

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	11
1. Opis zagadnienia	12
1.1. Źródła danych o połączeniach lotniczych	12
1.2. Parsowanie danych	13
1.2.1. Język znaczników XML	13
1.2.2. Notacja JSON	15
1.2.3. Język HTML	16
1.3. Wydajność wyszukiwania	16
2. Przegląd istniejących rozwiązań	18
2.1. Wyszukiwarka lotów Skyscanner	18
2.2. Wyszukiwarka lotów Google Flights	19
2.3. Porównanie aplikacji	22
3. Projekt aplikacji	24
3.1. Ogólna architektura aplikacji	24
3.2. Diagram UML	25
3.3. Baza danych	27
3.4. Część kliencka	27
3.5. Część serwerowa	28
3.6. Moduł wyszukiwania połączeń lotniczych	30
3.7. Komunikacja między komponentami aplikacji	31
3.7.1. Protokół HTTP	31
3.7.2. JDBC	32
3.7.3. I/O	32
4. Implementacja	33
4.1. Oprogramowanie po stronie klienta	33
4.1.1. Podział strukturalny	33
4.1.2. Moduły pobierania danych	34
4.1.3. Interfejs graficzny	35
4.2. Oprogramowanie po stronie serwera	39

4.2.1. Struktura pakietowa	39
4.2.2. Oprogramowanie wyszukujące połączenia lotnicze	40
4.2.3. Parsowanie danych	41
4.2.4. Zabezpieczenia	42
4.2.5. Kontrolery serwera	43
5. Testy	45
5.1. Testy jednostkowe	46
5.2. Testy integracyjne	47
6. Uwagi i wnioski	48

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 kolejnym 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 wcześniej wymienione zasoby oceniono z największym prawdopodobieństwem na ich użycie przez użytkowników aplikacji. 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

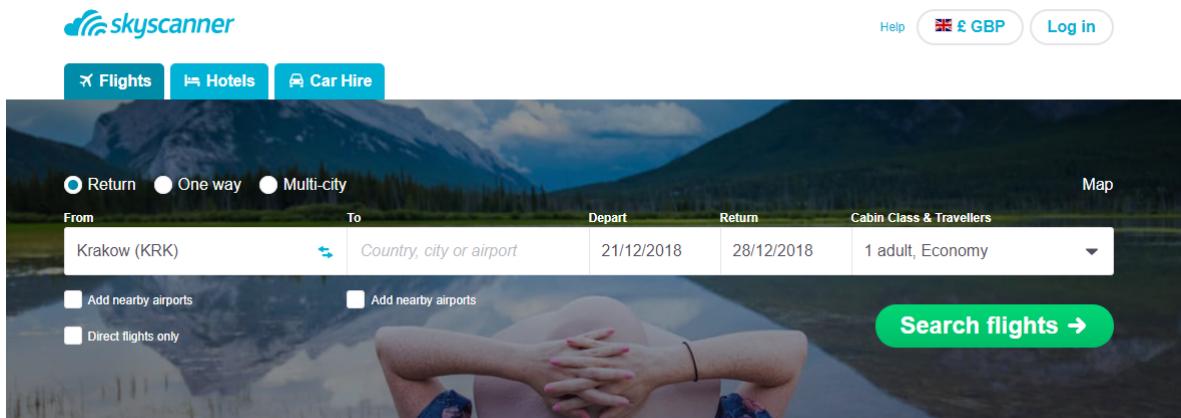
Analiza istniejących rozwiązań, **czyli** aplikacji wyszukujących połączenia lotnicze pozwoliła ~~nadać pracy bardziej preezyjne wymagania sprecyzować wymagania względem budowanego oprogramowania oraz zaprojektować jej ogólny przebieg~~. W i Internetie można znaleźć wiele aplikacji o ~~podobnych lub takich samych funkcjonalnościach~~ podobnej lub takiej samej dziedzinie zastosowania, jak opracowana aplikacja. W tym rozdziale **zostały** opisane są najbardziej znane wyszukiwarki lotów dostępnych na rynku.

2.1. Wyszukiwarka lotów Skyscanner

Pierwszym przykładem **została jest** aplikacja internetowa Skyscanner. **Jest to** wyszukiwarka lotów, która umożliwia użytkownikom szukanie **lotów** według ceny i lokalizacji. Oprócz **funkcji** wyszukiwania lotów, Skyscanner oferuje opcje **wyszukiwania hotelów** szukania hoteli blisko lotnisk oraz **wypożyczenia wypożyczania** auta w pobliżu lotniska docelowego. **Aplikacja ta została stworzona oraz wdrożona w 2002 roku. Od tego czasu firma Ctrip, która jest właścicielem tego produktu zatrudnia ponad 200 pracowników.** (Ta informacja nic nie wnosi do zagadnień związanych z tematem pracy.) Warto wspomnieć o **innej jej innej usłudze** **która udostępnia dane**, czyli udostępnianiu danych o połączeniach **lota** **z lotniczymi lotniczymi** zewnętrznym firmom i deweloperom programistom. (W języku polskim „deweloper” to człowiek, który prowadzi firmę budującą domy, więc nie ten wydział.) **Jej dane byłybrane pod uwagę w czasie** szukania źródeł danych lecz Skyseanner. 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 **z nich skorzystanie z nich**. Aplikacja ta jest dostępna w ponad 30 językach oraz używana przez 60 milionów użytkowników miesięcznie. **Aplikacja wiele razy nagradzana była za swoją funkcjonalność i użyteczność użytkownikom.** Skyscanner znajduje się pod adresem sieciowym: <https://www.skyscanner.net/>.

Opisywana aplikacja **ma dla swoich użytkowników szereg przydatnych funkcjonalności**, oferuje swoim użytkownikom szereg przydatnych funkcji. Korzystające z jej osoby mogą **korzystać z kompleksowego modułu** posługiwać się kompleksowym modułem wyszukiwania połączeń lotniczych. W panelu wyboru parametrów lotów można **wybierać określić** lotnisko wyłotu, lotnisko docelowe jak i daty **wylotu lotu**. Skyscanner oferuje **3 trzy sposoby** wyszukiwania lotów: lot powrotny, lot w jedną stronę oraz podróż wieloetapowa. Dodatkowo użytkownik jest w stanie sprecyzować liczbę osób podczas podróży jak i klasę biletu lotniczego. Ciekawą funkcjonalnością 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 o zadanych promieniu odległości. Oprócz bogatego 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 kilka sekund. Interfejs aplikacji Skyscanner zostanie zaprezentowany [na następnej stronie](#). Po wybraniu odpowiednich



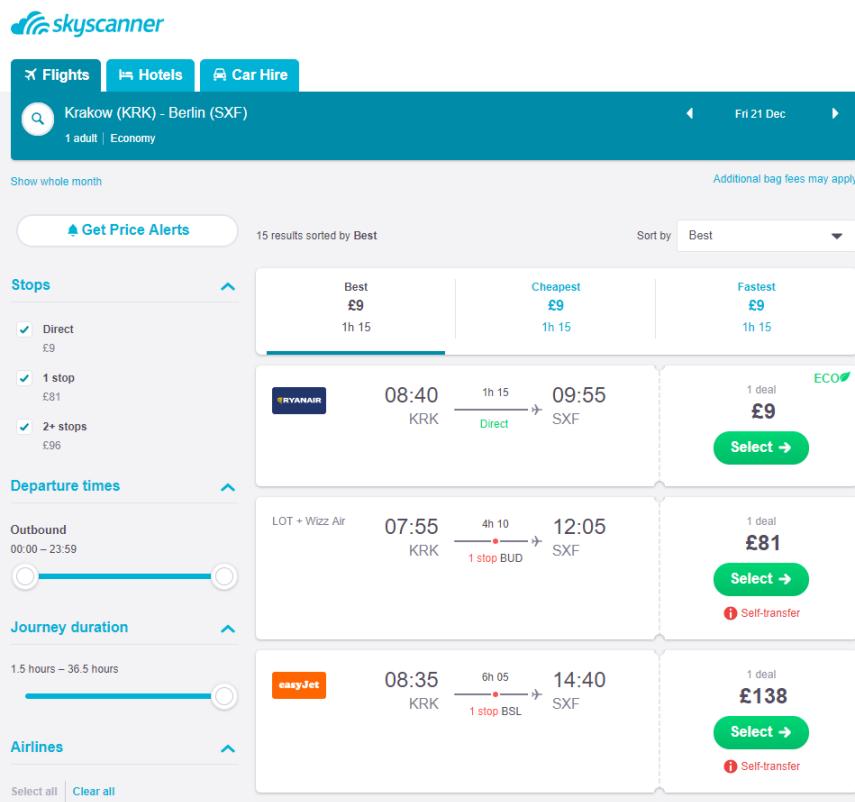
Rysunek 2.1.1: Panel wyszukiwania lotów aplikacji Skyscanner

parametrów należy nacisnąć zielony przycisk po prawej aby rozpocząć proces wyszukiwania lotów. Jego przykładowe 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 różnego rodzaju filtracji na otrzymanych wynikach.

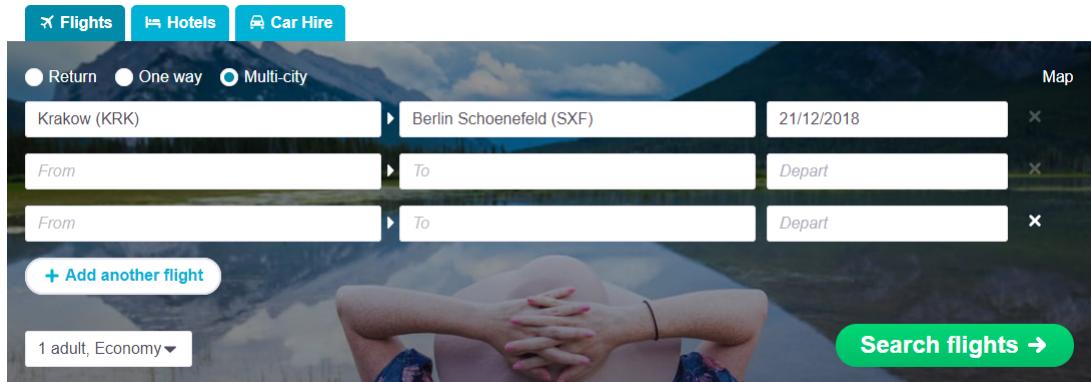
Warto zwrócić uwagę na panel podróży wieloetapowej, który jest dostępny po kliknięciu odpowiedniego przycisku powyżej pola do wyboru lotniska wylotowego. Można w nim wybrać do siedmiu osobnych połączeń lotniczych i na ich podstawie zaplanować swoją podróż. Każde z tych połączeń jest domyślnie połączeniem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

2.2. Wyszukiwarka lotów Google Flights

Kolejnym przedstawianym rozwiążaniem jest aplikacja internetowa Google Flights firmy Google. Jest to jeden z głównych internetowych produktów giganta z Kalifornii. W pełni integruje się z innymi usługami tej firmy co czyni go bardzo praktycznym systemem. Google Flights zostało wdrożone 13 października 2011 roku. Ma zbliżone funkcjonalności do wcześniej opisywanego konkurenta Skyscanner. Oferuje jednak pewne innowacje. Pozwala użytkownikowi na wyszukiwanie lotów określając okres czasowy go interesujący oraz budżet na jaki pozwala jego portfel. Największą zaletą tej aplikacji jest jej szybkość. Dzięki spokrewnieniu z silni-



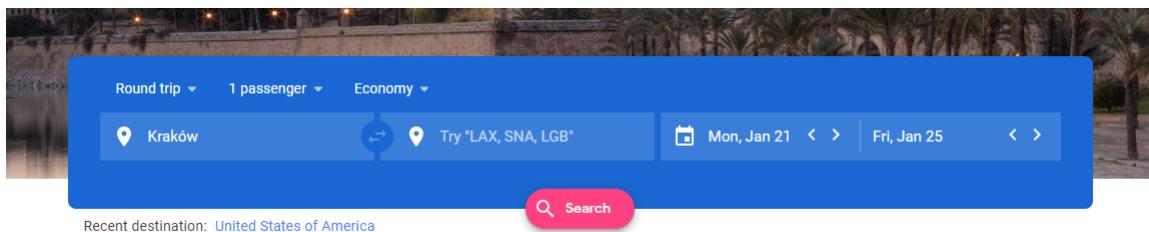
Rysunek 2.1.2: Wyniki wyszukiwania lotów aplikacji Skyscanner



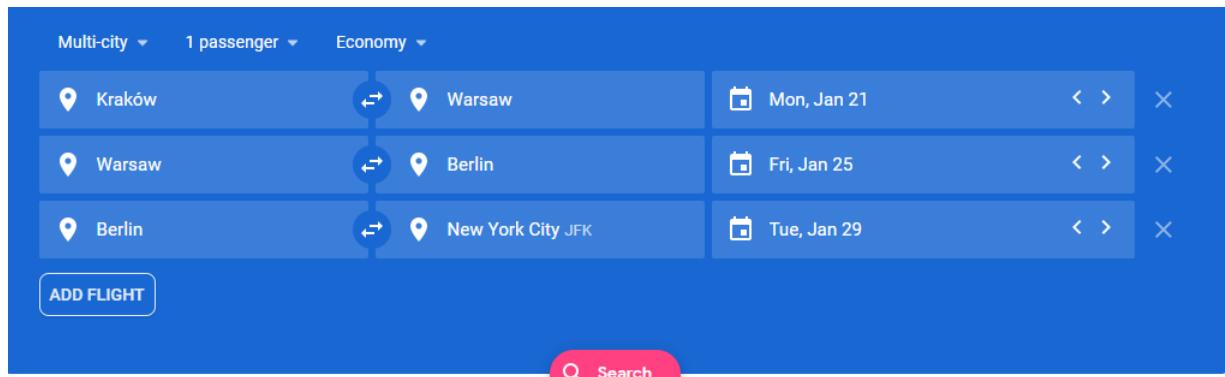
Rysunek 2.1.3: Komponent podróży wieloetapowej

kiem Google Search szybkość wyszukiwania połączeń lotniczych jest bezkonkurencyjna na rynku wyszukiwarek lotów. Szybkość ta została uzyskana przez użycie wyspecjalizowanych algorytmów szukających firmy Google oraz specjalnemu indeksowaniu danych. Oprócz samej funkcjonalności aplikacja prezentuje czytelny i responsywny interfejs, który zyskał aprobatę ze strony użytkowników.

Na rysunku 2.2.1 przedstawiono panel wyszukiwania lotów oferowany przez opisywaną aplikację. Dostępny jest też komponent podróży wieloetapowej który znalazł się też w poprzednio opisywanej aplikacji oraz w części praktycznej tej pracy.

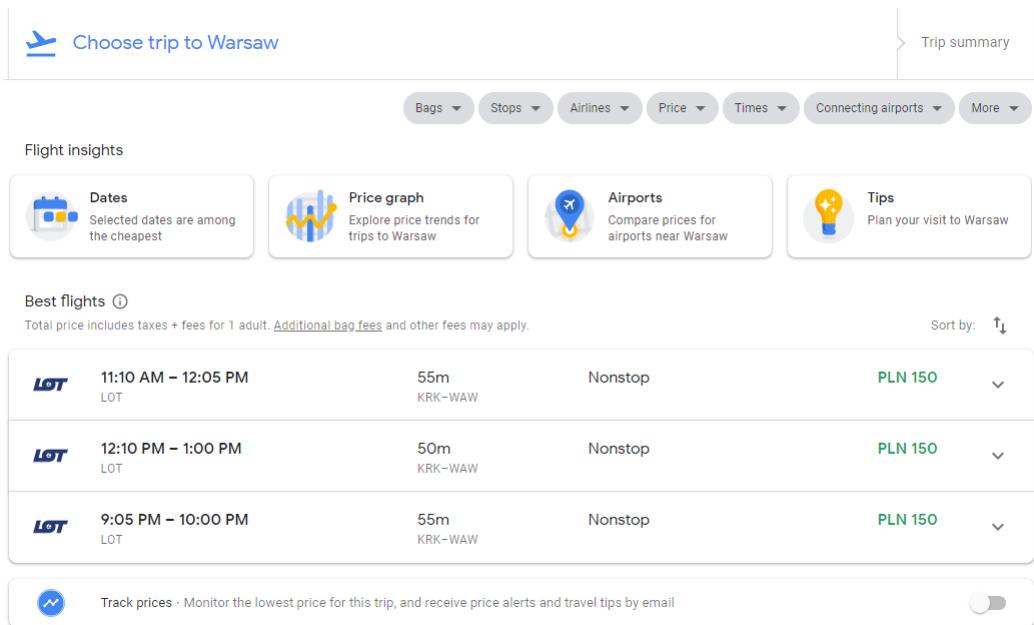


Rysunek 2.2.1: Panel wyszukiwania lotów aplikacji Google Flights



Rysunek 2.2.2: Komponent podróży wieloetapowej aplikacji Google Flights

Wszystkie produkty firmy Google, które można obejrzeć w przeglądarce internetowej charakteryzują się specyfcznym stylem interfejsu. Jest to platforma Material Design, która wyznacza specyfikację pojedynczych elementów aplikacji Google'a. Rysunek 2.2.3 przedstawia przykładowe wyniki wyszukiwania lotów. Wyszukane loty znajdują się w rozwijanej liście. Po kliknięciu w wybrany element lista rozwija się panel z dokładniejszymi informacjami o locie. Każdy element zaprezentowany na poniższym rysunku jest zgodny z wcześniej wspomnianą platformą Material Design. Widać na przykład zaokrąglenia przycisków charakterystyczne dla produktów marki Google.



Rysunek 2.2.3: Wyniki wyszukiwania lotów

2.3. Porównanie aplikacji

Obie opisane aplikacje są rozwiązaniami komercyjnymi, stworzonymi przez wieloosobowe zespoły programistów. Są systemami sprawnie działającymi, z czytelnym oraz zrozumiałym interfejsem graficznym. Swoją funkcjonalnością są do siebie identyczne, posiadają tą samą ofertę dla użytkownika jednak różnią się jakością jej wykonania.

Zaletami aplikacji Skyscanner jest z pewnością rzetelność jej danych. Jest to potwierdzone ze strony linii lotniczych, które wielokrotnie ją nagradzały. Warto też wspomnieć o jej systemie wyszukiwania lotów. Ilość wyszukanych lotów z przesiadkami często przekracza 100 wystąpień. Świadczy to o zaawansowanym silniku wyszukującym i analizującym różne przypadki połączeń. Za wady można uznać dosyć prosty i nieresponsywny interfejs użytkownika. Nie reprezentuje on poziomu nowoczesnych aplikacji.

Druga wyszukiwarka jest bardzo dopracowanym produktem światowego giganta. Jak wcześniej już wspomniano jego szybkość wyszukiwania danych jest bezkonkurencyjna. Aplikacja Skyscanner znacznie ustępuje jej na tym polu. Google Flights jest sprzężona z systemami sprzedawczymi większości przewoźników lotniczych. Daje to możliwość kupna biletu bezpośrednio z jej poziomu bez konieczności przechodzenia na zewnętrzną stronę linii lotniczej. Jej dużą zaletą z pewnością jest dopracowany interfejs zgodny ze standardami reprezentowanymi w rodzimych produktach. Gwarantuje on pełną responsywność oraz czytelność prezentowanych danych.

Wyszukiwarka Google Flights prezentuje się lepiej od aplikacji Skyscanner jednak

różnica je dzieląca nie jest duża. W opracowywanej aplikacji starano się w rzetelny sposób zrealizować podstawowe funkcjonalności obu opisywanych wcześniej aplikacji. Komercyjne aplikacje mają dużą przewagę biorąc pod uwagę szybkość wyszukiwania. W powstałej aplikacji czas odpowiedzi na żądania użytkownika jest zauważalnie większy. W warunkach akademickich trudno było o uzyskanie konkurencyjnych rezultatów.

3. Projekt aplikacji

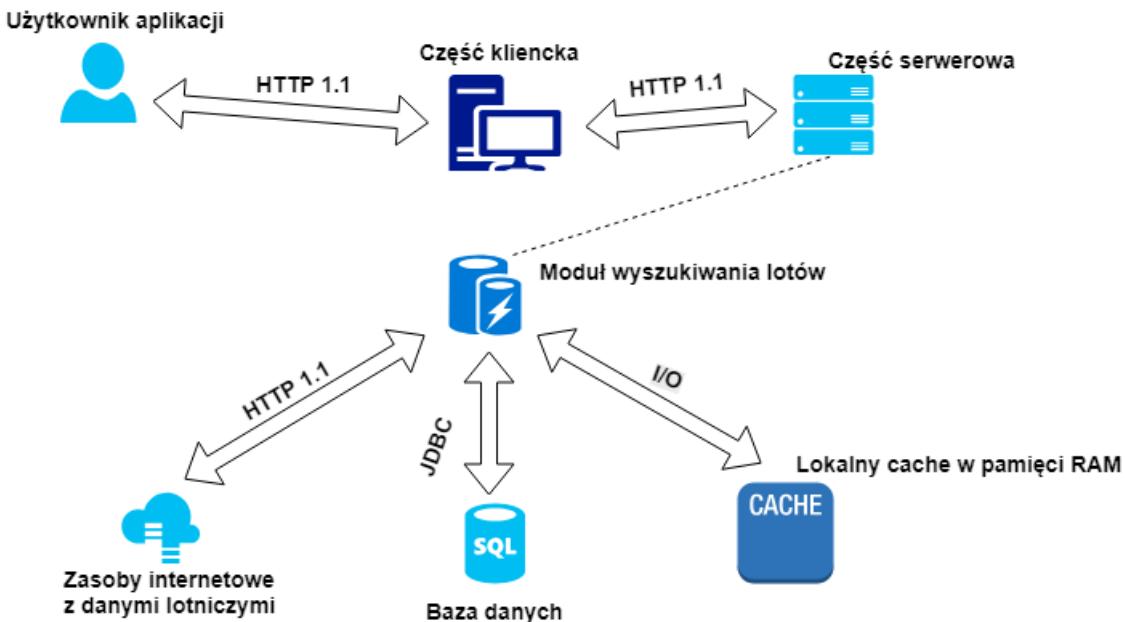
Następnym krokiem po analizie zagadnienia oraz zaznajomieniem się z przykładami istniejących rozwiązań było opracowanie projektu aplikacji. Zadanie to jest mocno związane z dziedziną architektury oprogramowania. Dobrze zaprojektowany system pozwala na uniknięcie wielu kosztownych problemów w przyszłości. Projektowanie aplikacji wymusza na osobie to wykonującej dostatecznie wcześnie rozważenia najważniejszych aspektów projektowania w odniesieniu do całego tworzonego systemu[5]. Wynikiem zrealizowania zadań analitycznych są wymagania postawione tworzonemu systemowi. Wynikiem tych działań mogą być takie elementy jak model dziedziny, model organizacyjny czy model wymagań . Projekt aplikacji ukierunkowuje zadania implementacyjne razem z projektowaniem szczegółowym, implementacją kodu, integracją modułów oraz testowaniem.

3.1. Ogólna architektura aplikacji

Projektowanie aplikacji wymagało stworzenia czytelnego schematu obrazującego poszczególne części systemu oraz sposoby komunikacji między nimi. Aplikację można podzielić na 3 logiczne części:

- Część serwerowa
- Część kliencka
- Baza danych

Każda część posiada unikalną dla siebie funkcjonalność. Ze względu na rozmiar pracy włożony w każde z nich, w tym rozdziale zostaną one krótko opisane. Część serwerowa w całym systemie jest odpowiedzialna za dostarczenie danych do interfejsu użytkownika inaczej nazywanego częścią kliencką. Znajduje się w nim złożony moduł wyszukiwania połączeń lotniczych. Moduł ten sam w sobie posiada złożoną architekturę i zostanie mu poświęcony dedykowany podrozdział. Część kliencka to internetowy interfejs użytkownika. Osadzony jest on w przeglądarce internetowej. W aplikacji nie zdecydowano się na podział względem ról użytkownika. Wynika to ze specyfiki projektowanej aplikacji. Każdy użytkownik ją odwiedzający posiada takie same uprawnienia do jej zasobów. Oprócz dwóch wspomnianych wcześniej modułów w skład architektury aplikacji wchodzi relacyjna baza danych, która jest nośnikiem danych dla tabeli lotnisk. Kluczowym zagadnieniem jest komunikacja między powyższymi komponentami. W stworzonej aplikacji istnieje kilka rodzajów transportu danych między modułami. Każdy z nich zostanie szczegółowo opisany w dalszej części tego rozdziału.



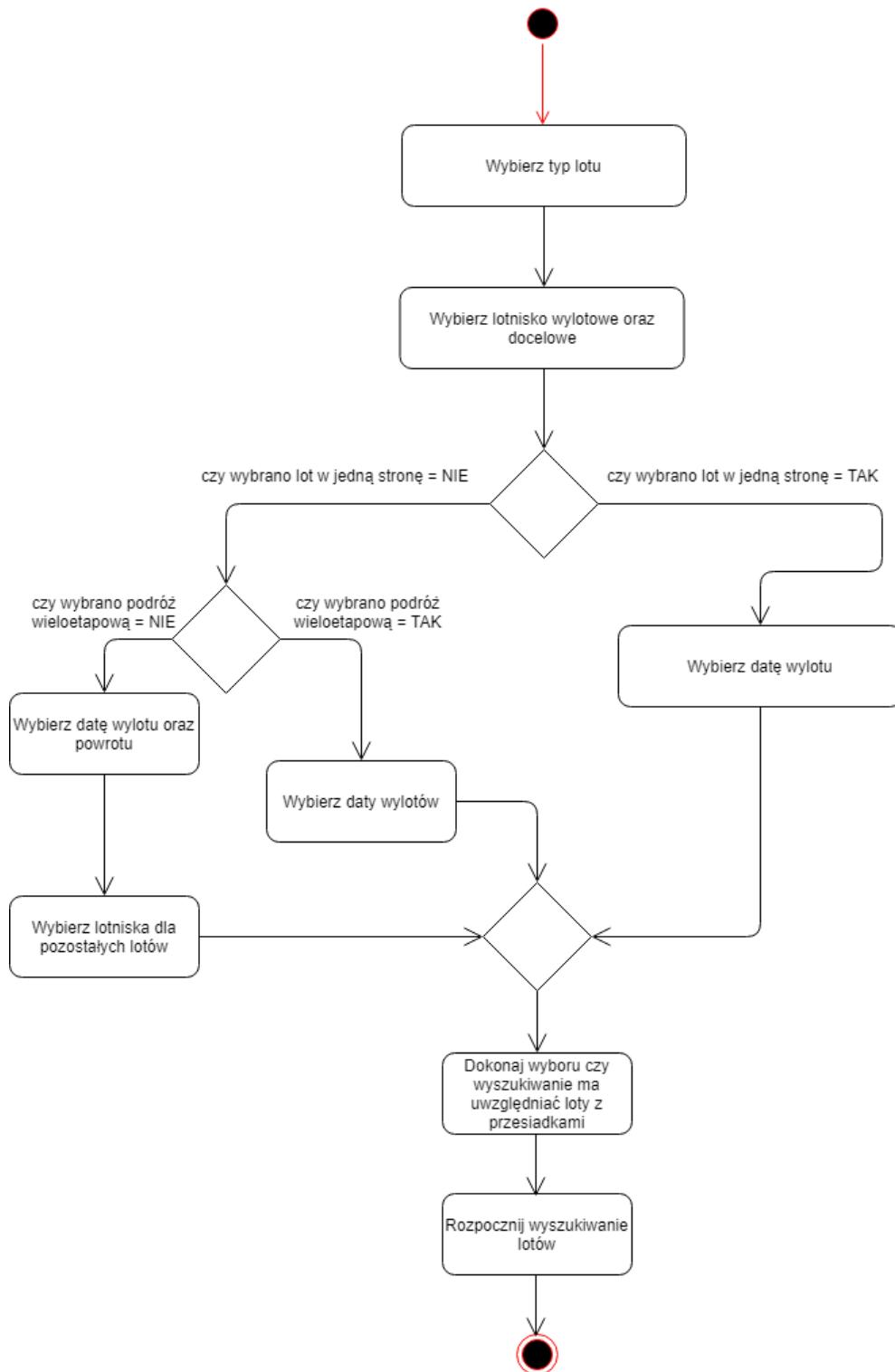
Rysunek 3.1.1: Schemat architektury stworzonej aplikacji

Na rysunku 3.1.1 zamieszczony został schemat architektury stworzonego systemu. Przedstawia on rozdział strukturalny aplikacji oraz drogę jaką musi odbyć żądanie użytkownika w celu uzyskania pożądanej odpowiedzi. Użytkownik po wypełnieniu odpowiednich formularzy w części klienckiej rozpoczyna proces wyszukiwania połączeń lotniczych. Oprogramowanie klienta po pobraniu od użytkownika niezbędnych parametrów takich jak miejsce wylotu, czy rodzaj lotu zwraca się do części serwerowej o wyszukanie lotów zgodnych z podanymi parametrami. Komunikacja między tymi modułami odbywa się protokołem HTTP. Po wywołaniu, część serwerowa zwraca się do modułu wyszukiwania lotów. Moduł ten sprawdza wszystkie źródła danych zamieszczone na schemacie 3.1.1. Zebrane dane są odpowiednio parsowane i przekazywane modułowi wyszukiwania który z kolei zwraca je kontrolerowi wywołanemu przez użytkownika. Kontroler następnie przekazuje dane do części klienckiej, która wyświetla je w graficznym interfejsie użytkownika.

3.2. Diagram UML

Kluczowym elementem podczas tworzenia oprogramowania jest komunikacja. Przełożenie złożonej logiki biznesowej na język informatyczny to zadanie trudne oraz wymagające zrozumienia wymagań potencjalnego odbiorcy tworzonej dokumentacji technicznej. Ważny jest więc czytelny sposób opisu oprogramowania, który byłby w stanie zrozumiale przedstawić swoją treść wszystkim osobom się z nim zaznajamiającym[6]. Podstawowym narzędziem w tej dziedzinie stał się język UML¹.

¹Unified Modelling Language



Rysunek 3.2.1: Diagram aktywności procesy wyboru parametrów wyszukiwania

Dla opracowanej aplikacji przygotowano diagram aktywności widoczny na rysunku 3.2.1. Obrazuje on przebieg wydarzeń które wykonywane są przez użytkownika po wczytaniu przez niego strony startowej. Jego najważniejszą funkcją jest ukazanie sekwencji kroków wykonywanych w części klienckiej.

3.3. Baza danych

Baza danych nie była sprecyzowana w wymaganiach pracy dyplomowej. Została ona użyta opcjonalnie jako miejsce dające szybki dostęp do danych o lotniskach. Użyto darmowego rozwiązania bazodanowego MySQL. Jest to system zarządzania relacyjnymi bazami danych rozwijany w przeszłości przez firmę MySQL AB, a obecnie przez korporację Oracle. Stanowi szybko działające oprogramowanie i obecnie jest jednym z popularniejszych serwerów baz danych dostępny na licencji GPL² jak i w wersjach komercyjnych [7].

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 3.3.1: Tabela danych lotnisk

Jedyna tabela znajdująca się w bazie danych to tabela lotnisk z parametrami wskazanymi powyżej.

3.4. Część kliencka

Aplikacja kliencka została zaimplementowana w technologii Angular 6, otwartym framework'u i platformie do tworzenia SPA³. Angular został w całości napisany w języku TypeScript, opiekę nad jego rozwojem sprawuje firma Google. Platforma ta umożliwia tworzenie stron internetowych których treść jest dynamicznie zmieniana bez przeładowywania strony. Przynosi to ogromne korzyści w wydajności aplikacji. Całość kodu napisanego w tym framework'u jest kompilowana do języka JavaScript a następnie renderowana w przeglądarce internetowej. W

²General Public License

³Single Page Application

skład stworzonej aplikacji klienckiej wchodzą dodatkowo dodane zewnętrzne biblioteki.

Są to:

- Material Design - pakiet oprogramowania od firmy Google do edytowania wyglądu strony
- MDBootstrap - zestaw narzędzi z gotowymi komponentami do tworzenia stron internetowych
- NgxSpinner - zewnętrzna biblioteka dodająca komponenty ładowania efektów wizualnych
- Bootstrap - pakiet oprogramowania do tworzenia responsywnego interfejsu

Implementacja części klienckiej jak i jej logiczny podział strukturalny znajdzie się w następnym rozdziale.

3.5. Część serwerowa

Do opracowania aplikacji serwerowej posłużono się językiem programowania Java w wersji 10 oraz pakietami oprogramowania Spring i Hibernate. Java jest w pełni obiektowym językiem programowania z ponad 20 letnią historią. Pierwotnie stworzona i rozwijana przez Jamesa Goslinga została przejęta przez korporację Oracle. Głównym założeniem tego języka jest sentencja "*Napisz raz, uruchom wszędzie*". Te słowa przemawiają za specjalnym mechanizmem kompilowania i uruchamiania kodu przez Javę. Proces zaczyna się od skompilowania plików o rozszerzeniu java do bytecode'u czyli specjalnej sekwencji bajtów rozumianej przez JVM⁴. Następnym krokiem jest uruchomienie bytecode'u przez wirtualną maszynę Javy, która komunikuje się z systemem operacyjnym[8].

Powstałe oprogramowanie serwerowe jest w dużej mierze oparte na zewnętrznych bibliotekach platformy Spring Framework w wersji 5. Spring jest otwartym źródłowo frameworkiem bazującym na wirtualnej maszynie Javy. Jego składowymi jest zbiór bibliotek, których głównym celem jest rozwiązywanie popularnych problemów programistycznych. Został stworzony w roku 2002 jako kandydat do zastąpienia oprogramowania spod znaku JavyEE⁵. W dzisiejszych czasach Spring jest rozwijany przez setki programistów z całego świata co czyni go stale nowoczesną technologią. W części serwerowej Sping Framework odpowiada za komunikację z bazą danych, udostępnienie danych o połączeniach lotniczych aplikacji klienckiej oraz za zabezpieczenia całego systemu.

W części serwerowej w celu przyspieszenia prac programistycznych użyto szeregu zewnętrznego oprogramowania. Należą do nich biblioteki:

- okhttp - klient protokołu HTTP do pobierania zasobów internetowych

⁴Java Virtual Machine

⁵Java Enterprise Edition

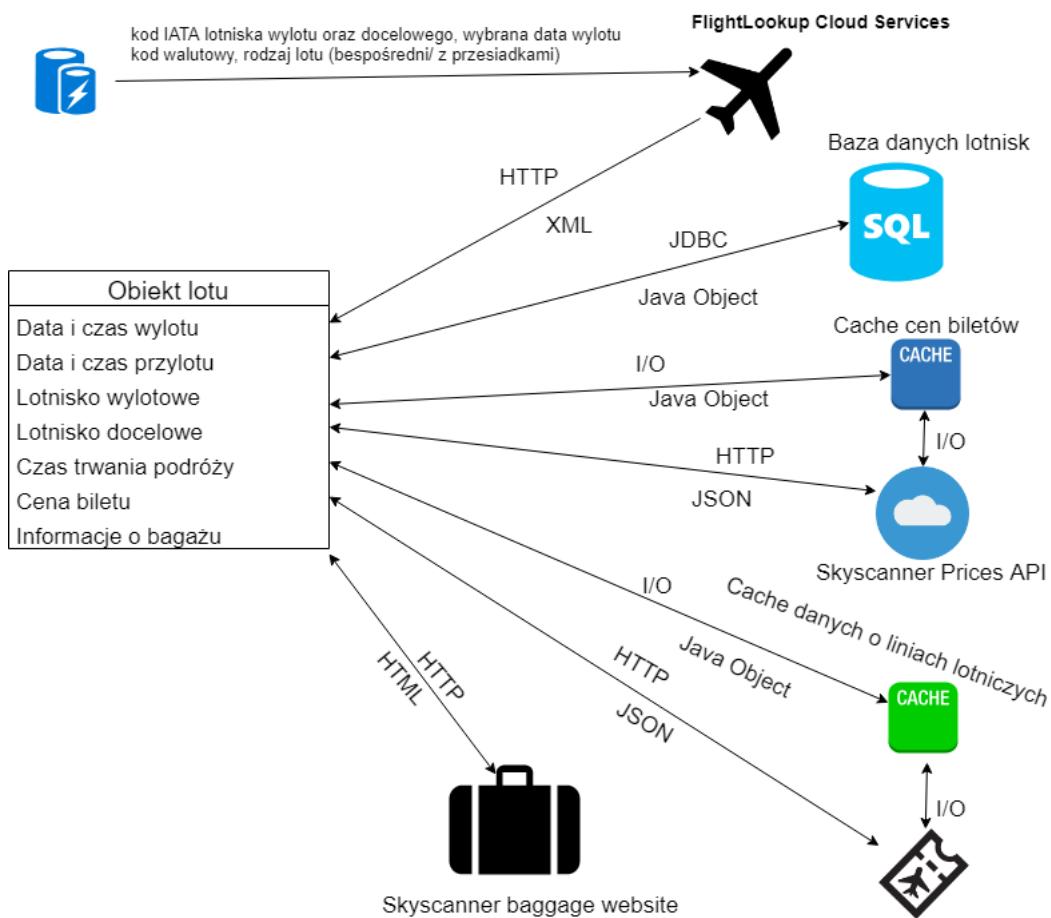
- gson - biblioteka do parsowania obiektów w notacji JSON
- jackson-dataformat-xml - biblioteka do parsowania języka znaczników XML
- jsoup - biblioteka do parsowania dokumentów HTML
- ehcache - oprogramowanie zarządzające danymi w pamięci podręcznej aplikacji
- commons-csv - biblioteka do parsowania plików CSV
- jaxb-api - biblioteka języka Java usunięta z głównej ścieżki modułowej w wersji JDK⁶ 9

W skład części serwerowej wchodzi też cały moduł wyszukiwania połączeń lotniczych. Ze względu na jego znaczenie zostanie mu poświęcony osobny podrozdział. Implementacja części serwerowej zostanie zaprezentowana w następnym rozdziale.

⁶Java Development Kit

3.6. Moduł wyszukiwania połączeń lotniczych

Rozwiążanie problemu wyszukiwania połączeń lotniczych było zadaniem trudnym i skomplikowanym. Wymagało to poradzenia sobie z problemem przeszukiwania wielu źródeł danych oraz konwertowania informacji przez nie zwracany do formy zrozumiałej dla części serwerowej oraz klienckiej. Zagadnienie to wymagało stworzenia wyodrębnionego modułu wyszukiwania połączeń lotniczych który został osadzony jako część oprogramowania serwerowego powstałej aplikacji.



Rysunek 3.6.1: Schemat architektoniczny modułu wyszukiwania lotów

Na rysunku 3.5.1 przedstawiono schemat zaprojektowanego modułu wyszukiwania połączeń lotniczych. Zobrazowano na nim przebieg procesu wyszukiwania z uwzględnieniem źródeł danych, sposobów ich uzyskiwania oraz rodzajów odpowiedzi. Górnny opis strzałek oznacza metodę zwrócenia się o dane, natomiast dolny opisuje rodzaj zwróconej treści. Wyszukiwanie rozpoczęło się od wysłania żądania z kontrolera serwerowego do omawianego modułu. Żądanie to musiało uwzględniać w sobie parametry wyszukiwania. Moduł swoją pracę rozpoczyna od pozyskania danych lotniczych z serwisu FlightLookup. Treść odpowiedzi stanowi większą część

całego obiektu lotu który ma zostać zwrócony. Po otrzymaniu tych danych moduł zwraca się do lokalnej bazy danych o udostępnienie danych lotnisk podając przy tym ich unikalne kody IATA⁷. Następnym krokiem jest pozyskanie ceny lotu. Czynność ta rozpoczyna się od sprawdzenia czy poszukiwana informacja nie znajduje się w cache'u. Jeśli takie dane są zwrócone, cache natychmiastowo zwraca odpowiedni zasób. Jeśli natomiast taka informacja nie jest dostępna, moduł zwraca się do serwisów firmy Skyscanner o udostępnienie cen na poszukiwany przelot. Analogiczna sytuacja dokonuje się w przypadku pozyskiwania danych o linii lotniczej obsługującej lot. Ostatnim etapem jest wyszukanie 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 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 narzucały sposoby komunikacji między nimi. Każde z nich zostanie w tym podrozdziale stosownie przedstawione.

3.7.1. Protokół HTTP

Protokół HTTP⁸ to jeden z najpopularniejszych i powszechnie przyjętych protokołów aplikacji w świecie internetu. Jest to wspólny język między serwerami a klientami umożliwiający modernistyczną sieć. Od prostego początku jako ciąg znaków stał się protokołem nie tylko dla przeglądarek internetowych ale praktycznie też dla każdego oprogramowania i aplikacji sprzętowych[9]. Cechą protokołu HTTP jest zapis i odczytywanie reprezentacji danych, co umożliwia budowę systemów niezależnych od przesyłanych danych. Początkowo protokół ten powstał z oznaczeniem wersji 0.9. Jest to obecnie dość stara wersja, używana jednak jeszcze przez niektóre serwery. HTTP 0.9 zostało podniesione do wersji 1.1 który jest najczęściej wykorzystywany protokołem w internecie. Standard HTTP/1.1 rozwiązał wiele niejednoznaczności protokołu, które znaleziono we wcześniejszych wersjach i wdrożył szereg kluczowych optymalizacji wydajnościowych, dodatkowe mechanizmy buforowania oraz enkrypcję transferu. Jako odpowiedź na ciągle zwiększącą się liczbę urządzeń podłączonych do internetu oraz wymagania wobec niego, w 2012 roku zainicjowano protokół HTTP/2. Jego głównym celem jest poprawa wydajności transportu i osiągnięcie zarówno mniejszych opóźnień i wyższej przepustowości.

⁷International Air Transport Association

⁸Hypertext Transfer Protocol

3.7.2. JDBC

JDBC⁹ to standardowy, niezależny od bazy danych interfejs do interakcji z dowolnym źródłem danych opartym na tabelaryczności. Przeważnie jest używany do współdziałania z relacyjnym systemem zarządzania bazami danych. Za pomocą tego interfejsu można też korzystać z dowolnego tabelarycznego źródła danych, takiego jak arkusz csv, plik tekstowy itd. Zazwyczaj używa się go do łączenia z bazą danych, wyszukiwania danych i ich aktualizowania. Umożliwia także wykonywanie procedur SQL w bazie danych przy użyciu składni niezależnej od bazy danych[10]. Używanie JDBC zwalnia użytkownika z konieczności uczenia się nowej składni za każdym razem pracy na innej bazie danych. JDBC dostarcza zestaw oprogramowania w języku Java do przetwarzania zestawu wyników zapytań SQL w sposób niezależny od bazy danych. W opracowanej aplikacji interfejs JDBC pośredniczył pomiędzy relacyjną bazą danych MySQL a częścią serwerową aplikacji.

3.7.3. I/O

Operacje zapisu oraz odczytu są podstawowymi częściami systemów operacyjnych wraz z językami programowania oraz ich bibliotekami. Prawie wszystkie programy komputerowe wykonują pewne operacje zapisu/odczytu inaczej zwany operacjami wejściowymi i wyjściowymi. W opracowanej aplikacji za wykonywanie tych operacji odpowiadał język Java ,który przez system operacyjny zwracał się do zasobów plikowych lub pamięci 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]

⁹Java Database Connectivity

4. Implementacja

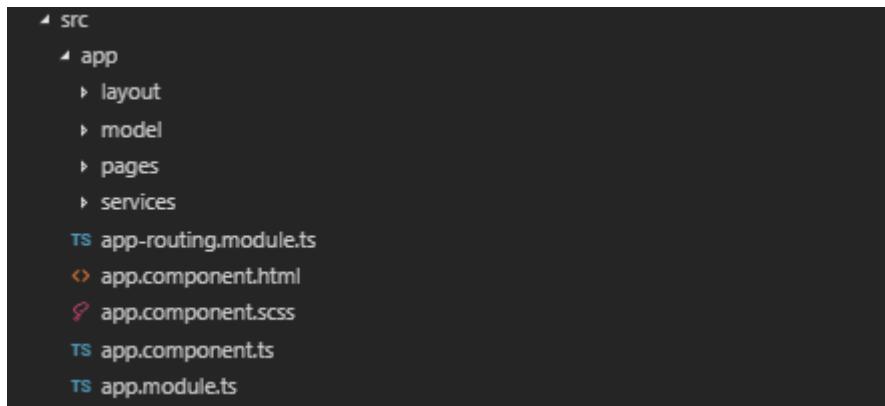
Następstwem stworzenia projektu aplikacji była implementacja oprogramowania zawartego w założeniach pracy dyplomowej. Zostanie ona szczegółowo opisana w tym rozdziale. Jego treść została stworzona na podstawie projektu aplikacji oraz zagadnień poruszonych w poprzednich rozdziałach. Stworzona aplikacja została podzielona według założonych wymagań opisanych na stronie trzeciej. Została 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.

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 pakiety:



Rysunek 4.1.1: Struktura plików części klienckiej

Pakiet *layout* przechowuje komponenty o wielokrotnym użyciu w aplikacji klienckiej. Katalog *model* zawiera odpowiedniki klas modelowych z częścią serwerową. Kolejnym elementem jest pakiet *pages*. Znajdują się w nim komponenty renderujące kompletne strony internetowe. Zostały zbudowane z segmentów stworzonych 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 odbierające dane z części serwerowej. Pierwszym z nich jest odpowiedzialny za połaczenie z modułem 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 powyższym listingu przedstawiono fragment oprogramowania pobierający wyniki wyszukiwania. Funkcja *getFlights* przyjmuje 5 parametrów które dostarcza jej użytkownik. Pierwszym procesem tej funkcji jest zbudowanie adresu URL który posłuży do zbudowania żądania pobrania danych. Jej działanie polega na złączeniu wcześniej wspomnianych parametrów oddzielając je znakiem "/" w roli separatora. Ostatnim krokiem jest stworzenia żądania GET protokołu HTTP.

Drugim zaimplementowanym serwisem jest prosta funkcja pobierająca dane o lotni skach. Jej działanie zostanie przedstawione na kolejnym listingu.

```

1 import { Injectable } from '@angular/core';
2 import { HttpClient, HttpErrorResponse } 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
16   getAirportsStartingWithPhrase(phrase: string): Observable<Airport[]>{
17     return this.http.get<Airport[]>(this.getAirportsUsingIncompletePhraseURL + phrase).
18       pipe(catchError(this.handleError));
19   }
20 }
```

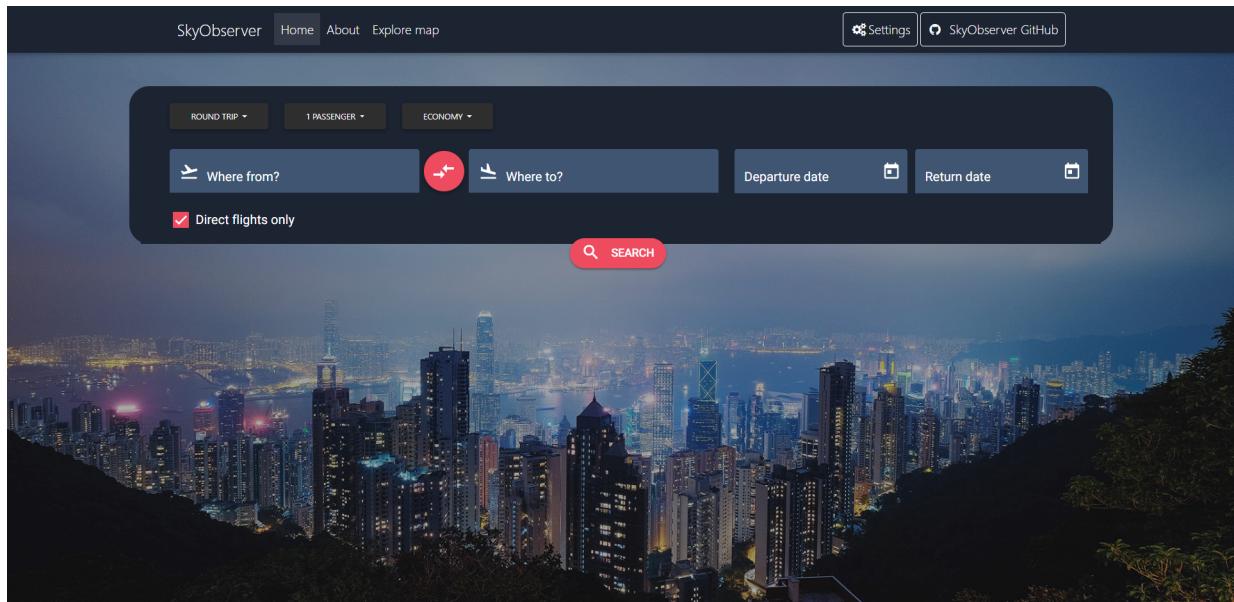
Listing 4.1.2: Kod źródłowy funkcji pobierającej dane lotnisk

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

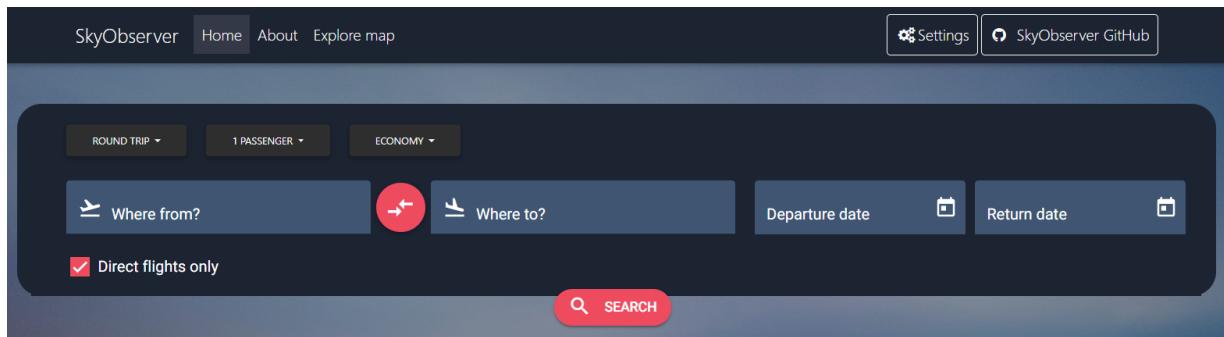
Walory szybkości oraz użyteczności aplikacji tracą na znaczeniu jeśli budowane oprogramowanie nie jest wyposażone w czytelny interfejs użytkownika. Jest on formą interakcji pomiędzy użytkownikiem a aplikacją. Stworzone środowisko graficzne jest grupą wzajemnie współpracujących przycisków oraz obrazów zapewniających podstawowe poruszanie się po aplikacji klienckiej w obrębie przeglądarki internetowej. W części klienckiej zaimplementowano responsywny wygląd z takimi komponentami jak panel wyszukiwania lotów czy panel podróży wieloetapowej. Ich ogólny kształt i funkcje zostały w części 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: Strona główna aplikacji klienckiej

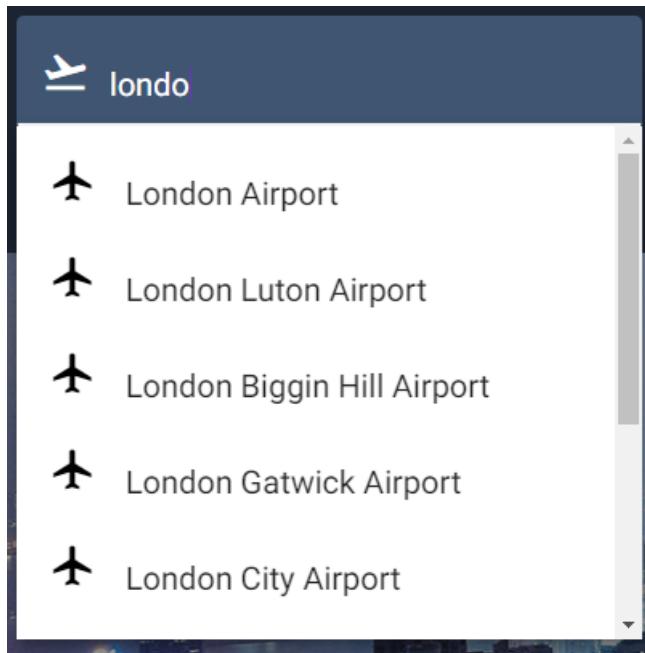
Rysunek 4.1.2 przedstawia stronę startową części klienckiej. Jego tło jest na całej szerokości oraz długości wypełnione obrazem miasta Hongkong. Górną część strony przedstawia pasek nawigacyjny aplikacji, panel ustawień oraz odwołanie do repozytorium kodu stworzonej pracy dyplomowej. Następnym komponentem aplikacji klienckiej jest panel wyszukiwania lotów.



Rysunek 4.1.3: Panel wyszukiwania połączeń lotniczych

Przedstawiony komponent obrazuje rozmieszczenie przycisków, pól wyboru parametrów wyszukiwania oraz dostępne opcje konfiguracyjne. Górną część panelu zawiera trzy przyciski konfiguracyjne. Odpowiadają one kolejna za wybór rodzaju lotu, ilość pasażerów oraz klasę poszukiwanego biletu lotniczego. W aplikacji założono trzy typy lotów: lot z możliwością powrotem, lot bezpośredni oraz podróż wieloetapowa. Kolejna część poniżej opisanych przycisków przedstawia pola do wyboru parametrów poszukiwania. Ich ilość zmienia się w zależności od wybranego typu lotu.

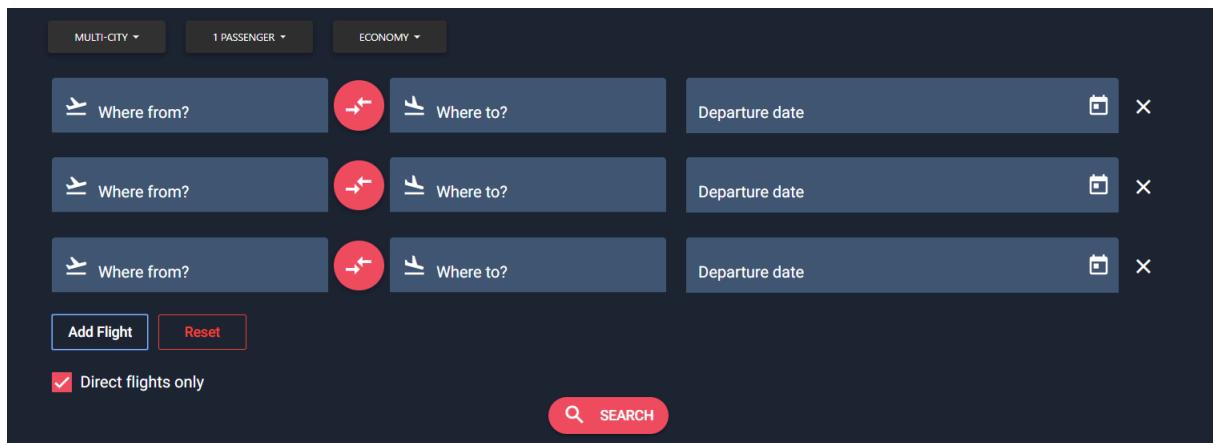
W elementach wyboru lotnisk został zaimplementowany mechanizm podpowiedzi wyszukiwania. Użytkownik po wpisaniu kilku pierwszych liter poszukiwanego miasta będzie mógł zobaczyć listę proponowanych lotnisk w obrębie tej miejscowości.



Rysunek 4.1.4: Komponent wyboru lotniska

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 w poprzednim podrozdziale. Użytkownikowi wyświetlane są pełne nazwy lotnisk. Po wybraniu poszukiwanego miejsca widoczna jest nazwa miasta wraz z unikalnym kodem IATA identyfikującym lotnisko. Elementem 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 w jedną stronę jest on niewidoczny a szerokość pierwszego jest rozszerzona. Kliknięcie w ikonę kalendarza wyzwala okno modalne z jego powiększonym widokiem. Szarym okręgiem jest na nim zaznaczony dzień dzisiejszy. Po wybraniu, data zapisywana jest w pamięci aplikacji. Ten zabieg służy do zabezpieczenia sytuacji przy ewentualnym wyborze daty powrotu. W takiej sytuacji aplikacja wskaże zisaną 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 4.1.5: Komponent podróży wieloetapowej

Rysunek 4.15 przedstawia wygląd opisywanego komponentu. Interfejs oferuje przyciski do dodawania kolejnych etapów podróży oraz ich usuwania.

Wspólnym elementem zarówno dla modułu wieloczęściowej 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 podrozdziale. Wygląd przykładowych wyników wyszukiwania zaprezentowano na obrazku 4.1.6 Wyświetla on listę 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.

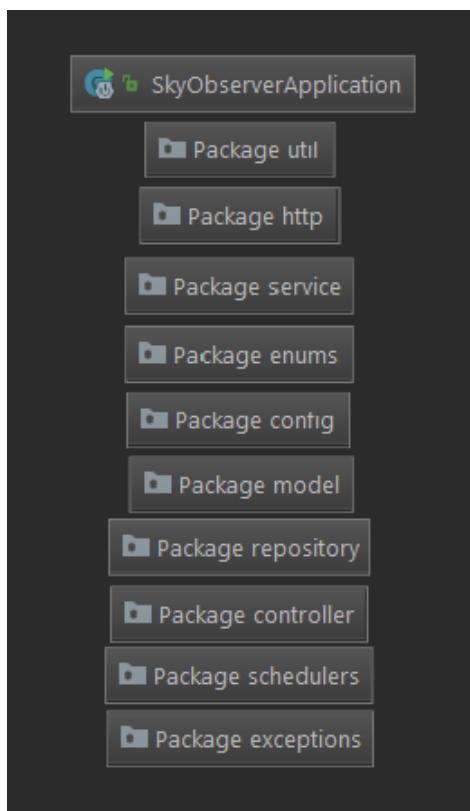
Rysunek 4.1.6: Wyniki wyszukiwania

4.2. Oprogramowanie po stronie serwera

Oprogramowanie serwerowe stanowi większą część całej aplikacji. Zostały w nim zaimplementowane mechanizmy parsowania, wyszukiwania i testowania danych lotniczych. Kwestie związane z użytymi technologiami przy tworzeniu serwera aplikacji zostały omówione w poprzednim rozdziale. W bieżącej części pracy zostanie opisana struktura części serwerowej, oprogramowanie odpowiedzialne za parsowanie różnych rodzajów danych jak i zaimplementowany moduł wyszukiwania.

4.2.1. Struktura pakietowa

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.



Rysunek 4.2.1: Podział strukturalny części serwerowej

Omawianie tych katalogów najlepiej zacząć od pakietu *config*. Jego zadaniem jest konfiguracja różnych parametrów serwera podczas jego startu. Ustawia on takie rzeczy jak klucze dostępne do zasobów lotniczych, parametry lokalnego cache'u czy też zabezpieczenia aplikacji. Warstwa modelowa aplikacji znajduje się w pakiecie *model*. Zaimplementowane zostały w nim klasy modelowe które opisują obiekty krążące po całej aplikacji. Związanym z tym pakietem jest katalog *repository*. Jest on miejscem na klasy implementujące pozyskiwanie obiektów odp-

wiadających klas modelowych. Niektóre z nich używają swoich analogicznych odpowiedników z pakietu *service*. Potrzeba ta wynika z konieczności parsowania niektórych formatów danych do obiektów 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 precyzyjnie określone. Dotyczy to lokalnych plików przechowywujących dane o lotniskach oraz bagażach. Kod odpowiedzialny za ich aktualizowanie jest opracowany w katalogu *schedulers*. Pozostałe pakiety *util*, *enums* oraz *exceptions* wprowadzają mniej znaczące dla aplikacji funkcje.

4.2.2. Oprogramowanie wyszukujące połączenia lotnicze

Moduł wyszukiwania połączeń lotniczych został już opisany w poprzednim rozdziale, bieżący zostanie poświęcony kwestiom jego implementacji. Na potrzeby jego stworzenia opracowano specjalny sposób tworzenia obiektu lotu. Standardowy proces powstawania obiektu przez konstruktor języka Java okazał się mało czytelny dla wielu obiektów wchodzących w skład jednostki połączenia lotniczego. Dla tego obiektu opracowano implementację wzorując się na wzorze projektowym o nazwie *Budowniczy*. Budowniczy to programistyczny wzorzec projektowy, który opisuje algorytm tworzenia 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 klasą wewnętrzną ją budującą. Jej implementacja została 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 odpowiedzialną za zbieranie danych o połączeniach lotniczych jest plik *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. Ze względu na złożoność oraz wielkość jego implementacji jego kod nie zostanie pokazany 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.

Opracowane 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 cen lotów
- FlightsParser - moduł parsujący złożone treści otrzymane w języku znaczników XML na analogiczne 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.

```
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 przedstawia sposób konwersji treści w formacie JSON. Do tego celu posłużono się biblioteka gson firmy Google. Wywołując metodę *fromJson* na obiekcie typu *Gson* w prosty sposób odbywa się konwersja obiektów Json na obiekty *Airline*. Funkcja jako wynik zwraca pierwszy element zparsowanej tablicy danych.

Translacja treści zwróconej z serwisu *Flight Lookup* otrzymanej w postaci XML odbywała się przy pomocy biblioteki Jackson. Pierwszym krokiem było stworzenie odpowiedników

elementów odpowiedzi na klasy modelowe w części serwerowej. Po tej czynności można było przystąpić do implementacji metody konwertującej.

```
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

Powyzsza implementacja jest podobna do poprzedniej przedstawionej na listingu 4.2.3. Metoda 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. Ustawi ono dostęp do serwera tylko z adresu części serwerowej podanego jako argument metody *addMapping*. Zabieg ten zabezpiecza dostęp do danych dla niepożądanych adresów sieciowych.

Mechanizm cachowania danych usprawniający wydajność aplikacji zminimalizował również szanse na skuteczny atak DDOS. Został on krótko opisany w rozdziale pierwszym w którym omówiono 3 kontenery cache'u zaimplementowane w aplikacji. Szybkość ich odpowiedzi uodparnia część serwerową na nadmiarowy ruch sieciowy ze stony użytkowników aplikacji. Zdefiniowano dla nich maksymalny rozmiar przechowywanych danych wynoszący 1GB dla każdego z kontenerów. Po przekroczeniu go, cache zostanie wyczyszczony.

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
15        Collection<MultiFlight> flights = multiFlightsRepository.searchForMultiFlights(
16            from, to, date, connection, currency);
17
18        if (flights.isEmpty()){
19            throw new FlightsNotFoundException();
20        }
21
22        return flights;
23    }
24}
```

Listing 4.2.5: Kontroler zwracający wyniki wyszukiwania lotów

Powyzszy kontroler został zaimplementowany w pakiecie *controller*. Dwie adnotacje nad deklaracją klasy 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 repozytorium lotów, które zwraca wyniki wyszukiwania w postaci kolekcji obiektów. W przypadku nie znalezienia żadnych połączeń lotniczych, metoda zwraca wyrzuca wyjątek, który implikuje zwróceniem odpowiedzi o kodzie 404.

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 ewoluuje 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 sprzężonym 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:

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

W stworzonej 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 dostarczająca zestaw narzędzi do testowania oprogramowania napisanego w języku Java. Znany i sprawdzony przez wielu programistów jest częścią wielu komercyjnych systemów. Drugi z nich, przedstawiony wcześniej w rozdziale trzecim dostarcza instrumenty do testowania jego oprogramowania serwerowego.

Podstawową metodą testującą oprogramowanie była asercja. Jest to predykat, który zwraca wartość logiczną. Najczęściej przyjmuje dwa argumenty a następnie porównuje je zwracając wynik[15]. Napisane testy znajdują się w części serwerowej aplikacji, testują one poprawność odpowiedzi serwera, informacje otrzymane z zewnętrznych serwisów, parsowanie różnych formatów danych jak i najbardziej podstawowe elementy aplikacji. Nazwy testów przemawiają za kodem źródłowym który testują. Dla przykładu 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);
18 }
```

Listing 5.1.2: Przykładowy test jednostkowy

Test zaprezentowany na listingu 5.1.2 inicjalizuje 3 obiekty typu *Flight* z przykładowymi wartościami cen przelotów. Obiekty te łączone są w kolekcję. Końcowa asercja sprawdza poprawne działanie metody *calculateTotalFlightPrice* porównując jej wynik z oczekiwana wartością. Warto zwrócić na trzeci argument jaki przyjmuje metoda *assertEquals*. Oznacza on dopuszczalną różnicę między oczekiwany a otrzymanym wynikiem.

5.2. Testy integracyjne

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 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 na listingu powyżej testuje funkcjonowanie repozytorium zwracające informacje o liniach lotniczych. W pierwszym kroku pobiera ono obiekt linii lotniczej. Po przypisaniu tych danych do obiektu, sprawdzane są poszczególne parametry zwróconych informacji.

Test o analogicznej funkcji sprawdzał też 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 zrozumienia logiki biznesowej związanej z pozyskiwaniem danych lotniczych. Do największych wyzwań podczas pracy z pewnością należało znalezienie odpowiednich źródeł danych udostępniających dane o lotach. Wysłano szereg zgłoszeń mailowych z prośbą o udostępnienie takich danych. W większości przypadków odpowiedzi były negatywne lub niesatysfakcyjne z powodu żądania od dostawcy pokaźnej zapłaty za korzystanie z usługi. W warunkach akademickich skorzystanie z takich rozwiązań było niemożliwe. Po wnikliwej analizie rynku danych lotniczych udało się znaleźć odpowiednie źródło informacji dla potrzeb stworzonej aplikacji.

Z dużą dozą pewności można stwierdzić iż opracowana aplikacja w pełni spełnia założone wymagania. Pozwala z całkiem dobrą szybkością wyszukiwać połączenia lotnicze oraz dostarczać wielu dodatkowych informacji z nimi powiązanych. W sposób logiczny i przemyślany zaprojektowano moduł wyszukiwania lotów. Opracowano dla niego specjalną architekturę, która została odzwierciedlona w kodzie źródłowym pod postacią wzorca projektowego. Dzięki temu zabiegowi aplikacja zachowuje potencjał łatwego i mało problemowego rozwoju.

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

Z uwagi na warunki akademickie aplikację można usprawnić na wielu płaszczyznach. W stworzonej pracy zabrakło takich dodatkowych funkcji jak możliwość wyszukiwania lotów również z pobliskich lotnisk od aktualnie wybranego czy też wprowadzenia wersji w języku polskim. Część serwerowa mogłaby zostać poprawiona w zakresie "czystości kodu" oraz poprawnego nazewnictwa jej elementów. Istotne znaczenie dla podniesienia atrakcyjności aplikacji miałoby 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 podczas studiów inżynierskich. Praca nad nią zaowocowała pogłębieniem technicznego sposobu myślenia oraz zrozumieniem wyzwań XXI wieku stawianym inżynierom informatykom.

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