Kolegium Nauk Przyrodniczych
Uniwersytet Rzeszowski

Przedmiot:

Wybrane Zagadnienia Współczesnej Informatyki

Tytuł projektu:

Aplikacja webowa do akwizycji i analizy danych z kanału RSS portalu gofin.pl (prawo pracy).

Wykonał:

Kamil Filar, Informatyka, rok III, lab1

Prowadzący: dr hab. inż. Krzysztof Pancerz, prof. UR

Rzeszów 2020

Spis treści:

1. Opis serwisu/aplikacji internetowej

Nazwa aplikacji: Aplikacja webowa do akwizycji i analizy danych z kanału RSS portalu gofin.pl (prawo pracy).

Aplikacja została stworzona z wykorzystaniem środowiska R oraz Apache Solr. Aplikacja webowa oparta jest o zaawansowane GUI (możliwość ustawiania parametrów) zrealizowane z wykorzystaniem pakietu Shiny. Ponadto posiada funkcjonalności takie jak:

- Możliwość pobierania wiadomości i magazynowania ich w Apache Solr.

- Możliwość analizy częstości występowania słów i wizualizacji chmury słów w wiadomościach zmagazynowanych w Apache Solr.

- Możliwość klasteryzacji wiadomości zmagazynowanych w Apache Solr oraz wizualizacji klastrów.

1.1 Główne funkcjonalności

1. Pobieranie danych z kanału RSS.
2. Magazynowanie ich w Apache Solr.
3. Pobieranie danych z Apache Solr.
4. Lematyzacja tekstu.
5. Analiza występowania danego słowa w zadanym zakresie danych.
6. Tworzenie chmury słów.
7. Analiza częstości występowania słów.
8. Klasteryzacja danych.

2. Szczegółowy opis funkcjonalności/wyglądu programu

2.1 Graficzny interfejs użytkownika

Graficzny interfejs użytkownika został zrealizowany z wykorzystaniem pakietu Shiny. Jego szczegółowy wygląd został zaprezentowany poniżej:

```
150 #UI aplikacji
151 ui <- fluidPage(
152 
153     includeCSS("style.css"),
154 
155     # Application title
156     tags$div(class="Title",
157         tags$p("Aplikacja webowa do akwizycji i analizy danych"),
158         tags$p("z kanalu RSS portalu"),
159         tags$p("Gofin.pl - prawo pracy")
160     ),
161     tags$div(class="Line1"),
162     tags$div(class="Line2"),
163 
164     sidebarLayout(
165         sidebarPanel(
166            tags$p("Procentowa zawartosc konkretnego slowa", class="SubTitle"),
167 
168            selectInput(inputId="SelectTab2",label=h4("Wybierz dane"),choices = c("Tytul"="Tytul","Zawartosc"="Zawartosc")
169                        selected = "Tytul",multiple = F),
170            textInput("slowo", h4("Slowo do wyszukania: ")),
171            selectInput(inputId="color1",label=h4("Wybierz kolor"),
172                        choices = c("Czerwony"="Czerwony","Niebieski"="Niebieski","Zielony"="Zielony"),
173                         selected = "Zielony",multiple = F),
174            actionButton("buttonA1","Start"),
175 
176         ),
177         mainPanel(
178             plotOutput("wykresA1")
179         ),
```

Rys. 1

```
194             numericInput("grid", h4("Odstep miedzy literami"), value = 5, min = 0, max = 25, step = 5),
```

Rys. 2

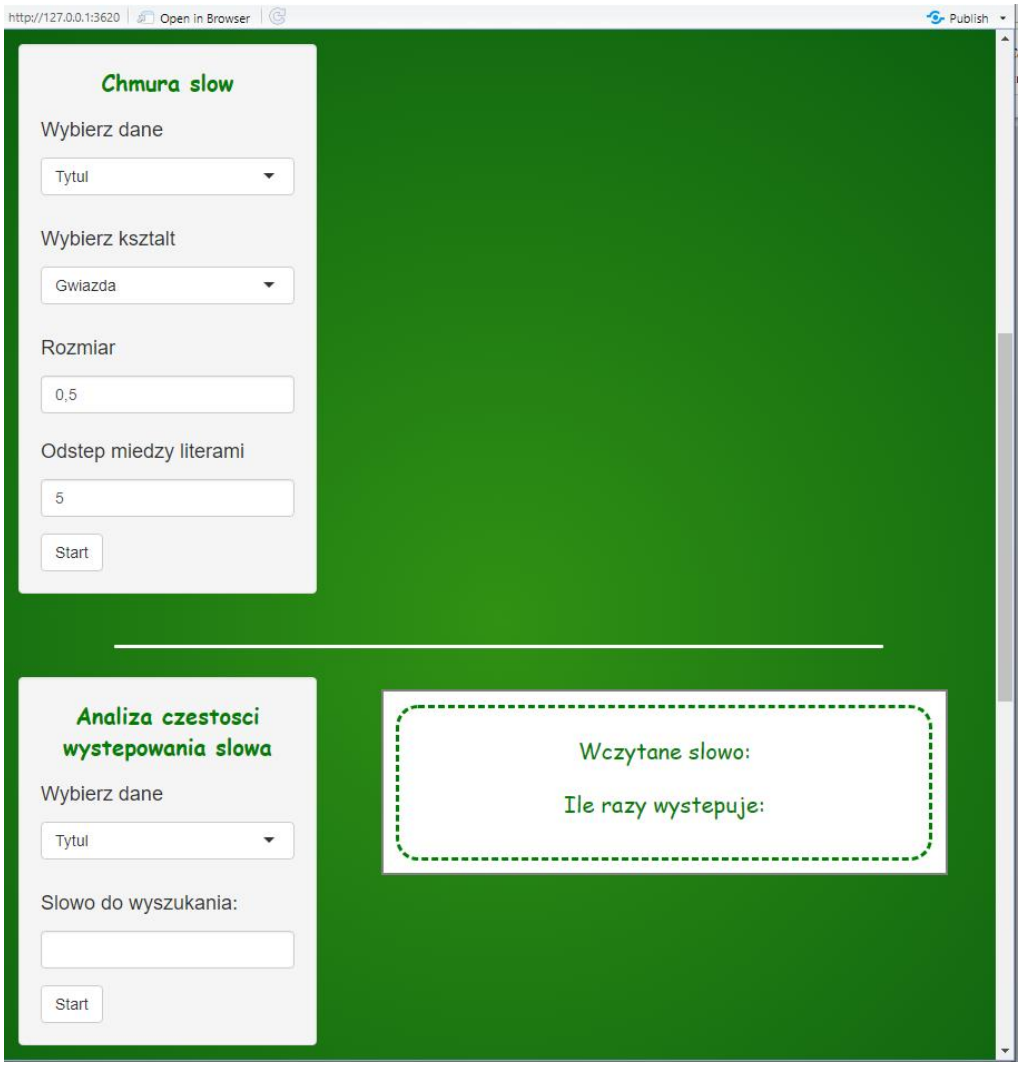
Rys. 1 przedstawia początek kodu UI. W linii 153 znajduje się załączony plik zawierający CSS, który w stanowi o wyglądzie aplikacji. W liniach od 156 do 160 znajduje się tytuł projektu, a poniżej tego są dwie linie służące do separacji między blokami w projekcie. Poniżej znajduje się pierwszy panel wchodzący w skład aplikacji. W aplikacji znajdują się cztery takie panele umożliwiające użytkownikowi analizę danych. Składają się z takich elementów jak:

- selectInput – służy do wybierania danych (168 linia Rys. 1)
- textInput – służy do wczytywania danych tekstowych (170 linia Rys. 1)
- actionButton – przycisk, który służy do uruchomienia funkcji (174 linia Rys. 1)
- numericInput – służy do wczytywania liczb z zakresu (194 linia Rys. 2)

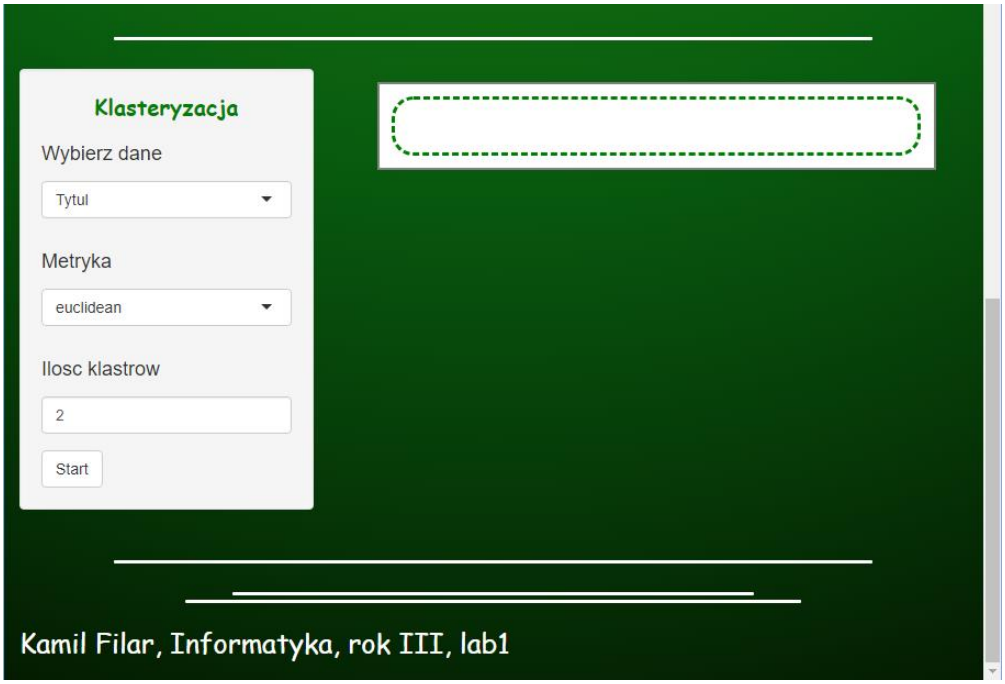
Wygląd UI został przedstawiony na Rys. 3, Rys. 4, Rys. 5 poniżej:


http://127.0.0.1:3620
Open in Browser
Publish
Aplikacja webowa do akwizycji i analizy danych
z kanalu RSS portalu
Gofin.pl - prawo pracy
Procentowa zawartosc konkretnego slowa
Wybierz dane
Tytul
Slowo do wyszukania:
Wybierz kolor
Zielony
Start

Rys. 3


http://127.0.0.1:3620
Open in Browser
Publish
Chmura slow
Wybierz dane
Tytul
Wybierz ksztalt
Gwiazda
Rozmiar
0,5
Odstep miedzy literami
5
Start
Analiza czestosci wystepowania slowa
Wybierz dane
Tytul
Slowo do wyszukania:
Start
Wczytane slowo:
Ile razy wystepuje:

Rys. 4


Klasteryzacja
Wybierz dane
Tytul
Metryka
euclidean
Ilosc klastrow
2
Start
Kamil Filar, Informatyka, rok III, lab1

Rys. 5

2.2 Apache Solr

```
19  #POBRANIE DANYCH ZE STRONY
20
21  aktualnosci <- tidyfeed(feed="http://www.rss.gofin.pl/prawopracy.xml")
22  #View(aktualnosci)
23  #WCZYTANIE WYBRANYCH DANYCH DO OBIEKTOW (SENSOWNYCH)
24  tytul_data <- aktualnosci$entry_title
25  zawartosc_data <- aktualnosci$entry_content
26  URL_data <- aktualnosci$entry_url
27
28  #View(aktualnosci) PODGLAD DANYCH (POMOCNICZO)
29
30  #DZIALANIE W SOLR
31  polaczenie <-SolrClient$new(host="127.0.0.1", port = 8983, path = "/solr/ProjektWZWI/select");
32
33  dane = data.frame(matrix(ncol=4, nrow = 10));
34  colnames(dane)[1] <- "id";
35  colnames(dane)[2] <- "Tytul";
36  colnames(dane)[3] <- "Zawartosc";
37  colnames(dane)[4] <- "URL"
38
39  dane$id<-c(1,2,3,4,5,6,7,8,9,10);
40  dane$Tytul<-c(tytul_data);
41  dane$Zawartosc<-c(zawartosc_data);
42  dane$URL<-c(URL_data);
43
44  solrium::add(x=dane, conn=polaczenie, name="ProjektWZWI", commit=TRUE);
45
```

Rys. 6

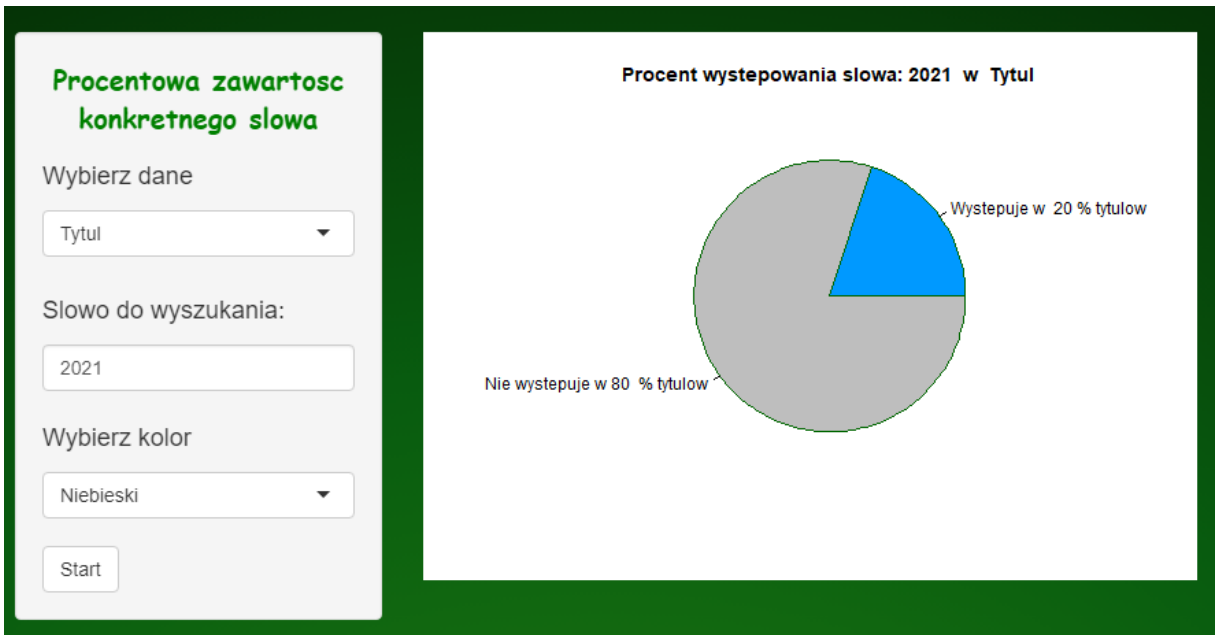
Na Rys. 6 została przedstawiona obsługa Apache Solr. W linii 21 wczytuje dane z kanału RSS za pomocą funkcji tidyfeed, następnie w liniach od 24-26 przypisuje obiektom (w celu ułatwienia sobie pracy) dane wczytane z kanału RSS, następnie w linii 34 tworzę połączenie z węzłem ProjektWZWI, tworzę obiekt do przechowywania danych i w linii 44 dodaję do Apache Solr moje dane. Pobieranie danych z Solr-a zostało opisane przy funkcjach, które to wykorzystują.

2.3 Występowanie danego słowa w wybranej tabeli

```
49  #FUNKCJA ZWRACA ILE % WIERSZY TABELI tabela2 ZAWIERA WCZYTANE slowo
50  analiza1<-function(slowo, tabela2){
51      dane1<-solr_search(conn = polaczenie, params = list(q=paste(tabela2,":",slowo),fl=paste(tabela2), rows=-1));
52      sprawdz_duplikaty <- duplicated(dane1)
53      View(sprawdz_duplikaty)
54      dane_koncowe <- unique(dane1)
55      ilosc_kolumn <- nrow(dane_koncowe)
56
57      return(ilosc_kolumn);
58  }
```

Rys. 7

Funkcja analiza1 służy do zliczenia ilości kolumn, w jakich zawiera się dane słowo. Funkcja przyjmuje dwa prametry: *slowo* oraz *tabela2*. Parametr słowo jest wczytywany w UI przez użytkownika i za pomocą logiki serwera zostaje przekazane do funkcji jako parametr przy wywołaniu, podobnie jest z drugim parametrem. Funkcja sprawdza czy są duplikaty (linia 52-54) w wczytanych danych z Apache Solr (samo wczytywanie jest zrealizowane za pomocą funkcji solr_search w linii 51) i wybiera tylko unikatowe wartości. Zwrócone wartości to ilość kolumn. Na Rys. 8 (poniżej) został zaprezentowany wygląd wyniku działania funkcji:



Rys. 8

2.4 Lematyzacja tekstu

```
lematyzacja<-function(tekst){
    #FUNKCJA ODPOWIEDZIALNA ZA LEMATYZACJE TEKSTU (KOPIA Z LAB4) sprawdzanie duplikatow!
    parametry<-list(lpmn="any2txt|wcrft2", text=tekst, user="FilarKamil04@gmail.com");
    odpowiedz<-POST("http://ws.clarin-pl.eu/nlprest2/base/process", body=parametry, encode="json", verbose());
    zawartosc<-content(odpowiedz, "text", encoding="UTF-8");
    xml<-xmlParse(zawartosc, encoding="UTF-8");
    slowa<-xpathSApply(xml, '//chunkList/chunk/sentence/tok/lex/base', xmlValue, encoding="UTF-8");

    return(paste(slowa, collapse=" "));
}
```

Rys. 9

Funkcja służąca do lematyzacji tekstu została przedstawiona na Rys. 9. Jest to funkcja, która została przedstawiona w laboratorium 4 podczas zajęć.

2.5 Przygotowanie danych do analizy

```
70  #FUNKCJA ODPOWIEDZIALNA ZA PRZYGOTOWANIE DANYCH DO ANALIZY
71  przygotowanie_danych<-function(dane){
72      stop<-as.vector(unlist(read.csv(file="stop_words_pl.txt", header=FALSE, sep=",", fileEncoding="UTF-8")));
73
74      dokumenty<-Corpus(VectorSource(stri_enc_toutf8(dane)));
75
76      dokumenty<-tm_map(dokumenty, removePunctuation, preserve_intra_word_dashes=TRUE);
77      dokumenty<-tm_map(dokumenty, removeNumbers);
78      dokumenty<-tm_map(dokumenty, removeWords, stop);
79
80      usun.znaki<-function(x) gsub("[â€“â€žâ€¢]", "", x);
81      dokumenty<-tm_map(dokumenty, usun.znaki);
82
83      for(d in 1:length(dokumenty))
84      {
85          dokumenty[[d]]$content<-lematyzacja(dokumenty[[d]]$content);
86          dokumenty[[d]]$content<-stri_enc_toutf8(dokumenty[[d]]$content);
87      }
88
89      tdm1<-TermDocumentMatrix(dokumenty);
90      m1<-as.matrix(tdm1);
91      v<-sort(rowSums(m1),decreasing=TRUE);
92      d<-data.frame(words=names(v), freq=v);
93
94      return(d);
95  }
```

Rys. 10

Na Rys. 10 została przedstawiona funkcja (wykorzystana z laboratorium 4), która przyjmuje parametr wejściowy *dane*, które są poddawane działaniom wewnątrz funkcji. Funkcja zwraca tabele z uporządkowanymi danymi w kolumnie słowa oraz częstotliwość ich występowania.

2.6 Chmura słów

```
 97  #FUNKCJA ODPOWIEDZIALNA ZA TWORZENIE CHMURY SLOW
 98  chmura<-function(tabela){
 99      dane<-solr_search(conn = polaczenie, params = list(q=paste(tabela,":*"),fl=paste(tabela), rows=-1));
100      sprawdz_duplikaty_chmura <- duplicated(dane)
101      View(sprawdz_duplikaty)
102      dane_koncowe <- przygotowanie_danych(dane)
103      return(dane_koncowe);
104  }
```

```
 98  #FUNKCJA ODPOWIEDZIALNA ZA TWORZENIE CHMURY SLOW
 99  chmura<-function(tabela){
100      dane<-solr_search(conn = polaczenie, params = list(q=paste(tabela,":*"),fl=paste(tabela), rows=-1));
101      sprawdz_duplikaty_chmura <- duplicated(dane)
102      View(sprawdz_duplikaty)
103      dane_z_solra <- przygotowanie_danych(dane)
104      dane_koncowe <- unique(dane_z_solra)
105      return(dane_koncowe);
106  }
```

Rys. 11

Funkcja *chmura* przyjmuje parametr tabela wczytany z UI przez użytkownika. Parametr wskazuje na tabelę z której solr pobiera dane do dalszego przetworzenia (linia 100 Rys.11). Następnie dane są sprawdzane pod kątem występowania duplikatów, a kolejnym krokiem jest zwrócenie unikatowych danych końcowych, które trafiają do przetworzenia w logice serwera (Rys. 12) gdzie w linii 300 została zastosowana funkcja tworząca chmurę słów (worldcloud2).

```
observeEvent(input$buttonA2,{

        output$WykresA2 <- renderWordcloud2({

            dane <- chmura(input$SelectTab1);

            if(input$SelectShape=="Gwiazda"){
                varshape = "star"
            }else if(input$SelectShape=="Diament"){
                varshape = "diamond"
            }else if(input$SelectShape=="Trojkat"){
                varshape = "triangle-forward"
            }

            wordcloud2(dane, size=input$size, gridSize = input$grid,
                       color = "random-dark", backgroundColor = "white",
                       shape = varshape)
        })

    });
```

Rys. 12

Wynik działania tej funkcji można zobaczyć na Rys. 13.

Rys. 14

2.7 Częstotliwość występowania danego słowa

```
#FUNKCJA PRZYGOTOWUJE DANE DO WYPISANIA ILOSCI DANEGO SLOWA I STWORZENIA WYKRESU 10 NAJPOPULARNIEJSZYCH
ilosc_slow<-function(tabela2){

    stop<-as.vector(unlist(read.csv(file="stop_words_pl.txt", header=FALSE, sep=",", fileEncoding="UTF-8")));

    dane<-solr_search(conn = polaczenie, params = list(q=paste(tabela2,":*"),fl=paste(tabela2), rows=-1));
    sprawdz_duplikaty_ilosc_slow <- duplicated(dane)
    view(sprawdz_duplikaty)

    dokumenty<-Corpus(VectorSource(stri_enc_toutf8(dane)));
    dokumenty<-tm_map(dokumenty, removePunctuation, preserve_intra_word_dashes=TRUE);
    dokumenty<-tm_map(dokumenty, removeNumbers);
    dokumenty<-tm_map(dokumenty, removeWords, stop);
    usun.znaki<-function(x) gsub("[â€“â€žâ€ť]", "", x);
    dokumenty<-tm_map(dokumenty, usun.znaki);

    for(d in 1:length(dokumenty))
    {
        dokumenty[[d]]$content<-lematyzacja(dokumenty[[d]]$content);
        dokumenty[[d]]$content<-stri_enc_toutf8(dokumenty[[d]]$content);
    }

    tdm1<-TermDocumentMatrix(dokumenty);
    m1<-as.matrix(tdm1);
    v<-sort(rowSums(m1),decreasing=TRUE);
    return(v);
}
```

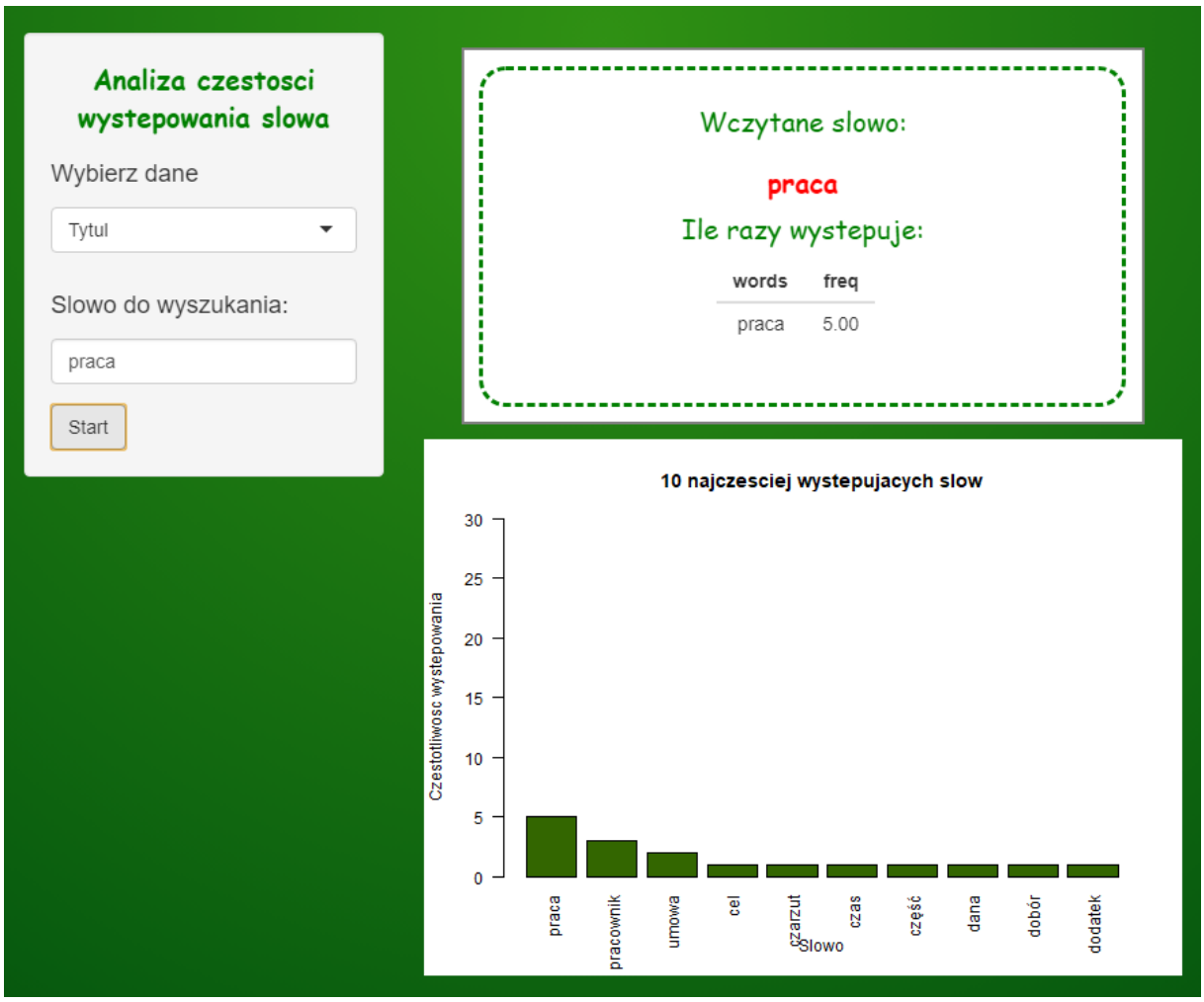
Rys. 15

Funkcja która przygotowuje dane do wyświetlenia danego słowa została przedstawiona na Rys. 15. Wykorzystuje funkcje lematyzującą tekst i zwraca posortowana i zliczona ilość słów. Następnie trafia do logiki serwera (Rys. 16) gdzie dane są przetwarzane. W liniach od 317 do 322 jest zawarty wykres, który wyświetla 10 najbardziej popularnych słów (wg. Występowania). W linii 328 Rys. 16 znajduje się tabela, która jest wyświetlana w UI.

```
    observeEvent(input$buttonA3,{
                    #WCZYTANY NUMER
                    wypisane_slowo2 <- eventReactive(input$buttonA3, {
                        input$slowo2
                    })
                    output$slowooutput <- renderText({
                        wypisane_slowo2()
                    })
                    #WCZYTANA ILOSC SLOW
                    dane <- data.frame(words=names(ilosc_slow(input$SelectTab3)), freq=ilosc_slow(input$SelectTab3))
                    output$WykresA3 <- renderPlot({
                        barplot(dane[1:10,]$freq, las = 2, names.arg = dane[1:10,]$word,
                                col ="#336600", main ="10 najczesciej wystepujacych slow",
                                ylab = "Czestotliwosc wystepowania", ylim = c(0,30),
                                xlab = "Slowo")
                    })

                   inputWORD <- input$slowo2
                    outputWORD <- filter(dane,
                                         words == toString(inputWORD)
                                         )
                    output$table <- renderTable({outputWORD})
                });
```

Rys. 16

Wygląd panelu z analizą częstotliwości występowania słowa został przedstawiony na Rys. 17.



Rys. 17

2.8 Klasteryzacja danych

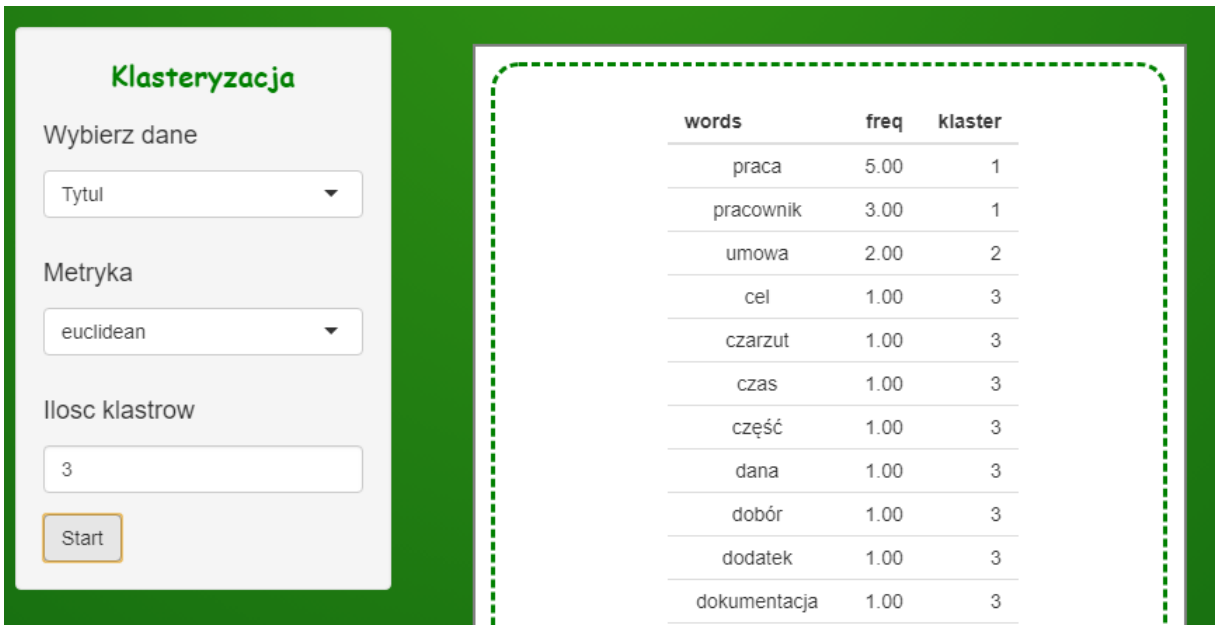
```
klasteryzacja <- function(tabela3, metryka, numberOFk){

    dane<-solr_search(conn = polaczenie, params = list(q=paste(tabela3,":*"),fl=paste(tabela3), rows=-1));
    sprawdz_duplikaty_klasteryzacja <- duplicated(dane)
    view(sprawdz_duplikaty)

    przetworzone_dane <- przygotowanie_danych(dane)
    dane_z_solra <- przygotowanie_danych(dane)
    dane_koncowe <- unique(dane_z_solra)

    klasteryzacja.h<-agnes(dane_koncowe, metric=paste(metryka), method="average");
    klaster<-cutree(klasteryzacja.h, k=paste(numberOFk));
    tabela.wynikowa.h<-cbind(dane_koncowe, klaster);

    plot(klasteryzacja.h, which.plots=2);

    return(tabela.wynikowa.h);
}
```

Rys. 18

W projekcie wykorzystałem klasteryzację hierarchiczną,. Funkcja klasteryzacja przyjmuje trzy parametry *tabela3*, *metryka*, *numberOFk*. Te parametry są wczytywane w UI przez użytkownika. Funkcja zwraca tabele z wynikami, która jest przetwarzana (Rys. 19) i wyświetlana (Rys. 20).

```
observeEvent(input$buttonA4,{
    output$table2<- renderTable(klasteryzacja(input$SelectTab4, input$SelectTab5, input$numberOFk))
})
```

Rys. 19



words	freq	klaster
praca	5.00	1
pracownik	3.00	1
umowa	2.00	2
cel	1.00	3
czarzut	1.00	3
czas	1.00	3
część	1.00	3
dana	1.00	3
dobór	1.00	3
dodatek	1.00	3
dokumentacja	1.00	3

Rys. 20