

Baza iNaturalist — występowanie gatunków zagrożonych  
na danym obszarze  
projekt z Pracowni informatycznej

Natalia Okopna  
nr albumu: 123454

prowadzący: dr hab. Wojciech Jakubowski

Styczeń 2022

# Spis treści

<b>1 Specyfikacja</b>	<b>3</b>
1.1 Wstępne założenia . . . . .	3
1.1.1 Cel aplikacji — sposób działania . . . . .	3
1.1.2 Technologia wykonania aplikacji . . . . .	3
1.1.3 Warunki techniczne . . . . .	3
1.2 Kolejne założenia oraz dane wejściowe i wyjściowe . . . . .	3
1.2.1 Dane wejściowe . . . . .	3
1.2.2 Dane wyjściowe . . . . .	4
1.3 Uzupełnienie danych wejściowych i wyjściowych oraz zapis informacji na dysku . . . . .	4
1.4 Końcowa postać specyfikacji . . . . .	4
1.4.1 Cel aplikacji . . . . .	4
1.4.2 Dane wejściowe . . . . .	4
1.4.3 Dane wyjściowe . . . . .	5
1.4.4 Technologia wykonania aplikacji . . . . .	5
1.4.5 Warunki techniczne . . . . .	5
<b>2 Projekt interface'u</b>	<b>6</b>
2.1 Schemat blokowy interface'u oraz jednego z modułów . . . . .	6
2.2 Frontend . . . . .	8
2.2.1 Pierwsza wersja . . . . .	8
2.2.2 Menu . . . . .	10
2.2.3 Okno wprowadzania obszaru . . . . .	11
2.2.4 Okna informacyjne . . . . .	13
2.2.5 Wczytywanie obserwacji . . . . .	13
2.2.6 Wczytywanie obserwacji z dysku . . . . .	14
2.2.7 Generowanie obserwacji z bazy . . . . .	15
2.2.8 Wizualizacja danych . . . . .	19
2.3 Backend . . . . .	20
2.3.1 Wprowadzanie nowego obszaru . . . . .	20
2.3.2 Wprowadzanie nazwy/taksonu . . . . .	22
2.3.3 Zapytanie do bazy . . . . .	23
2.3.4 Obserwacja w obszarze . . . . .	23
2.3.5 Tworzenie mapy . . . . .	23
2.3.6 Umieszczanie obszaru na mapie . . . . .	25
2.3.7 Umieszczanie obserwacji na mapie . . . . .	26
2.3.8 Znacznik . . . . .	26
2.3.9 Wyświetlanie obserwacji . . . . .	27
2.4 Zapisywanie obszarów i obserwacji . . . . .	28
2.4.1 Postać pliku json . . . . .	28

# 1 Specyfikacja

## 1.1 Wstępne założenia

Pierwszym krokiem w tworzeniu aplikacji było określenie podstawowych celów i założeń aplikacji. Pierwsza wersja wyglądała następująco:

### 1.1.1 Cel aplikacji — sposób działania

Stworzona zostanie aplikacja, której głównym zadaniem będzie prezentacja na danym obszarze obserwacji gatunków zagrożonych z określonej rodziny. Dane obserwacji będą pobierane w czasie rzeczywistym z bazy internetowej iNaturalist dostępnej pod adresem internetowym [www.inaturalist.org](http://www.inaturalist.org). Obszarami dostępnymi do zaznaczenia będą: kraje — do wyboru z listy dostępnych. Obserwacje występuowania będą obejmowały rzad *pieczarkowce*.

### 1.1.2 Technologia wykonania aplikacji

Kod programu będzie napisany w języku Python 3.8 z użyciem m. in. biblioteki *pyinaturalist*.

### 1.1.3 Warunki techniczne

Aplikacja będzie możliwa do uruchomienia na innych komputerach z zainstalowanym systemem *Windows*. Do użytkowania aplikacji wymagane jest połączenie internetowe.

## 1.2 Kolejne założenia oraz dane wejściowe i wyjściowe

Kolejnym założeniem do technologii wykonania aplikacji było wykorzystanie biblioteki *geopandas* — biblioteki służącej do tworzenia map. Jednak w dalszej części sprawozdania okaże się, że zostanie wykorzystane inne rozwiązanie tego zagadnienia.

Następnie stworzona została pierwsza wersja danych wejściowych i wyjściowych dla aplikacji.

### 1.2.1 Dane wejściowe

Użytkownik podaje w wyznaczonym miejscu interfejsu graficznego dane wejściowe w określony poniżej sposób.

- WSPÓŁRZĘDNE GEOGRAFICZNE OBSZARU

Użytkownik podaje minimum 3 pary współrzędnych geograficznych określających obszar, na którym będą wyświetlane obserwacje gatunku zagrozonego. Wprowadzając dane, użytkownik powinien pamiętać o następujących zasadach:

1. każda para współrzędnych ma być oddzielona przecinkiem i spacją
2. długość i szerokość każdej współrzędnej ma być oddzielona spacją,
3. współrzędne mają być wyrażone w stopniach z dokładnością do części milionowej — w przypadku podania współrzędnej z mniejszą dokładnością, aplikacja domyślnie dopisuje zera,
4. części dziesiętne współrzędnych mają być poprzedzone kropką,
5. ujemne dane mają być poprzedzone minusem,
6. zakres długości geograficznej to  $(-180.000000, 180.000000)$ ,
7. zakres szerokości geograficznej to  $(-90.000000, 90.000000)$ .

Przykładowe dane wejściowe współrzędnych użytkownika:

$-12.34565 \ 1.23456, -12.34500 \ 1.23450, -12.34300 \ 1.23440$ .

- **ZAKRES CZASU OBSERWACJI**

Użytkownik podaje datę początkową oraz datę końcową okresu, dla którego będą wyświetlane obserwacje gatunku zagrożonego. Daty podaje w formacie *DD.MM.RRRR*, gdzie DD oznacza dzień, MM — miesiąc, RRRR — rok.

### **1.2.2 Dane wyjściowe**

Po zatwierdzeniu danych wejściowych, zostają pobierane dane z bazy iNaturalist i są zapisywane do pliku na komputerze użytkownika. Następnie program wyświetla wyznaczony przez użytkownika obszar wraz z punktami obserwacji gatunku zagrożonego. Obok mapy pojawia się informacja o liczbie obserwacji oraz lista z danymi każdej obserwacji takimi jak: data obserwacji, dokładny gatunek, dane geograficzne, zdjęcie.

## **1.3 Uzupełnienie danych wejściowych i wyjściowych oraz zapis informacji na dysku**

Następnie została dodana dana wyjściowa nazwy obszaru — mogą być to nazwy zawierające spację i znaki polskie, bez znaków specjalnych oraz do danych wyjściowych zostały dodane informacje o zapisie na dysku: po zatwierdzeniu przez użytkownika wejściowych danych geograficznych, na dysku w osobnym folderze *obszary* zostają zapisane dane geograficzne w pliku tekstowym pod nazwą nazwy obszaru wprowadzonej przez użytkownika w formie zgodnej z wprowadzonymi danymi przez użytkownika, opisanej w poprzednim rozdziale. Następnie zostają pobrane dane z bazy iNaturalist i są zapisywane pod nazwą nazwy obszaru do pliku *log* na komputerze użytkownika. Pobierane są również zdjęcia z rozszerzeniem *jpg* obserwacji pod nazwami domyślnymi, takimi jak w bazie internetowej, do osobnego folderu *zdj.*

## **1.4 Końcowa postać specyfikacji**

Po stworzeniu projektu interface'u, należało zmienić w specyfikacji użyte biblioteki, format wprowadzanych danych przez użytkownika oraz na przykład wyszukiwane obiekty i zapisywane na dysku informacje.

### **1.4.1 Cel aplikacji**

Stworzona zostanie aplikacja, której głównym zadaniem będzie prezentacja gatunków zagrożonych na danym obszarze. Dane obserwacji będą pobierane w czasie rzeczywistym z bazy internetowej iNaturalist dostępnej pod adresem internetowym [www.inaturalist.org](http://www.inaturalist.org). Dostępne będzie również generowanie obserwacji z zapisanych wcześniej danych historycznych. Obszary obserwacji będą wyznaczone przez użytkownika. Obserwacje występujące będą obejmować dowolne gatunki określone nazwą systematyczną bądź taksonem.

### **1.4.2 Dane wejściowe**

Użytkownik podaje w wyznaczonym miejscu interfejsu graficznego dane wejściowe w określony poniżej sposób.

- **WSPÓŁRZĘDNE GEOGRAFICZNE I NAZWA OBSZARU**

Użytkownik podaje: nazwę obszaru — mogą być to nazwy zawierające spację i znaki polskie, bez znaków specjalnych oraz minimum 3 pary współrzędnych geograficznych określających obszar, na którym będą wyświetlane obserwacje gatunku zagrożonego. Wprowadzając dane, użytkownik powinien pamiętać o następujących zasadach:

1. każda para współrzędnych ma być zawarta w okrągłych nawiasach,
2. każda para współrzędnych ma być oddzielona przecinkiem i ewentualną spacją,

3. długość i szerokość każdej współrzędnej ma być oddzielona przecinkiem i ewentualną spacją,
4. współrzędne mają być wyrażone w stopniach,
5. dokładność jest zaokrąglana do 16-tego miejsca po przecinku,
6. części dziesiętne współrzędnych mają być poprzedzone kropką,
7. ujemne dane mają być poprzedzone minusem,
8. zakres długości geograficznej to  $(-180.00000000000000, 180.00000000000000)$ ,
9. zakres szerokości geograficznej to  $(-90.00000000000000, 90.00000000000000)$ .

Przykładowe dane wejściowe współrzędnych użytkownika:

$(-12.34565, 1.23456), (-12.34500, 1.23450), (-12.34300, 1.23440)$

- **ZAKRES CZASU OBSERWACJI**

Użytkownik podaje datę początkową oraz datę końcową okresu, dla którego będą wyświetlane obserwacje gatunku zagrożonego. Daty podaje w formacie *DD-MM-RRRR*, gdzie DD oznacza dzień, MM — miesiąc, RRRR — rok.

- **TAKSON/NAZWA**

Użytkownik zaznacza, czy podaje nazwę taksonomiczną czy nazwę systematyczną oraz wpisuje poszukiwany takson/nazwę.

#### **1.4.3 Dane wyjściowe**

Po zatwierdzeniu przez użytkownika wejściowych danych geograficznych oraz nazwy obszaru, na dysku do pliku typu *json* zostaje dosiany obszar. Z zapisanych w bazie obszarów, użytkownik wybiera jeden obszar oraz podaje datę i nazwę lub takson. Po zatwierdzeniu otrzymuje obserwacje pobrane dane z bazy iNaturalist. Program wyświetla wyznaczony przez użytkownika obszar wraz z punktami obserwacji gatunku zagrożonego. Obok mapy pojawia się informacja o liczbie obserwacji oraz lista z danymi każdej obserwacji takimi jak: data obserwacji, dokładny gatunek, dane geograficzne, zdjęcie. Jeżeli użytkownik wyrazi zgodę, obserwacje są zapisywane dopisywane pliku typu *json* na komputerze użytkownika. Pobierane są również zdjęcia z rozszerzeniem *jpg* obserwacji pod nazwami domyślnymi, takimi jak w bazie internetowej, do osobnego folderu *zdj*. Dostępna będzie również możliwość wczytywania wcześniej zapisanych obserwacji.

#### **1.4.4 Technologia wykonania aplikacji**

Kod programu będzie napisany w języku Python 3.8 z użyciem m. in. biblioteki *pyinaturalist* służącej do obsługi danych z bazy iNaturalist oraz *shapely* i *folium* odpowiadające za obrazowanie współrzędnych geograficznych.

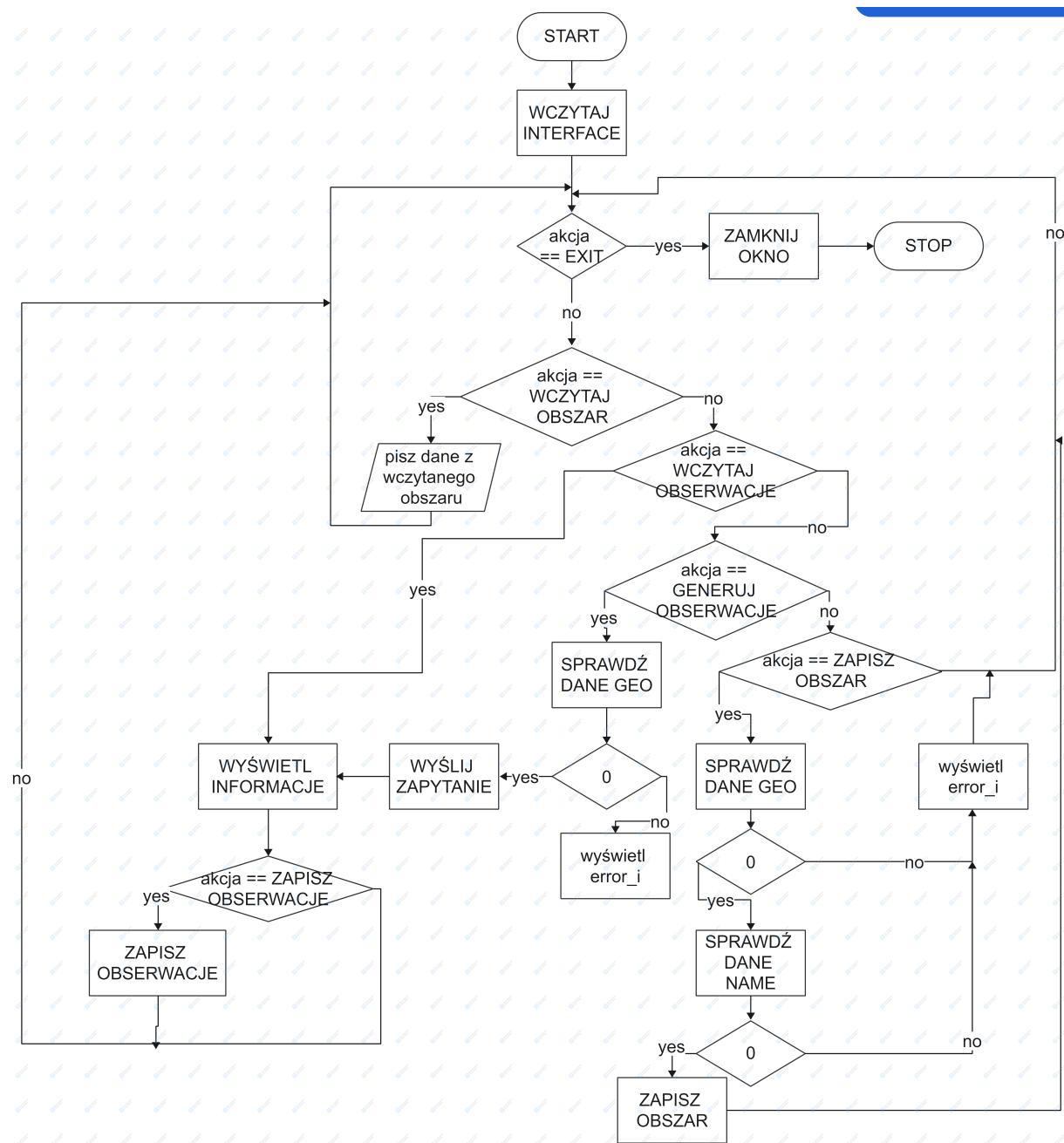
#### **1.4.5 Warunki techniczne**

Aplikacja będzie możliwa do uruchomienia na innych komputerach z zainstalowanym systemem *Windows*. Do użytkowania aplikacji wymagane jest połączenie internetowe.

## 2 Projekt interface'u

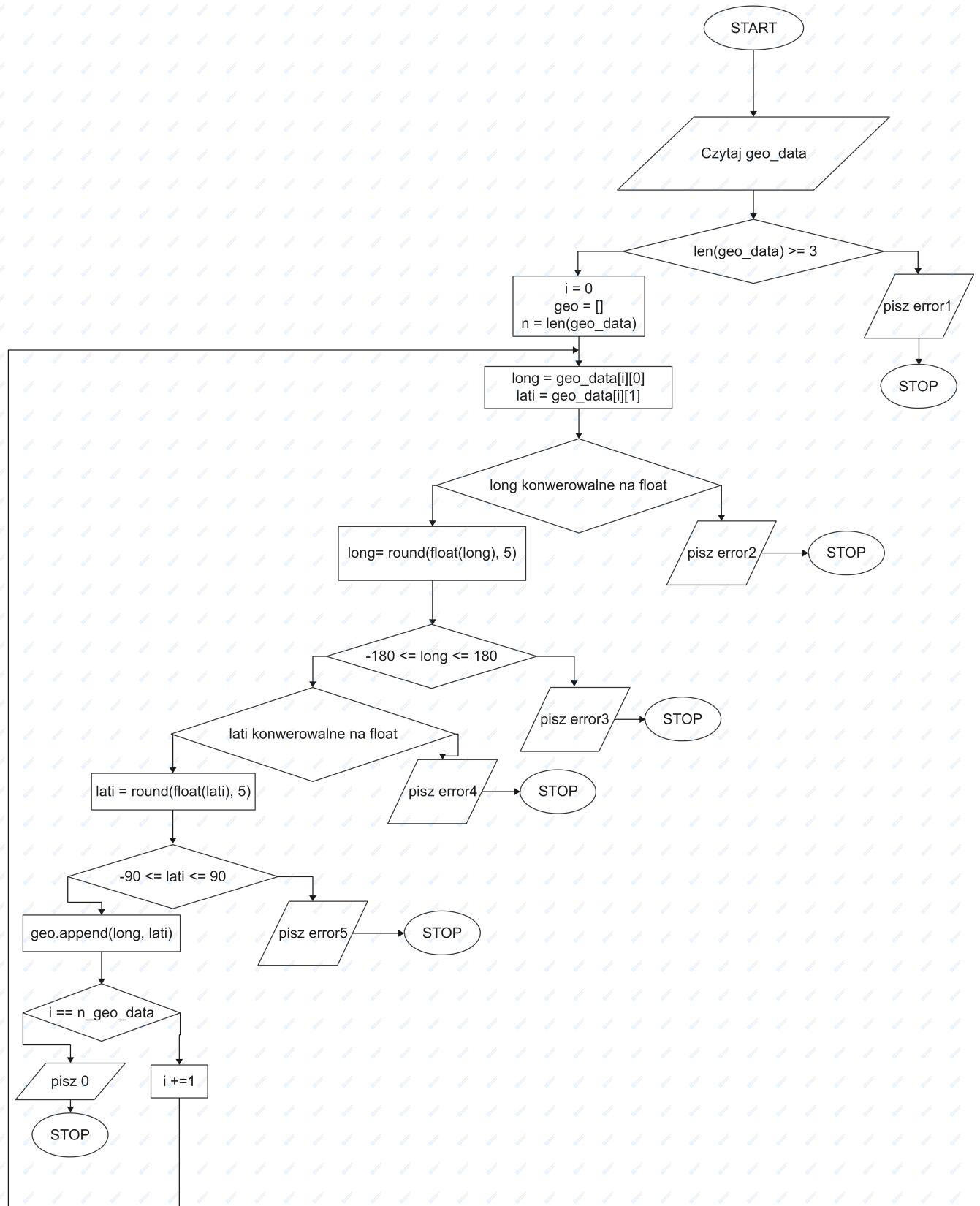
## 2.1 Schemat blokowy interface'u oraz jednego z modułów

Następnie został stworzony schemat blokowy interface'u oraz jednego z modułów. Już na tym etapie okazało się, że zaproponowane w specyfikacji rozwiązania muszą ulec zmianie. Dzieje się tak na przykład z postacią wprowadzonych przez użytkownika danych geograficznych.



Rysunek: Schemat blokowy interface'u

## SPRAWDŹ DANE GEO



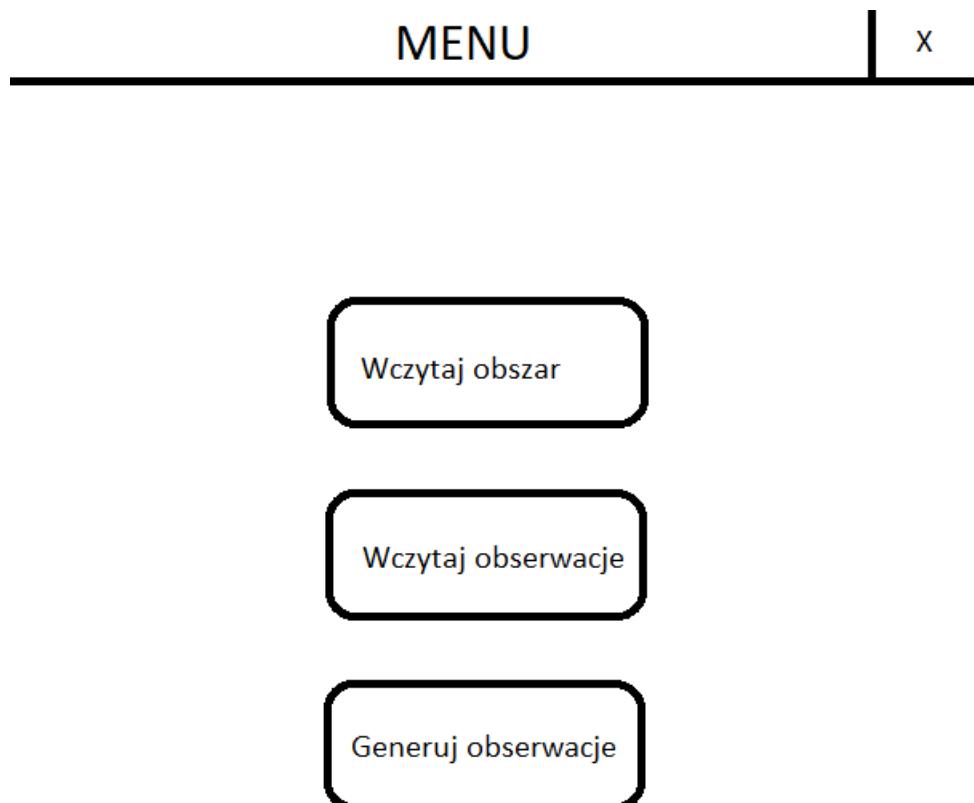
Rysunek: Schemat blokowy sprawdzający dane geograficzne

Szybko okazało się, że schemat blokowy nie jest wymagany, a zależy nam na dobrym zaplanowaniu front- i backendu.

## 2.2 Frontend

### 2.2.1 Pierwsza wersja

Pierwszym pomysłem było stworzenie menu, z którego dostępne będą opcje wczytywania obszaru i obserwacji



Rysunek: Menu

## NOWY OBSZAR

X

Dane geograficzne

Sprawdź

Data od - do

Nazwa obszaru

Wczytaj  
obszar z bazy

Zapisz obszar  
do bazy

Generuj  
obserwacje

Tutaj mozna sprawdzic  
wygenerowany obszar

Rysunek: Menu

## GENERUJ Z WCZYTANEGO OBSZARU

X

Dane geograficzne

(12.12345, 67.89087), (.....

Data od - do

Nazwa obszaru

park\_krajobrazowy\_gor\_stolowych

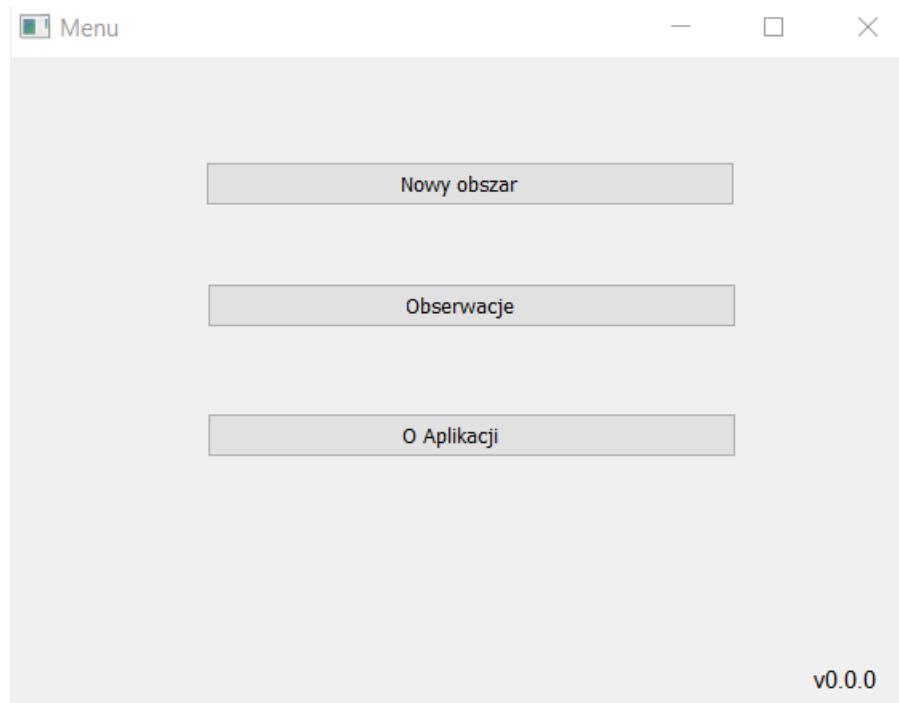
Generuj  
obserwacje

Mapa z wczytanym  
obszarem

Jednak interfejs skonstruowany w ten sposób był nieintuicyjny. Najbardziej rzucającym się w oczy problemem była możliwość generowania obserwacji w kilku miejscach. W związku z tym stworzony został nowy projekt interfejs'u.

### 2.2.2 Menu

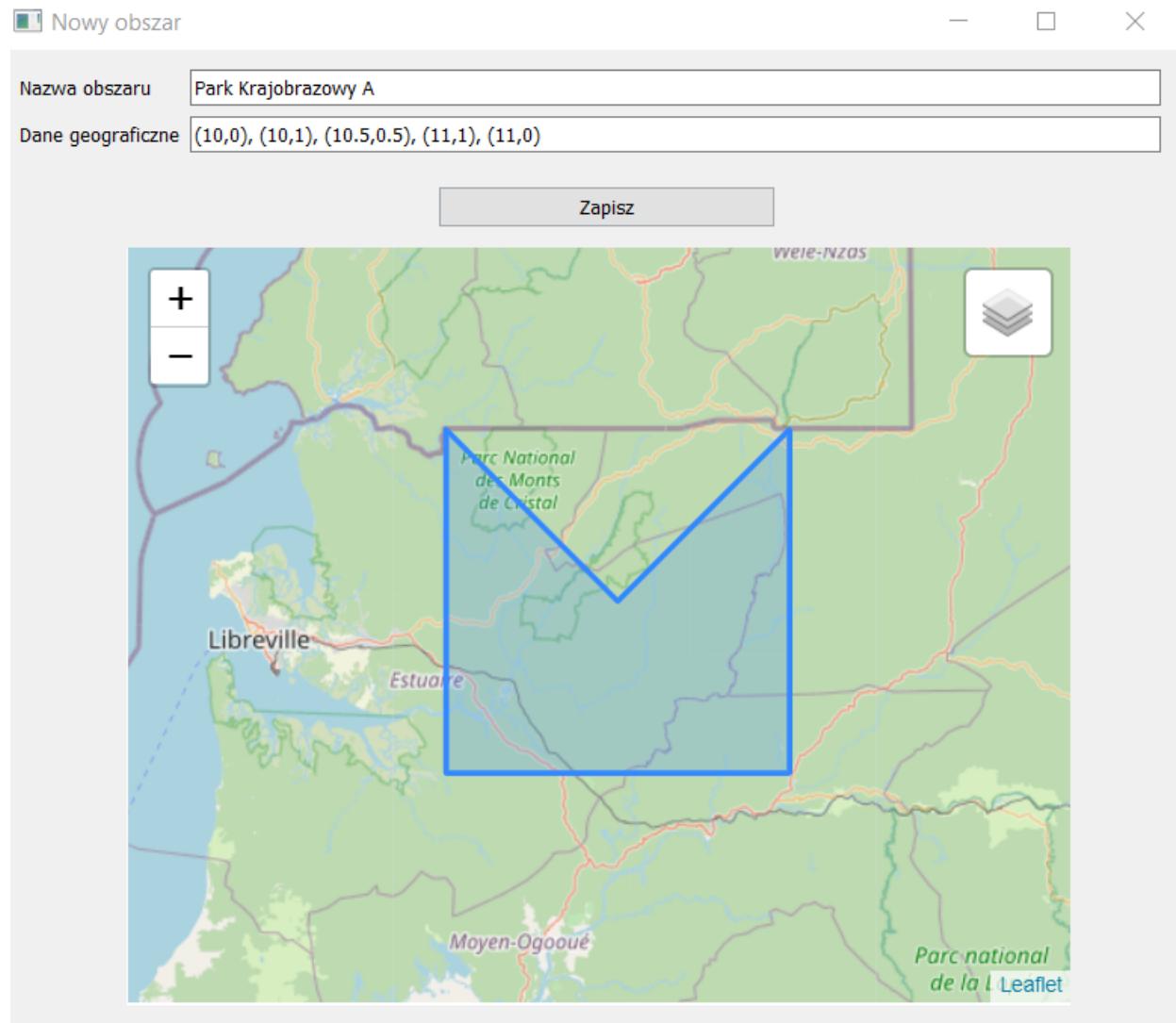
Po otwarciu aplikacji użytkownik dostaje kilka możliwości do wyboru. Zapis nowego obszaru, wczytanie informacji oraz wyświetlenie informacji o aplikacji.



Rysunek 1: Menu

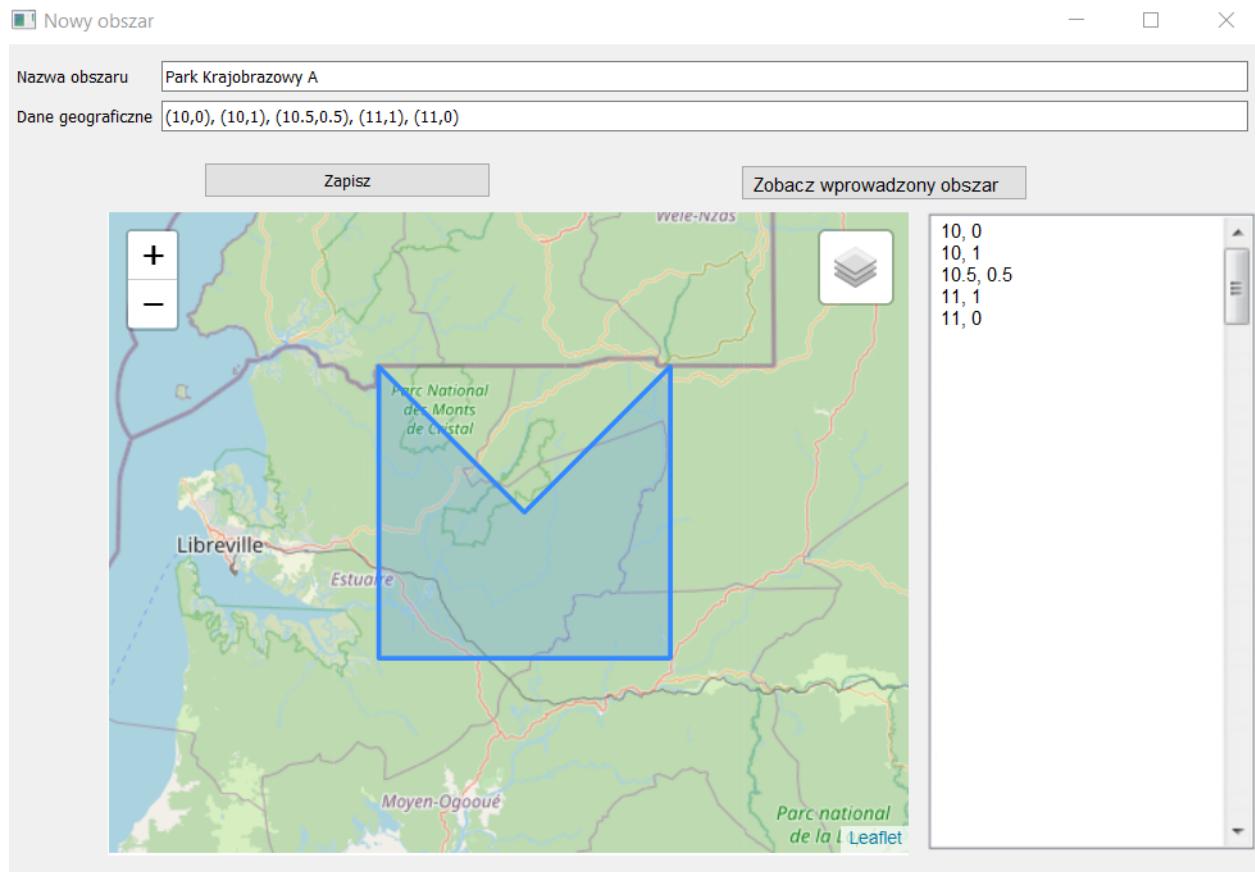
### 2.2.3 Okno wprowadzania obszaru

Po kliknięciu w menu głównym opcji *Nowy obszar* otwiera nam się następujące okno.



Rysunek 2: Okno 'Nowy obszar'

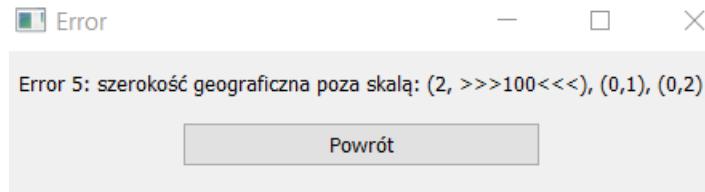
Rozważane było interaktywne generowanie obszaru, po wprowadzeniu każdego punktu. Jednak zostanie umieszczony przycisk *Zobacz wprowadzony obszar* oraz obszar przewijanej listy z wprowadzonymi punktami. Po wciśnięciu przycisku będzie generował się na mapie obszar oraz lista punktów lub okno błędu, takie jak po wciśnięciu przycisku *Zapisz*, omówione w dalszej części raportu.



Rysunek 3: Okno 'Nowy obszar z listą'

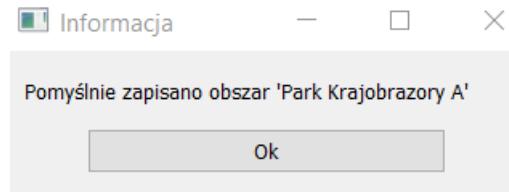
#### 2.2.4 Okna informacyjne

Po zatwierdzeniu obszaru rozpoczyna się proces walidacji oraz sprawdzenie, czy obszar jest spójny, omówione w następnym rozdziale. W razie błędu wyświetli się następujące okno.



Rysunek 4: Okno 'Error'

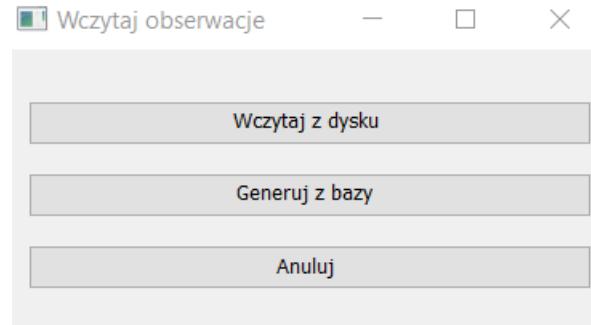
W przeciwnym razie obszar zostaje zapisany do bazy. Pojawia się okno z informacją.



Rysunek 5: Okno informujące o zapisaniu obrazu

#### 2.2.5 Wczytywanie obserwacji

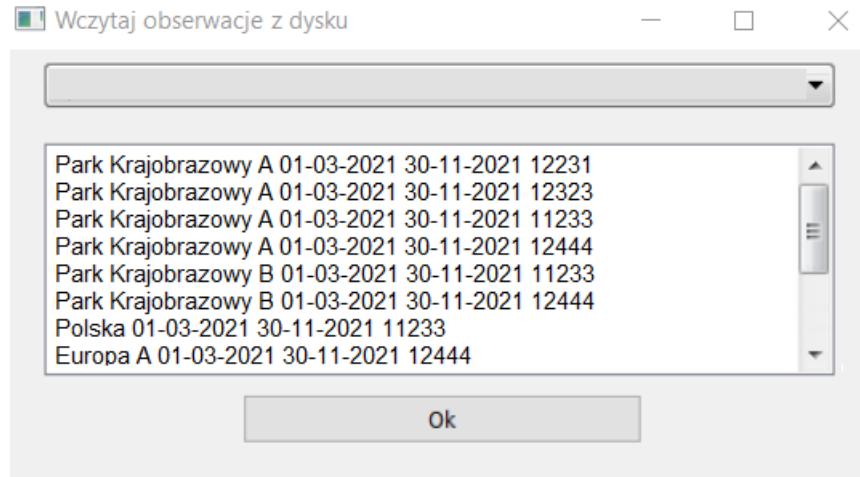
Po kliknięciu w menu głównym opcji *Obserwacje* wyświetla nam się następujące okno.



Rysunek 6: Wczytywanie obserwacji

### 2.2.6 Wczytywanie obserwacji z dysku

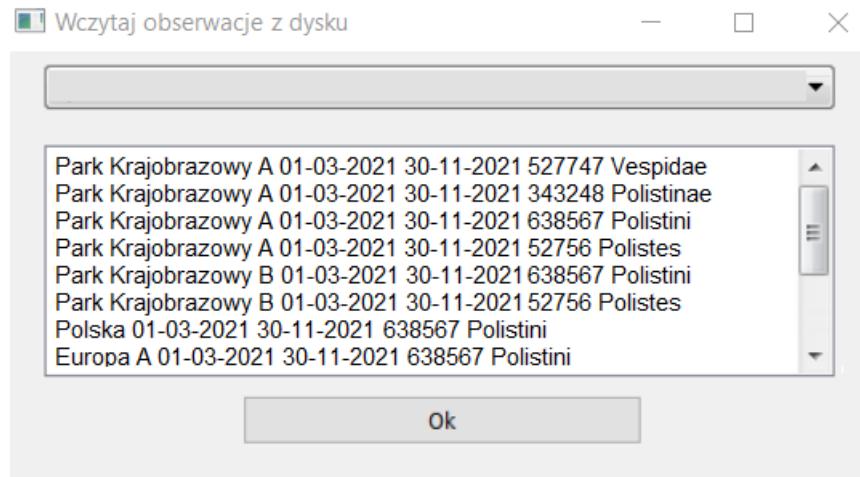
Po kliknięciu *Wczytaj z dysku* otwiera nam się okno *Wczytaj obserwacje z dysku*.



Rysunek 7: Wczytywanie obserwacji z dysku

Po kliknięciu *Ok* wyświetli się okno *Obserwacje* z wizualizacją danych omówioną w kolejnych punktach.

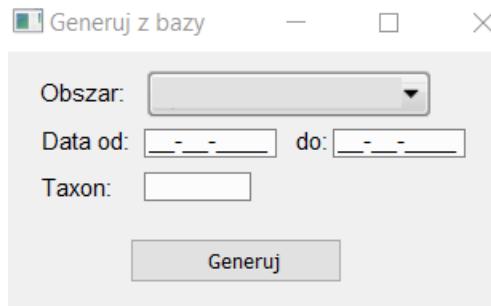
To okno zostało poprawione o wyświetlanie zarówno nazw taksonomicznych jak i systematycznych.



Rysunek 8: Wczytywanie obserwacji z dysku poprawione

### 2.2.7 Generowanie obserwacji z bazy

W pierwszej wersji po kliknięciu *Generuj z bazy* miało otwierać się następujące okno.

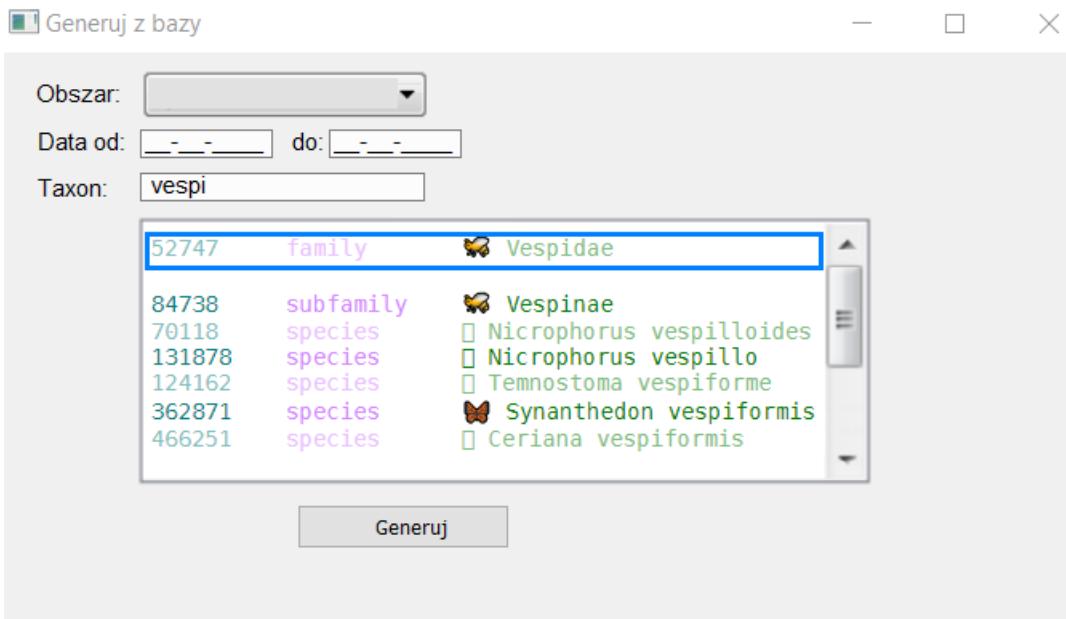


Rysunek 9: Generowanie z bazy

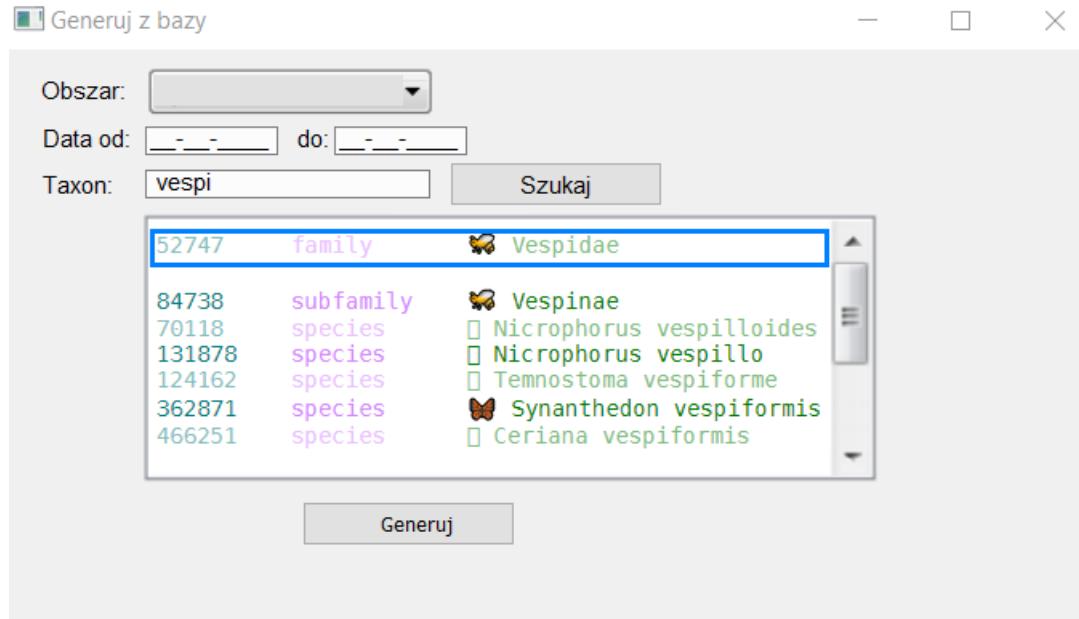
Możliwe błędy to: data spoza zakresu, zły takson, brak internetu. W razie błędów będą pojawiały się odpowiednie okna błędu. Po zatwierdzeniu danych i wcisnięciu przycisku *Generuj* zostaje wysyłane zapytanie do bazy i wygenerowanie obserwacji. Zostaje wyświetlane okno *Obserwacje* omówione w dalszej części.

Po konsultacji okazało się, że warto poprawić jego wygląd. Następna wersja tego okna była następująca.

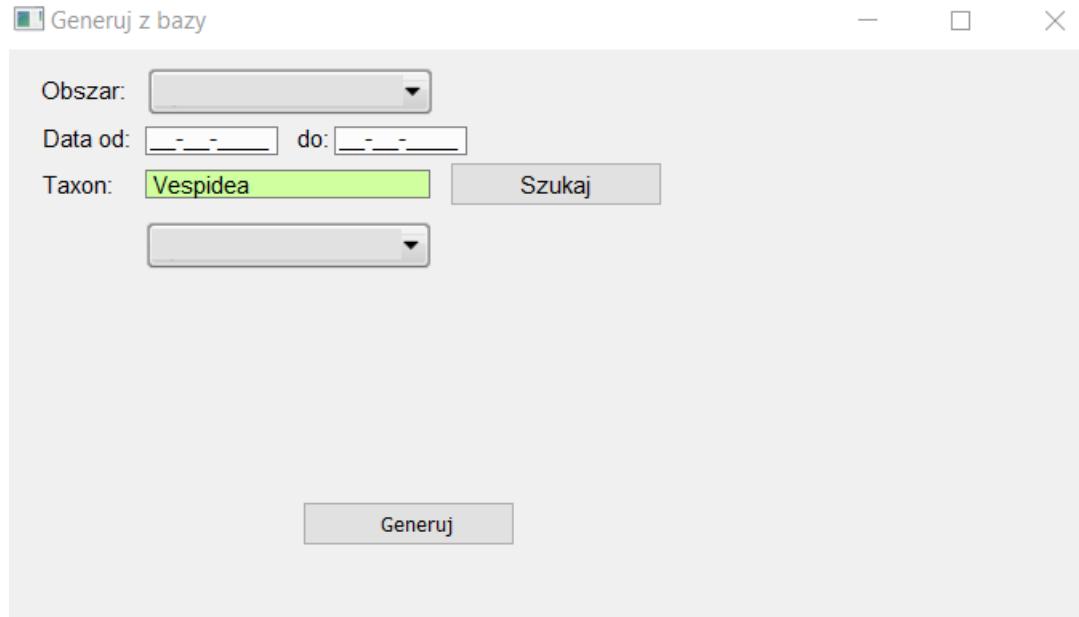
Użytkownik wpisując nazwę systematyczną może sprawdzić dostępne nazwy. Lista dostępnych nazw będzie generowana automatycznie.



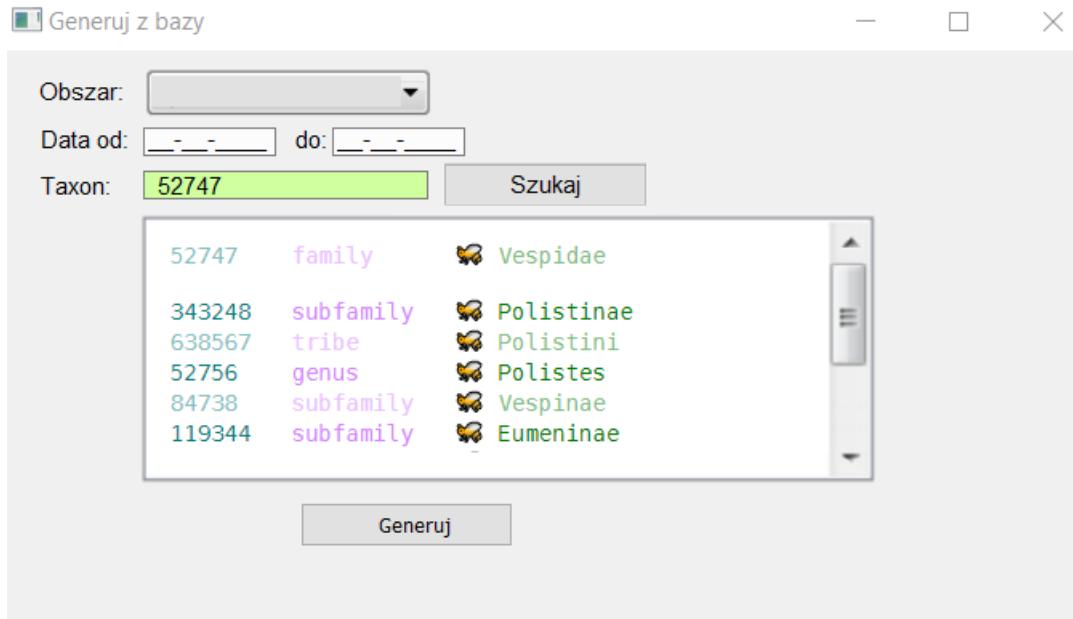
Bądź po wciśnięciu *Szukaj* — w zależności od tego, co się okaże lepsze, w trakcie tworzenia programu.



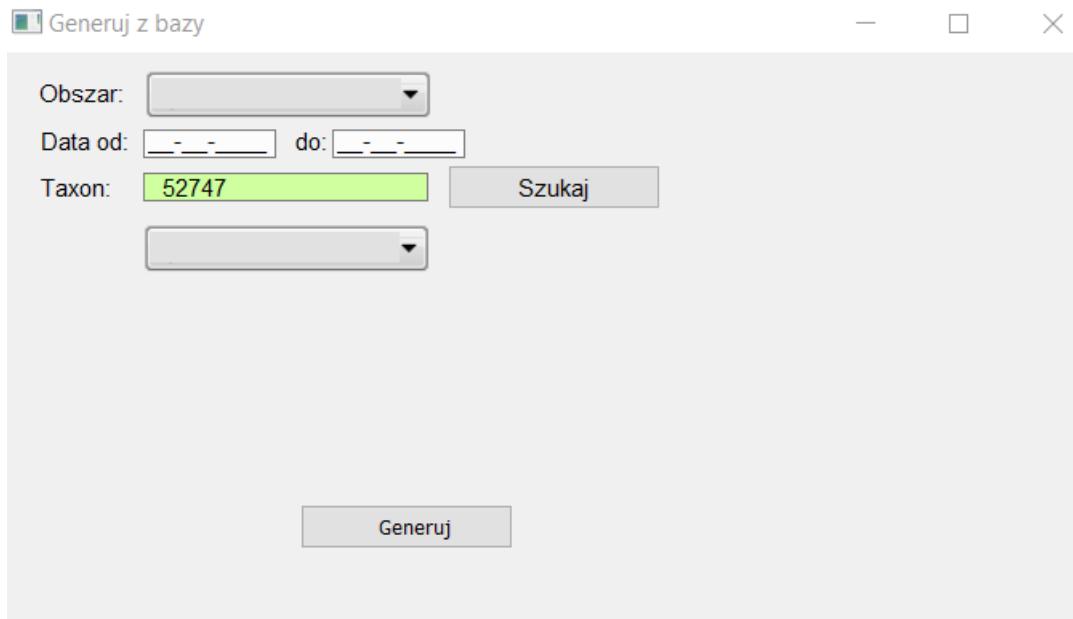
Po wybraniu lub wpisaniu odpowiedniej nazwy komórka podświetli się na zielono.



Dostępne będzie również wybranie taksonu. W tym przypadku użytkownik powinien podać poprawną nazwę taksonu. Na liście pojawi się szukana pozycja oraz jej dzieci takonomiczne.



Po wybraniu lub wpisaniu odpowiedniej pozycji, komórka podświetli się na zielono.



Jeśli przy wciśnięciu przycisku Generuj nazwa nie będzie podświetlona na zielono, pojawi się informacja w postaci okienka erroru.

Po kolejnej konsultacji okazało się, że bezpieczniej będzie stworzyć dodatkowe okno, w którym użytkownik deklaruje, czy będzie wpisywał takson, czy nazwę. Po wpisaniu taksonu, poniżej okna z wpisywaniem, pokaże się nazwa oraz inne ważne informacje do przypisanego taksonu.

Generuj z bazy

Obszar:

Data od:  do:

Takson  Nazwa

Takson/Nazwa:  Szukaj

52747	family	Vespidae
343248	subfamily	Polistinae
638567	tribe	Polistini
52756	genus	Polistes
84738	subfamily	Vespinae
119344	subfamily	Eumeninae

Generuj

## 2.2.8 Wizualizacja danych

Po wprowadzeniu obserwacji następuje wyświetlenie obserwacji. Poszczególne elementy okna są opisane w kolejnym rozdziale.

ID	Treszka ID	Nazwa	Obserwator	Ilustr.	Lokalizacja
30689463	78444	Species: <i>Silphium laciniatum</i> (Rocky Mountain Beesplant)	Aug 12, 2019	jcook	Johnston, IA, USA
30689506	47962	Species: <i>Rudbeckia triloba</i> (butterfly milkweed)	Aug 12, 2019	jcook	Johnston, IA, USA
30689603	60290	Species: <i>Verbena stricta</i> (verbena stricta)	Aug 12, 2019	jcook	Johnston, IA, USA
30689711	12121	Species: <i>Arenaria serpyllifolia</i> (butterfly milkweed)	Aug 12, 2019	jcook	Johnston, IA, USA
30689761	121260	Species: <i>Arenaria serpyllifolia</i> (butterfly milkweed)	Aug 12, 2019	jcook	Johnston, IA, USA
30689865	121268	Species: <i>Arenaria serpyllifolia</i> (butterfly milkweed)	Aug 12, 2019	jcook	Johnston, IA, USA
30689871	121270	Species: <i>Arenaria serpyllifolia</i> (butterfly milkweed)	Aug 12, 2019	jcook	Johnston, IA, USA
30689883	121276	Species: <i>Silphium laciniatum</i> (compass plant)	Aug 12, 2019	jcook	Johnston, IA, USA
30689910	121278	Species: <i>Silphium laciniatum</i> (compass plant)	Aug 12, 2019	jcook	Johnston, IA, USA

Rysunek 10: Obserwacje

Po wyjściu z okna *Obserwacje* pojawia się okno z zapytaniem, czy zapisać obserwację. O formie zapisu jest powiedziane w kolejnym rozdziale.

## 2.3 Backend

W odpowiedzi na potrzebę zaprogramowania obsługi obszarów oraz map, skorzystałem z biblioteki *shapely* oraz *folium*. Pierwsza z nich pozwala nie tylko na tworzenia obszarów, ale i na walidację danych, na przykład pod względem kryterium spójności. Druga biblioteka pozwala na tworzenie interaktywnych map. Dostępne są również przykłady użycia obu bibliotek do tworzenia takich obszarów jak poszczególne kraje, czy kontynenty. Nieoceniona w projekcie okazała się również biblioteka *pyinaturalist* pozwalająca na ściąganie danych z bazy iNaturalist. Dodatkową biblioteką, która będzie wprowadzona z drobnymi zmianami jest *ipypplot*. Biblioteka jest przeznaczona do użycia w notatnikach python'owskich, stąd potrzeba wprowadzenia niewielkich zmian w jej kodzie.

### 2.3.1 Wprowadzanie nowego obszaru

Aby wygenerować opcje na danym obszarze, taki obszar musi być wcześniej wprowadzony do bazy.

#### Tworzenie obszarów

Do tworzenia obszarów będziemy korzystać z biblioteki *shapely.geometry*. Tworzymy przykładowy obszar używając funkcji *shapely.geometry.Polygon*. Jako argument dajemy listę punktów danych geograficznych.

```
1 from shapely.geometry import Polygon
2
3 test = [(10,0), (10,1), (10.5,0.5), (11,1), (11,0)]
4 test = Polygon(test)
5 test
```

Listing 1: Przykładowy obszar

Otrzymujemy obszar:



Rysunek 11: Przykładowy obszar ‘test’

#### Obszar spójny

Sprawdzamy, czy wprowadzony przez użytkownika obszar jest poprawny za pomocą funkcji *shapely.geometry.Polygon.is\_valid*. Funkcja zwraca wartość logiczną *True*, gdy obszar jest spójny. Może więc być to obszar wklesły. Więcej można poczytać o tej funkcji w dokumentacji: <https://shapely.readthedocs.io/en/stable/manual.html#polygons>. Sprawdzamy poprawność stworzonego wcześniej obszaru Rysunek 11.

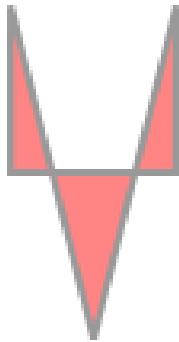
```
1 test.is_valid
2 #True
```

Listing 2: Obszar spójny

Tworzymy obszar, w którym linie między punktami nachodzą na siebie (obszar nie jest spójny).

```
1 test1 = [(0,0), (0,1), (0.5,-1), (1,1), (1,0)]
2 test1 = Polygon(test1)
3 test1
```

Listing 3: Obszar spójny



Rysunek 12: Obszar niespójny ‘test1’

Otrzymujemy figure: Sprawdzamy jego poprawność:

```
1 test1.is_valid
2 #False
```

Listing 4: Obszar niespójny

Funkcja *shapely.geometry.Polygon.is\_valid* zwraca wartość *False*. Czyli obszar nie jest poprawny.

### Walidacja danych

Aby korzystać z funkcji *shapely.Polygon* trzeba sprawdzić poprawność danych wprowadzonych przez użytkownika.

- Linia i punkt

W naszym programie obszar powinien zawierać minimum 3 punkty. Tworzymy funkcję *geo\_less\_than\_3*.

```
1 def geo_less_than_3(lista):
2     if len(lista) < 3:
3         return 1
4     return 0
```

Listing 5: geo\_less\_than\_3

Przykładowe wyjścia funkcji:

```
1 geo_less_than_3([(0, 0)])
2 #1
```

Listing 6: Za mało danych

```
1 geo_less_than_3([(0, 0), (0, 1)])
2 #1
```

Listing 7: Za mało danych

```
1 geo_less_than_3([(0, 0), (0, 1), (0, 2)])
2 #0
```

Listing 8: Minimalna liczba danych

- Inne znaki i zakres liczbowy. Sprawdzamy, czy dane wejściowe użytkownika to liczby i to liczby z zakresu szerokości i długości geograficznych świata. Tworzymy funkcję *geo\_float*, która zwraca numer błędu lub przekonwertowaną listę danych geograficznych.

```

1 def geo_float(lista):
2     lista_float = []
3     try:
4         for long, lati in lista:
5             try:
6                 long = round(float(long), 5)
7                 if long < -180 or long > 180:
8                     return 3
9             except:
10                 return 2
11             try:
12                 lati = round(float(lati), 5)
13                 if lati < -90 or lati > 90:
14                     return 5
15             except:
16                 return 4
17             lista_float.append((long, lati))
18         except:
19             return 0
20     return lista_float

```

Listing 9: geo\_float

Przykłady:  
wpisujemy błędne dane:

```

1 geo_float([('0', '0', '1'), ('0', '2')])
2 #0

```

Listing 10: za krótka krotka

```

1 geo_float([('1', '0', '4'), ('0', '1'), ('0', '2')])
2 #0

```

Listing 11: za długa krotka

```

1 geo_float([('0', 'a'), ('0', '1'), ('0', '2')])
2 #4

```

Listing 12: inne znaki

```

1 geo_float([('a', '0'), ('0', '1'), ('0', '2')])
2 #2

```

Listing 13: inne znaki

```

1 geo_float([('200', '0'), ('0', '1'), ('0', '2')])
2 #3

```

Listing 14: liczby poza skalą

```

1 geo_float([('2', '100'), ('0', '1'), ('0', '2')])
2 #5

```

Listing 15: liczby poza skalą

```

1 geo_float([('1', '0'), ('0', '1'), ('0', '2')])
2 #[(1.0, 0.0), (0.0, 1.0), (0.0, 2.0)]

```

Listing 16: poprawne dane

### 2.3.2 Wprowadzanie nazwy/taksonu

W programie za wyszukiwanie nazw i taksonów będzie odpowiedzialna funkcja `get_taxa` z biblioteki `pyinaturalist`.

### 2.3.3 Zapytanie do bazy

Zapytanie do bazy będzie kierowane poprzez funkcję `pyinaturalist.get_observation_species_counts`, która bierze jako argumenty takson, datę startową i końcową obserwacji oraz wartość 0-1 czy gatunek danej obserwacji ma być zagrożony w lokalizacji obserwacji oraz czy obserwacja jest zweryfikowana. Można też ustalić inne dodatkowe parametry o których więcej można poczytać w dokumentacji: [https://pyinaturalist.readthedocs.io/en/stable/modules/pyinaturalist.v1.observations.html#pyinaturalist.v1.observations.get\\_observation\\_species\\_counts](https://pyinaturalist.readthedocs.io/en/stable/modules/pyinaturalist.v1.observations.html#pyinaturalist.v1.observations.get_observation_species_counts). Otrzymujemy zestaw obserwacji w postaci słownika JSON.

### 2.3.4 Obserwacja w obszarze

Następnie należy sprawdzić, które obserwacje należą do danego. Tworzymy punkt przy użyciu funkcji `shapely.geometry.Point`.

```
1 from shapely.geometry import Point
2 p1 = Point(10.5, 0.2)
```

Listing 17: Dodanie obszaru

Następnie korzystamy z funkcji `shapely.geometry.Polygon.contains()`.

```
1 test.contains(p1)
2 #True
```

Listing 18: Dodanie obszaru

```
1 p2 = Point(10.5, 1.2)
2 test.contains(p2)
3 #False
```

Listing 19: Dodanie obszaru

Punkt `p1` należy do obszaru `test`, punkt `p2` nie należy.

Gdy mamy przefiltrowane obserwacje, zostaje wyświetlić je na mapie oraz w odpowiedni wizualnie sposób.

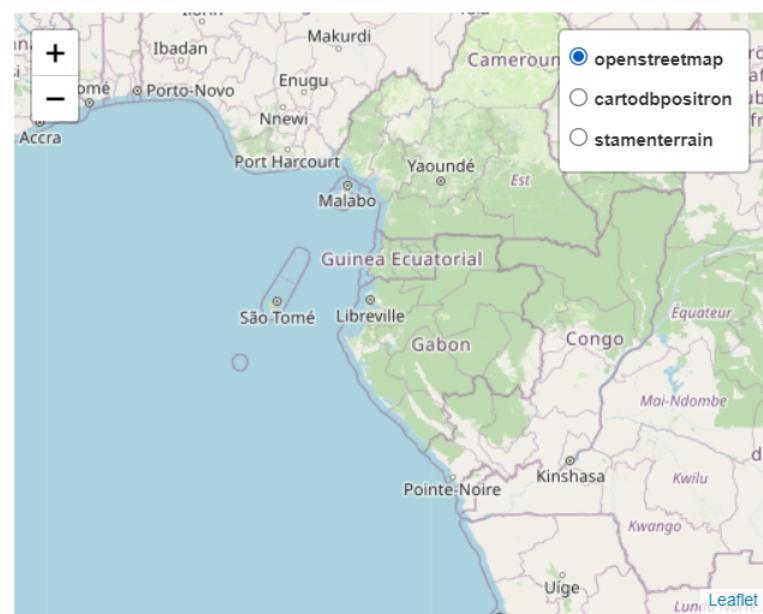
### 2.3.5 Tworzenie mapy

Aby umieścić obszary na mapie, skorzystamy z biblioteki `folium`: <https://python-visualization.github.io/folium/quickstart.html>.

Tworzymy obiekt `mapa` określając wymiary okna oraz widok. Domyślny tryb mapy to: `tiles = 'OpenStreetMap'`. W aplikacji będzie można zmienić tryb na '`cartodbpositron`' oraz '`Stamen Terrain`'.

```
1 import folium
2 mapa = folium.Map(width=500, height=400, location=[0,10], zoom_start=5)
3 folium.TileLayer('cartodbpositron').add_to(mapa)
4 folium.TileLayer('Stamen Terrain').add_to(mapa)
5 folium.LayerControl().add_to(mapa)
6 mapa
```

Listing 20: Mapa



Rysunek: Domyślny tryb



Rysunek: Tryb 'cartodbpositron'



Rysunek 13: Tryb 'Stamen Terrain'

### 2.3.6 Umieszczanie obszaru na mapie

Do mapy *mapa* dodajemy nasz obszar *test*.

```
1 import folium
2 mapa = folium.Map(width=500, height=400, location=[0,10], zoom_start=8)
3 folium.GeoJson(test).add_to(mapa) # dodanie obszaru
4 folium.TileLayer('cartodbpositron').add_to(mapa)
5 folium.TileLayer('Stamen Terrain').add_to(mapa)
6 folium.LayerControl().add_to(mapa)
7 mapa
```

Listing 21: Dodanie obszaru



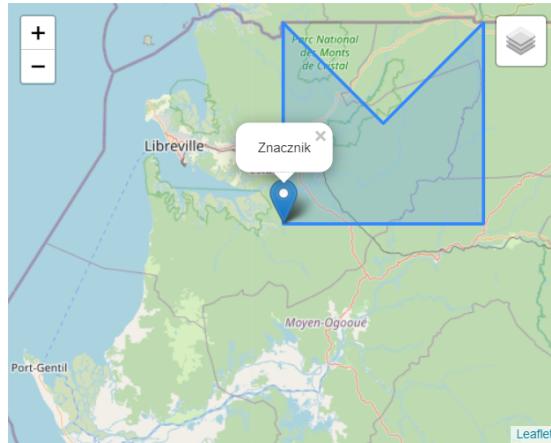
Rysunek 14: Mapa z obszarem

### 2.3.7 Umieszczanie obserwacji na mapie

Tworzymy obiekt punktu korzystając z funkcji *folium.Marker*.

```
1 folium.Marker([0, 10], popup="Znacznik").add_to(mapa)
2 mapa
```

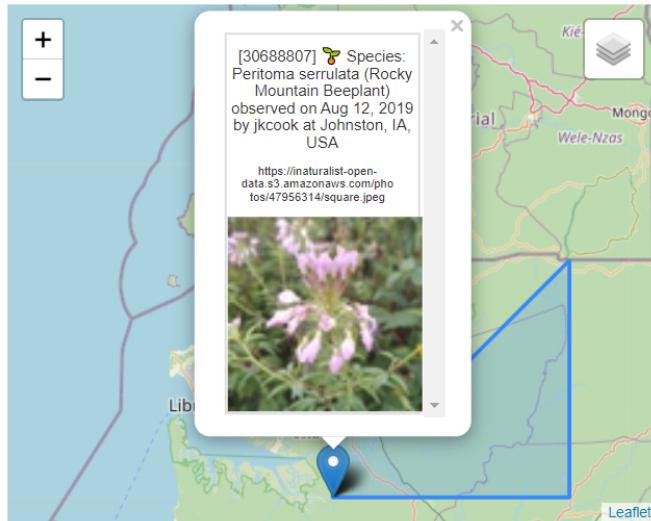
Listing 22: Dodanie obszaru



Rysunek 15: Mapa ze znacznikiem

### 2.3.8 Znacznik

Docelowo po kliknięciu na znacznik będziemy widzieli okno. Skorzystamy z biblioteki <https://github.com/karolzak/ipyplot> w zmienionej na nasze potrzeby formie. Ta część będzie uwzględniona dopiero w procesie tworzenia aplikacji w przyszłości.



Rysunek 16: Znacznik z informacjami

### 2.3.9 Wyświetlanie obserwacji

Oprócz mapy z obserwacjami (znacznikami) będzie obok widoczna metryczka z liczbą obserwacji i oknem ze wszystkimi obserwacjami w dwóch trybach:



Rysunek 17: Obserwacje z metryczką i zdjęciem

ID	Taxon ID	Taxon	Observed on	User	Location
30688807	78444	Species: <i>Peritom. serrulata</i> (Rocky Mountain Beeplant)	Aug 12, 2019	jkcook	Johnston, IA, USA
30688955	47912	Species: <i>Asclepias tuberosa</i> (butterfly milkweed)	Aug 12, 2019	jkcook	Johnston, IA, USA
30689111	60251	Species: <i>Verbena hastata</i> (blue vervain)	Aug 12, 2019	jkcook	Johnston, IA, USA
30689221	121968	Species: <i>Andropogon gerardi</i> (big bluestem)	Aug 12, 2019	jkcook	Johnston, IA, USA
30689306	121968	Species: <i>Andropogon gerardi</i> (big bluestem)	Aug 12, 2019	jkcook	Johnston, IA, USA
30689425	128701	Species: <i>Desmanthus illinoensis</i> (Illinois bundleflower)	Aug 12, 2019	jkcook	Johnston, IA, USA
30689463	121976	Species: <i>Silphium laciniatum</i> (compass plant)	Aug 12, 2019	jkcook	Johnston, IA, USA
30689506	126376	Species: .	Aug 12, 2019	jkcook	Johnston, IA, USA

Rysunek 18: Lista obserwacji z pełną informacją

## 2.4 Zapisywanie obszarów i obserwacji

Po wyjściu z okna *Obserwacje* pojawia się okno z zapytaniem, czy zapisać obserwację.

### 2.4.1 Postać pliku json

Obszary i obserwacje zapisują się w pliku typu json.

Pierwsza wersja była następująca.

```
1 {
2   'areas': [
3     {
4       'name': 'Park Krajobrazowy A',
5       'loc': [(12.32121, 13.12113), (12.12121, 13.12121), (12.12521,
6         13.12151)]
7         },
8         {
9           'name': 'Park Krajobrazowy B',
10          'loc': [(12.32121, 15.12113), (12.12121, 15.12121), (12.12521,
11            15.12151)]
12          }
13        ],
14   'observations': [
15     {
16       'area_name': '',
17       'loc': [(11.32121, 15.12113), (11.12121, 15.12121), (11.12521,
18         15.12151)],
19         'events': [
20           'Tutaj obserwacje jako słowniki json prosto z bazy pobierane
21           za pomocą funkcji pyinaturalist.get_observations() tak, aby były konwertowalne za
22           pomocą funkcji pyinaturalist.Observation.from_json_list()
23             ]
24         },
25         {
26           'area_name': 'Park Narodowy C',
27           'loc': [(1.32121, 15.12113), (1.12121, 15.12121), (1.12521, 15.12151)],
28           'events': [
29             'Jak wyżej
30           ]
31         }
32       }
33     ]
34 }
```

Listing 23: Schemat pliku JSON

Jednak po zmianach wprowadzonych w interfaç'e plik json będzie wyglądać w ten sposób.

```
1 {
2   'areas': [
3     {
4       'name': 'Park Krajobrazowy A',
5       'loc': [(12.32121, 13.12113), (12.12121, 13.12121), (12.12521,
6         13.12151)]
7         },
8         {
9           'name': 'Park Krajobrazowy B',
10          'loc': [(12.32121, 15.12113), (12.12121, 15.12121), (12.12521,
11            15.12151)]
12          }
13        ],
14   'observations': [
15     {
16       'area_name': 'Park Krajobrazowy A',
17       'data': {'start': '12-12-2021', 'stop': '20-12-2021'},
18       'events': [
19         'Tutaj obserwacje jako słowniki json prosto z bazy pobierane
20           za pomocą funkcji pyinaturalist.get_observations() tak, aby były konwertowalne za
21           pomocą funkcji pyinaturalist.Observation.from_json_list()
22             ]
23         },
24         {
25           'area_name': 'Park Krajobrazowy B',
26           'data': {'start': '12-12-2021', 'stop': '20-12-2021'},
27           'events': [
28             'Tutaj obserwacje jako słowniki json prosto z bazy pobierane
29               za pomocą funkcji pyinaturalist.get_observations() tak, aby były konwertowalne za
30               pomocą funkcji pyinaturalist.Observation.from_json_list()
31             ]
32         }
33       ]
34     ]
35   }
36 }
```

```

20
21     {
22         'area_name': 'Park Narodowy C',
23         'data': {'start': '12-12-2021', 'stop': '20-12-2021'},
24         'events': [
25             'Jak wyzej'
26         ]
27     }

```

Listing 24: Schemat pliku JSON

Postać JSONów obserwacji: [https://pyinaturalist.readthedocs.io/en/stable/user\\_guide.html#responses](https://pyinaturalist.readthedocs.io/en/stable/user_guide.html#responses)

The screenshot shows the iNaturalist API User Guide. On the left, there's a sidebar with links like 'User Guide', 'Examples', 'Endpoint Summary', 'API Reference', 'Contributing', 'Contributors', and 'History'. Below the sidebar, there's an advertisement for MongoDB Atlas. The main content area has a heading 'Observation response JSON' which is highlighted with a yellow oval. Below this, there's a section titled 'Previewing Responses' with some explanatory text and code examples. To the right, there's a sidebar with sections like 'CONTENTS', 'API Data vs Web UI', and various API-related topics.

And here is what that same observation looks like in JSON:

**Observation response JSON**

Previewing Responses

These responses can contain large amounts of response attributes, making it somewhat cumbersome if you just want to quickly preview results (for example, in a Jupyter notebook). For that purpose, the `pprint()` function is included to format response data as a condensed, color-highlighted table.

Examples:

Observations	Places	Places (with terminal colors)
<code>&gt;&gt;&gt; from pyinaturalist import get_observations, pprint</code>	<code>&gt;&gt;&gt; observations = get_observations(user_id='nicononc', per_page=5)</code>	<code>&gt;&gt;&gt; pprint(observations)</code>
ID	Taxon ID	Taxon
Observed on		User

Rysunek 19: Format obserwacji