso_country | text |
| iso_region | text |
| municipality | text |
| scheduled_service | text |
| gps_code | text |
| iata_code | text |
| local_code | text |
| home_link | text |
| wikipedia_link | text |
| keywords | text |

Rysunek 3.4.1: Tabela danych lotnisk

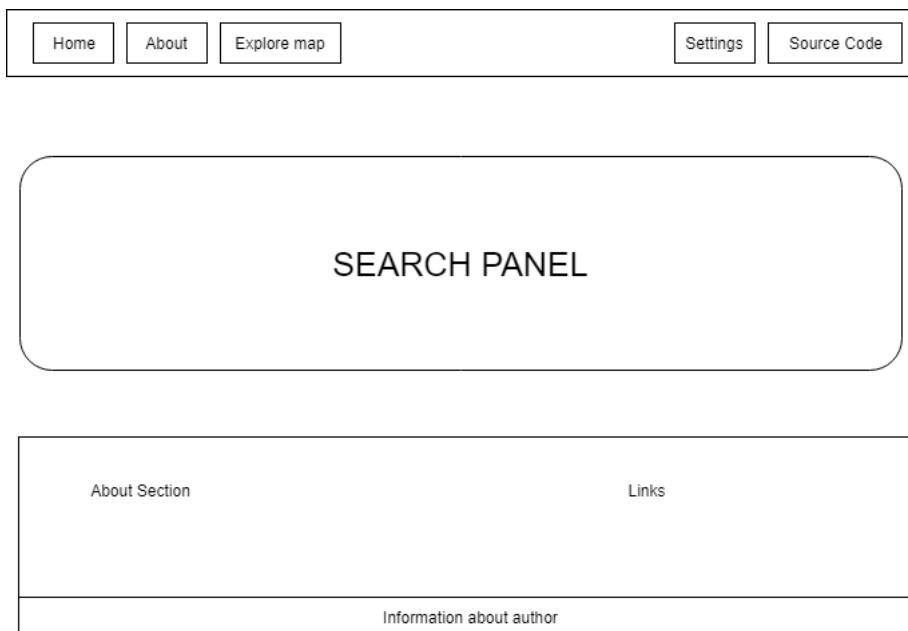
Rysunek 3.4.1 przedstawia jedyną tabelę znajdująca się w bazie danych, która przechowuje informacje dotyczące lotnisk. Tabela ta zawiera szereg parametrów takich jak na przykład: współrzędne lotniska, jego kody GPS i IATA, nazwę jak i jego typ. Zaprezentowane parametry są określone typami danych w jakich są przechowywane. Dane liczbowe są reprezentowane przez typ całkowity *int* lub przez typ podwójnej precyzji *double*. Parametry oznaczone typem *text* określają ciągi znaków.

3.5. Projekt interfejsu użytkownika

Klient aplikacji został wyposażony w intuicyjny interfejs użytkownika. Proces jego projektowania polega na tworzeniu schematów lub prototypów. W tym rozdziale zostaną opisane schematy interfejsu użytkownika przygotowane dla strony głównej aplikacji, panelu wyszukiwania lotów oraz dla strony z wynikami wyszukiwania lotów.

3.5.1. Projekt strony głównej

Prototyp strony głównej jest projektem widoku dla strony startowej aplikacji. Na rysunku 3.5.1 przedstawiono jego schemat.



Rysunek 3.5.1: Prototyp strony głównej aplikacji

Górna część strony przedstawia pasek nawigacji pozwalający na poruszanie się po aplikacji oraz konfiguruje jej ustawienia. Poniżej zobrazowany jest panel wyszukiwania lotów. Ze względu na jego złożoność został mu poświęcony osobny podrozdział. Dolna część strony ukazuje tak zwaną stopkę. Jest to komponent przeznaczony między innymi do zamieszczenia informacji o autorze aplikacji.

3.5.2. Projekt panelu wyszukiwania lotów

Panel wyszukiwania jest komponentem dynamicznym i podlegającym zmianom pod wpływem wyborów użytkownika. Z tego powodu zaprojektowano dla niego trzy różne widoki. Pierwszy prezentuje panel dla wybranego lotu z możliwością powrotu, drugi obrazuje go dla lotu bezpośredniego, a ostatni przedstawia panel podróży wieloetapowej. Na rysunku 3.5.2 zaprezentowano wygląd pierwszego wymienionego panelu.

The screenshot shows a flight search form with the following layout:

- Top row: "Round trip", "1 passenger", "Economy".
- Middle row: "Where from?", "Where to?", "Departure Date", "Return Date".
- Below middle row: A checked checkbox labeled "Direct flights only".
- Bottom right: A "Search" button.

Rysunek 3.5.2: Prototyp panelu wyszukiwania lotów z możliwością wybrania daty powrotu

Górna część ukazuje trzy przyciski wyboru typu lotu, liczby pasażerów oraz klasy poszukiwanego biletu. Poniżej zamieszczone są dwa pola do wyboru lotniska wylotowego oraz docelowego. Na prawo od nich znajdują się dwa pola wyboru dat dla wylotu oraz powrotu.

Dla lotu bezpośrednio wygląd panelu wyszukiwania zmienia się. Zaprezentowano to na rysunku 3.5.3 Można na nim zauważyc rozszerzone pole do wyboru daty wylotu. Opcja wyboru daty powrotu jest niewidoczna.

The screenshot shows a flight search form similar to the one above, but with expanded fields for direct flights:

- Top row: "Round trip", "1 passenger", "Economy".
- Middle row: "Where from?", "Where to?", "Departure Date".
- Below middle row: A checked checkbox labeled "Direct flights only".
- Bottom right: A "Search" button.

The "Departure Date" field is expanded into two separate fields: "Where from?" and "Where to?", each with its own "Departure Date" input field. There are also "X" icons next to the second "Departure Date" fields.

Rysunek 3.5.3: Prototyp panelu wyszukiwania lotów bezpośrednich

Na rysunku 3.5.4 przedstawiono projekt interfejsu dla komponentu podróży wieloetapowej. Można na nim dostrzec dodatkowe przyciski oraz pola. Komponent jest wyposażony w

The screenshot shows a complex travel component interface:

- Top row: "Round trip", "1 passenger", "Economy".
- Middle row: Two sets of "Where from?", "Where to?", and "Departure Date" fields. The second set has an "X" icon to its right.
- Bottom row: "Add Flight", "Reset", and a checked "Direct flights only" checkbox.
- Bottom right: A "Search" button.

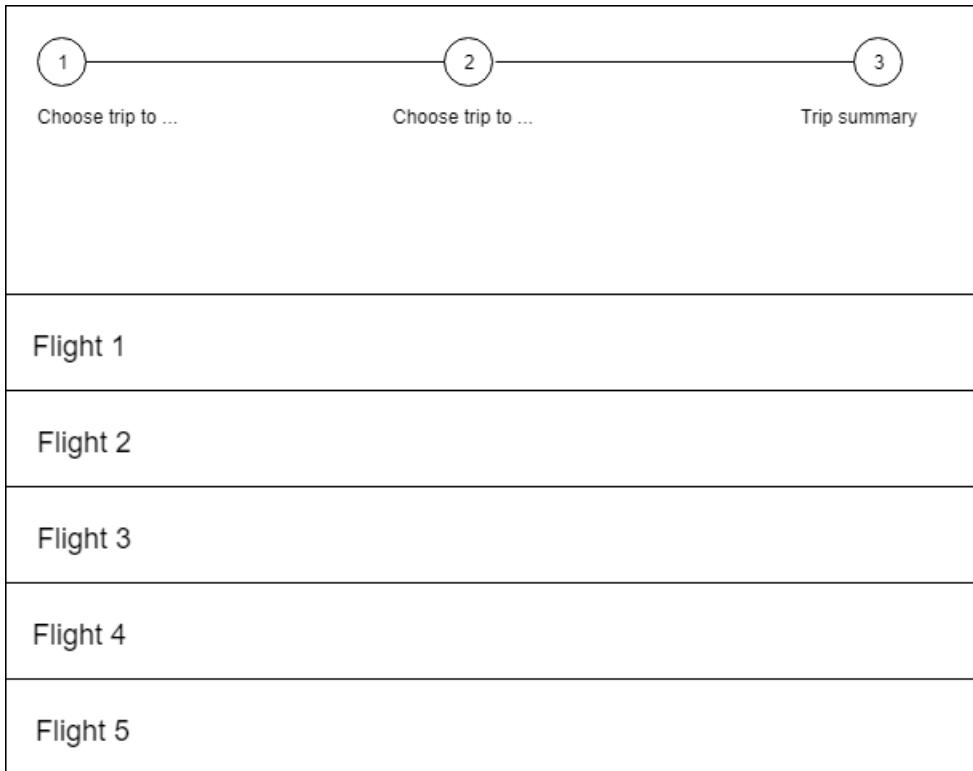
Rysunek 3.5.4: Prototyp komponentu podróży wieloetapowej

elementy wyboru lotnisk i daty wylotu, które można zwiększać lub zmniejszać odpowiednimi przyciskami. Przycisk oznaczony napisem *Add Flight* dodaje kolejne pola wyboru parametrów

lotu. Na prawo od daty wylotu można zauważyc ikonę do usuwania odpowiednich pól. Pozostałe elementy są zaczerpnięte z poprzednio opisanych schematów.

3.5.3. Projekt strony z wynikami wyszukiwania

Ostatnim prototyp dotyczy strony z wynikami wyszukiwania połączeń lotniczych. Zaprezentowano go na rysunku 3.5.5



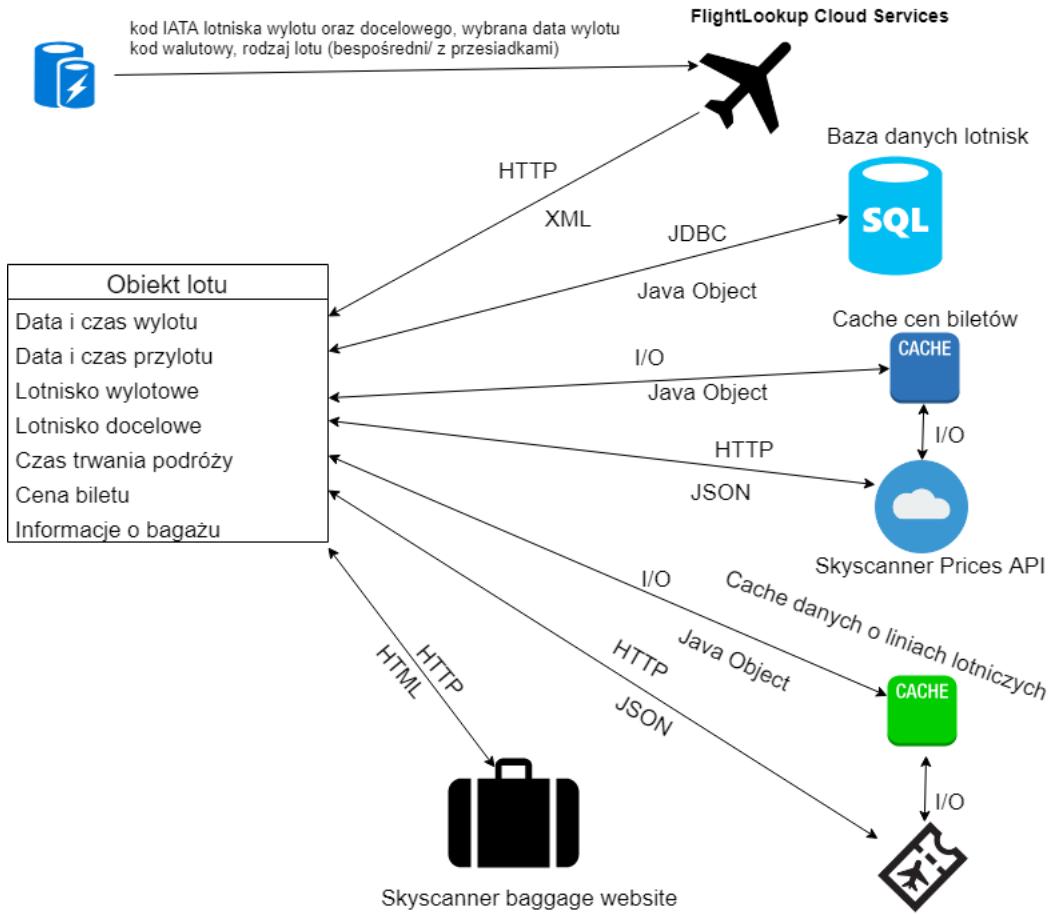
Rysunek 3.5.5: Prototyp komponentu podróży wieloetapowej

Górna część przedstawia zbiór przycisków, których liczba jest zależna od wybranego wcześniej typu lotu. Poniżej znajduje się lista wyszukanych lotów. Po kliknięciu w odpowiedni dla elementu listy przycisk powinien rozwinąć się panel z szczegółowymi informacjami o locie.

3.6. Moduł wyszukiwania połączeń lotniczych

Rozwiązywanie problemu wyszukiwania połączeń lotniczych jest zadaniem złożonym. Wymaga poradzenia sobie z problemem przeszukiwania wielu źródeł danych oraz konwertowania informacji przez nie zwracanych do postaci, w której mogą być przetwarzane zarówno przez stronę kliencką, jak i serwerową. W związku z tym wyodrębniono w strukturze aplikacji moduł, będący elementem części serwerowej oprogramowania, który jest odpowiedzialny za te czynności.

Na rysunku 3.6.1 przedstawiono schemat modułu wyszukiwania połączeń lotniczych wraz ze sposobami pozyskiwania danych do budowy obiektu przez niego zwrocanego. Napisy



Rysunek 3.6.1: Schemat architektury modułu wyszukiwania lotów

nad strzałkami są nazwami protokołów służących do realizacji zapytań, a te pod strzałkami oznaczają format treści zwróconej w odpowiedzi. Wyszukiwanie zaczyna się od wysłania żądania z kontrolera serwerowego do opisywanego modułu. Żądanie to musi uwzględniać parametry wyszukiwania. Moduł swoją pracę rozpoczyna od pozyskania danych lotniczych z serwisu FlightLookup. Treść odpowiedzi stanowi większą część całego obiektu lotu który jest zwracany. Po otrzymaniu tych danych moduł zwraca się do lokalnej bazy danych o udostępnienie danych lotnisk na podstawie ich unikalnych kodów IATA¹. Następnym krokiem jest pozyskanie ceny lotu. Czynność ta rozpoczyna się od sprawdzenia, czy poszukiwana informacja znajduje się w pamięci podręcznej. Jeśli takie dane są obecne, to pamięć podręczna zwraca odpowiedni zasób. Jeśli natomiast taka informacja nie jest dostępna, moduł zwraca się do serwerów firmy Skyscanner o udostępnienie cen za poszukiwany przelot. Analogicznie przebiega pozyskiwanie danych o linii lotniczej obsługującej lot. Ostatnim etapem wyszukiwania jest znalezienie informacji o dozwolonym bagażu podczas lotu. Jest to zrealizowane przez zwrócenie się do strony internetowej aplikacji Skyscanner posiadającej aktualne dane. Jej adres został wspomniany w rozdziale

¹International Air Transport Association

drugim. Po zebraniu wszystkich poszukiwanych informacji oraz sparsowaniu ich, obiekt lotu zwracany jest przez moduł wyszukiwania do kontrolera a następnie do części klienckiej gdzie użytkownik może obejrzeć wyniki całego procesu wyszukiwania.

3.7. Komunikacja między komponentami aplikacji

Opisane we wcześniejszych podrozdziałach komponenty narzucają określony sposób komunikacji między nimi. Każdy z nich zostanie przedstawiony w tym podrozdziale.

3.7.1. Protokół HTTP

Protokół HTTP² to jeden z najczęściej stosowanych protokołów aplikacji w Internecie. Jego oryginalnym zastosowaniem jest przesyłanie treści stron WWW. Jest on również stosowany w innych celach, np. jako protokół dla aplikacji sprzętowych.[9]. Cechą protokołu HTTP jest zapisywanie i odczytywanie reprezentacji danych, co umożliwia budowę systemów niezależnych od przesyłanych informacji. Protokół HTTP znalazł szerokie zastosowanie w stworzonej aplikacji. Wysłania żądania z parametrami wyszukiwania lotów możliwe było dzięki zbudowaniu odpowiedniego polecenia GET opisywanego protokołu. Serwer po obsłużeniu żądania odpowiadał treścią w formacie JSON również wykorzystując ten protokół. Dzięki niemu, możliwe było też pobieranie zasobów ze stron WWW oraz zewnętrznych baz danych.

3.7.2. JDBC

JDBC³ to standardowy, niezależny od bazy danych interfejs do interakcji z dowolnym źródłem danych w którym informacje przechowywane są w tabelach. Przeważnie jest używany do współdziałania z relacyjnym systemem zarządzania bazami danych. Za pomocą tego interfejsu można też korzystać z innych źródeł danych, jak arkusz csv, plik tekstowy itd. Zazwyczaj używa się go do łączenia z bazą danych, wyszukiwania danych i ich aktualizowania. Umożliwia on także wykonywanie procedur SQL przy użyciu składni niezależnej od bazy danych[10]. W opracowanej aplikacji interfejs JDBC pośredniczył pomiędzy relacyjną bazą danych a częścią serwerową aplikacji.

3.7.3. I/O

Podprogramy realizujące operacje zapisu oraz odczytu są podstawowymi częściami systemów operacyjnych oraz bibliotek języków programowania. Prawie wszystkie programy komputerowe wykonują pewne operacje zapisu/odczytu inaczej zwany operacjami wejściowymi i wyjściowymi. W opracowanej aplikacji za wykonywanie tych operacji odpowiada język Java ,który za pośrednictwem systemu operacyjnego korzysta z zasobów plikowych lub pamięci

²Hypertext Transfer Protocol

³Java Database Connectivity

podręcznej. Operacje I/O dotyczą systemu plików, który jest elementarnym składnikiem każdego systemu operacyjnego. Zarządza on archiwizacją danych oraz ich późniejszym odzyskiwaniem. System plików przetrzymuje dane w plikach, które zaś są przechowywane w katalogach. Dostęp do plików oraz katalogów uzyskuje się przez zdefiniowanie ścieżek dostępu, które lokalizują obiektu systemu plików.[11] Opracowany serwer wykorzystuje opisane operacje do odczytu pamięci podręcznej, plików CSV oraz plików tekstowych. Czynności te wspomagają pracę serwera pozwalając na szybszy dostęp do zasobów danych.

3.8. Podsumowanie

Zaprezentowane w tym rozdziale projekty oraz schematy pozwoliły zrozumieć sposoby działania aplikacji. Diagramy przypadków użycia oraz aktywności dostarczyły informacji o odbiorach aplikacji oraz o możliwościach jakie ona udostępnia. Schemat architektury trójwarstwowej podzielił aplikację na logiczne części objaśniając ich funkcje. Podrozdział dotyczący bazy danych pozwolił zrozumieć jej zastosowanie w powstałym systemie. Następnie zrealizowanie schematy interfejsu użytkownika przedstawiły ogólny zarys wymagań postawionych przed implementacją części klienckiej. Ze względu na swoją złożoność w kolejnej części tego rozdziału został omówiony moduł wyszukiwania lotów. Zaprezentował logikę stojącą za skomplikowanym procesem wyszukiwania. Ostatni podrozdział przybliżył rodzaje komunikacji między komponentami aplikacji oraz wyjaśnił ich użycie w aplikacji.

4. Implementacja

Następstwem stworzenia projektu aplikacji była implementacja oprogramowania zawartego w założeniach pracy dyplomowej. Została ona Implementacją aplikacji jest szczegółowo opisana w tym rozdziale. Jego treść została stworzona na podstawie projektu aplikacji oraz zagadnień poruszonych w poprzednich rozdziałach. Stworzona aplikacja Aplikacja została podzielona opracowana według założonych wymagań opisanych na stronie trzeciej otrzymanych na podstawie analizy problemu. Została jest ona podzielona na dwie logiczne części: stronę serwerową oraz kliencką. Opisano je ogólnie w poprzednim rozdziale, w tej części pracy skupiono się zagadnieniach związanych z implementacją powstałego oprogramowania. (To właśnie jest błędem w poprzednim rozdziale, że zawiera informacje na temat implementacji, nie projektu. To zdanie jest do poprawy.)

4.1. Oprogramowanie po stronie klienta

Część kliencka to osobna aplikacja osadzona w przeglądarce internetowej. Został w niej stworzony graficzny interfejs użytkownika oraz serwis odbierający dane od części serwerowej.

4.1.1. Podział strukturalny

Aplikacja kliencka została logicznie podzielona na cztery pakiety¹. (Ten rysunek nic nie wnosi do treści pracy.) Pakiet *layout* przechowuje komponenty o wielokrotnym użyciu, które są wielokrotnie używane w aplikacji klienckiej. Katalog *model* zawiera odpowiedniki klas modelowych z części serwerowej. Kolejnym elementem jest pakiet *pages*. Znajdują się w nim komponenty renderujące kompletne strony internetowe. Zostały, które są zbudowane z segmentów stworzonych umieszczonych w katalogu *layout*. Ostatnią częścią aplikacji klienckiej jest pakiet *services*. Zawiera on oprogramowanie odpowiedzialne za komunikowanie się z serwerem.

4.1.2. Moduły pobierania danych

W części klienckiej zaimplementowano serwisy (Proszę wyjaśnić, co ma Pan na myśli pisząc słowo „serwis”, bo jest ono dosyć wieloznaczne) odbierające dane z części serwerowej. Pierwszym z nich jest odpowiedzialny za połączenie z modelem wyszukiwania po stronie serwera. Zawiera on dane adresu sieciowego serwera udostępniającego zebrane wyniki wyszukiwania połączeń lotniczych.

```
1 import { Injectable } from '@angular/core';
2 import { baseUrl } from '../../../../../environments/environment';
3 import { HttpClient } from '@angular/common/http';
```

```

4 import { Observable } from 'rxjs';
5 import { MultiFlight } from 'src/app/model/interfaces/MultiFlight';
6
7 @Injectable({
8   providedIn: 'root'
9 })
10 export class FlightsService {
11
12   getFlightsURL: string = baseUrl + 'api/flights/';
13
14   constructor(private http: HttpClient) {
15   }
16
17   getFlights(originAirport: string, destinationAirport: string, departureDate: string,
18             typeOfConnection: string, currency: string) : Observable<MultiFlight[]>{
19     var requestUrl = this.getFlightsURL + originAirport + '/' + destinationAirport + '/'
20       + departureDate + '/' + typeOfConnection + '/' + currency;
21     return this.http.get<MultiFlight[]>(requestUrl);
22   }

```

Listing 4.1.1: Kod źródłowy funkcji pobierającej wyniki z serwera

Na **późszym** listingu 4.1.1 (Proszę tak odwoływać się w tekście do pozostałych listingów.) przedstawiono fragment **oprogramowania kodu** pobierający wyniki wyszukiwania. Funkcja *getFlights* przyjmuje **5 parametrów** pięć argumentów, które dostarcza jej użytkownik. **Pierwszym procesem** Pierwszą czynnością tej funkcji jest zbudowanie adresu URL, który posłuży jej do zbudowania żądania pobrania danych. Jej działanie polega na złączeniu wcześniej wspomnianych parametrów ~~oddzielając je znakiem "/" w roli separatora. Ostatnim krokiem jest stworzenia i stworzeniu~~ żądania GET protokołu HTTP.

Drugim zaimplementowanym serwisem jest prosta funkcja pobierająca dane o lotniskach. Jej ~~działanie zostanie przedstawione na kolejnym listingu~~ kod jest przedstawiony na **listingu** (Proszę uzupełnić.)

```

1 import { Injectable } from '@angular/core';
2 import { HttpClient, HttpHeadersResponse } from '@angular/common/http';
3 import { baseUrl } from '../../../../../environments/environment';
4 import { Airport } from '../../../../../model/interfaces/Airport';
5 import { Observable, throwError } from 'rxjs';

```

```

6 import { catchError, retry } from 'rxjs/operators';

7

8 @Injectable({
9   providedIn: 'root'
10 })
11 export class AirportsService {

12

13   getAirportsUsingIncompletePhraseURL: string = baseUrl + 'api/airports/
14     getAirportsStartingWith/';

15   getAirportsStartingWithPhrase(phrase: string): Observable<Airport[]>{
16     return this.http.get<Airport[]>(this.getAirportsUsingIncompletePhraseURL + phrase)
17       .pipe(catchError(this.handleError));
18   }
}

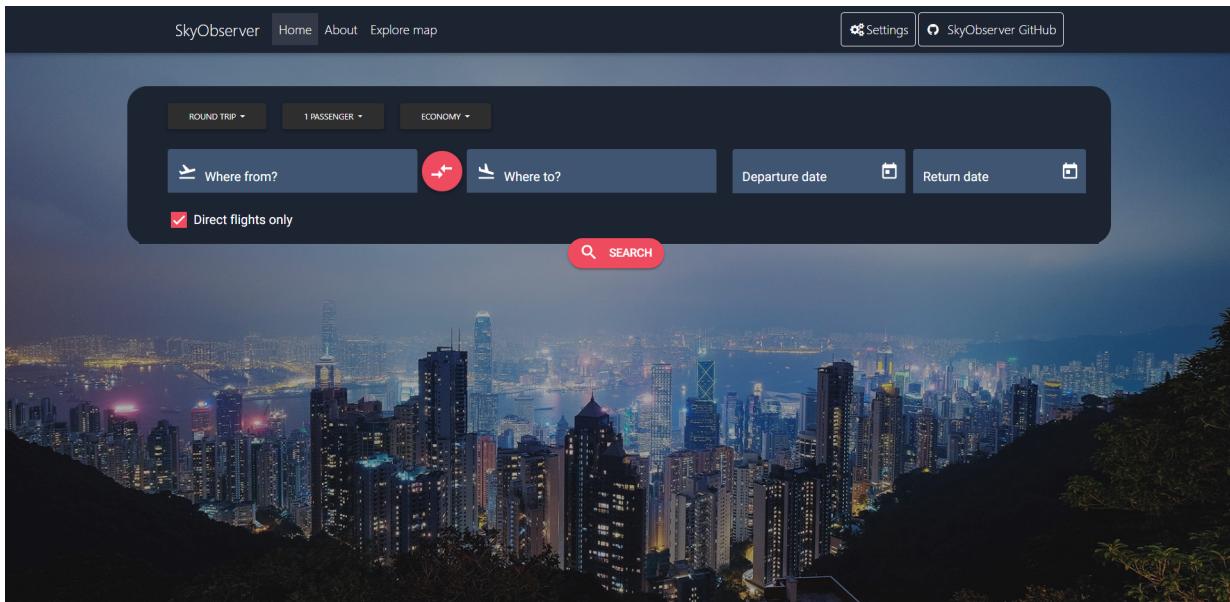
```

Listing 4.1.2: Kod źródłowy funkcji pobierającej dane lotnisk

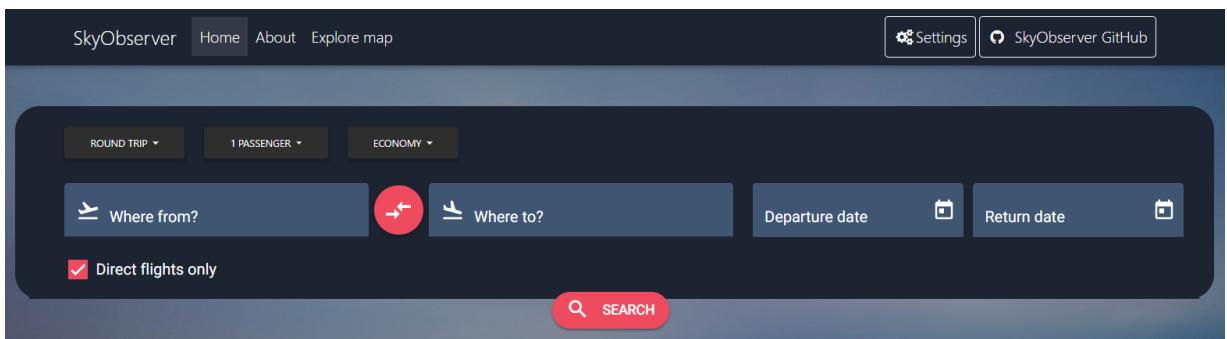
Działanie przedstawionego kodu źródłowego rozpoczyna się od stworzenia adresu URL pod którym część serwerowa udostępnia poszukiwane dane. Do zmiennej *baseUrl* oznaczającej domenę serwera zostaje dodana część sparametryzowana dla danych o lotniskach. Funkcja zwraca obiekty typu *Airport* oraz dodatkowo obsługuje błędy zwrócone z drugiej części aplikacji.

4.1.3. Interfejs graficzny

Walory **szynkości efektywności** oraz użyteczności aplikacji tracą na znaczeniu jeśli budowane oprogramowanie nie jest wyposażone w czytelny interfejs użytkownika. **Jest on formą interakcji Umożliwia on sprawną interakcję** pomiędzy użytkownikiem a aplikacją. **Stworzone Opracowane** środowisko graficzne jest grupą wzajemnie współpracujących przycisków oraz obrazów zapewniających **podstawowe poruszanie się nawigacje** po aplikacji klienckiej w obrębie przeglądarki internetowej. W części klienckiej zaimplementowano responsywny wygląd z takimi komponentami jak panel wyszukiwania lotów **ezy** oraz panel podróży wieloetapowej. Ich **ogólny** kształt i funkcje zostały **w czesiej częściowo** zaczerpnięte z rozwiązań rynkowych aby zwiększyć **konkurencyjność** aplikacji klienckiej. W bieżącym podrozdziale zostanie przedstawiony **stworzony** interfejs aplikacji wraz z krótkim opisem. Rysunek 4.1.2 przedstawia stronę startową części klienckiej. Jego tło jest na całej szerokości oraz długości wypełnione obrazem miasta Hongkong. Góra część strony przedstawia pasek nawigacyjny aplikacji, panel ustawień oraz odwołanie do repozytorium kodu stworzonej pracy dyplomowej. Następnym komponentem aplikacji klienckiej jest panel wyszukiwania lotów. Przedstawiony (**Gdzie?**) komponent obrazuje

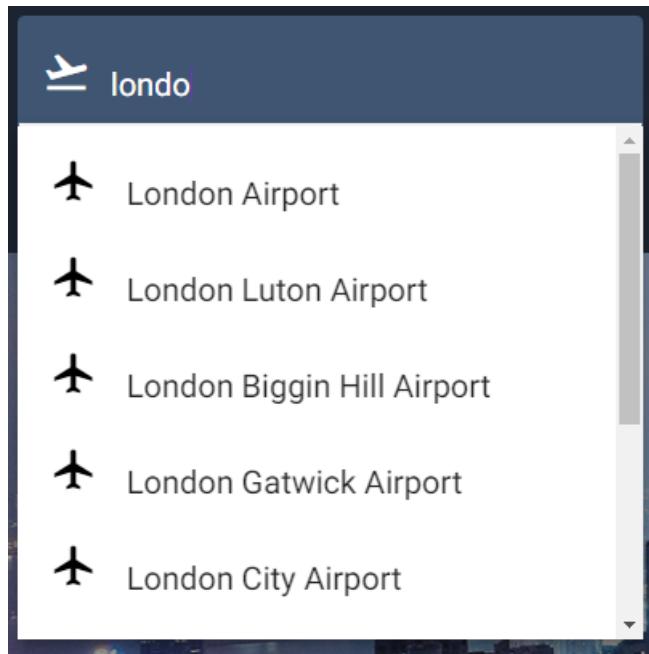


Rysunek 4.1.1: Strona główna aplikacji klienckiej



Rysunek 4.1.2: Panel wyszukiwania połączeń lotniczych

rozmieszczenie przycisków, pól wyboru parametrów wyszukiwania oraz dostępne opcje konfiguracyjne. Górną część panelu zawiera trzy przyciski konfiguracyjne. Odpowiadają one kolejna kolejno za wybór rodzaju lotu, ilość liczby pasażerów oraz klasę klasy poszukiwanego biletu lotniczego. W aplikacji założono trzy typy lotów: lot z możliwością powrotem, lot bezpośredni oraz podróż wieloetapową. Kolejna część, poniżej opisanych przycisków przedstawia pola do wyboru parametrów poszukiwania wyszukiwania. Ich ilość zmienia się w zależności od wybranego typu lotu. W elementach wyboru lotnisk został zaimplementowany mechanizm podpowiedzi wyszukiwania. Użytkownik po wpisaniu kilku pierwszych liter poszukiwanego miasta będzie mógł zobaczyć listę proponowanych lotnisk w obrębie tej miejscowości. Operacja ta jest dokonywana poprzez pobranie wpisanych liter a następnie zgłoszenia żądania do części serwerowej, która zwraca wyniki z bazy danych. Implementacja tego mechanizmu znalazła się jest opisana w poprzednim podrozdziale. Użytkownikowi wyświetlane są pełne nazwy lotnisk. Po wybraniu poszukiwanego miejsca widoczna jest nazwa miasta wraz z unikalnym kodem IATA



Rysunek 4.1.3: Komponent wyboru lotniska

identyfikującym lotnisko. Elementem **umieszczonym** pomiędzy polami dotyczącymi wyboru lotnisk jest przycisk zamiany pól. Jego funkcją jest zamiana informacji zawartych w polach miejsca odlotu oraz przylotu. Ostatnimi elementami są pola wyboru dat lotu docelowego oraz powrotnego. Obecność drugiego jest uzależniona od wybranego typu lotu. W przypadku gdy wyszukiwany jest tylko **lot** w jedną stronę **jest on element ten jest niewidoczny** a szerokość pierwszego jest **rozszerzona większa**. Kliknięcie w ikonę kalendarza wzywa okno modalne z jego powiększonym widokiem. Szarym okręgiem jest na nim zaznaczony dzień **dzisiejszy bieżący**. Po wybraniu, data zapisywana jest w pamięci aplikacji. Ten zabieg służy **do zabezpieczenia sytuacji przy ewentualnym wyborze daty powrotu** przeciwdziałaniu pomyłkom użytkownika przy wyborze daty powrotu. **W takiej sytuacji Jeśli taka się pojawi, to** aplikacja wskaże zapisaną datę jako najwcześniejszą wartość przy wyborze daty lotu powrotnego.

Ważnym komponentem w który włożono **dużo pracy** jest moduł podróży wieloetapowej. Umożliwia on zaplanowanie podróży złożonej z maksymalnie pięciu lotów w jedną stronę. Rysunek 4.15 przedstawia wygląd opisywanego komponentu. Interfejs oferuje przyciski do dodawania kolejnych etapów podróży oraz ich usuwania. (**Skoro ten komponent jest ważny, to dlaczego poświęcił Pan mu tak mało miejsca w pracy?**)

Wspólnym elementem zarówno dla **wszystkich** modułu **podróży wieloetapowej podróży jak i podróży w jedną stronę i powrotnych** jest przycisk wyszukiwania. Po kliknięciu go, aplikacja pobiera wprowadzone przez użytkownika parametry wyszukiwania i przekazuje je dalej do modułu odpytującego serwer. Moduł ten został przedstawiony w poprzednim podroz-

MULTI-CITY ▾ 1 PASSENGER ▾ ECONOMY ▾

Where from? Where to? Departure date X

Where from? Where to? Departure date X

Where from? Where to? Departure date X

Add Flight Reset Direct flights only SEARCH

Rysunek 4.1.4: Komponent podróży wieloetapowej

dziale. Wygląd przykładowych wyników Przykładowe wyniki wyszukiwania zaprezentowano na obrazku rysunku 4.1.6 Wyświetla on listę Jest to lista połączeń lotniczych pomiędzy wybranymi miejscami. Każdy element z tej listy jest rozwijalny, co oznacza że po kliknięciu na niego wysunie się dodatkowy obszar ze szczegółowymi informacjami o locie. W zależności od typu lotu wyświetlana jest zmienna liczba komponentów do wyboru połączenia lotniczego na wybranej trasie.

SkyObserver Home About Explore map Settings SkyObserver GitHub

① Choose trip to Warsaw (WAW) ② Trip summary

| | | | | |
|---------------------|--------------|-------------------|---------|---------------------|
| LOT Polish Airlines | 6:30 - 10:00 | 2h 30m LHR-WAW | Nonstop | No price data found |
| British Airways | 8:25 - 11:50 | 2h 25m LHR-WAW | Nonstop | No price data found |

6:30 AM . London Heathrow Airport (LHR)
Travel time: 2h 30m
10:00 AM . Warsaw Chopin Airport (WAW)

Information about Baggage not found
Airline: LOT Polish Airlines

Select this flight

Rysunek 4.1.5: Wyniki wyszukiwania

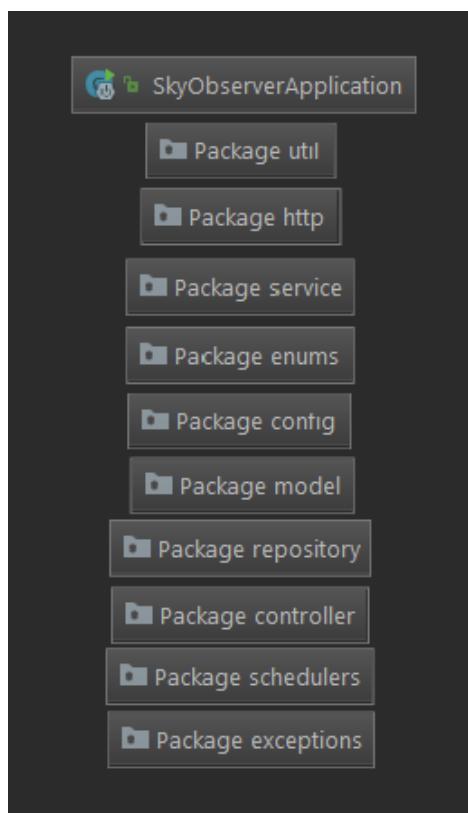
4.2. Oprogramowanie po stronie serwera

Oprogramowanie serwerowe stanowi większą część aplikacji. Zostały w nim zaimplementowane mechanizmy parsowania, wyszukiwania i testowania danych lotniczych. (Mechanizmy testowania powinny być oddzielone od kodu produkcyjnego!) Kwestie związane

z użytymi technologiami przy tworzeniu serwera aplikacji zostały omówione w poprzednim rozdziale. (Po pierwsze, w pracy **pisemnej** można coś opisać, nie zaś omówić. Po drugie, nie opisuje się technologii w rozdziale o projekcie aplikacji.) W bieżącej części pracy zostanie opisana struktura części serwerowej, oprogramowanie odpowiedzialne za parsowanie różnych rodzajów danych, jak i zaimplementowany moduł wyszukiwania.

4.2.1. Podział na pakiety

(Zmieniłem tytuł podrozdziału.) Proces powstawania części serwerowej rozpoczął się od podzielenia tej części aplikacji na logiczne pakiety których nazwa związana byłaby z ich rolą. Poniższy rysunek prezentuje podział aplikacji. Kod części serwerowej został podzielony na pakiety, których nazwa określa ich rolę w aplikacji. Ten podział został przedstawiony na rysunku 4.2.1. Omawianie tych katalogów najlepiej zacząć od pakietu *config*. Jego zadaniem Zadaniem



Rysunek 4.2.1: Podział strukturalny części serwerowej

pakietu *config* jest konfiguracja różnych parametrów serwera podczas jego startu. Ustawa on takie **rzeczy dane** jak klucze dostępowe do zasobów lotniczych, parametry **lokalnego cache'a** czy pamięci podręcznej oraz **też** zabezpieczenia aplikacji. Warstwa modelowa aplikacji znajduje się w **pakiecie model**. Zaimplementowane zostały w nim klasy modelowe które opisują obiekty krążące (Jakie pakiety??? Proszę to poprawić.) po całej aplikacji. Związanym z tym pakietem jest katalog *repository*. Jest on miejscem na klasy implementujące pozyskiwanie obiektów

odpowiadających klas modelowych. (To zdanie jest niezrozumiałe. O jakie „odpowiadające klasy modelowe” chodzi? Proszę to poprawić.) Niektóre z nich używają swoich analogicznych odpowiedników z pakietu *service*. Potrzeba ta wynika z konieczności parsowania niektórych formatów danych do obiektów na obiekty języka Java. Pakietem który udostępnia obiekty modelowe dla części klienckiej jest katalog *controller*. Kod odpowiedzialny za wysyłanie żądań HTTP zaimplementowany jest w części *http*. W części serwerowej zaimplementowano też moduły których wywołanie jest zaplanowane i preezyjnie określone, które są wywoływane cyklicznie. Dotyczy to lokalnych plików Aktualizują one lokalne pliki przechowywujących przechowujące dane o lotniskach oraz bagażach. Kod odpowiedzialny za ich aktualizowanie jest opracowany Ich kod jest umieszczony w katalogu *schedulers*. Pozostałe pakiety, czyli *util, enums* oraz *exceptions* wprowadzają zawierającą mnóstwo znaczące dla aplikacji funkcje.

4.2.2. Oprogramowanie wyszukujące połączenia lotnicze

Módul Projekt modułu wyszukiwania połączeń lotniczych został już opisany w poprzednim rozdziale, bieżący zostanie jest poświęcony kwestiom jego implementacji. Na potrzeby jego stworzenia opracowano tego modułu opracowano specjalny sposób tworzenia obiektu lotu. Standardowy proces powstawania obiektu przez konstruktora języka Java związany z wywołaniem konstruktora okazał się mało czytelny (Co to znaczy mało czytelny? Czy miał Pan na myśli „nieprzydatny”, czy jeszcze coś innego??) dla wielu obiektów wchodzących w skład jednostki połączenia lotniczego. Dla tego obiektu opracowano implementację wzorującą się na wzorze projektowym o nazwie Do tworzenia tego obiektu zastosowano wzorzec projektowy Budowniczy. Budowniczy to programistyczny wzorzec projektowy, który opisuje algorytm tworzenia uzyskiwania złożonego obiektu z innych. Algorytm ten powinien być niezależny od składowych obiektu. Ponadto proces budowania powinien zezwalać na budowę różnych form tworzonego obiektu.[12] Klasa modelowa *Flight* została wyposażona w klasę wewnętrzną ja budującą jej obiekty. Jej i Implementacja tej klasy została jest przedstawiona na listingu 4.2.1.

```
1 @JsonSerialize  
2 public class Flight {  
3     private String departureTime;  
4     private String arrivalTime;  
5     private Airport originAirport;  
6     private Airport destinationAirport;  
7     private String duration;  
8     private CachedPrice price;  
9     private Airline airline;  
10    private Baggage baggage;
```

```
11  
12     ...  
13 }
```

Listing 4.2.1: Fragment klasy Flight

Klasą a odpowiedzialną a za zbieranie danych o połączeniach lotniczych jest umieszczona w pliku *FlightsRepository*. W jej kodzie źródłowym zaimportowano wszystkie pozostałe repozytoria danych. Jego działanie polega na ustaleniu każdej jego składowej widocznej na [listingu 4.2.2](#) pobierając odpowiednie dane z analogicznych repozytoriów tych składowych. Działanie tej klasy polega na zapisaniu w jej atrybutach, widocznych na [listingu 4.2.1](#), wartości pobranych z odpowiednich repozytoriów. Ze względu na złożoność oraz wielkość jego implementacji jego kod nie zostanie pokazany jej kod nie jest umieszczony w całości w tym rozdziale.

4.2.3. Parsowanie danych

~~Podrozdział ten jest odwołaniem do swojego odpowiednika w rozdziale trzecim. Zostanie w nim zwrócona szczególna uwaga na implementacje komponentów odpowiedzialnych za parsowanie różnych rodzajów danych. W tym podrozdziale opisane są implementacje mechanizmów parsowania formatów danych przedstawionych w rozdziale trzecim. Opracowane. Opracowano następujące klasy parsujące:~~

- AirlineParser - parser danych o liniach lotniczych konwertujący obiekty JSON na obiekty Java,
- AirportsParser - komponent pozyskujące obiekty z plików CSV,
- BaggagesParser - moduł parsujący dokumenty HTML,
- PricesParser - klasa konwertująca obiekty JSON dotyczące zawierające cen lotów,
- FlightsParser - moduł parsujący złożone treści dane otrzymane w języku znaczników XML na analogiczne odpowiadające im obiekty języka Java.

~~Wymienione komponenty zostaną krótko opisane wraz z przedstawieniem ich implementacji. Ze względu na prostą złożoność jako pierwszy zostanie opisany parser danych linii lotniczych. Parser linii lotniczych ma najprostszą budowę z nich wszystkich i dlatego zostanie przedstawiony jako pierwszy.~~

```
1 public class AirlineParser {  
2     private static final int FIRST_ELEMENT = 0;  
3     private Gson gson = new Gson();  
4  
5     public Airline getAirlineObjectFromJSONResponse(String jsonObject) {  
6         Airline[] airlines = gson.fromJson(jsonObject, Airline[].class);
```

```

7     return airlines[FIRST_ELEMENT];
8 }
9 }
```

Listing 4.2.2: Implementacja parsowania treści JSON

Powyższy listing Listing ... przedstawia sposób kod dokonujący konwersji treści w formacie JSON. Do tego celu posłużono się biblioteką `gson` firmy Google. Wywołując metodę `fromJson` na obiekcie typu `Gson` w prosty sposób odbywa się konwersja obiektów `Json` na obiekty `Airline`. Konwersja obiektów JSON na obiekty `Airline` odbywa się poprzez wywołanie metody `fromJson` na obiekcie klasy `Gson`. Funkcja jako wynik zwraca pierwszy element zparsowanej przekonwertowanej tablicy danych.

Translacja treści zwrotnej z serwisu *Flight Lookup* otrzymanej w postaci XML odbywała odbywa się przy pomocy biblioteki Jackson. Pierwszym jej krokiem było jest stworzenie odpowiedników elementów odpowiedzi na klasy modelowe w części serwerowej. (To zdanie jest niezrozumiałe, proszę poprawić.) Po tej czynności można było przystąpić do implementacji metody konwertującej wywoływaną jest metoda konwertująca, której kodu zawiera listing ...

```

1 public class FlightsParser {
2     private JacksonXmlModule xmlModule = new JacksonXmlModule();
3
4     public OTA_AirDetailsRS getDeserializedXML(String XML) throws IOException {
5         xmlModule.setDefaultUseWrapper(false);
6         ObjectMapper objectMapper = new XmlMapper(xmlModule);
7         return objectMapper.readValue(XML, OTA_AirDetailsRS.class);
8     }
9 }
```

Listing 4.2.3: Implementacja parsowania treści XML

Powyższa implementacja Kod klasy `FlightsParser` jest podobna y do poprzedniej przedstawionej na listingu 4.2.3. Metoda (Która?) jako argument przyjmuje ciąg znaków XML. Do konwersji używana jest klasa `ObjectMapper` pochodząca ze wcześniej wspomnianej biblioteki Jackson. Wynikiem zwróconym przez funkcję jest pojedynczy obiekt `OTAAirDetailsRS` zawierający w sobie listę wszystkich wyszukanych lotów. Ze względu na złożoność kodu źródłowego pozostałych parserów ich implementacja nie zostanie przedstawiona. Kod ten można znaleźć w dołączonym repozytorium, w pakiecie *service*.

4.2.4. Zabezpieczenia

Sześćdziesiąt miliardów to cena, jaką branża informatyczna corocznie wydaje na bezpieczeństwo swoich systemów. Tak wysoka kwota jest jednak wydatkiem koniecznym do skutecznej ochrony powstałego oprogramowania oraz danych jego użytkowników[13]. W części serwerowej opracowano zabezpieczenie ograniczające dostęp do udostępnianych danych lotniczych.

```
1 @Configuration
2 @EnableWebMvc
3 public class CorsConfiguration implements WebMvcConfigurer {
4
5     @Override
6     public void addCorsMappings(CorsRegistry registry) {
7         registry.addMapping("localhost:4200/**");
8     }
9 }
```

Listing 4.2.4: Implementacja zabezpieczanie dostępu do serwera

Listing 4.2.4 przedstawia implementację tego zabezpieczenia. Ustawia ono dostęp do serwera tylko z adresu części serwerowej podanego jako argument metody *addMapping*. Zabieg ten **zabezpiecza dostęp chroni przed dostępem** do danych dla z niepożądanych adresów sieciowych.

Machanizm **ezechowania danych pamięci podręcznej**, usprawniający wydajność aplikacji zminimalizował również szanse na skuteczny atak DDOS. Został on krótko opisany w rozdziale pierwszym w którym **omówiono 3 przedstawiono trzy kontenery eache'u pamięci podręcznej** zaimplementowane w aplikacji. **Szybkość Wydajność** ich odpowiedzi uodparnia część serwerową na **nadmiarowy nadmierny ruch sieciowy ze stony strony** użytkowników aplikacji. Zdefiniowano dla nich dla każdego z kontenerów maksymalny rozmiar przechowywanych danych wynoszący 1GB dla każdego z kontenerów. Po jego przekroczeniu go, eache zostanie **wyczyszczony** pamięć podręczna zostaje wyczyszczona.

4.2.5. Kontrolery serwera

W części serwerowej stworzono specjalny komponent odpowiedzialny za obsługę żądań wyszukiwania lotów oraz zwrócenie im poszukiwanych wyników.

```
1 @RestController
2 @RequestMapping("/flights")
3 public class FlightsController {
4
5     @Autowired
6     private MultiFlightsRepository multiFlightsRepository;
```

```

7
8 @GetMapping(value = "/{from}/{to}/{date}/{connection}/{currency}", produces =
9   MediaType.APPLICATION_JSON_VALUE)
10 public Collection<MultiFlight> searchForFlights(@PathVariable String from,
11   @PathVariable String to, @PathVariable String date, @PathVariable String
12   connection, @PathVariable String currency) throws AirportsNotFoundException,
13   IOException {
14   Collection<MultiFlight> flights = multiFlightsRepository.searchForMultiFlights(
15     from, to, date, connection, currency);
16   if (flights.isEmpty()){
17     throw new FlightsNotFoundException();
18   }
19   return flights;
20 }
21 }
```

Listing 4.2.5: Kontroler zwracający wyniki wyszukiwania lotów

Powyższy Ten kontroler został **zaimplementowany zaimplementowany** w pakiecie *controller*. Dwie adnotacje nad deklaracją klasy (**listing ...**) tworzą w serwerze specjalne miejsce, w którym po podaniu odpowiednich parametrów kontroler zostanie wywołany. Parametry te opisane są nad deklaracją metody *searchForFlights*. Funkcja pobiera z adresu URL kody IATA lotniska wylotowego oraz docelowego, datę wylotu, typ połączenia i kod walutowy. Przedstawione parametry są przekazywane **w-wywołaniu klasy klasie** (Parametry przekazuje się **metodzie, nie klasie. Klasie nie są wywoływane. Proszę poprawić dopisując, która metoda tej klasy otrzymuje wspomiane parametry.**) repozytorium lotów, które a zwraca wyniki wyszukiwania w postaci kolekcji obiektów. W przypadku nie znalezienia żadnych połączeń lotniczych, metoda **zwraea** wyrzuca wyjątek, który **implikuje skutkuje** zwróceniem odpowiedzi o kodzie 404. (**To nie jest dobry sposób sygnalizowania braku wyników. Taka sytuacja nie jest wyjątkowa. Wyjątkiem jest awaria serwera, nie zaś zwrócenie przez niego pustej listy wyników.**) (Brak podsumowania rozdziału.)

5. Testy

Proces powstawania oprogramowania jest zbiorem czynności opartych na wielu czynnikach. Patrząc wstecz, programowanie jako fach ma jedynie kilka dekad. Tworzenie oprogramowania można potraktować jako proces ewolucyjny gdzie ewoluje, w którym zmienia się ono z upływem czasu. Programiści szybko dochodzą do wniosków, że pisanie kodu jest trudne i podatne na liczne błędy. Wiele projektów zawodzi, ponieważ zespoły napotykają trudności ze złożonością tworzonych systemów. W wyniku tych niedociągnięć projekt nie zostaje skończony w terminie a koszt jego finalizacji przekracza początkowe oczekiwania[14]. Rozwój oparty na sprężonym testowaniu tworzonego kodu jest jedną z praktyk efektywnego rozwoju oprogramowania. Testy potwierdzają funkcje aplikacji oraz jej użyteczność dla użytkownika. Standardowy podział ze względu na przeznaczenie można przedstawić następująco: Opracowaną aplikację poddano testom. W inżynierii oprogramowania proces testowania podzielony jest na kilka etapów:

- testy jednostkowe - potwierdzają poprawne sprawdzają działanie najmniejszych elementów systemu takich jak metody czy lub funkcje,
- testy integracyjne - sprawdzają poprawność integracji tworzonego oprogramowania z zewnętrznymi systemami współpracy komponentów oprogramowania,
- testy systemowe - badają takie cechy oprogramowania jak efektywność lub niezawodność,
- testy akceptacyjne - zapewniają o spełnieniu oczekiwania klienta, który zamówił oprogramowanie.
- testy regresywne - potwierdzają brak wpływu napisanego oprogramowania na pozostałe elementy systemu

(Testy regresywne, to nie etap testowania oprogramowania, a jeden z rodzajów testów. Nie należy do tej klasyfikacji.) W stworzonej przypadku opracowanej aplikacji zdecydowano się na przetestowanie kluczowych fragmentów oprogramowania odpowiedzialnych za podstawowe funkcje systemu. Testy zaimplementowane zostały przy użyciu frameworków JUnit oraz Spring. Biblioteka JUnit to zewnętrzny pakiet oprogramowania dostarczający zestaw narzędzi do testowania oprogramowania napisanego programów napisanych w języku Java. Znany i sprawdzony przez wielu programistów jest częścią wielu komercyjnych systemów. Drugi z nich, przedstawiony wcześniej w rozdziale trzecim Platforma Spring dostarcza instrumenty mechanizmów do testowania jego oprogramowania serwerowego.

Podstawową metodą testującą oprogramowanie była asercja. Metody testujące oprogramowanie zawierają asercje. Jest to predykat, który zwraca. Są to predykaty, które zwracają wartość logiczną. Najczęściej przyjmuje przyjmują one dwa argumenty a następnie porównuje

porównują je **zwracając i zwracają wynik**[15]. Napisane testy znajdują się w części serwerowej aplikacji, t. Testują one poprawność odpowiedzi serwera, **informacje otrzymane informacji otrzymanych** z zewnętrznych serwisów, **parsowanie prasowania** różnych formatów danych jak i najbardziej **podstawowe elementy podstawowych elementów aplikacji**. (Już o tym wspomniałem: kod testujący powinien być oddzielony od produkcyjnego, więc jeśli jest on częścią serwera, to jest to błąd.) Nazwy testów **przemawiają za powiązane są z kodem źródłowym** który testują. Dla przykładu Przykładowo, test dla klasy *FlightRepository* będzie nazywał się *FlightsRepositoryTest*.

5.1. Testy jednostkowe

Podstawowymi testami były testy jednostkowe. Na poniższym listingu ... przedstawiono **zaimplementowany** kod testujący poprawność działania funkcji zwracającej adres żądania HTTP.

```
1 @Test
2 public void shouldReturnValidCacheKey() {
3     assertEquals("WAW/LHR/20190110/DIRECT/PLN",
4         multiFlightsRepository.buildCacheKey("WAW", "LHR", "20190110", "DIRECT", "PLN"));
5 }
```

Listing 5.1.1: Przykładowy test jednostkowy

Bardziej złożonym przypadkiem jest test odpowiedzialny za sprawdzenie poprawnej ceny całej podróży.

```
1 @Test
2 public void shouldReturnProperSumOfTotalFlightPrice() {
3     Flight firstFlight = new Flight.Builder()
4         .setPrice(new CachedPrice(200.43))
5         .build();
6
7     Flight secondFlight = new Flight.Builder()
8         .setPrice(new CachedPrice(800.99))
9         .build();
10
11    Flight thirdFlight = new Flight.Builder()
12        .setPrice(new CachedPrice(436.67))
13        .build();
14
15    List<Flight> flightList = List.of(firstFlight, secondFlight, thirdFlight);
16    assertEquals(1438.09, multiFlightsRepository.calculateTotalFlightPrice(flightList),
17        0.001);
```

Listing 5.1.2: Przykładowy test jednostkowy

Test zaprezentowany na listingu 5.1.2 inicjalizuje 3 inicjuje trzy obiekty typu *Flight* z przykładowymi wartościami cen przelotów. Obiekty te łączone są w kolekcję. Koncepcja sprawdza poprawne działanie metody *calculateTotalFlightPrice* porównując jej wynik z oczekiwana wartością. Warto zwrócić uwagę na trzeci argument jaki przyjmuje metoda *assertEquals*. Oznacza on dopuszczalną różnicę między oczekiwany a otrzymanym wynikiem. (Przechowywanie kwot w zmiennych typu *double* lub *float* to proszenie się o kłopoty. Od tego jest klasa *BigDecimal*.)

5.2. Testy integracji

(Zmieniłem tytuł podrozdziału, bo to nie są testy integracyjne, tylko testy integracji z systemami zewnętrznymi, które są częścią testów systemowych.) Specyfikacja projektu narzucała konieczność przetestowania integracji napisanego oprogramowania z zewnętrznymi systemami. Testowano poprawność pobierania danych oraz ich oczekiwane parametry. Wszystkie klasy z pakietu *repository* zostały przetestowane w ten sam sposób analogiczny do siebie.

```
1 @Test
2 public void shouldReturnProprtAirlineObjectFromAPI() throws IOException {
3     Airline airline = airlineRepository.getAirlineFromApi("AA");
4     assertEquals(airline.getNameAirline(), "American Airlines");
5     assertEquals(airline.getStatusAirline(), "active");
6     assertEquals(airline.getCallsign(), "AMERICAN");
7     assertEquals(airline.getNameCountry(), "United States");
8 }
```

Listing 5.2.1: Przykładowy test integracyjny

Oprogramowanie przedstawione Kod przedstawiony na listingu powyżej testuje funkcjonowanie repozytorium zwracającego informacje o liniach lotniczych. W pierwszym kroku pobiera ono obiekt linii lotniczej. Po przypisaniu tych danych do obiektu, sprawdzane są poszczególne parametry zwróconych informacji. (Pobiera obiekt i przypisuje dane do obiektu? Proszę poprawić.)

Test o analogicznej funkcji sprawdzał też Analogiczny test sprawdza repozytorium zwracające ceny lotów.

```
1 @Test
```

```
2 public void shouldReturnValidPriceObject() throws IOException {
3     CachedPrice price = pricesRepository.getFlightPrice("PLN", "WAW", "LHR", "20190128",
4         "20190130");
5     assertNotNull(price);
6     assertEquals(price.getCurrency(), "PLN");
7     assertEquals(price.getOriginPointOfRoute(), "WAW");
8     assertEquals(price.getDestinationPointOfRoute(), "LHR");
9 }
```

Listing 5.2.2: Przykładowy test integracyjny

6. Uwagi i wnioski

Opracowanie aplikacji wyszukującej połączenia lotnicze ~~było zadaniem trudnym i wymagającym bardzo dużej ilości pracy. Droga do stworzenia oprogramowania spełniającego wszystkie wymagania przedstawione w celach pracy wymagała głębokiego wymaga~~ zrozumienia logiki biznesowej związanej z pozyskiwaniem danych lotniczych. Do największych wyzwań podczas tworzenia pracy z pewnością należało znalezienie odpowiednich źródeł danych udostępniających dane informacje o lotach. Wysłano szereg zgłoszeń mailowych z prośbą o udostępnienie takich danych. W większości przypadków odpowiedzi były negatywne lub niesatysfakcjonujące z powodu żądania od dostawcy przez dostawcę pokaźnej zapłaty za korzystanie z usługi. (Jeśli faktycznie byłoby tak jak Pan napisał, to dostawca by Panu płacił, a nie odwrotnie. Trzeba było się zgodzić.) W warunkach akademickich skorzystanie z takich rozwiązań było niemożliwe. Po wnikliwej analizie rynku danych lotniczych udało się znaleźć odpowiednie źródło informacji dla potrzeb stworzonej tworzonej aplikacji.

Z dużą dozą pewności można stwierdzić iż opracowana aplikacja w pełni spełnia Opracowana aplikacja spełnia założone wymagania. Pozwala z całkiem dobrą szybkością relatywnie szybko wyszukiwać połączenia lotnicze oraz dostarcza wielu dodatkowych informacji z nimi powiązanych. W sposób logiczny i przemyślany zaprojektowano moduł wyszukiwania lotów Moduł wyszukiwania lotów został zaprojektowany tak, aby działał efektywnie. Opracowano dla niego specjalną architekturę, która została odzwierciedlona w kodzie źródłowym pod postacią wzorca projektowego Jakiego?. Dzięki temu zabiegowi aplikacja zachowuje potencjał łatwego i mało problemowego rozwoju. Ułatwiony jest dalszy rozwój aplikacji.

Zaimplementowany system został wyposażony w bogaty Aplikacja została wyposażona w interfejs użytkownika bazujący na technologii Angular. Opracowany w nowoczesnej technologii Angular, poziomem dobrego stylu i wyglądu dorównuje komercyjnym aplikacjom internetowym. Część pomysłów na jego kreatywną realizację zaczerpnięto z obecnych wyszukiwarek dostępnych na rynku.

Z uwagi na warunki akademickie a Aplikację można usprawnić na wielu płaszczyznach wiele sposobów. W stworzonej pracy zabrakło takich dodatkowych funkcji jak możliwość wyszukiwania lotów również z pobliskich lotnisk od aktualnie wybranego z lotnisk położonych w pobliżu aktualnie wybranego miejsca, czy też wprowadzenia wersji w języku polskim. Część serwerowa mogłaby zostać poprawiona w zakresie "czystości kodu" oraz poprawnego nazewnictwa jej elementów. Kod części serwerowej mógłby zostać poddany refaktoryzacji. Istotne znaczenie dla podniesienia atrakcyjności aplikacji miałyby też zwiększenie poziomu jej

responsywności. Część kliencka wymaga w tym zakresie **lekkich** poprawek.

~~Stworzonej aplikacji internetowej nadano nazwę *SkyObserver*¹. Jest ona zwieńczeniem zebranej wiedzy i doświadczenia podeczas studiów inżynierskich. Praca nad nią zaowocowała pogłębieniem technicznego sposobu myślenia oraz zrozumieniem wyzwań XXI wieku stawianym inżynierom informatykom.~~

~~(Bibliografia powinna zawierać także miejsce i rok wydania książki. Proszę sprawdzić odwołania do literatury, szczególnie tam gdzie wykreślałem fragmenty pracy. Wszystkie pozycje muszą być cytowane. Trochę jest ich mało.)~~

¹SkyObserver - obserwator nieba

Bibliografia

[1] Elliotte Rusty Harold, W.Scott Means „*XML. Almanach*” . Wydawnictwo O'REILLY

[2] JSON

<https://www.json.org/>

Stan na dzień 03.01.2019

[3] HTML, Wikipedia

<https://en.wikipedia.org/wiki/HTML>

Stan na dzień 04.01.2019

[4] Daniel Wind „*Instant Effective Caching with Ehcache*” . Wydawnictwo Packt

[5] Christine Hofmeister, Robert Nord „*Tworzenie architektury oprogramowania*” . Wydawnictwo Naukowo-Techniczne Warszawa

[6] Diagramy UML | Michał Wolski

<https://www.michalwolski.pl/diagramy-uml>

Stan na dzień 16.01.2019

[7] Marcin Lis „*MySQL Darmowa baza danych - Ćwiczenia praktyczne*” . Wydawnictwo Helion

[8] Simon Roberts „*Java Programming Basics*” . Wydawnictwo Addison-Wesley Professional

[9] Ilya Grigorik „*HTTP Protocols*” . Wydawnictwo O'REILLY

[10] Kishori Sharan „*Beginning Java 8 APIs, Extensions and Libraries Swing, JavaFX, JavaScript, JDBC and Network Programming APIs*” . Wydawnictwo Apress

[11] Jeff Friesen „*Java I/O, NIO, NIO.2*” . Wydawnictwo Apress

[12] John Vlissides, Ralph Johnson, Richard Helm, Erich Gamma „*Design Patterns: Elements of Reusable Object-Oriented Software*” . Wydawnictwo Addison-Wesley Professional

[13] Vincent Liu, Bryan Sullivan „*Web Application Security*” . Wydawnictwo McGraw-Hill

[14] Shekhar Gulati, Rahul Sharma „*Java Unit Testing with JUnit 5*” . Wydawnictwo Apress

[15] Asercja, Wikipedia

[https://pl.wikipedia.org/wiki/Asercja_\(informatyka\)](https://pl.wikipedia.org/wiki/Asercja_(informatyka))

Stan na dzień 14.01.2019