Przypadki użycia i diagramy UML

April 2020

1 Scenariusze przypadków użycia

1.1

- **Nazwa PU: createAdminAccount**
- **Cel**: Dodanie nowego konta administratora.
- **Warunki poczatkowe**: Uruchomienie strony internetowej w architekturze typu klient-serwer. Pierwsze konto jest domyślnie wbudowane w system.
- **Warunki końcowe**: Wprowadzenie danych administratora o unikalnym loginie i haśle w ramach użytkowników.
- **Scenariusz**:
  1. Należy podać dane administratora: login, hasło, imie, nazwisko.
  2. Należy sprawdzić, czy wprowadzane dane sa unikalne za pomoca wywołania PU: isDataCorrect, przekazujac login i hasło (login i hasło musza być unikalne). Jeśli zachodzi konflikt wprowadzonych danych, należy zakończyć przypadek użycia, w przeciwnym razie należy zapisać dane.

1.2

- **Nazwa PU: createClientAccount**
- **Cel**: Dodanie nowego konta klienta.
- **Warunki poczatkowe**: Uruchomienie strony internetowej w architekturze typu klient-serwer.
- **Warunki końcowe**: Wprowadzenie danych klienta o unikalnym loginie i haśle w ramach użytkowników.
- **Scenariusz**:
  1. Należy podać dane klienta: login, hasło, imie, nazwisko.
  2. Należy sprawdzić, czy wprowadzane dane sa unikalne za pomoca wywołania PU: isDataCorrect, przekazujac login i hasło (login i hasło musza być

unikalne). Jeśli zachodzi konflikt wprowadzonych danych, należy zakończyć przypadek użycia, w przeciwnym razie należy zapisać dane.
3. Należy przypisać do klienta poczatkowo pusta liste posiadanych biletów.

1.3

- **Nazwa PU: isDataCorrect**
- **Cel**: Sprawdzenie poprawności danych nowego administratora .
- **Warunki poczatkowe**: Uruchomienie z PU createAdminAccount lub createClientAccount.
- **Warunki końcowe**: Zwraca wynik, określajacy, czy podane date sa unikatowe (poprawne) w obrebie użytkowników.
- **Scenariusz**:
  1. Porównuje loginy i hasła pozostałych użytkowników z podanymi.
  2. W przypadku znalezienia użytkownika o takim samym loginie lub haśle PU kończy przegladanie danych pozostałych użytkowników i zwraca informacje o niepoprawności danych.
  3. W przeciwnym przypadku, po przejrzeniu danych, zwracana jest informacja o poprawności danych.

1.4

- **Nazwa PU: addFlight**
- **Cel**: Dodanie nowego lotu.
- **Warunki poczatkowe**: Uruchomienie strony internetowej w architekturze typu klient-serwer.
- **Warunki końcowe**: Wprowadzenie danych dt. lotu o unikalnym id.
- **Scenariusz**:
  1. Należy podać atrybuty lotu: id, miejsce i czas odlotu, miejsce i czas przylotu, cena, liczba miejsc w samolocie.
  2. Należy wywołać PU existsFlight - należy sprawdzić, czy lot o podanym id już istnieje. Jeśli istnieje to należy wywołać PU isTimeCorrect - sprawdzenie czy czas jest w poprawnej formie. Jeśli czas jest poprawny to należy dodać lot do bazy danych. Jeśli któryś z przypadków jest niepoprawny to należy zakończyć PU.

1.5

- **Nazwa PU: existsFlight**
- **Cel**: Sprawdzenie unikatowości id nowego lotu.

- **Warunki poczatkowe**: Uruchomienie z PU addFlight, removeFlight, buyTicket lub cancelTicket.
- **Warunki końcowe**: Zwraca wynik, określajacy, czy podane id istnieje w obrebie lotów.
- **Scenariusz**:
  1. Porównuje id pozostałych lotów z podanym.
  2. W przypadku znalezienia lotu o takim samym id PU kończy przegladanie danych pozostałych użytkowników i zwraca informacje o niepoprawności danych. W przeciwnym przypadku, po przejrzeniu danych, zwracana jest informacja o poprawności danych.

1.6

- **Nazwa PU: isTimeCorrect**
- **Cel**: Sprawdzenie poprawności czasu odlotu i przylotu nowego lotu.
- **Warunki poczatkowe**: Uruchomienie z PU addFlight.
- **Warunki końcowe**: Zwraca wynik, określajacy, czy czas przylotu nastepuje po odlocie.
- **Scenariusz**:
  1. Porównanie czasu przylotu i odlotu.
  2. Jeśli czas przylotu jest wiekszy od odlotu to zwracana jest informacja o poprawności, w przeciwnym przypadku zwracana jest informacja o niepoprawności danych.

1.7

- **Nazwa PU: removeFlight**
- **Cel**: Usuniecie lotu o podanym id.
- **Warunki poczatkowe**: Uruchomienie strony internetowej w architekturze typu klient-serwer.
- **Warunki końcowe**: Zniknecie danych dt. lotu o podanym id z bazy danych.
- **Scenariusz**:
  1. Należy sprawdzić czy dany lot wogóle istenieje - wywołanie PU exists-Flight.
  2. Jeśli lot istnieje to go usuwamy z bazy danych, jeśli nie to PU kończy działanie.

1.8

- **Nazwa PU: buyTicket**
- **Cel**: Zakupienie biletu na dany lot przez klienta.
- **Warunki poczatkowe**: Uruchomienie strony internetowej w architekturze typu klient-serwer.
- **Warunki końcowe**: Utworzenie i dodanie biletu do listy biletów klienta oraz zwiekszenie liczby zarezerwowanych miejsc.
- **Scenariusz**:
  1. Należy sprawdzić czy dany lot w ogóle istenieje - wywołanie PU existsFlight.
  2. Jeśli lot nie istnieje to to PU kończy działanie. Jeśli istenieje to kontynuujemy działanie.
  3. Sprawdzamy czy sa dostepne miejsca wywołujac PU isFreeSeats. Jeśli sa to generujemy bilet za pomoca PU createTicket, dodajemy go do biletów klienta i zwiekszamy liczbe zarezerwowanych miejsc. Jeśli nie to nie możemy kupić biletu, wiec PU kończy działanie niepowodzeniem.

1.9

- **Nazwa PU: createTicket**
- **Cel**: Dodanie nowego biletu.
- **Warunki poczatkowe**: Uruchomienie z PU buyTicket.
- **Warunki końcowe**: Wprowadzenie danych dt. nowego biletu o unikalnym id do bazy danych.
- **Scenariusz**:
  1. Należy podać atrybuty biletu: id, id lotu i login klienta.
  2. Należy wywołać PU existsTicket. Należy sprawdzić, czy bilet o podanym id już istnieje. Jeśli tak, należy zakończyć PU. Jeśli nie to należy dodać bilet do bazy danych.

1.10

- **Nazwa PU: isFreeSeats**
- **Cel**: Sprawdzenie czy sa wolne miejsca w danym istniejacym locie.
- **Warunki poczatkowe**: Uruchomienie z PU buyTicket.
- **Warunki końcowe**: Zwraca wynik, określajacy, czy sa wolne miejsca w samolocie.

- **Scenariusz**:
  1. Jeśli liczba miejsc zarezerwowanych jest mniejsza od liczby wszystkich miejsc to zwracamy informacje o istenieeniu miejsca, jeśli jest wieksza to zwracana jest informacja o braku miejsca.

1.11

- **Nazwa PU: cancelTicket**
- **Cel**: Anulowanie biletu na dany lot przez klienta.
- **Warunki poczatkowe**: Uruchomienie strony internetowej w architekturze typu klient-serwer.
- **Warunki końcowe**: Usuniecie biletu i zmniejszenie liczby zarezerwowanych miejsc na dany lot.
- **Scenariusz**:
  1. Należy sprawdzić czy dany lot w ogóle istenieje - wywołanie PU existsFlight.
  2. Należy sprawdzić czy dany bilet w ogóle istenieje - wywołanie PU existsTicket.
  3. Jeśli lot lub bilet nie istnieje to PU kończy działanie niepowodzeniem. W przeciwnym przypadku zostaje zmniejszona liczba zarezerwowanych miejsc oraz bilet zostaje usuniety z listy biletów klienta.

1.12

- **Nazwa PU: existsTicket**
- **Cel**: Sprawdzenie czy bilet o zadanym id istnieje.
- **Warunki poczatkowe**: Uruchomienie z PU buyTicket lub cancelTicket.
- **Warunki końcowe**: Zwraca wynik, określajacy, czy podane id istnieje w obrebie biletów.
- **Scenariusz**:
  1. Porównuje id pozostałych lotów z podanym.
  2. W przypadku znalezienia lotu o takim samym id PU kończy przegladanie danych pozostałych użytkowników i zwraca informacje o niepoprawności danych. W przeciwnym przypadku, po przejrzeniu danych, zwracana jest informacja o poprawności danych.

1.13 Dodatek

Nie opisałem tutaj logowania i wylogowywania sie użytkowników(przy logowaniu wywolujemy PU User Authentication dla sprawdzenia poprawnosci danych logowania). Nie ujałem też czesci funkcjonalności z pierwszego zadania, ponieważ

jeszcze nie wiem czy bede je implementował ze wzgledu na rozmiar programu (m.in. usuwanie konta - analogicznie do reszty wywołujemy PU existsAccount i usuwamy).

2 Diagram przypadków użycia