AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE
Wydział Fizyki i Informatyki Stosowanej

Praca magisterska

Eryk Jarocki i Arkadiusz Michalik

kierunek studiów: **informatyka stosowana**
specjalność: **Grafika komputerowa i przetwarzanie obrazów**

Aplikacja do analizy i eksploracji danych

Opiekun: **dr inż. Barbara Kawecka-Magiera**

Kraków, wrzesień 2020

Oświadczenie studenta

Uprzedzony(-a) o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2018 r. poz. 1191 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, wideogram lub nadanie.", a także uprzedzony(-a) o odpowiedzialności dyscyplinarnej na podstawie art. 307 ust. 1 ustawy z dnia 20 lipca 2018 r. Prawo o szkolnictwie wyższym i nauce (Dz. U. z 2018 r. poz. 1668 z późn. zm.) „Student podlega odpowiedzialności dyscyplinarnej za naruszenie przepisów obowiązujących w uczelni oraz za czyn uchybiający godności studenta.", oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Jednocześnie Uczelnia informuje, że zgodnie z art. 15a ww. ustawy o prawie autorskim i prawach pokrewnych Uczelni przysługuje pierwszeństwo w opublikowaniu pracy dyplomowej studenta. Jeżeli Uczelnia nie opublikowała pracy dyplomowej w terminie 6 miesięcy od dnia jej obrony, autor może ją opublikować, chyba że praca jest częścią utworu zbiorowego. Ponadto Uczelnia jako podmiot, o którym mowa w art. 7 ust. 1 pkt 1 ustawy z dnia 20 lipca 2018 r. – Prawo o szkolnictwie wyższym i nauce (Dz. U. z 2018 r. poz. 1668 z późn. zm.), może korzystać bez wynagrodzenia i bez konieczności uzyskania zgody autora z utworu stworzonego przez studenta w wyniku wykonywania obowiązków związanych z odbywaniem studiów, udostępniać utwór ministrowi właściwemu do spraw szkolnictwa wyższego i nauki oraz korzystać z utworów znajdujących się w prowadzonych przez niego bazach danych, w celu sprawdzania z wykorzystaniem systemu antyplagiatowego. Minister właściwy do spraw szkolnictwa wyższego i nauki może korzystać z prac dyplomowych znajdujących się w prowadzonych przez niego bazach danych w zakresie niezbędnym do zapewnienia prawidłowego utrzymania i rozwoju tych baz oraz współpracujących z nimi systemów informatycznych.

.....

(czytelny podpis studenta)

Kraków, 1 września 2020

Praca magisterska "Aplikacja do analizy i eksploracji danych" wykonana przez Eryka Jarockiego i Arkadiusza Michalika, studentów drugiego roku studiów drugiego stopnia na kierunku informatyka stosowana na specjalności grafika komputerowa i przetwarzanie obrazów.

Temat pracy magisterskiej: **Aplikacja do analizy i eksploracji danych.**

Opiekun pracy: dr inż. Barbara Kawecka-Magiera
Recenzenci pracy:
Miejsce praktyki dyplomowej: WFiIS AGH, Kraków

Program pracy magisterskiej i praktyki dyplomowej

1. Omówienie realizacji pracy z opiekunem.
2. Zebranie literatury potrzebnej do realizacji pracy.
3. Praktyka dyplomowa:
    - Zapoznanie się z tematyką uczenia maszynowego.
    - Zapoznanie się z dostępnymi narzędziami potrzebnymi do realizacji pracy.
    - Analiza istniejących narzędzi o podobnej tematyce.
    - Sporządzenie szkieletu projektu.
    - Sporządzenie sprawozdania z praktyki dyplomowej.
4. Implementacja projektu.
5. Testowanie zastosowanych rozwiązań oraz ich optymalizacja.
6. Sporządzenie dokumentacji projektu oraz kodu źródłowego.
7. Opracowanie redakcyjne pracy.

Termin oddania w dziekanacie: 1 września 2020

....

(podpis kierownika katedry) (podpis opiekuna)

Merytoryczna ocena pracy przez opiekuna:

Merytoryczna ocena pracy przez recenzenta:

Spis treści

1 Wstęp

1.1 Wprowadzenie

W dzisiejszych czasach firmy oraz instytucje naukowe przechowują, przetwarzają i analizują ogromne ilości informacji zawartych w bazach i hurtowniach danych. Zmagazynowane dane określające działania przedsiębiorstwa, instytucji i ich klientów pozwalają na szeroką analizę trendów, przewidywań rozwoju biznesu, oceny klienta, ale również na przewidywanie zagrożeń finansowych.

Eksploracja danych to proces odkrywania uogólnionych reguł, wzorców i zależności oparty o metody statystyczne i techniki sztucznej inteligencji. Wiedza ta nie wynika bezpośrednio z samych danych, ale z faktu, iż to właśnie takie, a nie inne dane znalazły się razem w jednej bazie danych.

1.2 Cel i zakres pracy

Celem pracy było stworzenie aplikacji internetowej zapewniającej zintegrowane środowisko do przygotowania danych niezbędnych do eksploracji, wykonania analizy na wybranym zbiorze oraz wizualizacji uzyskanych wyników. Aplikacja umożliwia użytkownikowi dokonanie takich kroków uczenia maszynowego jak przygotowanie danych, zastosowanie wybranego algorytmu na danych oraz analiza wyników.

System udostępnia użytkownikowi możliwość wyboru metody eksploracji danych. Wśród nich możemy wyróżnić takie metody jak *klasyfikacja, regresja, klasteryzacja* oraz *korelacja*. Każda z nich daje użytkownikowi różne możliwości na przykład, metoda klasyfikacji będzie niezbędna do zdiagnozowania chorób na podstawie wcześniejszej klasyfikacji schorzeń. Metoda regresji to technika bardzo podobna do klasyfikacji. Podstawową różnicą pomiędzy regresją i klasyfikacją jest to, że w klasyfikacji przewidywana zmienna przyjmuje wartość kategoryczną, natomiast w regresji jej celem jest przewidzenie zmiennej przyjmującej wartość ciągłą. Kolejną metodą jest grupowanie tworzące skończoną liczbę podzbiorów klas, grup obiektów posiadających podobne cechy. Metoda ta często wykorzystywana w wyszukiwarkach internetowych do tworzenia grup tematycznych powiązanych dokumentów bądź grupowanie klientów ze względu na podobieństwo zakupionych produktów lub zrealizowanych zamówień. Aplikacja pozwala również wykryć współzależność cech (korelacja) pokazującą związek, jaki zachodzi pomiędzy nimi. W wyniku analizy korelacji zmiennych uzyskujemy trzy podstawowe informacje: istotność statystyczną związku, siłę oraz kierunek związku. Szczegółowy opis oraz sposób użycia powyższych metod znajduje się w rozdziale 7 *Podręcznik użytkownika*.

Eksploracja danych to prężnie rozwijająca się dziedzina nauki. Na rynku powstaje coraz więcej narzędzi usprawniających ten proces decyzyjny wykorzystujący metody analizy danych. Wśród znanych firm zajmujących się tworzeniem oprogramowania komercyjnego wyróżniamy

HP, IBM, Microsoft, czy *Oracle*. Na ogół tworzą one jednak dedykowane rozwiązania na potrzeby odbiorców. Oprócz rozwiązań komercyjnych dostępne są również programy niekomercyjne. Do najbardziej popularnych zaliczamy *Rapid Miner*[1], *CMSR DATA Miner*[2], *Databionic ESOM Tools*[3], *ELKI*[4], *KNIME*[5], *Mloss*[6] oraz wiele innych. Można zauważyć, że darmowych programów do eksploracji danych jest wiele i nie zostały tutaj zamieszczone wszystkie. Wadą tych programów jest natomiast to, że interfejs graficzny większości z nich sprawia, iż są one trudne do zrozumienia oraz obsługi przez klienta nieposiadającego wiedzy z dziedziny eksploracji danych. Dlatego wymagają one dobrej znajomości tematów związanych z tą dziedziną, nie gwarantując przy tym prawidłowości działania. Przykładem może być program *SenticNEt API*[7]. Sam autor zaznacza, że program nie gwarantuje pewności działania i nie ponosi odpowiedzialności za szkody powstałe w wyniku jego działania.

Celem zrealizowanej pracy magisterskiej jest stworzenie aplikacji służącej do magazynowania i eksploracji danych posiadającej przejrzysty i intuicyjny interfejs graficzny. Zamierzeniem autorów jest, aby nawet klienci nieposiadający specjalistycznej wiedzy z zakresu metod eksploracji danych, mogli poradzić sobie z obsługą programu. Program został stworzony jako aplikacja webowa i jest dostępny pod adresem `https://www.edufront.pl/`. Dzięki temu każdy użytkownik będzie miał do niego dostęp bez konieczności pobierania oraz instalowania go na własnym komputerze. Na stronie głównej systemu zamieszczono podręcznik użytkownika udostępniający informację, w jaki sposób działa aplikacja, jak jej użyć oraz jakie daje możliwości.

Dodatkowo w aplikacji użyto algorytmów metod eksploracji danych znajdujących się w bibliotece *Scikit-learn*[8] do uczenia maszynowego oprogramowania dla języka Python. Biblioteka ta zawiera algorytmy analizy danych takie jak: *klasyfikacja, regresja*, czy też *grupowanie*. Takie rozwiązanie pozwoliło zwiększyć prawdopodobieństwo uzyskanych wyników analiz jako prawidłowe, powiększając tym grono zainteresowanych użytkowników działaniem aplikacji.

Aplikacja to narzędzie służące do eksploracji danych odkrywające wzorce, reguły i zależności danych, wykorzystująca uczenie maszynowe do zapamiętywania pewnych wzorców i zachowań, których maszyna doświadczyła podczas procesu uczenia. Zastosowanie systemu znajdziemy w wielu dziedzinach, w których należy dokonać analizy i oceny dużej ilości danych, gdy człowiek sam nie jest w stanie szybko ich przeanalizować. Wiedza pozyskana z rezultatów analiz może zostać wykorzystana w procesach decyzyjnych, jako pomoc w dokonaniu ostatecznej oceny przedstawionych wyników.

[1] *Rapid Miner* - https://rapidminer.com/ (licencja AGPL).

[2] *CMSR DATA Miner* - http://www.roselladb.com/starprobe. htm (licencja akademicka, wersja darmowa na 6 miesięcy).

[3] *Databionic ESOM Tools* - http://databionic-esom.sourceforge.net/ (licencja GNU GPL).

[4] *ELKI* - http://elki.dbs.ifi.lmu.de/ (licencja AGPL).

[5] *KNIME* - http://www.knime.org/ (licencja GNU GPL).

[6] *Mloss* - http://mloss.org/software/ (licencja GNU GPL).

[7] *SenticNEt API* - http://sentic.net/api/ (Darmowy z umieszczeniem informacji: "Copyright © 2012 Yuri Malheiros").

[8] *Scikit-learn* - https://scikit-learn.org/stable/ (licencja BSD).

Praca została podzielona na rozdziały zawierające opis wykorzystanych technologii, sposób ich implementacji, informacje niezbędne do wdrożenia aplikacji na serwer oraz podręcznik użytkownika. Na końcu pracy znajduje się bibliografia, natomiast w załączniku zamieszczono dokumentację kodu wygenerowaną za pomocą narzędzi *javadoc, jspd* i *sphinx* oraz kod aplikacji.

1.3 Wymagania funkcjonalne

Przed przystąpieniem do implementacji kodu źródłowego podjęto analizę wymagań funkcjonalnych. Należało określić zakres podjętych prac, oszacować ich czasochłonność, określić przewidywane koszty z nimi związane oraz czas wykonania. Proces ten umożliwił zidentyfikowanie i opisanie pożądanego zachowania systemu.

Dzięki temu procesowi wyznaczono techniczne wymagania projektu. Aplikację podzielono na mikroserwisy, separując w ten sposób jej funkcjonalności. Interfejs graficzny aplikacji został wydzielony do serwisu frontendowego *Datamaster-ui*. Funkcjonalność obsługi danych użytkownika została zaimplementowana w serwisie *Datamaster-service*, natomiast algorytmy uczenia maszynowego oraz edycji danych przeznaczonych do eksploracji obsługiwane są w serwisie *Datamaster-engine*.

Po analizie tematu eksploracji danych, przeanalizowaniu dostępnych już rozwiązań postawiono następujące wymagania funkcjonalne:

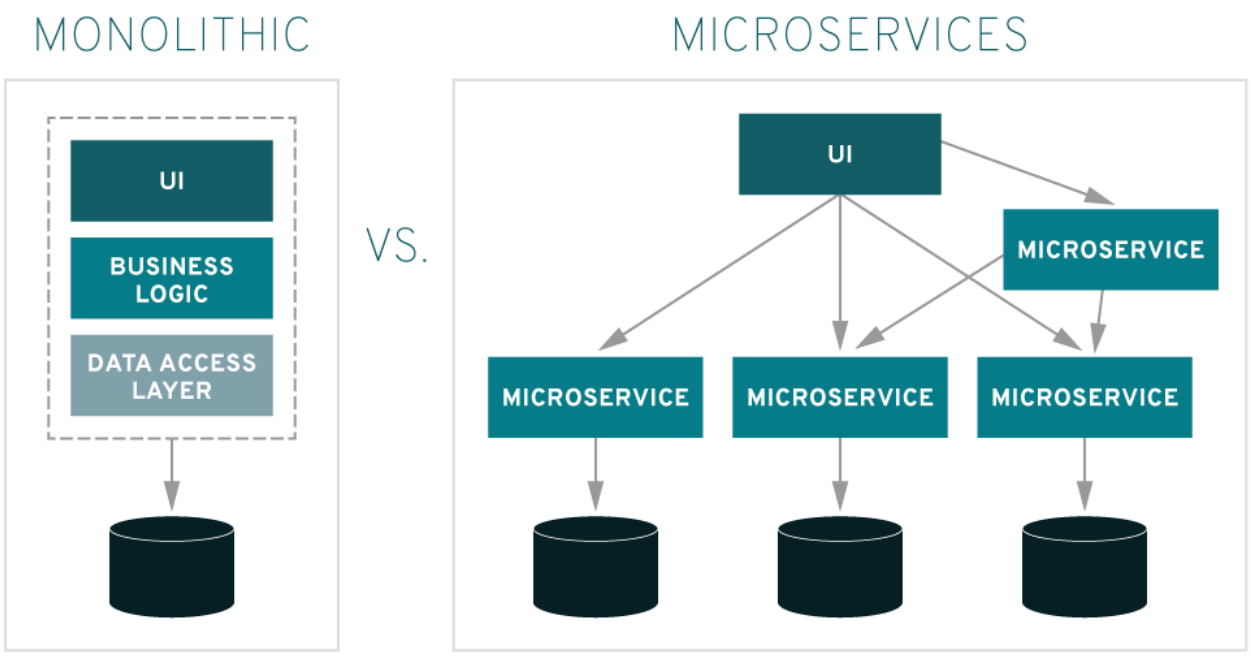
1. Stworzenie konta użytkownika w serwisie.
2. Importowanie danych do analizy z pliku csv lub z bazy danych serwisu.
3. Edycja zaimportowanych danych oraz ich zapisywanie do bazy danych lub na dysk.
4. Analiza danych przy użyciu metod uczenia maszynowego.
5. Zapisywanie rezultatów analiz do pliku bądź do bazy danych.
6. Zarządzanie zapisanymi danymi.
7. Stworzenie galerii wyników analiz.

Przedstawiona lista powyżej pozwala stwierdzić, jakie usługi ma oferować system oraz czego można się spodziewać po jej działaniu.

2 Struktura aplikacji

Aplikacja została podzielona na mniejsze projekty zgodnie z *architekturą mikroserwisów* [9]. Na poniższym rysunku znajduje się porównanie struktury *aplikacji monolitycznej* z aplikacją o *architekturze mikroserwisów*.

[9] *Architektura mikroserwisów* - Alternatywne rozwiązanie do tworzenia aplikacji monolitycznych. W architekturze mikroserwisów aplikacja podzielona jest na mniejsze systemy podzielone ze względu na ich funkcjonalności.



Rysunek 1: Porównanie aplikacji monolitycznej z architekturą mikroserwisów.
Źródło: `https://www.redhat.com/cms/managed-files/monolithic-vs-microservices.png`

W architekturze mikroserwisów każdy z nich wykonuje zadania przydzielone tylko jemu. Komponenty zawarte w mikroserwisach można prosto wymieniać oraz niezależnie rozwijać i instalować. Wszystkie projekty wchodzą w skład większej aplikacji, którą w prosty sposób można dalej skalować oraz rozszerzać o nowe funkcjonalności. W dzisiejszych czasach powszechne jest dzielenie aplikacji na trzy niezależne elementy, a są nimi: *frontend (warstwa prezentacji), backend (warstwa logiki biznesowej)* oraz *baza danych* (warstwa gromadzenia danych). W przypadku naszej aplikacji elementy te podzielono zgodnie z architekturą mikroserwisów na mniejsze projekty, z których każda jest odpowiedzialna za wykonanie określonych przez nią zadań. Podział oraz opis stworzonych w ten sposób serwisów znajdziemy w następnym rozdziale 2.1 *Projekty*.

2.1 Projekty

Aplikację podzielono na trzy projekty posiadające osobne repozytoria w serwisie *Github*. W ten sposób oddzielono funkcjonalności logiki biznesowej do wykonywania operacji na danych przeznaczonych do obliczeń oraz do obsługi danych użytkownika. Ponadto wydzielono również logikę odpowiedzialną za graficzny interfejs użytkownika. Poniżej w punktach wymieniono stworzone serwisy wraz z ich krótkim opisem:

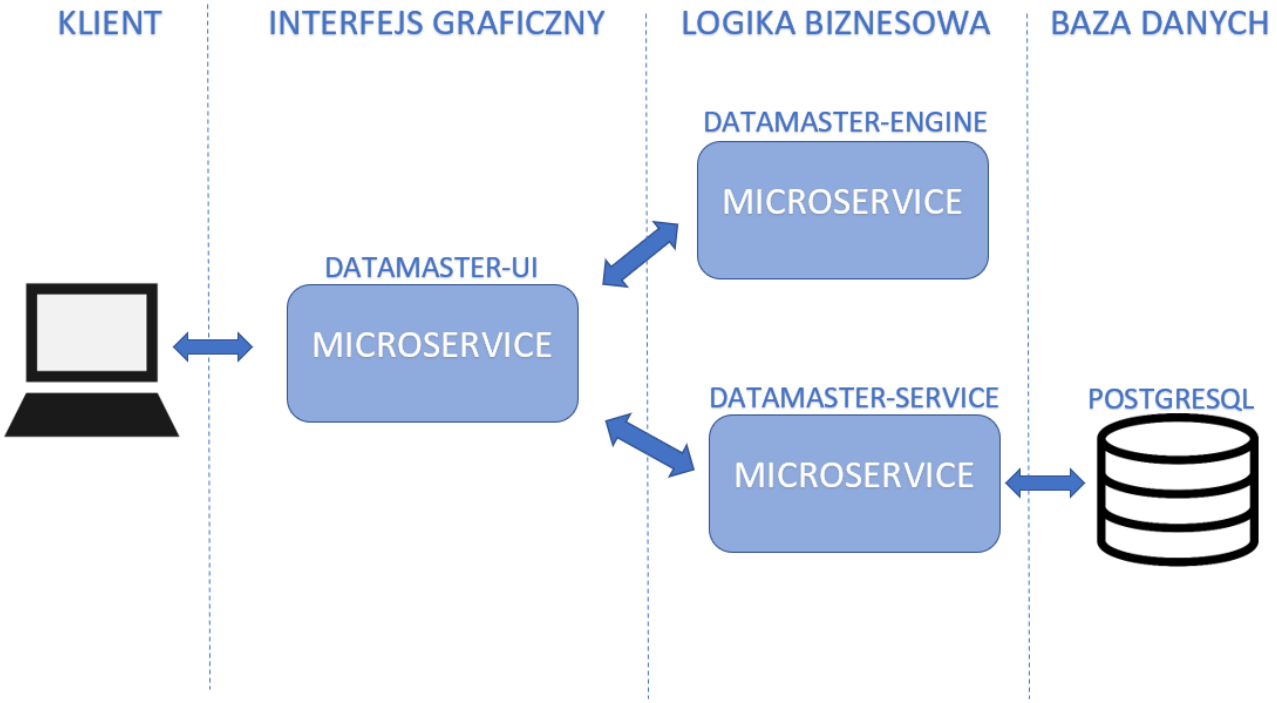
1. *Datamaster-ui* - Serwis frontendowy napisany w języku *JavaScript* (*React.js*) pełniący funkcję graficznego interfejsu użytkownika.

2. *Datamaster-service* - Serwis backendowy napisany w języku *Java* (*Spring* framework) służący do zarządzania bazą danych, autoryzacji i autentykacji użytkownika oraz zapisywania

dokumentów.

3. *Datamaster-engine* - Serwis backendowy napisany w języku *Python* (*Flask* framework) służący do wykonywania analizy, eksploracji oraz edycji danych.

Diagram przedstawiający podział projektów ze względu na ich przeznaczenie wraz z symbolicznym przedstawieniem komunikacji między serwisami znajduje się poniżej:



Rysunek 2: Struktura aplikacji.
Źródło: opracowanie własne.

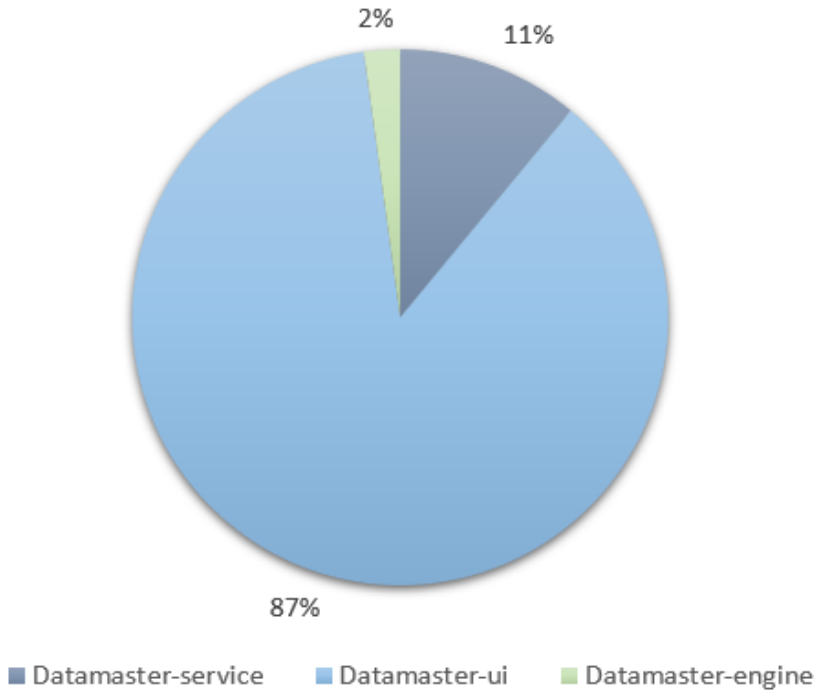
Serwis *Datamaster-ui* komunikuje się z serwisami logiki biznesowej oraz wyświetla interfejs graficzny aplikacji. Serwis *Datamaster-engine* wykonuje operacje na danych oraz wykonuje algorytmy eksploracji danych na żądanie klienta. *Datamaster-service* odpowiada za zabezpieczenie aplikacji oraz magazynuje informacje takie jak informacje o użytkowniku, zapisane wyniki analiz i zbiory danych komunikując się z bazą danych.

Jak zostało wcześniej wspomniane, projekty posiadają oddzielne repozytoria w serwisie *Github*:

1. *Datamaster-ui* - `https://github.com/erykjarocki/datamaster-ui`
2. *Datamaster-service* - `https://github.com/ArekMich/datamaster-service`
3. *Datamaster-engine* - `https://github.com/erykjarocki/datamaster-engine`

W serwisie dostępny jest podgląd historii prac nad danym projektem oraz wiele statystyk takich jak np. użyte języki programowania.

Dokonano również porównania rozmiaru projektów na podstawie ilości zaimplementowanych linijek kodu, z których się składają.



Rysunek 3: Wykres procentowego udziału mikroserwisów w projekcie na podstawie zaimplementowanych ilości linijek kodu.

Źródło: opracowanie własne.

Na wykresie widać, że projekt frontendowy stanowi większość aplikacji. Wynika to z konieczności zaimplementowania wielu komponentów, które budują graficzny interfejs użytkownika wraz z ich logicznym połączeniem. Warto jednak zaznaczyć, że najmniejszy projekt tj. *Datamaster-engine* składa się w bardzo niewielkim stopniu z kodu typu *boilerplate* czyli takiego, który nie wymaga wkładu pracy, a stanowi bardziej powtarzający się schemat konieczny do zaimplementowania. Kod pisany właśnie w tym najmniejszym projekcie wymagał najwięcej czasu, gdyż implementował logikę biznesową związaną z analizą, edycją oraz eksploracją danych. Sumarycznie cały projekt stworzony jest z około 41.5 tys. linijek kodu.

2.2 Sekcje aplikacji

Frontendowy projekt *Datamaster-ui* został podzielony na sekcje. Dzięki nim użytkownik znacznie szybciej zrozumie działanie aplikacji, a programiście ułatwi proces implementacji odpowiednich komponentów aplikacji. Dokładny opis każdej z sekcji oraz przykłady ich zastosowań znajdziemy w rozdziale 7 *Podręcznik użytkownika*, natomiast poniżej lapidarnie wyjaśniono każdą z nich.

1. *Home* - Strona powitalna aplikacji.
2. *Login* - Logowanie użytkownika.

3. *Register* - Rejestracja użytkownika.

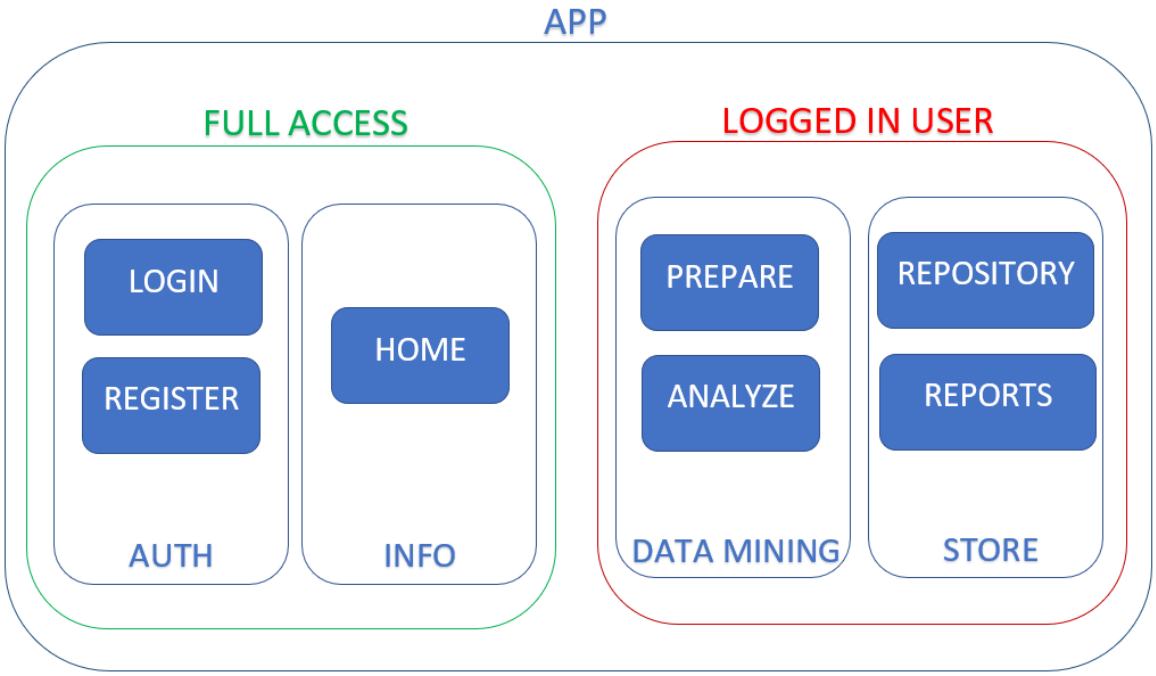
4. *Prepare* - Edytowanie zaimportowanych danych oraz ich zapisywanie.

5. *Analyze* - Analiza danych oraz wyświetlanie wyników analiz.

6. *Reports* - Przechowywanie wyników analiz oraz galeria wyników.

7. *Repository* - Przechowywanie zapisanych danych.

Na poniższym diagramie pogrupowano sekcje ze względu na ich przeznaczenie oraz dostępność (czy należy być zalogowanym, by móc się dostać do danej sekcji). Funkcję informacyjną pełni sekcja *Home*. Sekcje *Login* oraz *Register* służą do logowania się do serwisu, oraz tworzenia w nim konta. Do operacji na danych przeznaczone są sekcje *Prepare* oraz *Analyze* natomiast wyświetlanie zmagazynowanych danych jest dostępne w sekcjach *Repository* i *Reports*.



Rysunek 4: Sekcje aplikacji wraz z kontrolą dostępu.
Źródło: opracowanie własne.

3 Podział prac

3.1 Redakcja rozdziałów

3.1.1 Arkadiusz Michalik

1. Redakcja rozdziałów: 1, 3.2.1, 4.1, 6.2, 6.3, 7.1, 7.2, 7.5, 7.6, 9

3.1.2 Eryk Jarocki

1. Redakcja rozdziałów: 3.2.2, 4.2, 4.3, 5, 6.1, 7.3, 7.7, 7.8

3.1.3 Wspólnie zredagowane

1. Redakcja rozdziałów: 1.3, 2, 3.1, 7.4, 8

3.2 Wykonane zadania

3.2.1 Arkadiusz Michalik

1. Współtworzenie serwisu frontendowego Datamaster-ui.
   (a) Stworzenie sekcji *Login* i *Register*.
   (b) Stworzenie sekcji *Repository*.
   (c) Stworzenie sekcji *Reports*.
   (d) Stworzenie panelu do zapisywania, pobierania, bądź przejścia do analizy w sekcji *Prepare*.
   (e) Implementacja analizy klasteryzacji i korelacji w sekcji *Analyze*.
   (f) Implementacja widoku rezultatów analiz w sekcji *Analyze*.
   (g) Dokumentacja kodu źródłowego projektu.
2. Stworzenie serwisu backendowego *Datamaster-service*.
   (a) Stworzenie struktury projektu.
   (b) Stworzenie struktury oraz zarządzanie relacyjną bazą danych *PostgreSQL*.
   (c) Implementacja autentykacji oraz autoryzacji użytkownika.
   (d) Implementacja zarządzania plikami analiz.
   (e) Implementacja zarządzania wynikami analiz.
   (f) Zaprojektowanie oraz implementacja *Restful Web Services*.
   (g) Optymalizacja zapytań bazodanowych.
   (h) Dokumentacja kodu źródłowego projektu.
3. Współtworzenie serwisu backendowego Datamaster-engine.
   (a) Implementacja wstępnego przetwarzania danych.
   (b) Implementacja klasteryzacji.
   (c) Implementacja korelacji.
   (d) Implementacja generowania wykresów.
   (e) Dokumentacja kodu źródłowego projektu.

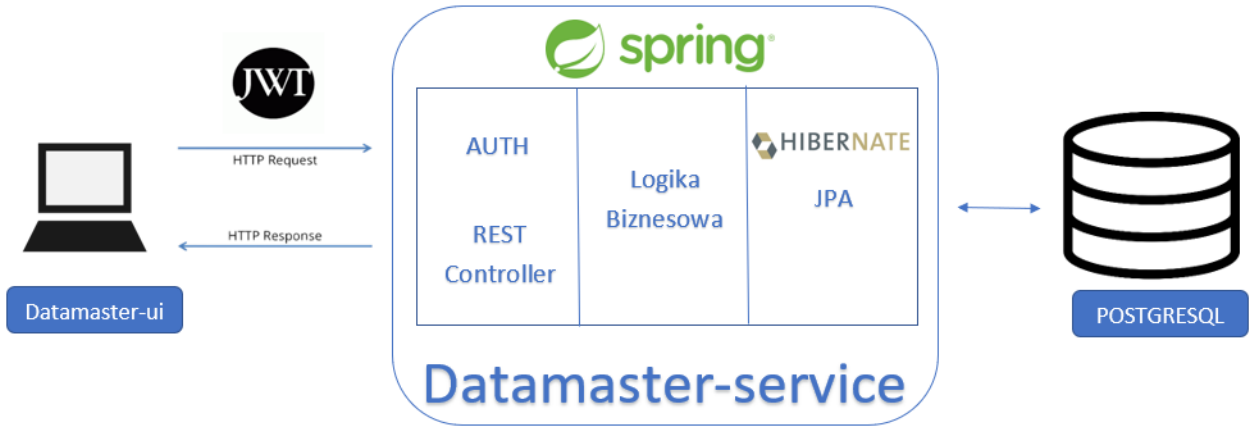
3.2.2 Eryk Jarocki

1. Współtworzenie serwisu frontendowego *Datamaster-ui*.
   (a) Stworzenie struktury projektu.
   (b) Naprawianie błędów w zakresie całego projektu.
   (c) Implementacja funkcjonalności renderowania po stronie serwera.
   (d) Optymalizacja procesu kompilacji plików pod kątem wdrożenia aplikacji na serwer.
   (e) Implementacja importowania oraz edytowania danych w sekcji *Prepare*.
   (f) Tworzenie reużywalnych komponentów.
   (g) Implementacja analizy klasyfikacji i regresji w sekcji *Analyze*.
   (h) Implementacja importu danych w sekcji *Analyze*.
   (i) Implementacja sekcji *Home*.
2. Współtworzenie serwisu *Datamaster-engine*.
   (a) Stworzenie struktury projektu.
   (b) Implementacja klasyfikacji.
   (c) Implementacja regresji.
   (d) Implementacja obsługi błędów.
3. Umieszczenie aplikacji na serwerze
   (a) Obsługa serwera
   (b) Konteneryzacja mikroserwisów.
   (c) Połączenie mikroserwisów.
   (d) Implementacja obsługi zapytań do serwera.
   (e) Optymalizacja działania aplikacji.
4. Współtworzenie serwisu *Datamaster-service*.
   (a) Stworzenie profilu produkcyjnego serwisu.

4 Serwisy aplikacji

4.1 Serwis Datamaster-service

Głównym zastosowaniem tego serwisu jest zabezpieczenie aplikacji, a także zapisywanie oraz dostęp do zasobów przechowywanych w bazie danych.



Rysunek 5: Struktura projektu *Datamaster-service*.
Źródło: opracowanie własne z użyciem ikon.
Ikona *JWT*: `https://cdn.worldvectorlogo.com/logos/jwt.svg`
Ikona *Spring*:
`https://spring.io/images/spring-logo-9146a4d3298760c2e7e49595184e1975.svg`
Ikona *Hibernate*: `https://hibernate.org/images/hibernate-logo.svg`

Aplikację backendową *Datamaster-service* z zapleczem *Spring Boot*[10] zbudowano przy użyciu narzędzia *Spring Initialzr*[11]. Wykorzystuje *Spring Boot Starter Web*[12], gdyż jest on aplikacją internetową o architekturze *REST API*[13]. Komunikuje się z serwisem *Datamaster-ui* przy użyciu protokołu HTTP do wymiany danych. Do zabezpieczenia aplikacji wykorzystano *Spring Security*[14] wraz z uwierzytelnianiem *JWT*[15]. System pozwala również na utrwalanie oraz przechowywanie danych w relacyjnej bazie danych *PostgreSQL* wykorzystując w tym celu framework *Spring Data JPA*[16] oraz Hibernate ORM do mapowania obiektowo-relacyjnego. W rozdziale *4.1.2 Struktura projektu* szerzej wytłumaczono pojęcia opisane powyżej oraz strukturę projektu.

4.1.1 Wyróżnione standardy i technologie

4.1.1.1 Spring Framework

[10] *Spring Boot* - framework Javy bazujący na Springu stworzony do budowania mikroserwisów.
[11] *Spring Initialzr* - narzędzie do szybkiego generowania struktury projektu Spring Boot (`https://start.spring.io/`)
[12] *Spring Boot Starter Web* - Starter do tworzenia aplikacji internetowych, w tym aplikacji RESTful przy użyciu Spring MVC. Używa Tomcat jako domyślnego kontenera osadzonego.
[13] *REST API (ang. Representational state transfer)* - styl architektury oprogramowania wywiedziony z doświadczeń przy pisaniu specyfikacji protokołu HTTP dla systemów rozproszonych.
[14] *Spring Security* - wysoce konfigurowalna platforma do uwierzytelniania oraz kontroli dostępu
[15] *JWT* - JSON Web Token to otwarta, zgodna ze standardem branżowym metoda RFC 7519 do bezpiecznego reprezentowania roszczeń między dwiema stronami (`https://jwt.io/`).
[16] *Spring Data JPA* - narzędzie ułatwiające wdrażanie repozytoriów opartych na JPA.

Spring jest frameworkiem aplikacyjnym oraz kontenerem odwrócenia sterowania *(ang. Inversion-of-Control, IoC)* dla aplikacji Java. Zawiera zbiór funkcjonalności, które stanowią bardzo solidny fundament dla wszystkich nowoczesnych aplikacji opartych o język Java. Spring pomaga developerom tworzyć proste, przenośne, szybkie i elastyczne systemy oraz aplikacje oparte na *JVM*[17][1]. Jego głównymi zaletami są:

- MNOGOŚĆ MOŻLIWOŚCI - tworzenie łatwego, czystego kodu z możliwością testowania wybranego komponentu spośród infrastruktury aplikacji. Możliwość realizowania zadań bez wymyślania prostych oraz gotowych już rozwiązań.
- PRZENOŚNOŚĆ ROZWIĄZAŃ - aplikacje napisane w Springu działają na każdym środowisku z JVM i można je uruchomić samodzielnie na serwerze aplikacyjnym, czy też w chmurze.
- ZAPEWNIONE WSPARCIE - framework Spring jest dobrze wspieraną technologią o szerokim zasięgu używalności. Dostarcza programiście kompleksowy, otwarty i spójny model oprogramowania.

4.1.1.2 Spring Boot

Programiści wykorzystujący framework Spring sami musieli zadbać o zachowanie kompatybilności pomiędzy używanymi bibliotekami. Częstym problemem podczas konfiguracji *API*[18] był brak zachowania zależności pomiędzy nimi. Dodatkowo framework ten wymaga posiadania zewnętrznego serwera, który pozwala na wgranie aplikacji i jej weryfikację. Wszystkie te etapy, czyli konfiguracja, wgranie, uruchomienie i testowania zajmowały wiele czasu. Dlatego z pomocą przychodzi nam *Spring Boot* [2] [3].

Spring Boot jest rozwiązaniem *"convention-over-configuration"*[19] bazującą na *Springu* i jest on tzw. *"starterem"* dostarczającym programiście zbiór wszystkich wymaganych bibliotek oraz konfiguracji. Zawiera także wbudowany serwer aplikacyjny *Tomcat*. Ostatnim elementem budowy aplikacji za pomocą *Spring Boot* jest:

- dodanie odpowiedniej zależności do pliku konfiguracyjnego *pom.xml*[20].
- dodanie *"rodzica"*, po którym aplikacja będzie dziedziczyć zależności.

Istnieje już gotowe narzędzie generujące projekt i jest nim wspomniany wcześniej *Spring Initializr*.

[17] *JVM (ang. Java Virtual Machine)* Wirtualna maszyna oraz środowisko zdolne do wykonywania kodu bajtowego Javy.

[18] *API (ang. application programming interface)* - Interfejs programistyczny aplikacji to zbiór reguł ściśle opisujący, w jaki sposób programy lub podprogramy komunikują się ze sobą.

[19] *Convention Over Configuration* - strategia pozwalająca tworzyć złożone rozwiązania programistyczne z użyciem niewielkiej ilości kodu.

[20] *POM (ang. Project Object Model)* - podstawowa jednostka pracy w Maven. Znajdujący się w głównym katalogu projektu jako pom.xml.

4.1.1.3 Spring Security

Spring Security to szkielet budowy aplikacji typu *enterprise*, który oferuje takie funkcjonalności jak: autentykacja, autoryzacja, zarządzanie sesjami oraz wiele innych udogodnień związanych z bezpieczeństwem aplikacji. Framework opiera się na architekturze, w której wykorzystywane są filtry. Każde zapytanie klienta przechodzi przez zestaw takich filtrów, które decydują czy udzielić dostęp do zasobu, odmówić czy przekazać decyzję dalej.

Spring Security zapewnia wsparcie dla wielu procesów autentykacji takich jak: *JWT, LDAP, OAuth, OpenID*. Dzięki temu jest jednym z najpopularniejszych frameworków służących do zabezpieczenia aplikacji [4].

4.1.1.4 JSON Web Token

JWT to mechanizm, wykorzystywany w kontekście webowych API, aplikacji WWW oraz mobilnych. Jeden z najpopularniejszych systemów autoryzacji obok *Basic Authentication*[21]. Jego zastosowanie możemy znaleźć w takich standardach jak np. *OpenID Connect*[22], czy *OAuth2.0*[23]. Istnieje wiele bibliotek obsługujących JWT, a sam standard mocno wspiera mechanizmy kryptograficzne.

JWT to w dużym skrócie metoda zapisu tzw. *deklaracji (ang. claims)* przy użyciu *JSONa*[24]. *Deklaracje* to najczęściej najprostsze pary typu *klucz-wartość*. Jego formalna definicja określona jest w dokumencie *RFC*[25] dostępny pod adresem `https://tools.ietf.org/html/rfc7519`. Czytamy w nim:

"JSON Web Token(JWT) to prosta, bezpieczna dla URL-a metoda reprezentowania deklaracji, które są przesyłane pomiędzy dwoma stronami. Deklaracje są zakodowane w postaci obiektu JSON, który jest używany jako zawartość (payload) struktury JSON Web Signature(JWS) lub jako fragment struktury JSON Web Encryption (JWE)."[26]

W teorii mamy do wyboru dwie struktury *JWS* lub *JWE*. Natomiast w praktyce najczęściej korzysta się z *JWS* i to właśnie ta struktura powszechnie jest nazywana JWT. Przykładowy

[21] *Basic Authentication* - Uwierzytelnianie podstawowe to prosty schemat uwierzytelniania wbudowany w protokół HTTP. Klient wysyła żądanie HTTP z nagłówkiem *Authorization*, który zawiera słowo *Basic*, po którym następuje spacja i łańcuch znaków *"nazwa_użytkownika":"hasło"* zakodowane algorytmem Base64.

[22] *OpenID Connect* - warstwa tożsamości zbudowana na podstawie struktury OAuth 2.0. Umożliwia aplikacjom innych firm na weryfikację tożsamości użytkownika końcowego i uzyskiwanie podstawowych informacji o profilu użytkownika.

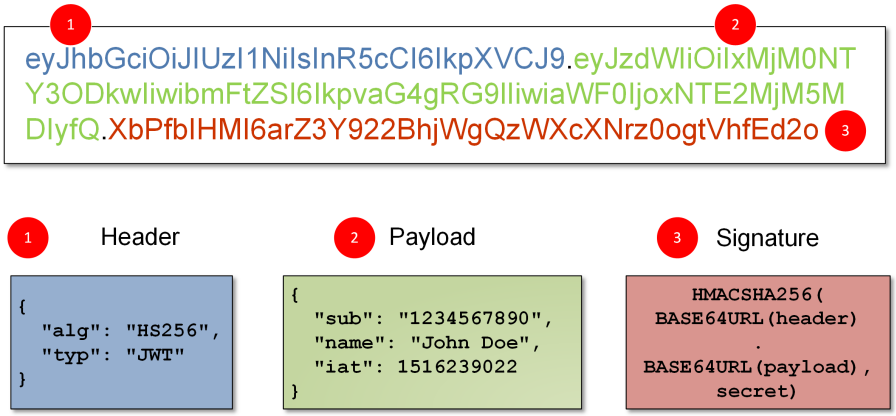
[23] *OAuth2.0* - standardowy protokół autoryzacji. Koncentruje się na prostocie obsługi klienta, zapewniając jednocześnie określone przepływy autoryzacji dla aplikacji internetowych, aplikacji komputerowych, telefonów komórkowych i urządzeń domowych.

[24] *JSON (ang. JavaScript Object Notation)* - prosty format wymiany danych. Zapis i odczyt danych w tym formacie jest łatwy do opanowania przez ludzi.

[25] *RFC (ang. Request for Comments)* - zbiór technicznych oraz organizacyjnych dokumentów mających formę memorandum związanych z Internetem, oraz sieciami komputerowymi.

[26] M. Jones, J. Bradley, N. Sakimura, *JSON Web Token (JWT)* z 05.2015 roku, https://tools.ietf.org/html/rfc7519 (dostęp: 14.07.2020).

JSON Web Token zaprezentowano na rysunku 1.



Rysunek 6: Przykład prostego JWT.
Źródło: `https://research.securitum.com/wp-content/uploads/sites/2/2019/10/jwt_ng1_en.png`

Zaprezentowany JWT posiada długi ciąg znaków rozdzielony kropkami. Kropki dzielą go na 3 sekcje, których struktura prezentuje się następująco:

```
Header . Payload . Signature
```

- *nagłówek (ang. header)* – przechowuje on informacje na temat algorytmu szyfrowania oraz typie tokena.
- *zawartość (ang. payload)* – dowolny przekazywany ładunek za pomocą deklaracji. Istnieją trzy typy deklaracji: *registered, public* i *private*. Najczęściej są w nim przechowywane informacje na temat roli i praw użytkownika, czy też długości życia tokena.
- *podpis (ang. signature)* – podpis cyfrowy, który składa się z zaszyfrowanego *Header* i *Paylod*. Stanowi on sumę kontrolną.

Wszystkie wymienione elementy struktury JWT są zakodowane algorytmem *Base64URL*[27] podobnym do *Base64*[28] (z wyjątkiem drobnych różnic znakowych). Token, który trafia do serwera od klienta jest parsowany, a następnie weryfikowany pod kątem prawidłowości, uprawnień, ważności, *TTL*[29] i innych. Na podstawie tokena, który jest przechowywany po stronie klienta,

[27] *Base64URL* - modyfikacja głównego standardu *Base64*, której celem jest możliwość wykorzystania wyniku kodowania jako nazwy pliku lub adresu URL. Base64URL jest opisany w *RFC 4648 § 5* (`https://tools.ietf.org/html/rfc4648#section-5`).

[28] *Base64* - rodzaj kodowania transportowego, zmodyfikowanego pod kątem zwiększenia przenośności wersja kodowania *uuencode*. Kodowanie to zostało zdefiniowane w dokumencie *RFC 4648* (`https://tools.ietf.org/html/rfc4648`).

[29] *TTL (ang. Time to live)* - okres ważności pakietu danych lub innych danych (np. rekordu DNS), stosowany w sieciach komputerowych.

można uzyskać dostęp do zasobów serwera. Najczęściej JWT jest stosowany do autoryzacji przy dostępie do API. Mamy możliwość rozkodować go również za pomocą serwisu `https://jwt.io/`. Dzięki temu procesowi widzimy w pełni czytelne sekcje nagłówka oraz zawartości [5].

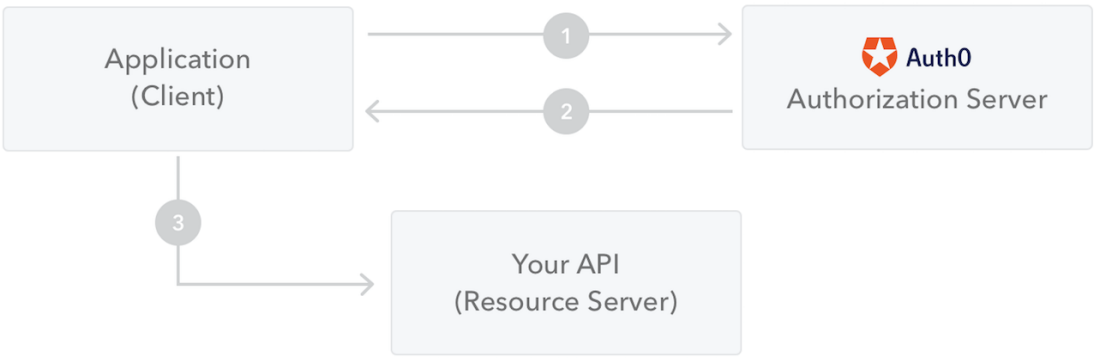
Wykorzystanie JSON Web Token w aplikacji datamaster-service

Podczas autentykacji, gdy użytkownik pomyślnie zaloguje się do aplikacji przy użyciu poświadczeń, zostanie zwrócony token. Ilekroć użytkownik chce uzyskać dostęp do chronionej ścieżki lub zasobu, agent użytkownika powinien wysłać JWT zawierający w nagłówku *Authorization* wraz ze schematem *Bearer*. Treść nagłówka powinna wyglądać następująco:

```
Authorization: Bearer <token>
```

W niektórych przypadkach może to być *bezstanowy (ang. stateless)* mechanizm autoryzacji. Chronione ścieżki serwera sprawdzą poprawność JWT w nagłówku Authorization, a jeśli jest obecny oraz prawidłowy, użytkownik będzie miał dostęp do chronionych zasobów. Gdy JWT zawiera niezbędne dane, potrzeba wykonania zapytania do bazy danych w celu wykonania niektórych operacji może być ograniczona, choć nie zawsze tak jest.

Jeżeli token zostanie wysłany w nagłówku Authorization, udostępnianie zasobów między źródłami (*CORS*[30]) nie będzie stanowiło problemu, ponieważ nie wykorzystuje *plików cookie*. Poniższy diagram pokazuje, w jaki sposób JWT jest tworzony i wykorzystywany do uzyskiwania dostępu do zasobów API:



Rysunek 7: Diagram przepływu JSON Web Tokena.
Źródło: `https://cdn2.auth0.com/docs/media/articles/api-auth/client-credentials-grant.png`

[30] *Cross-Origin Resource Sharing (CORS)* - mechanizm, który wykorzystuje dodatkowe nagłówki HTTP do informowania przeglądarek, aby dały aplikacji internetowej działającej w jednym miejscu, dostęp do wybranych zasobów z innego źródła. Aplikacja internetowa wykonuje żądanie HTTP między źródłami, gdy żąda zasobu, który ma inne pochodzenie (domenę, protokół lub port) od własnego.

1. Aplikacja lub klient żąda autoryzacji do serwera. Proces ten odbywa się poprzez jeden z przepływów autoryzacji. Na przykład typowa aplikacja internetowa zgodna z *OpenID Connect* przejdzie przez punkt końcowy */oauth/authorize* przy użyciu przepływu kodu autoryzacji.

2. Po udzieleniu autoryzacji serwer zwraca token dostępu do aplikacji.

3. Aplikacja korzysta z tokena dostępu, aby uzyskać dostęp do chronionego zasobu API.

Należy pamiętać, że w przypadku podpisanych tokenów wszystkie informacje zawarte w tokenie są udostępniane użytkownikom lub innym podmiotom, nawet jeśli nie są w stanie ich zmienić. Oznacza to, że nie należy umieszczać tajnych informacji w tokenie.

4.1.1.5 RESTful Web Services

REST (Representational State Transfer) API to jeden z modeli, z których możemy korzystać całymi latami, kompletnie nie zdając sobie sprawy z tego, jak wyglądają podstawy jego działania. Został on stworzony do komunikacji, pomiędzy urządzeniami wykorzystując istniejące już protokoły. Dzięki temu programista nie musi dołączać dodatkowych bibliotek, a aplikacja będzie mogła wykorzystać istniejące już protokoły jak na przykład *HTTP*. *HTTP* jest powszechnie dostępnym i wykorzystywanym protokołem komunikacji. W przypadku *RESTful API* jest on najczęściej wykorzystywanym protokołem.

REST sam w sobie nie jest frameworkiem, jest to zbiór zasad, dobrych praktyk, które działają jako drogowskaz. Dlatego aplikacje oparte o *RESTful API* są bardzo elastyczne. Mogą obsługiwać wiele rodzajów połączeń oraz zwracać kilka typów danych. Zachowanie zgodności z architekturą *RESTful* spoczywa na barkach programisty, właśnie dlatego przy ocenie danego api należy wspierać się sześcioma kluczowymi elementami, które zostały zdefiniowane przez *Roy'a Fielding'a*:

1. *Client-Server* – architektura typu Klient – Serwer.

2. *Cache* – buforowanie danych po stronie klienta tam, gdzie to możliwe.

3. *Stateless* – poszczególne połączenia są od siebie niezależne.

4. *Uniform Interface* – separacja implementacji po stronie klienta oraz serwera poprzez zastosowanie jednolitego interfejsu.

5. *Layered System* – dzielnie architektury REST na współdziałające ze sobą warstwy.

6. *Code on Demand* – pozwala na przesyłanie nowych fragmentów kodu poprzez api.

Spring Boot zapewnia bardzo dobre wsparcie przy tworzeniu RESTful Web Services. Zbudowanie aplikacji w oparciu o tę architekturę wymaga dodania zależności *spring-boot-starter-web* do pliku konfiguracyjnego kompilacji.

Przed przystąpieniem do budowy RESTful Web Services zaleca się znajomość następujących adnotacji:

- *Rest Controller (adnotacja @RestController)* - odpowiedzialny za definiowanie RESTful web services. Obsługuje JSON, XML i niestandardową odpowiedź.
- *Request Mapping (adnotacja @RequestMapping)* - odpowiedzialny za definiowanie identyfikatora *URI* żądania w celu uzyskania dostępu do punktów końcowych *REST Endpoints*. Odpowiedzialny również za odebranie oraz wytworzenia obiektu żądania.
- *Request Body (adnotacja @RequestBody)* - odpowiedzialny za zdefiniowanie typu zawartości ciała żądania.
- *Path Variable (adnotacja @PathVariable)* - służy do definiowania niestandardowego lub dynamicznego identyfikatora *URI* żądania.
- *Request Parameter (adnotacja @RequestParam)* - służy do odczytu parametrów żądania z adresu *URL* żądania.

W architekturze REST odwołujemy się do tzw. *endpoints (punktów końcowych)*. Są to adresy, które pozwalają na dostęp do informacji, powinny jednoznacznie wskazywać, do jakiego zasobu się odwołują. Z ich budowy powinniśmy wiedzieć, jaki konkretnie zasób otrzymamy. Struktura punktu końcowego powinna mieć następującą budowę:

```
http://<adresIpServera>:<portAplikacji>/<scieckaWywolania>
```

Największą z zalet korzystania z tej architektury jest przede wszystkim uniwersalność. Istnieje możliwość utworzenia jednego API, z którego będzie korzystała zarówno aplikacja, jak i strona internetowa [6].

4.1.1.6 Swagger

Swagger jest to biblioteka zapewniająca zestaw narzędzi niezbędnych do projektowania, tworzenia, dokumentowania i wizualizacji usług REST API. Sprawia on, że projektowanie oraz obsługa interfejsu API jest znacznie prostsze. Nazwa *Swagger* wywodzi się od konkretnego produktu, jakim jest *Swagger UI* (`https://swagger.io/tools/swagger-ui/`). Genialny w swoich możliwościach i łatwy w zastosowaniu.

Interfejs *Swagger UI* pozwala każdemu, zarówno zespołowi programistów, jak i użytkownikom aplikacji na wizualizację zasobów API i korzystanie z nich bez konieczności posiadania zewnętrznych aplikacji. Ponadto jest on generowany automatycznie na podstawie specyfikacji *OpenAPI*[31]. Przez co zwalnia programistów z potrzeby aktualizacji dokumentacji API.

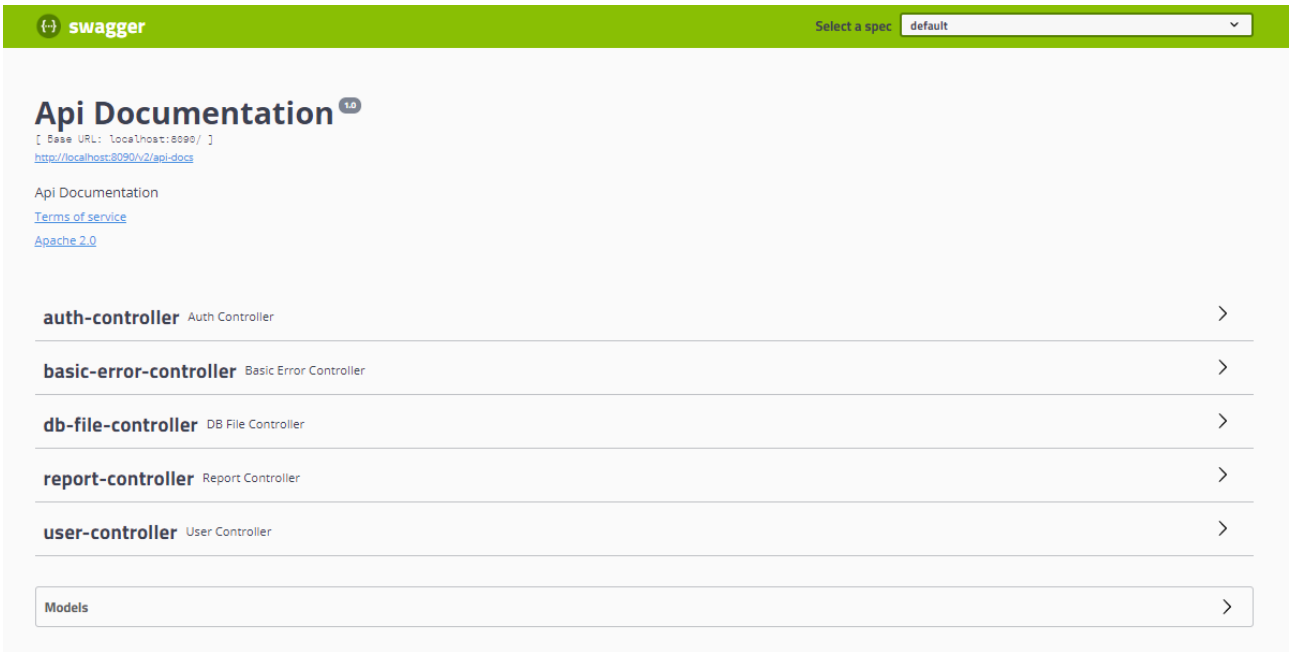
[31] *OpenApi Specification (OAS)* - definiuje standardowy, niezależny od języka interfejs do RESTful API, który umożliwia zarówno ludziom, jak i komputerom odkrywanie i zrozumienie możliwości usługi bez dostępu do kodu

Integracja *Swagger UI* z aplikacją napisaną w *Spring Boot* jest bardzo prosta. Wystarczy dodać dwie zależności:

- *springfox-swagger2*
- *springfox-swagger-ui*

Pierwsza z nich dostarcza możliwości wykorzystania narzędzia w podstawowej wersji, natomiast druga dostarcza nakładkę graficzną. Po dodaniu zależności tworzymy klasę konfiguracyjną zawierającą dwie adnotacje: *@Configuration* oraz *@EnableSwagger2*.

Uruchomienie aplikacji przy domyślnej konfiguracji *Spring Boot* pozwala przejść pod wskazany adres `localhost:8080/swagger-ui.html`, pod którym mamy dostępną dokumentację REST API.



Rysunek 8: Interfejs Swagger UI aplikacji datamaster-service.

Swagger UI pozwalana na wywołanie wszelkich metod REST z odpowiednimi parametrami, ciałem oraz nagłówkiem. Dokumentacja w postaci *JSON* dostępna jest pod adresem `localhost:8080/v2/api-docs` [7].

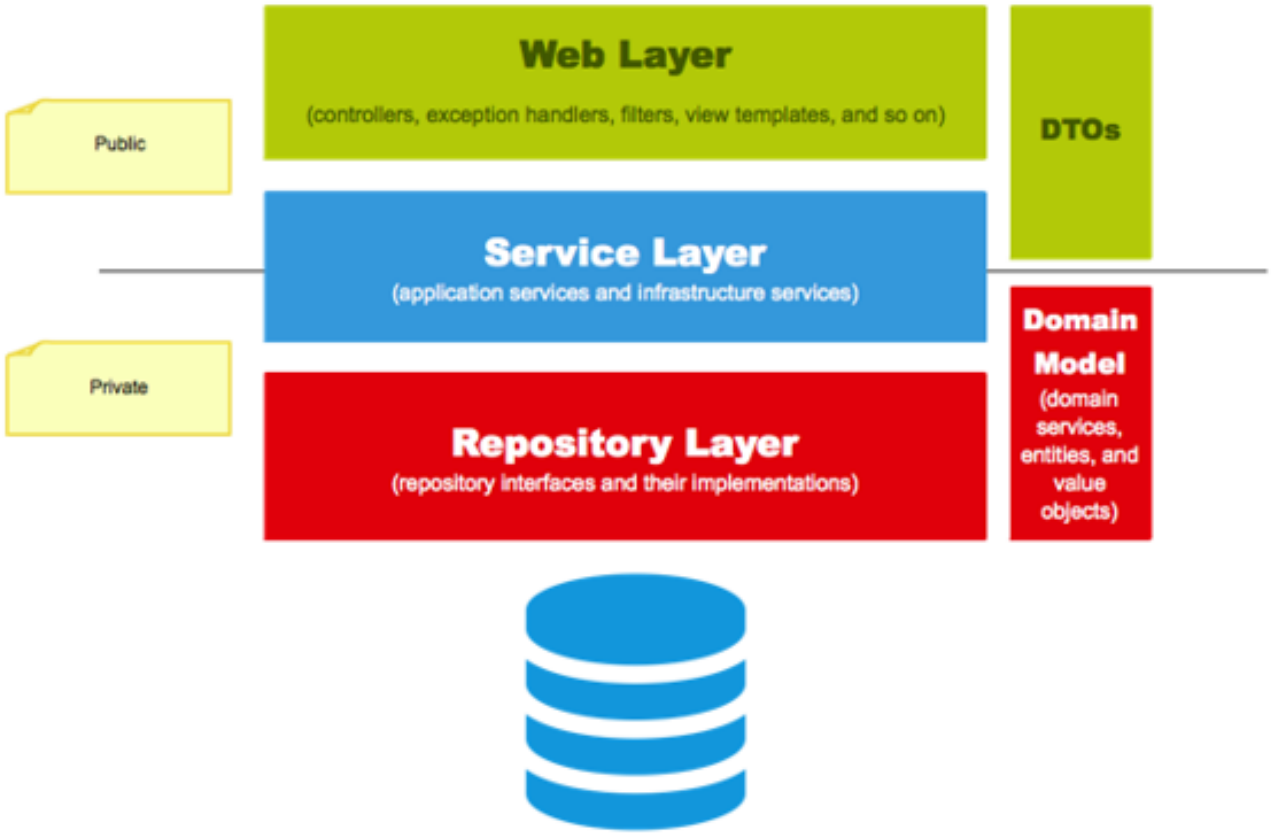
4.1.2 Struktura projektu

4.1.2.1 Architektura serwisu

Datamaster-service został podzielony na odpowiednie warstwy logiczne, dzięki którym można znacznie prościej zarządzać oraz skalować aplikację. Warstwy podzielone są zgodnie z ich

źródłowego, dokumentacji lub kontroli ruchu sieciowego. Prawidłowo zdefiniowany pozwala klientowi zrozumieć usługę zdalną i wchodzić z nią w interakcję przy minimalnej logice implementacji.

obowiązkami, zapewniając przy tym dobrą organizację kodu źródłowego oraz ułatwiając pracę programiście pod względem ich wyszukiwania. Architektura projektu wykracza również poza organizację kodu. Wprowadzenie warstw do aplikacji wymusza ograniczenia komunikacyjne między nimi. Dzięki czemu zachowujemy zasadę *hermetyzacji*[32] w projekcie. Każda z warstw po wymianie danych ma wpływ tylko na współpracujące z nią inne warstwy.



Rysunek 9: Architektura warstwowa serwisu datamaster-service.
Źródło: `https://www.petrikainulainen.net/wp-content/uploads/spring-web-app-architecture.png`

Logika aplikacji została podzielona na trzy warstwy (*ang. 3-tier architecture*):

- *Warstwa internetowa (ang. Web Layer)* to najwyższa warstwa aplikacji internetowej. Odpowiada za przetwarzanie danych wejściowych użytkownika i zwracanie poprawnej odpowiedzi z powrotem do użytkownika. Musi ona również obsługiwać wyjątki wyrzucane przez inne warstwy. Ponieważ warstwa internetowa jest punktem wejścia naszej aplikacji, musi dbać o uwierzytelnianie i działać jako pierwsza linia obrony przed nieautoryzowanymi użytkownikami.

[32] *Hermetyzacja (enkapsulacja)* - pozwala ograniczyć dostęp do danych składowych, metod obiektów klas poprzez ich ukrycie modyfikatorami dostępu.

- *Warstwa usług (ang. Service Layer)* znajduje się poniżej warstwy internetowej. Działa jako granica transakcji i zawiera zarówno usługi aplikacyjne, jak i infrastrukturalne. Usługi aplikacji zapewniają publiczne API warstwy usług. Działają również jako granica transakcji i są odpowiedzialne za autoryzację. Usługi infrastrukturalne zawierają *"kod hydrauliczny"*, który komunikuje się z zasobami zewnętrznymi, takimi jak *systemy plików*, *bazy danych* lub *serwery poczty e-mail*. Często te metody są używane przez więcej niż jedną usługę aplikacji.

- *Warstwa repozytorium (ang. Repository Layer)* to najniższa warstwa aplikacji internetowej. Odpowiada za komunikację z używanymi nośnikami danych.

Wymienione wyżej warstwy dzielimy również pod względem dostępności. Warstwy publiczne chronią warstwy prywatne. Jeżeli ujawni się model domenowy światu zewnętrznemu, nie można go zmienić bez zerwania innych rzeczy, które od niego zależą. Jeżeli wykorzystuje się *DTO*, można zmieniać model domenowy, o ile nie wprowadza żadnych zmian w *DTO*. Dodatkowo Model domeny określa wewnętrzny model aplikacji. W przpadku wystawienia takiego modelu światu zewnętrznemu klienci musieliby wiedzieć, jak go używać. Innymi słowy, klienci aplikacji musieliby zająć się rzeczami, które do nich nie należą. Korzystanie z *DTO* pozwala ukryć ten model przed klientami aplikacji, zapewniając łatwiejsze i czystsze API.

Architektura aplikacji wykorzystuje również wzorzec projektowy *Data Transfer Object* należący do grupy *wzorców dystrybucji*. Głównym zadaniem DTO jest transfer danych oraz ich ochrona pomiędzy systemami. Przykładem może być pakiet *com.agh.dataminingservice.payload* występujący w aplikacji i zawierający klasy DTO. Klient ma do nich dostęp, gdyż musi wiedzieć w jaki sposób skonstruować ciało żądania.

Ostatnim elementem architektury, który ma wpływ na działanie aplikacji jest *model domenowy (ang. Domain Model)*. Jest on koncepcyjnym modelem, który obejmuje zarówno zachowanie, jak i dane. Jest reprezentacją znaczących pojęć ze świata rzeczywistego odnoszących się do domeny, które należy modelować w oprogramowaniu. W aplikacji datamaster-service istnieje wiele klas należących do modelu domenowego. Znajdziemy je w pakiecie *com.agh.dataminingservice.model*. Wśród nich możemy wyróżnić takie modele jak *User, Role, Report, DBFile* oraz wiele innych.

4.1.2.2 Struktura katalogów - Pakiety aplikacji

Kod źródłowy projektu rozdzielony jest na pakiety Java. Dzięki nim zarządzanie plikami programu zwiększa wydajność programisty oraz pozwala na grupowanie klas, interfejsów, typów wyliczeniowych czy też adnotacji. W skład serwisu *datamaster-service* wchodzą następujące pakiety:

- *com.agh.dataminingservice.config*

Pakiet ten zawiera klasy konfiguracyjne projektu. Dzięki nim korzystamy z automatycznego uzupełniania danych podczas zapisu, wykorzystujemy wszystkie zabezpieczenia serwisu niezbędne do poprawnego działania aplikacji. Możemy również zbudować narzędzie do projektowania oraz obsługi interfejsu API, jakim jest *Swagger* oraz włączyć dostęp *CORS* do API z klienta serwisu datamaster-ui.

- *com.agh.dataminingservice.controller*

  Każda z klas znajdująca się w tym pakiecie zawiera adnotację *@RestController*. Oznaczamy nią klasy będące Restful-owymi kontrolerami czyli takie, które będą obsługiwały zapytania wysyłane poprzez przeglądarkę od klienta do API datamaster-service.

  W aplikacji możemy wyróżnić następujące kontrolery:

  – AuthController - Rest API do logowania oraz rejestracji użytkowników.

  – DBFileController - CRUD[33] Rest API do zapisywania oraz zarządzania plikami użytkowników w bazie danych.

  – ReportController - CRUD Rest API do zapisywania oraz zarządzania raportami użytkowników w bazie danych.

  – UserController - Rest API odpowiedzialne za zarządzanie informacjami o istniejących profilach użytkowników.

- *com.agh.dataminingservice.exception*

  Pakiet exception zawiera klasy będące wyjątkami rzucanymi w trakcie działania programu tzw. *RuntimeException*. Stworzone na potrzeby poprawnego działania aplikacji w momencie wystąpienia nieoczekiwanych lub nieprawidłowych efektów pracy programu. Większość wyjątków zaimplementowanych w aplikacji oprócz przekazywanej wiadomości zwrotnej zawiera również status odpowiedzi.

- *com.agh.dataminingservice.model*

  Model jest pakietem zawierającym klasy, typy wyliczeniowe zdefiniowane przez *JPA*[34], czyli standard z rodziny tzw. *ORM (ang. Object-Relational Mapping)*, co oznacza mapowanie z modelu obiektowego do modelu relacyjnego. JPA stanowi zbiór kontraktów tzn. opisów, jakie funkcje mają być realizowane i w jaki sposób.

  Dzięki JPA klasy oznaczone adnotacją *@Entity* zostaną zmapowane do encji bazodanowej. W naszej aplikacji posiadamy 5 standardowych encji: *users, roles, user-roles, reports* oraz *files*.

[33] *CRUD (create, read, update, delete)* - cztery podstawowe funkcje w aplikacjach korzystających z pamięci trwałej, które umożliwiają zarządzanie nią.

[34] *Java Persistence API (JPA)* - oficjalny standard mapowania obiektowo-relacyjnego (ORM) firmy Sun Microsystems dla języka programowania Java.

- *com.agh.dataminingservice.model.audit*

  Pakiet audit wywodzący się z pakietu model zawiera klasy abstrakcyjne odpowiedzialne za automatyczne wypełnianie pól danych, jakimi są data utworzenia, modyfikacji obiektu encji oraz pól zawierających identyfikator osoby tworzącej, modyfikującej wybrany obiekt encji. Aplikacja wykorzystuje klasy pakietu audit w klasie User, którego encja bazodanowa *users* zawiera wymienione powyżej pola.

- *com.agh.dataminingservice.payload*

  Payload to pakiet zawierający klasy odpowiedzialne za transfer danych pomiędzy systemami. Stworzony na bazie *DTO*. Pakiet ten jest kontenerem ułatwiającym dostęp, ochronę oraz przesyłanie danych.

  Wśród najważniejszych klas żądań tego pakietu wyróżniamy: *LoginRequest, SignUpRequest* oraz *UploadReportsRequest*.

  Natomiast klasy odpowiedzi z serwera przesyłające dane do klienta stanowią: *ApiResponse, FileDto, JwtAuthenticationResponse, ReportResponse, UploadFileResponse, UploadReportsResponse, UserIdentityAvailability, UserProfile, UserReportsResponse, UserRepositoryFiles* oraz *UserSummary*.

  Można zauważyć, że ilość klas stworzonych na potrzeby odpowiedzi z serwera jest znacznie większa niż klas żądań. Spowodowane jest to faktem, że często żądania do serwera nie potrzebują konkretnych struktur danych do przetworzenia zapytania na serwerze. Natomiast liczba klas odpowiedzi dlatego jest znacznie większa, ponieważ aplikacji chroni dostęp do informacji o strukturach bazy danych, zabezpiecza informacje newralgiczne takie jak np. hasło użytkownika przed wyciekami.

- *com.agh.dataminingservice.repository*

  Interfejsy znajdujące się w pakiecie *repository* zawierają adnotację *@Repository*. Dzięki temu aplikacja przekazuje *Spring Data*, że dany interfejs odpowiada za dostęp do danych. Wszystkie rozszerzają interfejs*JpaRepository*, które samo w sobie zawiera funkcjonalności *CrudRepository* oraz *PagingAndSortingRepository*. Taka konfiguracja pozwala na udostępnienie metod do tworzenia, aktualizacji czy też usuwania rekordów z bazy bez ich definicji w utworzonym interfejsie.

  Aplikacja datamaster-service wyróżnia następujące interfejsy: *UserRepository, RoleRepository, ReportRepository* oraz *DBFileRepository*. Ostatni wymieniony wykorzystuje niestandardowe zapytanie oznaczone adnotacją *@Query*, które wybiera pliki z informacjami o identyfikatorze, nazwie i typie przy użyciu zapytania *JPQL*[35]. Dzięki takim zabiegom

[35] *Java Persistence Query Language (JPQL)* - język zdefiniowany w specyfikacji JPA. Służy do tworzenia zapytań dotyczących encji do przechowywania w relacyjnej bazie danych. JPQL jest rozwijany w oparciu o składnię SQL, ale nie wpływa on bezpośrednio na bazę danych.

jesteśmy w stanie kilkukrotnie zmniejszyć czas wykonywanych zapytań oraz odpowiedzi z bazy do serwera, co znacząco wpływa na jakość działania aplikacji.

- *com.agh.dataminingservice.security*

  Security to pakiet zawierający klasy oraz definicje adnotacji odpowiedzialne za konfigurację zabezpieczeń, poprawne działanie *Spring Security* oraz *JSON Web Token* w aplikacji.

  W skład klas odpowiedzialnych za działanie *Spring Security* oraz *JWT* wchodzą:

  – *JwtTokenProvider* - Narzędzie do generowania i weryfikacji JWT.

  – *JwtAuthenticationFilter* - Niestandardowy filtr uwierzytelniania Spring Security.

  – *JwtAuthenticationEntryPoint* - Klasa służy do zwracania nieautoryzowanego błędu 401 klientom próbującym uzyskać dostęp do chronionego zasobu bez uwierzytelnienia.

  – *UserPrincipal* - Spring Security używa informacji przechowywanych w obiekcie UserPrincipal do przeprowadzenia uwierzytelnienia i autoryzacji.

  – *CustomUserDetailsService* - W celu uwierzytelnienia użytkownika lub przeprowadzenia testów opartych na rolach. Spring Security musi w jakiś sposób załadować szczegóły użytkowników. W tym celu implementuje interfejs o nazwie *UserDetailsService*, który ma jedną metodę ładującą użytkownika na podstawie nazwy *loadUserByUsername(String)*.

  Dodatkowo pakiet security posiada definicję adnotacji *CurrentUser*. Spring Security zapewnia adnotację o nazwie *AuthenticationPrincipal*, która umożliwia dostęp do aktualnie uwierzytelnionego użytkownik w aplikacji. Nasza adnotacja opakowuje *AuthenticationPrincipal*, aby być niezależna od frameworka Spring.

- *com.agh.dataminingservice.service*

  Ostatnim typem stereotypu Spring wykorzystanym w aplikacji jest adnotacja *@Service*. Wszystkie klasy nią oznaczone znajdziemy w pakiecie service. Mówi nam ona, że wybrana klasa jest serwisem w warstwie logiki biznesowej. Projekt definiuje dwie klasy:

  – *DBFileStorageService* - Klasa odpowiedzialna za przetwarzanie magazynu plików.

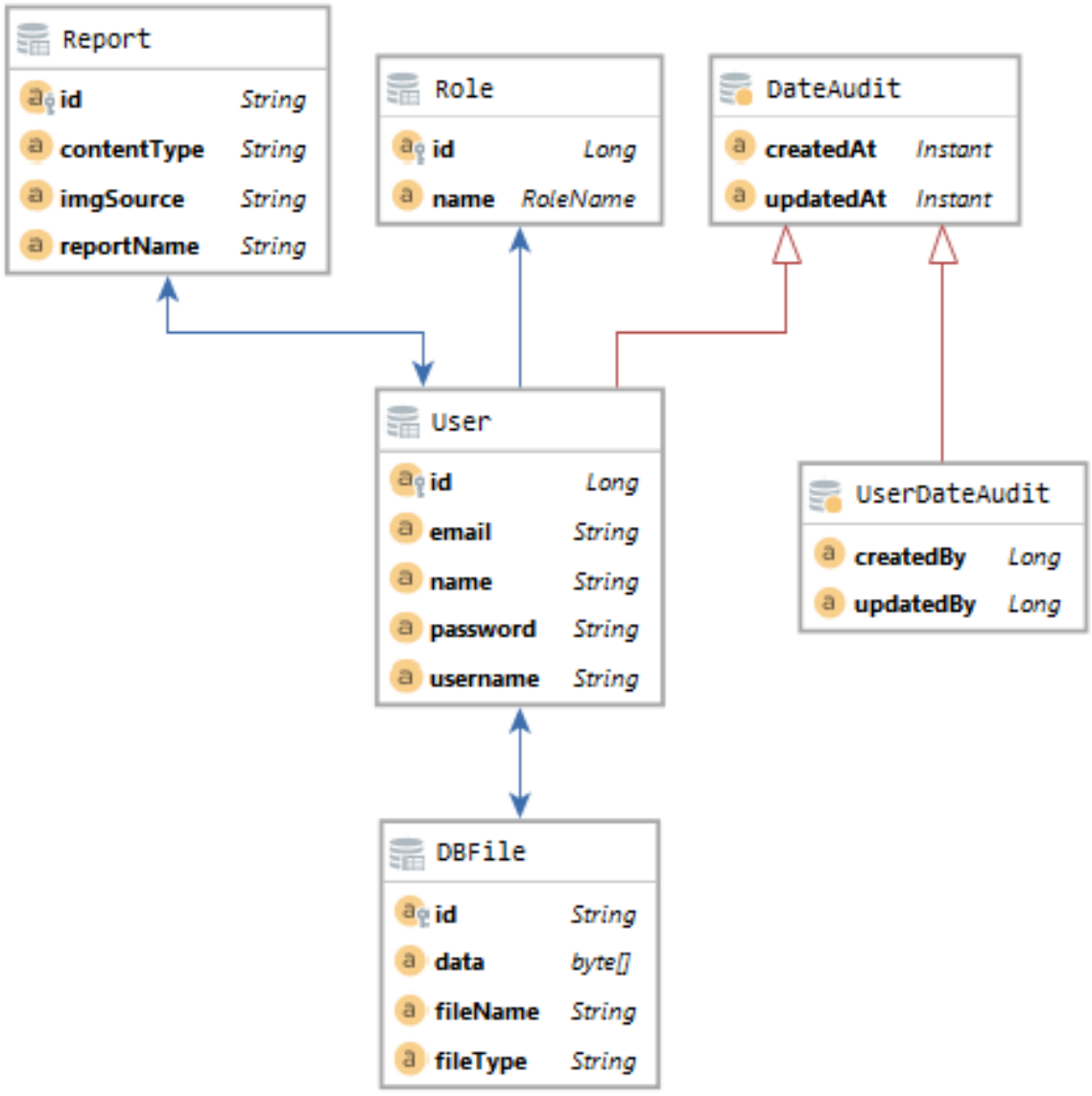
  – *ReportStorageService* - Klasa odpowiedzialna za przetwarzanie plików raportów.

- *com.agh.dataminingservice.util*

  Pakiet util zawiera pomocnicze klasy oraz interfejsy użyteczne w działaniu aplikacji. Wśród nich znajdziemy *AppConstants* ze stałymi aplikacji, czy też *ModelMapper* do mapowania obiektów modelu.

4.1.2.3 Struktura modelu domenowego

Struktura bazy danych generowana jest na podstawie domenowego modelu aplikacji. Jest to możliwe dzięki zastosowaniu *JPA*, czyli Java Persistence API oraz *ORM* czyli Object-Relational Mapping. *JPA* określa mechanizmy pozwalające na zarządzanie zawartością bazy danych poprzez obiekty Javowe. Innymi słowy, instancje klas odpowiadają wierszom w bazie danych. Natomiast *ORM* czyli *"mapowanie obiektowo-relacyjne"* pozwala zmapować obiekty Javowe na zawartość bazy danych. Najczęściej stosowaną implementacją *JPA* jest *Hibernate*[36] i to właśnie on został wykorzystany w tym celu.



Rysunek 10: Diagram związków modelu domenowego aplikacji.

Przy pomocy adnotacji zastosowanych w klasach pakietu *com.agh.dataminingservice.model*

[36] *Hibernate* - framework do realizacji warstwy dostępu do danych (ang. persistence layer). Zapewnia on przede wszystkim translację danych pomiędzy relacyjną bazą danych a światem obiektowym.

mapowany jest obiekt klasy na wiersz w tabeli. Przykładem może być obiekt klasy *User* mapowany do encji o nazwie *users*.

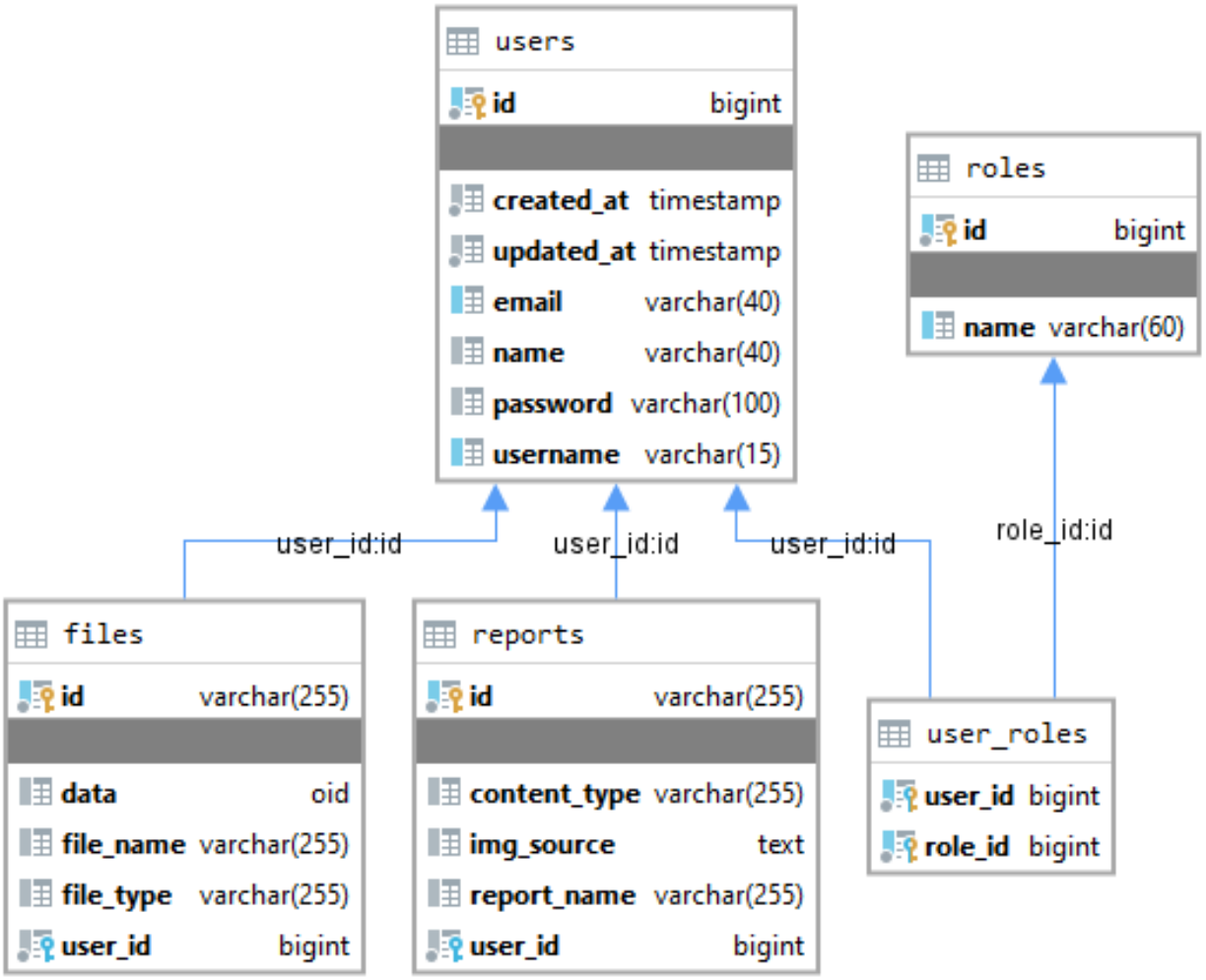
Schemat przedstawiony na rysunku 5 prezentuje diagram związków klas modelu aplikacji wygenerowany na podstawie kodu Javy. Dodatkowa konfiguracja *ORM*, dostęp do bazy oraz zarządzanie nią znajduje się w zasobach aplikacji *(ang. resources)*, a dokładniej w plikach *application-dev.properties* oraz *application-prod.properties*.

Kierunek strzałek znajdujący się na diagramie oznacza dostęp jednego modelu do drugiego. Na przykład model *User* ma dostęp do modelu *Role*. W przeciwnym kierunku dostępność ta już nie jest możliwa, czyli obiekt klasy *Role* nie może uzyskać informacji o instancji klasy *User*. Jeżeli jednak weźmiemy pod uwagę model *User* oraz *Report* zauważymy relację obustronną, czyli jednakowo instancja *User* ma dostęp do raportów zapisanych w zbiorze *Report*. Tak samo raport ma dostęp do instancji użytkownika w klasie *User*.

Kolor strzałek występujący pomiędzy modelami aplikacji również ma znaczenie. Czerwony oznacza, że relacja modeli opiera się na implementacji lub rozszerzeniu jednego modelu przez drugi. Przykładem jest klasa *User* rozszerzająca klasę *DateAudit*. Kolor niebieski oznacza natomiast relacje występujące w systemach bazodanowych, takie jak: *jeden-jeden*, *jeden-wielu*, czy też *wiele-wielu*.

4.1.2.4 Diagram związków encji (ERD)

Baza danych zarządza 5 encjami: *users, roles, user_role, files* oraz *reports*. Zostały one stworzone przy użyciu narzędzia *Hibernate* mapując model domenowy aplikacji do encji bazodanowych.



Rysunek 11: Diagram zwiazków encji (ERD) bazy danych PostgrSQL.

Diagram związków encji prezentuje jakie relacje występują między nimi. W celu lepszej analizy systemu opisano każdą z nich:

- *users-roles*: relacja *wiele do wielu (ang. Many to many).*

  W systemie zarządzania relacyjnymi bazami danych takie relacje są zwykle realizowane za pomocą *tabeli asocjacyjnej*[37] przy użyciu dwóch kluczy obcych. W naszej bazie danych tabela asocjacyjna zwana inaczej tabelą łączenia wyrażona jest przez encję *user_roles*. Zawiera ona dwa klucze obce *user_id* oraz *role_id*.

  Aplikacja wykorzystuje proces autoryzacji, czyli nadaje podmiotowi dostęp do zasobu. Jest to możliwe dzięki zastosowaniu koncepcji *"ról"*. Każdy użytkownik systemu musi posiadać spis ról przypisanych do niego. Użytkownikowi przyznawany jest dostęp do wybranej akcji tylko i wyłącznie wtedy, gdy posiada on wymaganą rolę. W systemie każdy użytkownik

[37] *Tabela asocjacyjna* - struktura pozwalająca na przechowywanie par typu klucz-wartość. Dzięki kluczowi w prosty sposób znajdziemy wartość z nią skojarzoną.

może mieć więcej niż jedną rolę. Dlatego wykorzystano w tym celu tabelę asocjacyjną z relacją wiele do wielu, ponieważ rolę można przypisać do wielu użytkowników systemu.

- *users-files*: relacja *jeden do wielu (ang. One to many).*

  Każdy użytkownik systemu ma prawo do przechowywania oraz zarządzania plikami danych. Najczęściej pliki te są formatu CSV. Natomiast zdarzają się wyjątki takie jak podręcznik użytkownika zapisany w formacie PDF. System nie narzuca ograniczeń do rodzaju przechowywanych plików.

  Relacja, jakiej użyto w tym wypadku to jeden do wielu, oznacza ona, że każdy użytkownik ma prawo posiadać więcej niż jeden plik. Natomiast pliki zapisane przez użytkownika mogą należeć tylko do niego.

- *users-reports*: relacja *jeden do wielu (ang. One to many).*

  W tym przypadku mamy podobną sytuację jak w relacji powyżej *users-files*. Raporty kodowane są w Base64 i przechowywane w formacie PNG. Ułatwia to proces modyfikacji pliku do innych formatów takich jak na przykład PDF. Raportami są wyniki analiz zapisane przez użytkownika do aplikacji. Dlatego tutaj również zastosowano relację jeden do wielu, ponieważ użytkownik może zapisać i zarządzać wieloma raportami. A każdy pojedynczy wynik analizy należy tylko do użytkownika, który go zapisał.

4.1.2.5 System zarządzania relacyjną bazą danych

Aplikacja zbudowana jest z dwóch profili: *developerskiego* oraz *produkcyjnego*. Oba korzystają z relacyjnej bazy danych *PostgreSQL*. Baza developerska dostępna jest pod linkiem `jdbc:postgresql://157.230.25.219:5432/postgres`. Natomiast baza produkcyjna znajduje się w serwisie *ElephantSQL* pod linkiem `jdbc:postgresql://isilo.db.elephantsql.com:5432/ydtgkgoc`.

ElephantSQL automatyzuje każdą część konfiguracji i działania klastrów PostgreSQL. Dostępny na wszystkich głównych platformach chmurowych i aplikacyjnych na całym świecie. Zapewnia w pełni zautomatyzowane tworzenie kopi zapasowych wykonywanych codziennie, które są przechowywane w magazynie plików w chmurze, aby były zawsze dostępne. Dlatego właśnie zdecydowano się na użycie tego serwisu, jako platformy wykorzystującej dostęp do relacyjnej bazy danych PostgreSQL.

4.2 Serwis Datamaster-ui

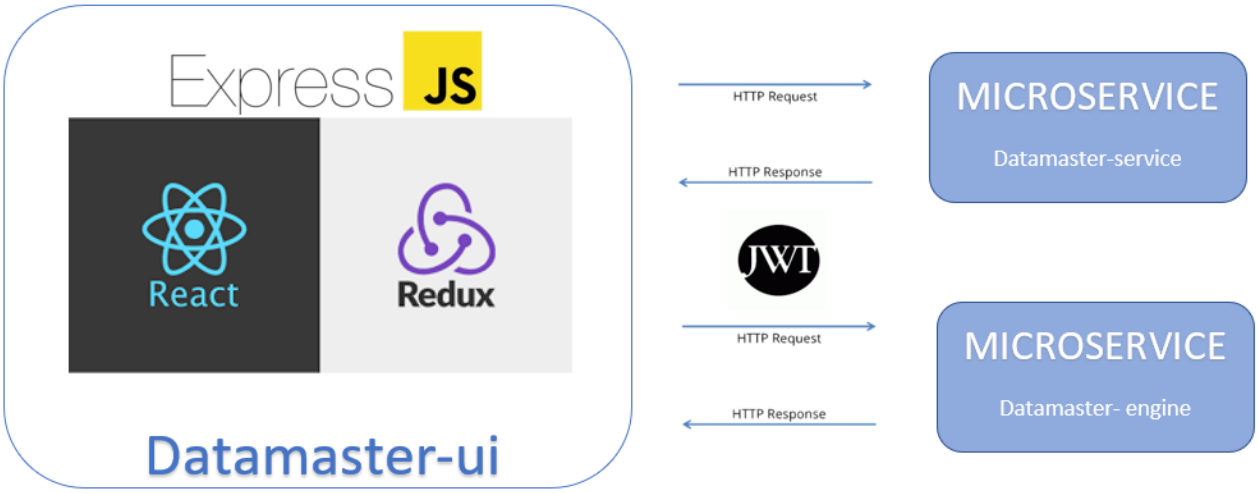
Serwis ten stanowi warstwę prezentacyjną stworzonej aplikacji. Został napisany przy użyciu frameworku *React.js*[38], *Express.js*[39] oraz biblioteki *Redux*[40]. Kod projektu został napisany przy

[38] *React.js* - framework *JavaScript* do pisania aplikacji frontendowych.

[39] *Express.js* - framework do aplikacji webowych udostępniający między innymi możliwość serwowania aplikacji.

[40] *Redux* - biblioteka do zarządzania stanem aplikacji.

użyciu *React'a* wraz z dodatkiem *Redux* do zarządzania globalnym stanem aplikacji, natomiast uruchamiany jest przy użyciu frameworku *Express.js* na porcie 8080. W celu pobrania informacji wykonywane są zapytania do mikroserwisów *Datamaster-engine* oraz *Datamaster-service*. Poniżej przedstawiono grafikę obrazującą sposób działania serwisu:



Rysunek 12: Struktura projektu *Datamaster-ui*
Źródło: opracowanie własne z użyciem ikon.
Ikona *JWT*: `https://cdn.worldvectorlogo.com/logos/jwt.svg`
Ikona *React i Redux*: `https://pushed.to/cokeSchlumpf/rethink-it/posts/web/best-practice-react-redux-project-structure.md`
Ikona *Express.js*: `https://medium.com/javascript-in-plain-english/3-express-js-features-you-need-to-know-8f78b0035f33`

4.2.1 Wyróżnione standardy i technologie

4.2.1.1 React.js

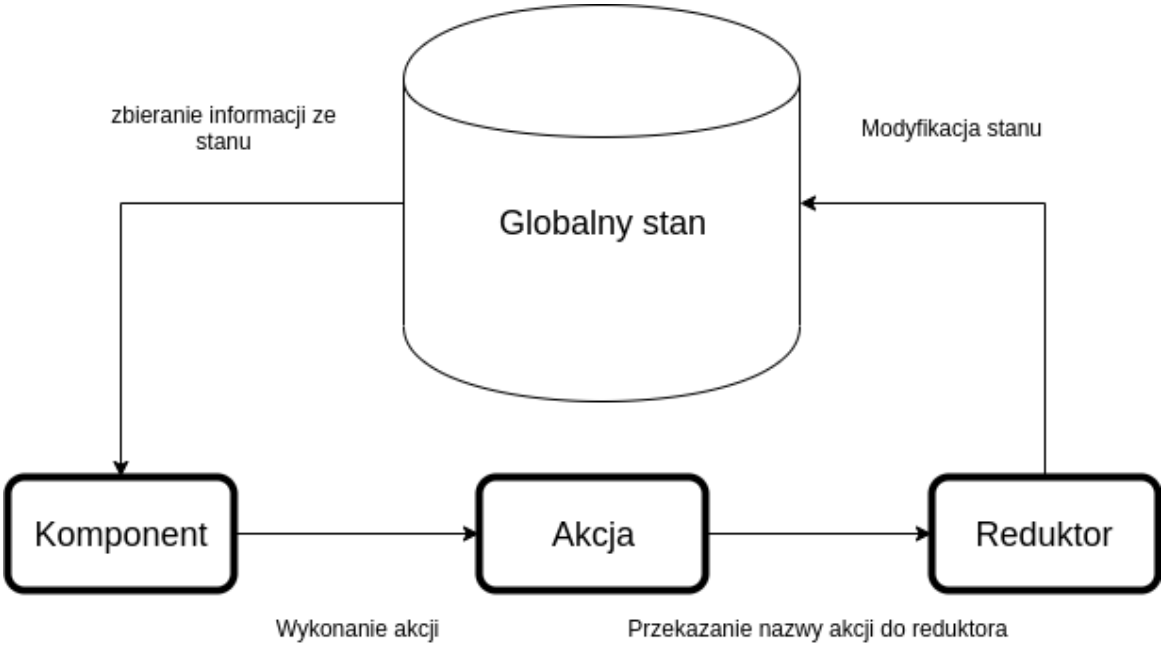
Do napisania interfejsu użytkownika wykorzystano technologię *React.js*, która umożliwia tworzenie komponentów (fragmentów wizualnych strony). Technologia ta wykorzystuje standard *JSX*[41] do renderowania komponentów. Komponenty można definiować za pomocą funkcji oraz klas *JavaScript*. Podstawową różnicą między tymi dwoma podejściami jest to, że komponenty funkcyjne są mniejsze, posiadają mniej tak zwanego *lukru składniowego*[42] oraz umożliwiają pisanie prostszego, bardziej modularnego kodu. W projekcie zdecydowano się użyć wyłącznie komponentów funkcyjnych z dodatkiem *Hooks*[43] wprowadzonym w wersji 16.8 frameworka

[41] *JSX* - standard stanowiący mieszaninę języka *html* i *JavaScript*
[42] *lukier składniowy* - przekształcenia składniowe mające na celu uproszczenie tworzonego kodu przez programistę.
[43] *Hooks* - dodatek do Reacta umożliwiający pisanie komponentów funkcyjnych z możliwościami komponentów klasowych (wcześniej nie było to możliwe).

React.js. Dzięki temu łatwiej można było wyodrębnić powtarzające się fragmenty kodu oraz tworzyć bardziej przejrzysty kod.

4.2.1.2 Redux

Redux jest technologią wprowadzającą funkcjonalność zarządzania globalnym stanem aplikacji. Stan ten można definiować jako pojemnik na dane, który jest obiektem *JavaScript* z odpowiednimi własnościami zdefiniowanymi przez programistę. Ma on swój ustalony stan początkowy, który następnie jest modyfikowany w odpowiedzi na zmiany w aplikacji takie jak np. zalogowanie się do systemu. W celu modyfikacji stanu programista musi najpierw zdefiniować tzw. akcję, która najczęściej sprowadza się do kodu wykonującego żądanie do serwera. W odpowiedzi na akcję oraz to, czy jest ona w trakcie, powiodła się, bądź też nie powiodła się, zostanie zmodyfikowany globalny stan aplikacji. Za tą modyfikację będzie odpowiedzialny tzw. reduktor, który posiada definicje dostępnych akcji oraz sposoby modyfikacji stanu w odpowiedzi na każdą z nich.



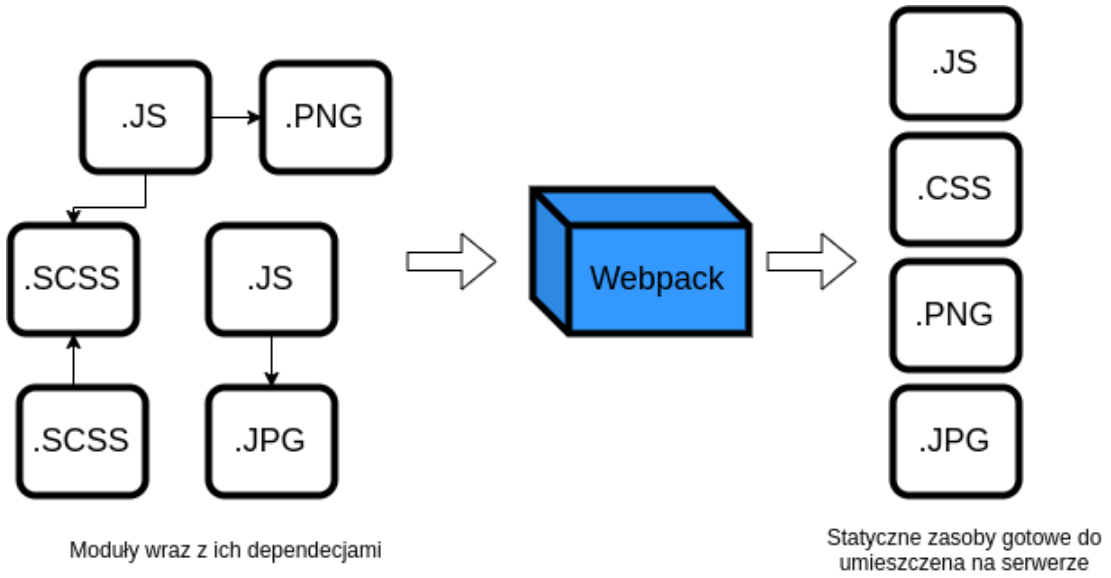
Rysunek 13: Diagram działania narzędzia *Redux*.
Źródło: opracowanie własne.

Redux umożliwia więc rozdzielenie logiki wykonywania akcji od pozostałego kodu aplikacji oraz wprowadza możliwość odwoływania się do stanu aplikacji w prosty sposób z dowolnego komponentu. Poza tym ułatwia również kontrolę nad aplikacją dzięki wypisywaniu logów wykonywanych akcji.

4.2.1.3 Webpack

Technologia ta zajmuje się przetwarzaniem modułów i ich zależności na statyczne pliki. Posiada

również wiele funkcjonalności pozwalających na optymalizację tworzonych plików wynikowych. Poniższy obrazek prezentuje działanie tego narzędzia.



Rysunek 14: Diagram działania narzędzia *Webpack* [8].
Źródło: opracowanie własne.

Wszelki kod *JavaScript* aplikacji jest modyfikowany i umieszczony najczęściej w jednym lub kilku plikach wynikowych. Style komponentów definiowane w języku *Sass* są przetwarzane do języka css zrozumiałego dla przeglądarki. Modyfikowane są również obrazki i czcionki. Wśród funkcjonalności *Webpack'a* które wykorzystano w projekcie, można wymienić:

- Zmniejszanie rozmiaru plików wynikowych.

  Przykładem może być zmieniony plik wynikowy *css*, aby zajmował mniej miejsca lub zoptymalizowana wielkość obrazka kosztem jego jakości.

- Dzielenie kodu na części.

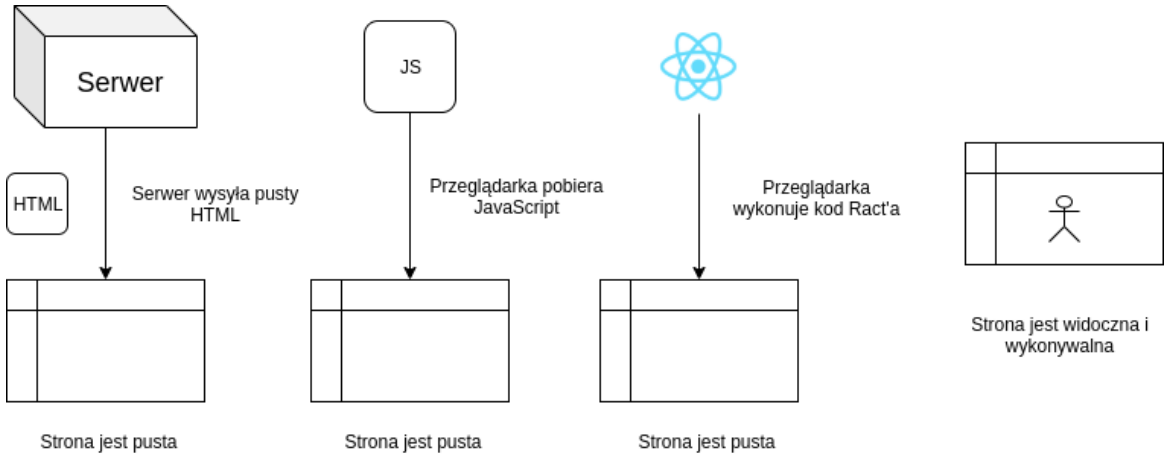
  Kod wynikowy *JavaScript* jest dzielony na wiele plików, dzięki czemu zoptymalizowany jest mechanizm pobierania plików z serwera. W przypadku jednego dużego pliku wynikowego, w sytuacji zmiany kodu w dowolnym miejscu w aplikacji, użytkownik musiałby pobierać cały plik jeszcze raz. W przypadku podzielonych plików zmianie ulegnie tylko ten plik, który powstał z obszaru kodu, w którym nastąpiła zmiana.

- Haszowane nazwy plików.

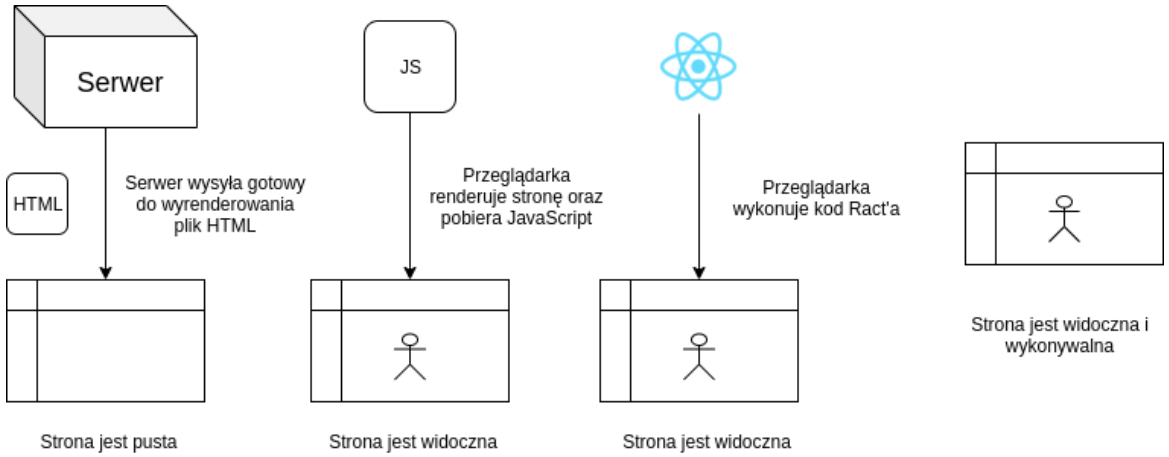
  Nazwy plików są generowane z haszem na podstawie kodu źródłowego. Oznacza to, że w przypadku zmiany kodu, zmianie również ulegnie nazwa pliku, dzięki czemu tylko zmienione części aplikacji zostaną pobrane z serwera, a pozostałe zostaną wyjęte z pamięci podręcznej.

4.2.1.4 Renderowanie po stronie serwera

Podstawowym podejściem do renderowania strony internetowej utworzonej przy użyciu frameworku *React.js* jest renderowanie po stronie klienta. Taką funkcjonalność mamy dostępną automatycznie po skorzystaniu ze skryptpów udostępnionych przez twórców *Reacta*. Renderowanie to polega na wysłaniu do klienta pliku *html* z elementem *<body>* zawierającym tylko jeden element potomny * *. Następnie wykonywany jest kod *JavaScript*, który umieszcza w tym elemencie zawartość strony. Cały proces nie jest natychmiastowy, przez co użytkownik przez krótką chwilę będzie widział wyłącznie pustą białą stronę w przeglądarce. Rozwiązaniem tego problemu jest renderowanie po stronie serwera, które wykorzystano w serwisie *Datamaster-ui*. Zamiast wysyłać plik *html* bez właściwej zawartości, po stronie serwera ta zawartość jest tworzona, a następnie wysyłany jest pełny plik *html*. W ten sposób zamiast białej, pustej strony użytkownik zobaczy na ekranie właściwą zawartość aplikacji. Stona jednak nie będzie wykonywalna (nie będzie się dało wykonać żadnej akcji) dopóki kod *React'a* nie zostanie wykonany. Dodatkowo renderowanie po stronie serwera jest korzystne w kontekście indeksowania strony przez wyszukiwarki internetowe. Poniższe diagramy prezentują oba te podejścia do renderowania strony.



Rysunek 15: Diagram renderowania po stronie klienta. [9]
Źródło: opracowanie własne.



Rysunek 16: Diagram renderowania po stronie serwera. [9]
Źródło: opracowanie własne.

4.2.2 Struktura projektu

4.2.2.1 Pliki konfiguracyjne

W głównym folderze projektu znajdują się pliki konfiguracyjne. Są to między innymi:

- *Webpack*

  Są to pliki, których pierwszy człon nazwy to *webpack.* takie jak *webpack.common.js*, *webpack.dev.server.js*, *webpack.prod.client.js*, *webpack.prod.server.js*. Pliki te zawierają reguły dotyczące przetwarzania modułów dla różnych trybów aplikacji.

- *Eslint*

  Eslint jest dodatkiem do projektu mającym na celu ułatwienie utrzymywania pewnej zdefiniowanej jakości kodu. Plik konfiguracyjny tego dodatku to *.eslintrc* i zawiera definicje reguł, które powinny być przestrzegane przez programistów podczas tworzenia oprogramowania.

- Szablon *html*

  Szablon html wskazuje na to, jak ma wyglądać plik html wygenerowany przez *Webpack'a*.

- *JsDoc*

  JsDoc jest dodatkiem służącym do generowania dokumentacji kodu *JavaScript*. Posiada plik konfiguracyjny *jsdoc.conf.json*.

- *Docker*

  Docker jest narzędziem służącym między innymi do ułatwienia uruchomienia aplikacji w dowolnym środowisku dzięki technologii wirtualizacji warstwy systemu operacyjnego. Więcej o *Dockerze* napisane jest w rozdziale 5.2.1. Plik konfiguracyjny *Dockera* to *Dockerfile*.

- *Docker compose*

  Docker compose służy do połączenia mikroserwisów w jedną aplikację. Plik konfiguracyjny tej technologii to *docker-compose.yaml*. Więcej o tej technologii napisane jest w rozdziale 5.2.2.

4.2.2.2 Kod *React*

Kod źródłowy *React.js* został umieszczony w folderze *src*. Folder ten podzielony jest na podfoldery, z których najważniejsze to:

- *components*

  Zawiera komponenty wspólne dla widoków aplikacji.

- *global*

  Zawiera globalne zmienne, style, oraz funkcje.

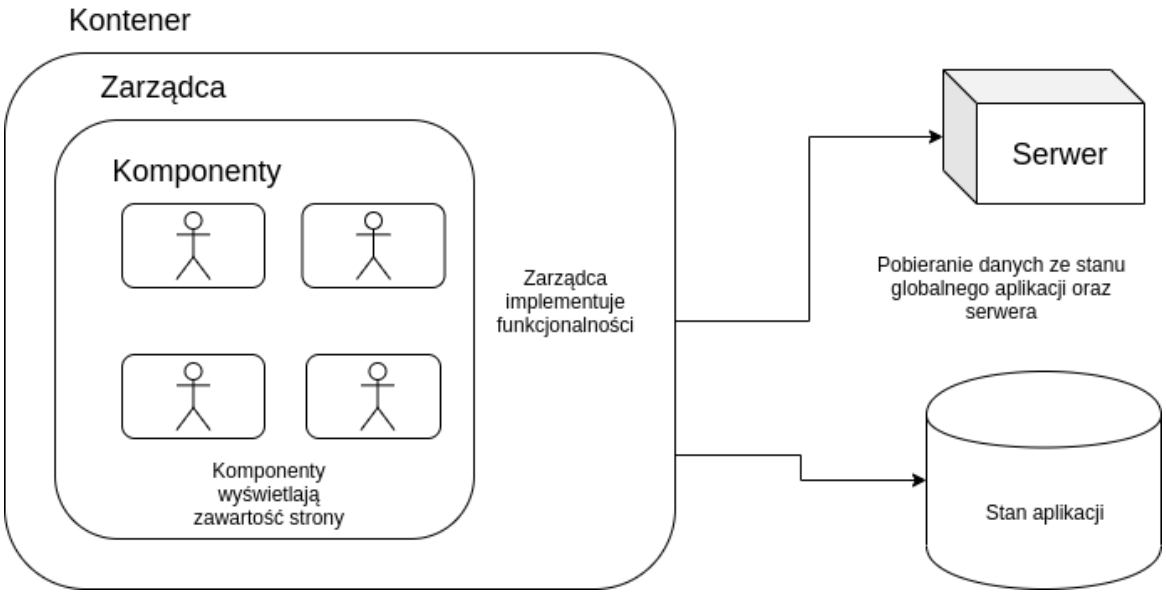
- *pages*

  Zawiera pod foldery z definicjami stylu, komponentów oraz funkcjonalności dla danego widoku aplikacji.

- *index*

  Zawiera punkty wejścia aplikacji osobno dla renderowania po stronie klienta oraz serwera.

Do tworzenia widoków wykorzystano architekturę kontenerów, zarządców i komponentów. Kontener stanowi warstwę pobierania danych z serwera oraz odpowiada za komunikację z globalnym stanem aplikacji. Zarządca wykonuje akcje, posiada implementacje niezbędnych funkcji do działa nia aplikacji, a komponent pełni warstwę prezentacyjną. Takie rozwiązanie nie jest oficjalnym sposobem implementowania widoków przy użyciu frameworku *React*. Najczęściej implementowany jest jedynie kontener i komponenty jednak w projekcie zdecydowano się wprowadzić dodatkową warstwę zarządcy.



Rysunek 17: Architektura kontener, zarządca, komponent.
Źródło: opracowanie własne.

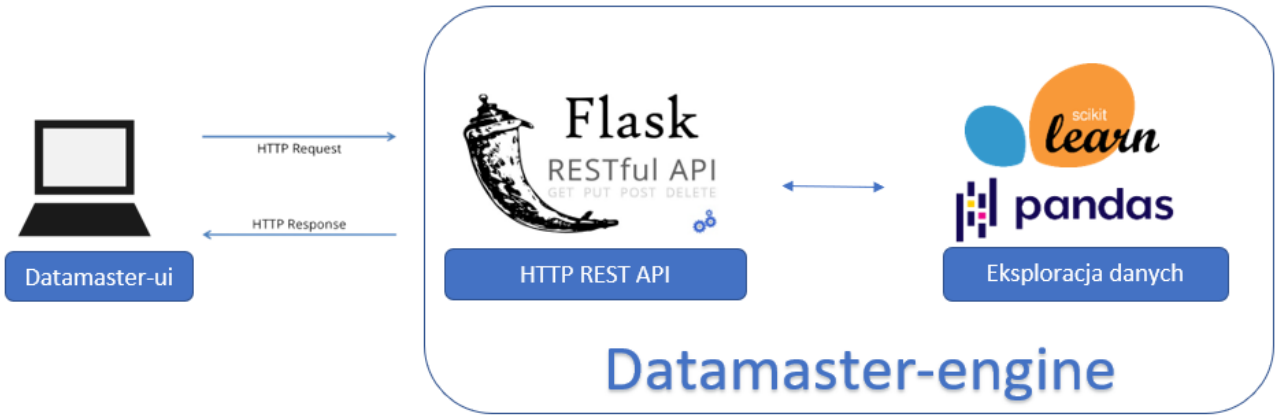
4.2.2.3 Serwowanie aplikacji

W folderze serwer znajdują się pliki odpowiedzialne za renderowanie aplikacji po stronie klienta oraz serwera. Do zaimplementowania tej funkcjonalności użytko frameworku *Express.js*.

4.3 Serwis Datamaster-engine

Serwis *Datamaster-engine* odpowiedzialny jest za różnorakie operacje wykonywane na danych. Wśród nich można wyróżnić operacje na danych w tablicy takie jak usuwanie kolumn, wybieranie podzbioru danych ze zbioru początkowego lub wypełnianie wartości brakujących w tabeli. Oprócz tego w serwisie tym zostały zaimplementowane funkcje systematyzujące algorytmy uczenia maszynowego w tym przygotowania danych do analizy oraz wykonywania analizy. Serwis ten stanowi więc warstwę obliczeniową aplikacji.

Na poniższym diagramie zaprezentowano sposób komunikacji pomiędzy serwisem prezentacji danych *Datamaster-ui* a serwisem obliczeniowym *Datamaster-enigne*.



Rysunek 18: Struktura projektu *Datamaster-engine*
Źródło: opracowanie własne z użyciem ikon.
Ikona *Flask*: `https://flask.palletsprojects.com/en/1.1.x/`
Ikona *Sklearn*: `https://github.com/scikit-learn/scikit-learn/blob/master/doc/logos/scikit-learn-logo.svg`
Ikona *Pandas*: `https://github.com/pandas-dev/pandas`

4.3.1 Wyróżnione standardy i technologie

4.3.1.1 Flask

Flask to lekki framework WSGI (Web Server Gateway Interface[44]) aplikacji webowej. Został zaprojektowany tak, aby rozpoczęcie pracy było szybkie i łatwe, z możliwością skalowania do złożonych aplikacji. Zaczęło się jako proste opakowanie wokół Werkzeug i Jinja i stało się jedną z najpopularniejszych platform aplikacji internetowych Python.

Upraszcza on projektowanie, zapewniając przejrzysty schemat łączenia adresów URL, źródła danych, widoków i szablonów. Domyślnie dostajemy również deweloperski serwer WWW, nie musimy instalować żadnych dodatkowych narzędzi typu LAMP (WAMP).

4.3.1.2 Scikit-learn (Sklearn)

Scikit-learn jest biblioteką napisaną w języku *Python*. Posiada szeroki zestaw algorytmów z dziedziny uczenia maszynowego i statystyki matematycznej. Wykorzystuje między innymi takie biblioteki jak *NumPy*, czy *matplotlib*.

4.3.1.3 Pandas

Pandas jest biblioteką napisaną w języku *Python* do analizy i manipulacji danymi. Jej głównym

[44]https://wsgi.readthedocs.io/en/latest/what.html

komponentem jest tzw. *DataFrame* stanowiący tabelę z danymi wraz z wieloma funkcjonalnościami takimi jak: operacje na danych w tabeli czy ich analiza statystyczna.

4.3.2 Struktura projektu

Do projektu zaliczamy plik do obsługi punktów końcowych aplikacji zdefiniowany z użyciem *frameworka Flask*, pliki logiki biznesowej wykorzystujące biblioteki takie jak *Scikit-learn* i *Pandas* oraz pliki ustawień. Poniżej znajduje się dokładniejszy opis struktury aplikacji.

- Obsługa punktów końcowych

  Za obsługę punktów końcowych odpowiedzialny jest plik *app.py*.

- Logika biznesowa

  Logika biznesowa została podzielona na osobne pliki ze względu na przeznaczenie. Są to wszystkie pliki znajdujące się w folderze głównym projektu poza *app.py*. Zaliczamy do nich pliki takie jak: *clustering.py, prediction.py, transform.py, graphUtils.py, validators.py* oraz wiele innych plików pomocniczych.

- Pliki ustawień

  W skład plików ustawień wchodzą takie pliki jak *requirements.txt* służący do zdefiniowania potrzebnych zależności projektowych, *Dockerfile* czyli plik do tworzenia *obrazka Dockerowego*[45] serwisu oraz pliki ustawień do generowania dokumentacji znajdujące się w folderze *docs*.

Dokumentacja projektu *Datamaster-engine* znajduje się pod adresem `https://edufront.pl/docs/engine`.

4.3.3 Szczegóły implementacji

4.3.3.1 Konfiguracja Flaska

Instalujemy mikroframework z użyciem menadżera pakietów pip. Następnie w folderze aplikacji tworzymy główny plik aplikacji, w którym importujemy moduł flask i tworzymy nową aplikację.

```
from flask import Flask
app = Flask(__name__)
```

Tworzymy katalog główny "/" i odpowiadającą mu funkcję obsługi żądań (request handler)

```
@app.route("/")
def main():
    return "Welcome!"
```

[45] *obrazek Dockerowy* - szablon do tworzenia odizolowanego środowiska do uruchamiania aplikacji tzw. *kontenera Dockerowego*

Następnie sprawdzamy, czy uruchomiony plik jest głównym programem.

```
if __name__ == "__main__":
    app.run()
```

Po czym uruchamiamy aplikację.

```
python app.py
```

Tak skonfigurowana aplikacja skieruje przeglądarkę na adres http://localhost:5000/, na której zostanie wyświetlona wiadomość powitalna.

4.3.3.2 Serwowanie punktów końcowych

Serwowanie punktów końcowych aplikacji (ang. *endpoints*) odbywa się w pliku głównym aplikacji tj. *app.py*. Definicja punktów rozpoczyna się od dyrektywy, gdzie w nawiasach zdefiniowana jest kolejno ścieżka do danego punktu końcowego oraz dozwolone metody żądania.

```
\emph{@app.route(engine_ep('fillNans'), methods=['POST'])}
```

Zdefiniowane jest łącznie osiem ścieżek do których można wysyłać zapytania, a operacje które są wykonywane w ramach tych ścieżek to operacje na tablicach danych oraz wykonywanie algorytmów uczenia maszynowego.

4.3.3.3 Przykładowy kod przygotowania danych

W ramach przykładu podany zostanie kod wykonujący uzupełnianie brakujących wartości w zbiorze danych na podstawie pozostałych danych.

```
@app.route(engine_ep('fillNans'), methods=['POST'])
def fill_nans_for_column():
    req = request.get_json()
    df = pd.DataFrame(np.array(req['table'][1:]), columns=req['table'][0])
    setup = req['setup']
    column = setup['column']
    fillingMode = setup['fillingMode']

    df[column].replace(r'^\s*$|^-$', np.nan, regex=True, inplace=True)
    df[column] = df[column].transform(pd.to_numeric, errors='ignore')

    if fillingMode == 'mode':
        df[column].fillna(df[column].mode()[0], inplace=True)
    if fillingMode == 'median':
        df[column].fillna(df[column].median(), inplace=True)
```

```
    if fillingMode == 'mean':
        df[column].fillna(df[column].mean(), inplace=True)

    headers = [list(df)]
    values = df.to_numpy()
    matrix = np.array(np.append(headers, values, axis=0))
    response = json.dumps({"matrix": matrix.tolist()})

    return Response(response, status=200, mimetype="application/json")
```

Powyższy kod stanowi kompletny punkt końcowy serwisu. Dozwoloną metodą żądania dla tego punktu jest *POST*, w ciele którego przesyłamy całą tabelę z danymi wraz z podaną nazwą kolumny, dla której będziemy wypełniać brakujące wartości oraz metodę wypełniania (medianą, modą lub wartością średnią).
Wewnątrz funkcji zdefiniowana jest zmienna *df*, która wskazuje na obiekt *DataFrame* obsługujący dane z tabeli. Dzięki niemu w łatwy sposób możemy przekształcać dane. W celu zastąpienia danych brakujących musimy najpierw wszelkie dane, które chcemy traktować jako brakujące zastąpić jakąś wspólną wartością. W tym przypadku jest to wartość *np.nan*. Warto wspomnieć, że wartości w tabeli, które zostały przesłane do tego punktu końcowego, są typu *string*, więc najpierw muszą być zamienione na liczby.

```
    df[column].replace(r'^\s*$|^-$', np.nan, regex=True, inplace=True)
    df[column] = df[column].transform(pd.to_numeric, errors='ignore')
```

Następnie w zależności od wybranego trybu wypełniania danych wykonywany jest odpowiedni kod.

```
    if fillingMode == 'mode':
        df[column].fillna(df[column].mode()[0], inplace=True)
    if fillingMode == 'median':
        df[column].fillna(df[column].median(), inplace=True)
    if fillingMode == 'mean':
        df[column].fillna(df[column].mean(), inplace=True)
```

Ostatecznie dane są przekształcane, a następnie odesłane do klienta.

4.3.3.4 Przykładowy kod analizy danych

W ramach przykładu zaprezentowano fragment kodu wykonujący analizę klasyfikacji na danych przesłanych przez użytkownika. Poniżej zamieszczony jest punkt końcowy tej analizy.

```
@app.route(engine_ep('classification'), methods=['POST'])
def get_classification():
```

```
req = request.get_json()
options = req['options']
typesSettings = req['typesSettings']
df = pd.DataFrame(np.array(req['data'][1:]), columns=req['data'][0])
classWeight = options['classWeights']
if classWeight == 'None':
    classWeight = None
bootstrap = options['bootstrap']
if bootstrap == 'true':
    bootstrap = True
else:
    bootstrap = False

models = {"logistic": LogisticRegression(max_iter=options['logistic_max_iter'],
                                         penalty=options['logistic_penalty'],
                                         solver=options['logistic_solver'],
                                         class_weight=classWeight
                                         ),
          "forest":
              RandomForestClassifier(n_estimators=options['forest_class_n_estimators'],
                                       class_weight=classWeight,
                                       bootstrap=bootstrap,
                                       criterion=options['forest_class_criterion']
                                       ),
          "knn": KNeighborsClassifier(n_neighbors=options["neighbors"],
                                      metric=options['metric'])}

score_value, score_3std, df_predicted_table, params, discard =
    predictionModel(df, options, typesSettings, models, 'classification')

return {
    "score": score_value,
    "score_3std": score_3std,
    "predicted_table": castDataFrameToList(df_predicted_table),
    "params": params
    }
```

W ciele żądania wysyłane są dane w postaci tabeli, opcje *options* ustawień algorytmu oraz ustawienia typów danych w kolumnach *typesSettings*. Następnie po definicji dostępnych algorytmów wykonywana jest funkcja główna tego punktu końcowego *predictionModel* z której zwracane są wyniki analizy odsyłane później do klienta. Ciało funkcji *predictionModel* zamieszczone jest

poniżej:

```
def predictionModel(df, options, typesSettings, models, type):
  model = models[options["model"]]
  num_predict = options["numPredict"]
  direction = options["offsetPredict"]
  targetColumn = options["targetColumn"]
  scalerName = options["scaler"]
  validationMethod = options["validationMethod"]
  validationFolds = options["validationFolds"]
  includedColumns = options["includedColumns"]
  testSize = options["testSize"]

  df = convertTypes(df, typesSettings)

  df, targetColumn = getOnlyIncludedColumnsAndNewTarget(df, includedColumns,
      targetColumn)
  original_rows_to_predict_x = getRowsToPredictWithNoTarget(df, num_predict,
      direction, targetColumn)

  x, y = splitTargetColumnAndData(df, targetColumn)
  x = dummifyDataFrame(x)
  x = scaleData(x, scalerName)
  rows_to_predict_x, x, y = separateRowsToPredictFromTheRest(x, y, num_predict,
      direction)
  # split data

  X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=testSize)
  model.fit(X_train, y_train)
  score = model.score(X_test, y_test)
  score3Std = None
  arrayY = np.array(y_test)
  arrayP = np.array(model.predict(X_test))

  if validationMethod == 'cross':
    calculated_score = cross_val_score(model, x, y, cv=validationFolds)
    score = np.mean(calculated_score)
    score3Std = 2 * np.std(calculated_score) * 100

  # final model on whole dataset
  model.fit(x, y)
```

```
  # predict selected rows
  predict_table = pd.DataFrame([])
  if rows_to_predict_x is not None:
    predicted_col = model.predict(rows_to_predict_x)
    predicted_col_df = pd.DataFrame(predicted_col,
        columns=[df.columns[targetColumn]])

    if type == 'classification':
      predicted_proba_col = model.predict_proba(rows_to_predict_x)
      predicted_proba_col = [round(np.max(proba) * 100, 2) for proba in
          predicted_proba_col]
      predicted_proba_col_df = pd.DataFrame(predicted_proba_col,
          columns=['Probability [%]'])
      predict_table = pd.concat([predicted_col_df, predicted_proba_col_df,
          original_rows_to_predict_x.reset_index()],axis=1)
    else:
      predict_table = pd.concat([predicted_col_df,
          original_rows_to_predict_x.reset_index()],axis=1)

  return score, score3Std, predict_table, model.get_params(), {"y_test": arrayY,
      "y_pred": arrayP}
```

W funkcji tej po definicji opcji ustawień następuje konwersja typów danych do typów zdefiniowanych przez użytkownika:

```
df = convertTypes(df, typesSettings)
```

Kolejnym krokiem jest usunięcie kolumn, które nie zostały zaznaczone do analizy oraz odseparowanie kolumny docelowej klasyfikacji.

```
df, targetColumn = getOnlyIncludedColumnsAndNewTarget(df,
    includedColumns,targetColumn)
```

Następnie dane bez kolumny docelowej są kopiowane w celu późniejszego zamieszczenia ich w tabeli wynikowej.

```
original_rows_to_predict_x = getRowsToPredictWithNoTarget(df, num_predict,direction,
    targetColumn)
```

Później następuje dalsza transformacja danych w tym wykonanie *kodowania gorącojedynkowego* na danych oraz ich przeskalowanie, gdy użytkownik zdefiniował takie skalowanie.

```
x = dummifyDataFrame(x)
```

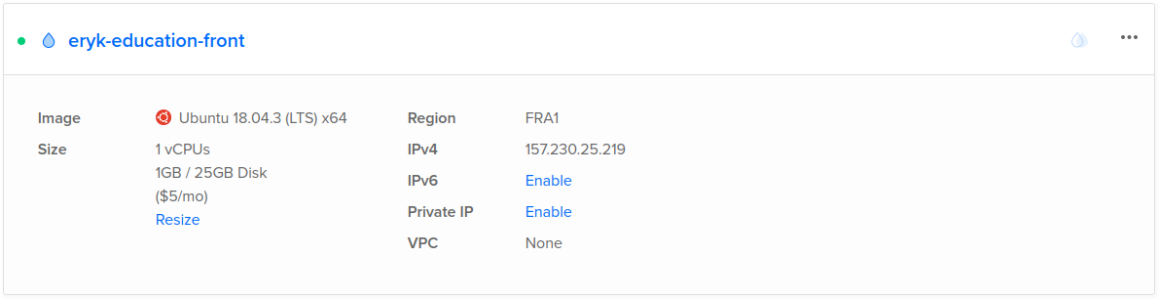
```
x = scaleData(x, scalerName)
```

Kolejne fragmenty kodu to już wykonywanie obliczenia modelu oraz jego walidacja. Końcowym etapem jest budowa tabeli z obliczonymi wynikami oraz połączenie jest z tabelą wejściową danych.

5 Wdrożenie aplikacji na serwer

5.1 Hosting

Aplikacja umieszczona jest na deweloperskiej chmurze *DigitalOcean*. Skorzystano z najmniejszej możliwej usługi tzw. *Droplet*. Wybrana opcja jest maszyną wirtualną z następującymi parametrami:



Rysunek 19: Szczegóły techniczne maszyny wirtualnej, na której umieszczona jest aplikacja.

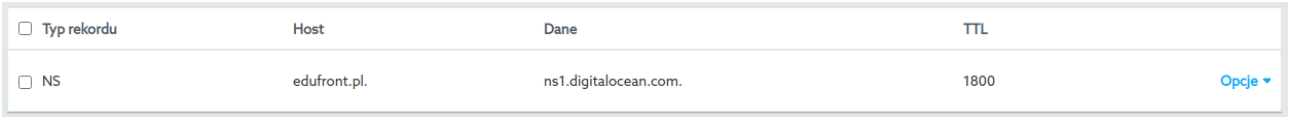
Dodano również rekordy *DNS*[46] w celu udostępnienia domeny w sieci pod nazwą *www.edufront.pl* lub po prostu *edufront.pl*.

Type	Hostname	Value	TTL (seconds)	
A	www.edufront.pl	directs to 157.230.25.219	3600	More
A	edufront.pl	directs to 157.230.25.219	3600	More
NS	edufront.pl	directs to ns1.digitalocean.com.	1800	More
NS	edufront.pl	directs to ns2.digitalocean.com.	1800	More
NS	edufront.pl	directs to ns3.digitalocean.com.	1800	More

Rysunek 20: Rekordy *DNS* ustawione w panelu kontrolnym *DigitalOcean*.

Nazwa domeny została zarezerwowana w portalu *home.pl* na którym również dodano rekordy *DNS* wskazujące na serwer *DigitalOcean*, na którym znajduje się aplikacja.

[46]DNS (ang. Domain name system)-system nazw sieciowych odpowiadający na zapytania o nazwy domen [10].



Rysunek 21: Rekordy *DNS* ustawione w panelu kontrolnym *home.pl*.

5.2 Architektura

5.2.1 Konteneryzacja

Aplikacja składa się z pięciu mikrosewisów, z których cztery wykorzystywane są w wersji produkcyjnej aplikacji. Każdy mikroserwis działa w tzw. *kontenerze Dockerowym*, który stanowi kontrolowane środowisko do uruchomienia aplikacji. Wirtualizuje on warstwę systemu operacyjnego, dzięki czemu niezależnie od środowiska serwera, będziemy w stanie uruchomić aplikację. W tym celu zainstalowano *Docker deamon* udostępniający interfejs *Dockera*, a następnie stworzono tzw. *Obrazki Dockerowe* (*Docker images*), które stanowią szablon do stworzenia kontenera. Obrazki są tworzone na podstawie plików *Dockerfile* definiujących obrazki. Przykład takiego pliku dla mikroserwisu *Datamaster-ui*, znajduje się poniżej:

```
FROM node:14.3.0-buster
WORKDIR ui
COPY package.json ./
RUN npm install
COPY . ./
EXPOSE 8080
ARG NODE_ENV=production
ENV NODE_ENV=${NODE_ENV}
RUN npm run prod:server
CMD ["node", "server/indexServer.js"]
```

Plik ten zawiera procedury potrzebne do stworzenia obrazka. Zawiera on definicję środowiska, jakie zostanie stworzone dla aplikacji wraz z pakietami potrzebnymi do jej uruchomienia (w tym przypadku *Node.js* przy dyrektywie *FROM*). Następne instrukcje wykonują kopiowanie zawartości danej aplikacji do kontenera, ustawiają zmienne środowiskowe, uruchamiają skrypty instalujące potrzebne zależności, budują aplikacje oraz uruchamiają ją.
Poza kontenerami stworzonymi dla serwisów aplikacji tj. *Datamaster-ui*, *Datamaster-service*, *Datamaster-engine* stworzono również osobny kontener dla procesu odpowiedzialnego za komunikację pomiędzy światem zewnętrznym, a poszczególnymi serwisami. Wykorzystane narzędzie, które umożliwia zarządzanie tą komunikacją to *Nginx* [47]. Stworzono również kontener na relacyjną bazę danych *Postgresql*[48] wykorzystywaną jedynie podczas rozwoju aplikacji. Do celów

[47] *Nginx* - program posiadający wiele funkcjonalności m.in. *reverse proxy*, *caching*, serwowanie stron.
[48] *Postgresql* - system zarządzania relacyjnymi bazami danych.

produkcyjnych wykorzystano trwałą bazę danych zapisaną na serwerach dostawcy *ElephantSQL*.

5.2.2 Połączenie mikroserwisów

Połączenie mikroserwisów wraz z ich uruchomieniem jako jeden serwis realizowane jest za pomocą narzędzia *Docker Compose*[49]. Narzędzie to wymaga pliku *yaml* zawierającego definicje poszczególnych charakterystyk mikroserwisów oraz innych funkcjonalności takich jak na przykład sieć między kontenerami. Poniżej przedstawiony jest fragment kodu z pliku *docker-compose.yaml*:

```
services:
  engine-service:
    container_name: engine
    restart: always
    build: ../Datamaster-engine/.
    ports:
      - 8000:8000   # original port : destination port
    networks:
      - ui-network
      - backend-network
```

Powyższy fragment kodu definiuje serwis *Datamaster-engine*. W polu nazwy kontenera *container-name* podana jest jego nazwa, przy użyciu której będzie można się do niego odwoływać z pozostałych serwisów. Do udostępnienia komunikacji między serwisami zdefiniowano wspólne dla nich sieci.

```
networks:
  ui-network:
  backend-network:
```

Pole *build* dotyczy wskazania na plik *Dockerfile*, na podstawie którego zostanie zbudowany *obrazek Dockerowy*[50]. Pole *ports* wskazuje na to, jak mają być mapowane porty z wewnątrz kontenera na zewnątrz.
Oprócz sieci zdefiniowano również tak zwane wolumeny (*Docker Volumes*) służące do dzielenia plików pomiędzy kontenerami. Konieczność stworzenia takich wolumenów wynikała z tego, że pliki dokumentacji serwisów *Datamaster-service* oraz *Datamaster-engine* są serwowane przez serwis *Datamaster-ui*. Aby to serwowanie było możliwe, serwis musi mieć dostęp do plików znajdujących się w innych kontenerach.

```
volumes:
```

[49] *Docker compose* - narzędzie do definiowania i uruchamiania aplikacji składających się z wielu *kontenerów Dockerowych*

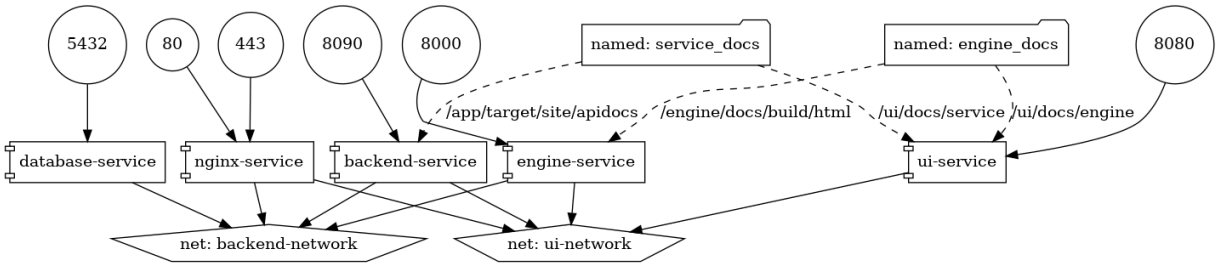
[50] *obrazek Dockerowy* - szablon tworzony na podstawie pliku emphDockerfile dla *kontenera Dockerowego* aplikacji

```
  service_docs:
  engine_docs:
```

```
volumes:
  - service_docs:/app/target/site/apidocs:ro

volumes:
  - engine_docs:/engine/docs/build/html:ro
```

Efekt połączenia przedstawia poniższy diagram pokazujący porty, sieci oraz wolumeny zdefiniowane dla poszczególnych kontenerów.



Rysunek 22: Diagram połączonych serwisów aplikacji.

5.2.3 Obsługa zapytań

Zapytania kierowane do serwera obsługiwane są przez warstwę oprogramowania *Nginx*, a sposób działania tej warstwy określony jest w jej pliku konfiguracyjnym.

5.2.3.1 Kierowanie zapytań

W pliku konfiguracyjnym blok *location* służy do ustawienia obsługi danego zapytania. Skonfigurowano obsługę następujących lokacji:

- */*

  Obsługa zapytań do serwisu *Datamaster-ui*. Gdy ścieżka nie istnieje, serwis ten wyświetli stronę 404.

- */api*

  Obsługa zapytań do serwisu *Datamaster-service*

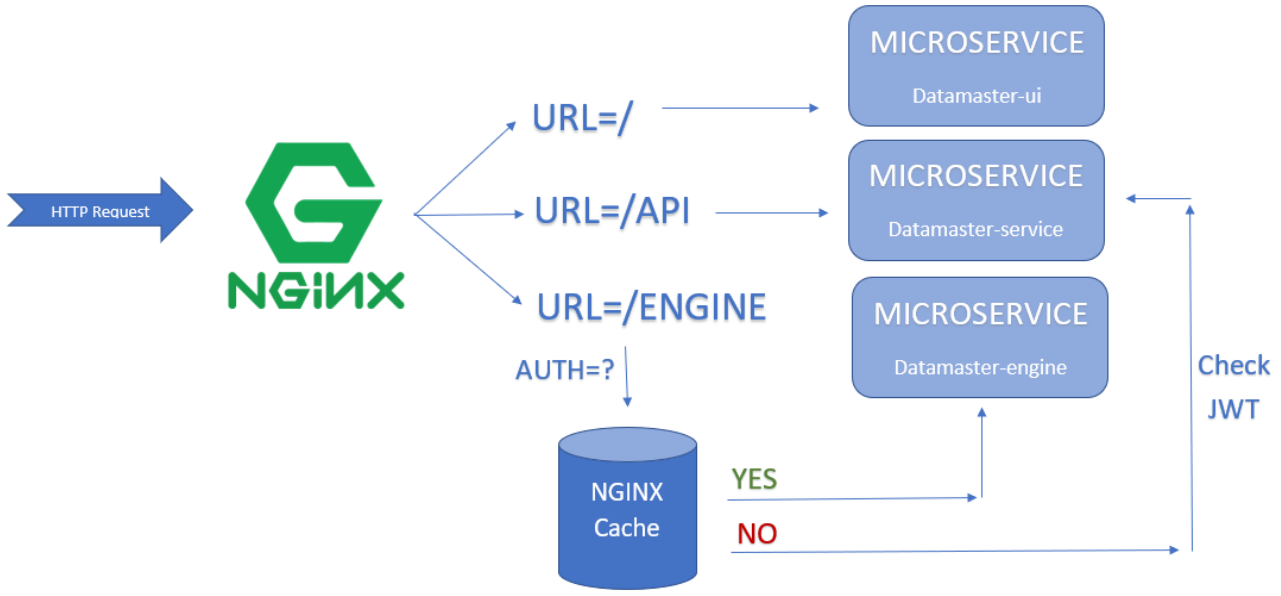
- */engine*

  Obsługa zapytań do serwisu *Datamaster-engine*

Przykładowy kod konfiguracji dla lokacji */api* wygląda następująco:

```
location /api {
        proxy_pass http://backend:8090;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
}
```

Linijka *proxy-pass* określa gdzie przekierować wszystkie zapytania do serwisu z lokacji */api*. W tym przypadku zapytania przekierowywane są do mikroserwisu *Datamaster-service*, a dzięki temu, że została zdefiniowana sieć między kontenerem *nginx-service*, a kontenerem *backend-service*, jest możliwość komunikacji z tym serwisem poprzez nazwę *backend* jako alias do użycia w kodzie.
Zapytania kierowane do lokacji */engine* są najpierw kierowane do mikroserwisu *Datamaster-service* w celu weryfikacji *JWT tokena*, a po pomyślnej weryfikacji do mikroserwisu *Datamaster-engine*.



Rysunek 23: Kierowanie zapytań wewnątrz *Nginx'a*.
Źródło: opracowanie własne z użyciem ikony *Nginx*: `https://guides.wp-bullet.com/set-custom-conditional-http-headers-in-nginx-based-on-host-domain/`

Powyższa grafika obrazuje opisany wcześniej sposób działania. W następnym paragrafie opisano również sposób działania oraz implementację pamięci podręcznej *Nginx'a*.

5.2.3.2 Optymalizacja poprzez pamięć podręczną

Zapytania kierowane do serwisu *Datamaster-service* w celu walidacji *JWT Tokena* są zapisywane do pamięci podręcznej serwera w celu ich optymalizacji.

```
proxy_cache_path /var/cache/nginx/mycache levels=1:2 keys_zone=cache_resp:10m
    max_size=2m;

location = /auth {
    internal;
    proxy_pass            http://backend:8090/api/user/me;
    proxy_pass_request_body off;
    proxy_set_header      Content-Length "";
    proxy_set_header      X-Original-URI $request_uri;
    proxy_cache           cache_resp; # Enable caching
    proxy_cache_lock      on;             # Duplicate tokens must wait
    proxy_cache_valid     200 5m;        # How long to use each response
    proxy_ignore_headers  Cache-Control Expires Set-Cookie;
}
```

Taka odpowiedź serwera pozostaje ważna przez 5 minut. Później, przy próbie zapytania do mikroserwisu *Datamaster-engine*, *Nginx* przekieruje je do *Datamaster-service* w celu sprawdzenia poprawności tokena. Gdy operacja zakończy się sukcesem, wówczas odpowiedź serwera ponownie zostanie dodana do pamięci podręcznej na 5 minut.

Zapytania kierowane po statyczne zasoby serwisu *Datamaster-ui* takie jak pliki *JavaScript*, *CSS* i zdjęcia, są zapisywane w pamięci podręcznej po stronie użytkownika. Reguły dla tego mechanizmu określone są w pliku konfiguracyjnym *Nginx*.

```
location ~* \.(js|css|png|jpg|jpeg|gif|ico|svg)$ {
        expires 1d;
        add_header Cache-Control "public no-transform";
        proxy_pass http://ui:8080;
        proxy_http_version 1.1;
}
```

W linijce ze słowem kluczowym *expires* określone jest, jak długo dane trzymane w pamięci są uznawane za aktualne. Ten czas ustawiono na jeden dzień. Dzięki takiemu rozwiązaniu, zanim zostanie wysłane zapytanie do serwera po dany zasób, najpierw zostanie sprawdzona zawartość pamięci podręcznej, co przyspieszy cały proces.

6 Dokumentacja kodu

6.1 Datamaster-ui

Kod serwisu *Datamaster-ui* został udokumentowany za pomocą narzędzia *jspd*, które generuje pliki *html* z opisem kodu. Tak wygenerowany kod został umieszczony pod adresem `https://edufront.pl/docs/ui`. Dokumentacja posiada listę komponentów oraz funkcji po prawej stronie interfejsu użytkownika. Wybór danego elementu z listy wyświetli krótki opis komponentu wraz z linkiem do kodu źródłowego aplikacji.

6.2 Datamaster-service

Dokumentację serwisu *Datamaster-service* wykonano przy użyciu *Javadoc*. Narzędzie to ułatwia programiście pisanie dokumentacji technicznej projektu. Przetwarza ono specjalne komentarze umieszczone w kodzie na strony www, automatycznie dołączając informacje o nazwach komentowanych klas, metodach, czy też polach. Komentarze napisane w Javadoc muszą rozpoczynać się specjalnym znakiem /** oraz kończyć */, aby zostały odpowiednio przetworzone [11].
Do wygenerowania dokumentacji Javadoc w postaci stron *html* wykorzystano wtyczkę *maven-javadoc-plugin*. Wtyczka ta jest wydajna i ułatwia płynne generowanie złożonych dokumentów. Wyniki wygenerowanej dokumentacji serwisu zostały umieszczone pod adresem `https://edufront.pl/docs/service`.

6.3 Datamaster-engine

Kod źródłowy ostatniego serwisu *Datamaster-engine* został wykonany przy użyciu *Sphinx* oraz *Rinohtype*.
Sphinx to narzędzie, które ułatwia tworzenie inteligentnej i przejrzystej dokumentacji, napisane przez *Georga Brandla* na licencji *BSD*[51]. Pierwotnie stworzony dla dokumentacji języka *Python*, jednak posiada on również doskonałe narzędzia do dokumentowania projektów oprogramowania w wielu innych językach. Działanie Sphinx opiera się na konwersji plików źródłowych *reStructuredText*[52] (rozszerzenie *rst*) najczęściej do dokumentu *HTML*, ale posiada on również możliwość konwersji do takich formatów jak np.: *LaTeX, PDF, ePUB, man*.
System Sphinx dostarcza wbudowany skrypt *sphinx-quickstart*, który po uruchomieniu pyta się użytkownika o konfigurację. Na podstawie odpowiedzi generuje odpowiednie katalogi oraz pliki konfiguracyjne niezbędne do utworzenia dokumentacji [12].
Rinohtype to wysokopoziomowa biblioteka PDF dla języka Python obecnie w fazie beta na

[51] *Licencja BSD (Berkeley Software Distribution Licenses)* - jedne z licencji zgodnych z zasadami wolnego oprogramowania. Skupiają się na prawach użytkownika.

[52] *reStructuredText* - język znaczników oraz analizator składniowy, przeznaczony do szybkiego tworzenia prostych stron internetowych, samodzielnych dokumentów i dokumentacji technicznej.

licencji *GNU AGPL 3.0*[53]. Pomaga zautomatyzować tworzenie wszelkiego rodzaju dokumentów, od faktur po długie złożone dokumenty techniczne. W połączeniu ze Sphinx, Rinohtype stanowi nowoczesną alternatywę dla LaTeX. Sphinx pomaga w pisaniu dużych, ustrukturyzowanych dokumentów i obsługuje wiele różnych formatów wyjściowych, w tym przeszukiwalny HTML. Rinohtype zapewnia zaplecze Sphinx, które umożliwia generowanie profesjonalnie złożonych dokumentów PDF [13].
Wygenerowana dokumentacja w postaci strony HTML znajduje się pod adresem: `https://edufront.pl/docs/engine/`.

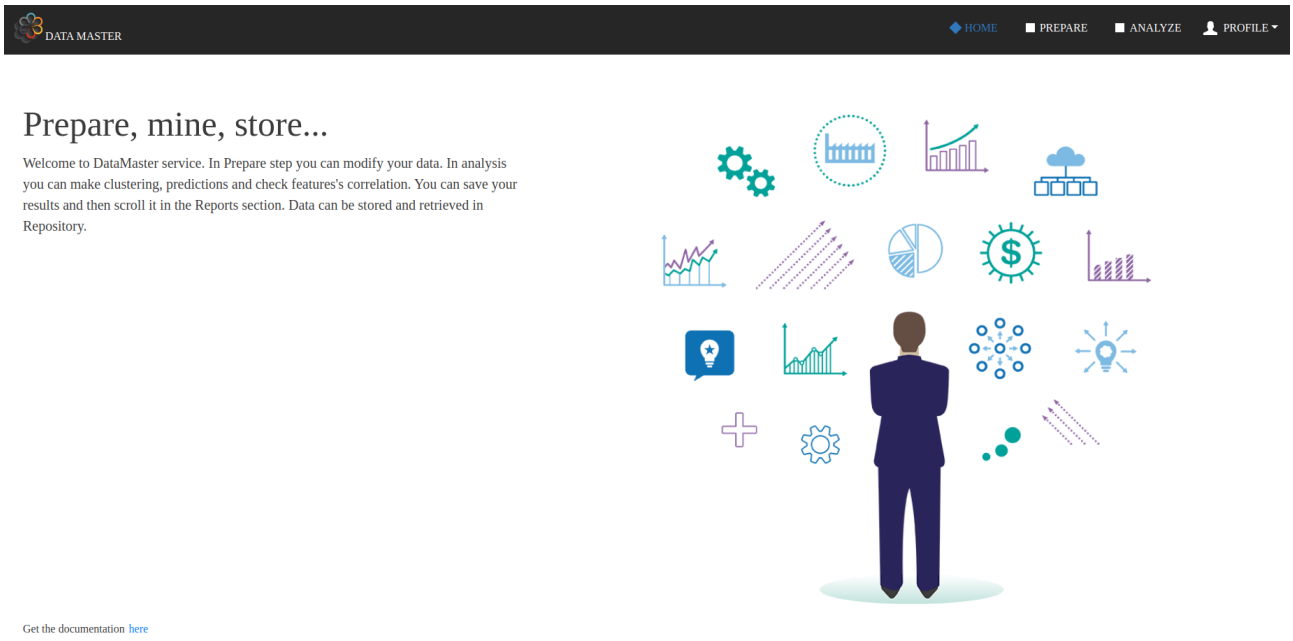
7 Podręcznik użytkownika

W tym rozdziale omówione są wszystkie sekcje aplikacji wraz z tym jak z nich korzystać. W celu dokonania analizy testowej należy zalogować się na konto testowe *user* hasłem *password*. Konto to posiada zapisane przykładowe dane *titanic.csv* zawierające szczegóły osób uczestniczących w rejsie popularnego statku wraz z informacją o tym, czy przetrwali katastrofę. Następnie należy przejść do sekcji *Analyze* oraz wybrać metodę importu danych *import from repository*. W tabeli z danymi należy wybrać przycisk z ikoną analizy (dokument z lupą) przy danych *titanic.csv*. Kolejnym krokiem jest wybór typu analizy oraz przeprowadzenie jej. To jak przeprowadzić analizę opisane jest w punkcie 7.4. Przykładowy wynik analizy klasyfikacji dla tych danych można zobaczyć w sekcji *Reports*, do której można się dostać będąc zalogowanym z menu rozwijalnego *Profile* na pasku nawigacyjnym.

7.1 Sekcja Home

Po wejściu do aplikacji znajdujemy się w sekcji *Home* pod domyślną ścieżką `https://edufront.pl/`.

[53] *GNU Affero General Public License* - Powszechna licencja publiczna jest wolną licencją typu Copyleft na oprogramowanie i inne rodzaje prac. Zaprojektowaną specjalnie w celu zapewnienia współpracy ze społecznością w przypadku oprogramowania serwera sieciowego.

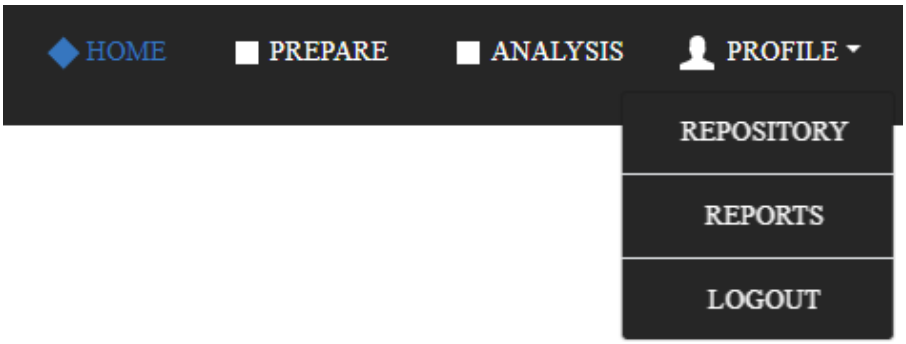


Rysunek 24: Sekcja *Home*.

Sekcja *Home* stanowi wprowadzenie do aplikacji. Zawiera grafikę tematyczną oraz krótki opis najważniejszych funkcjonalności aplikacji. W dolnej części strony znajduje się link z dokumentacją do pobrania. Na pasku nawigacyjnym widzimy różne sekcje takie jak *Home, Prepare, Analyze, Profile*. Sekcja, w której użytkownik aktualnie się znajduje, zaznaczona jest kolorem niebieskim. Sekcja *Home* dostępna jest dla użytkownika bez konieczności autentykacji. Gdy nie jest on zalogowany wówczas próba przejścia do sekcji *Analyze* lub *Prepare* przekieruje go do strony logowania. Użytkownik może się zalogować lub wylogować korzystając z listy rozwijalnej *Profile* na pasku nawigacyjnym.

Rysunek 25: Pasek nawigacyjny aplikacji widziany przez niezalogowanego użytkownika.

Jeżeli użytkownik jest już zalogowany, rozwijalny panel *Profile* zmienia swoją zawartość z sekcji *Login* i *Register* na sekcje *Repository*, *Reports* oraz *Logout*.

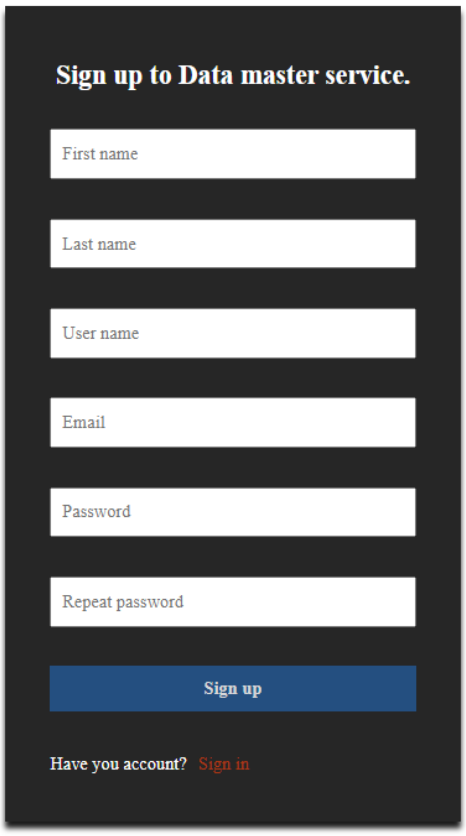


Rysunek 26: Pasek nawigacyjny aplikacji widziany przez zalogowanego użytkownika.

7.2 Rejestracja i uwierzytelnianie

7.2.1 Rejestracja

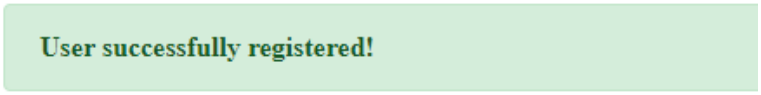
Strona rejestracji znajduje się pod linkiem `https://edufront.pl/register` bądź w panelu nawigacyjnym sekcji *Home* pod przyciskiem *Profile*. Każdy użytkownik musi uzupełnić wszystkie dane w formularzu rejestracyjnym w celu dopełnienia procesu rejestracji.



Rysunek 27: Formularz rejestracyjny.

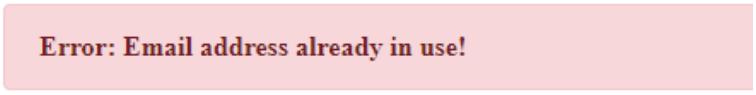
Poprawne wypełnienie formularza skutkuje utworzeniem konta użytkownika oraz przejściem do

strony logowania w celu uwierzytelnienia. W celach informacyjnych zostanie również wyświetlony komunikat o poprawnym utworzeniu konta.



Rysunek 28: Komunikat potwierdzający prawidłowe utworzenie konta.

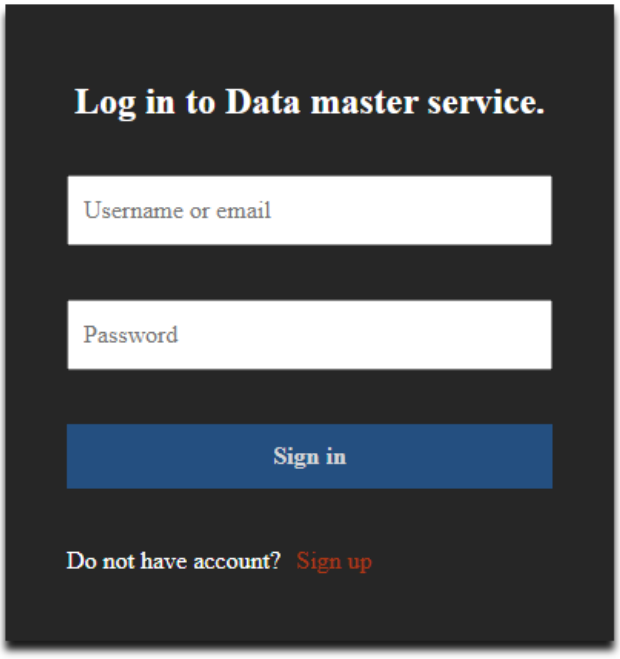
Próba wprowadzenia niekompletnych, nieprawidłowych lub istniejących już danych zostanie zablokowana przez system, a użytkownik otrzyma komunikat o błędzie. Jednym z nich jest komunikat dotyczący użytego już adresu e-mail.



Rysunek 29: Komunikat błędu dotyczący użytego już adresu email.

7.2.2 Logowanie

Przejście do strony logowania odbywa się z sekcji *Home* bądź przez link `https://edufront.pl/login`. Użytkownik posiadający konto w aplikacji ma możliwość zalogowania się do niej. Zalogowanie się daje możliwość korzystania ze wszystkich sekcji aplikacji. Do tego procesu potrzebujemy jedynie wprowadzić poprawny login (*username* lub *e-mail*) oraz hasło w formularzu logowania.

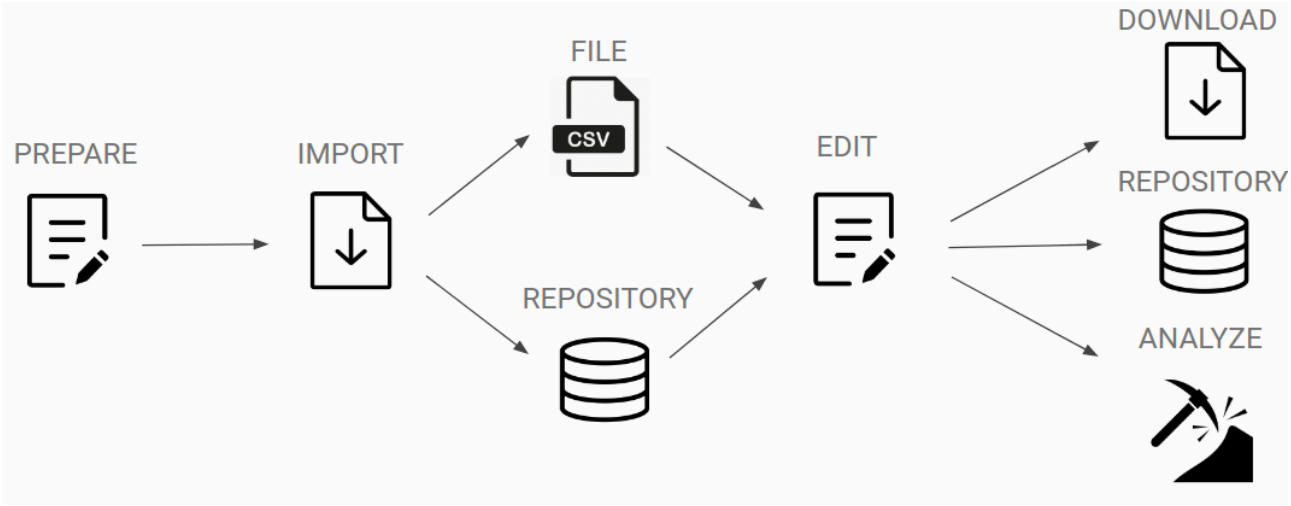


Rysunek 30: Formularz logowania.

W przypadku gdy użytkownik nie posiada konta i zostanie przekierowany do strony logowania, istnieje możliwość przejścia do formularza rejestracji poprzez naciśnięcie czerwonego tekstu *Sign up* znajdującego się na dole formularza logowania. Niepoprawne dane logowania zostaną zakomunikowane użytkownikowi za pomocą błędu uwierzytelnienia, co zobrazowano na poniższym rysunku.

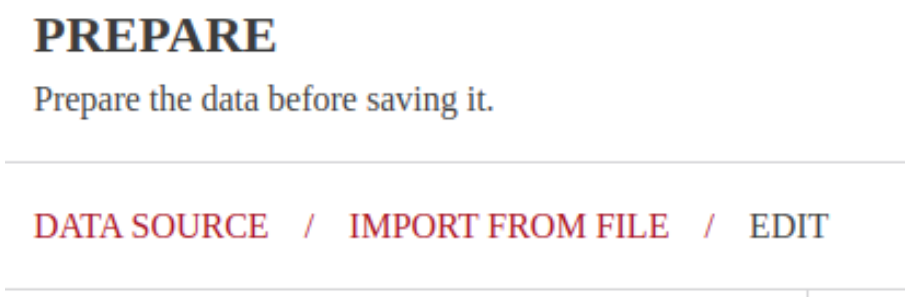
Rysunek 31: Komunikat błędu podczas uwierzytelniania.

7.3 Sekcja przygotowania danych - Prepare



Rysunek 32: Schemat procesu przygotowania danych w sekcji Prepare.
Źródło: opracowanie własne.

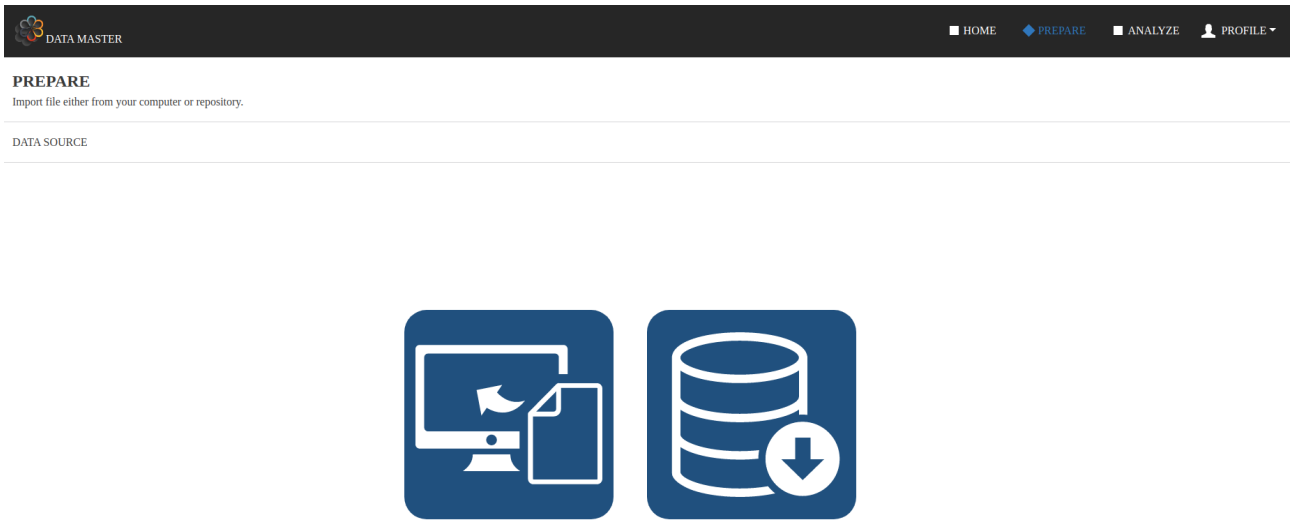
Sekcja *Prepare* dostępna jest pod linkiem `https://edufront.pl/prepare`. W tej sekcji użytkownik ma możliwość edytowania zaimportowanych danych w formie pliku csv. Sekcja ta podzielona jest na kroki, w których użytkownik podejmuje kolejne działania w celu edycji danych. W górnej części strony znajduje się nagłówek opisujący krótko krok, w którym się znajdujemy oraz wyświetlający dużą czcionką nazwę sekcji. Płynną nawigację po tej sekcji umożliwia komponent wyświetlający obecny krok przygotowania danych. Poniżej zamieszczony jest zrzut ekranu nagłówka oraz tego komponentu.



Rysunek 33: Nagłówek sekcji *Prepare* w kroku wyboru źródła danych.

7.3.1 Krok wyboru źródła danych - *Data source.*

Pierwszym krokiem sekcji przygotowania danych jest wybór źródła danych. Użytkownik ma do wyboru import danych z dysku własnego komputera lub z repozytorium serwisu. Po kliknięciu jednego z dwóch obrazków użytkownik zostanie przeniesiony do kolejnego kroku tj. importu danych.

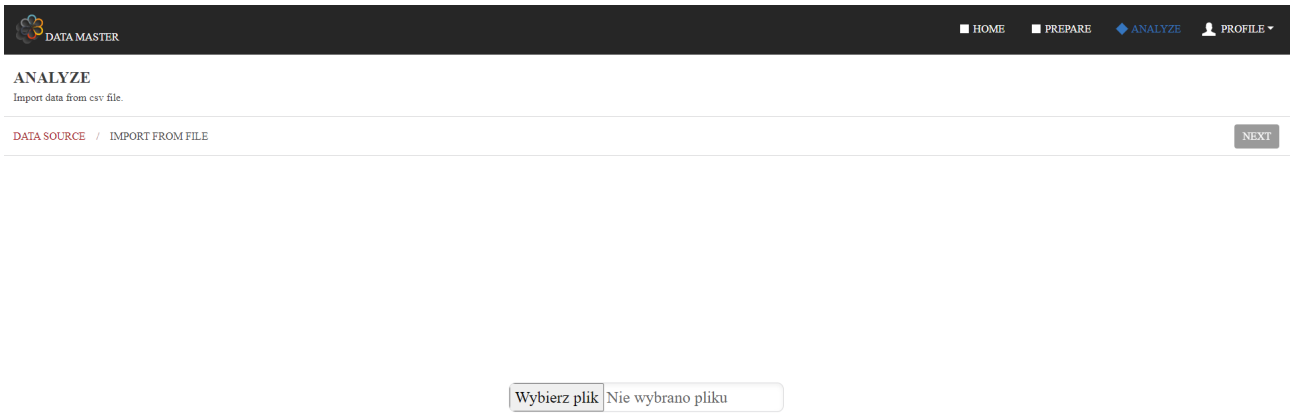


Rysunek 34: Strona wyboru źródła danych w sekcji *Prepare*.

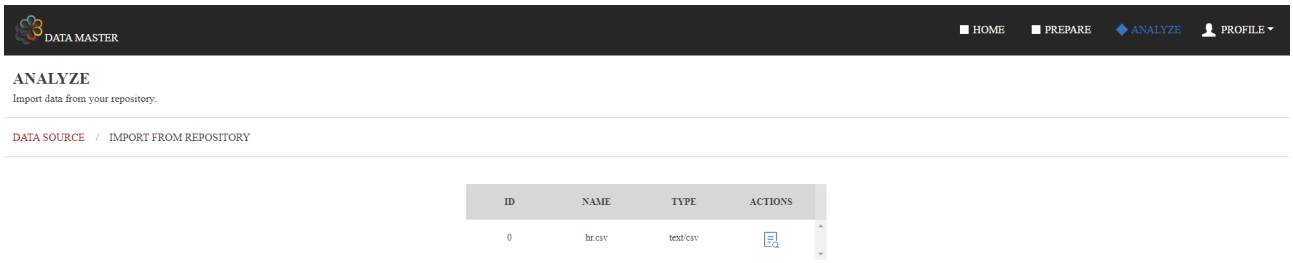
7.3.2 Krok importu danych - *Import from file/repository*

Krok ten występuje w dwóch wersjach: importu pliku z zasobów komputera oraz importu pliku z repozytorium aplikacji. W pierwszym przypadku, po przyciśnięciu przycisku *Choose file* zostanie wyświetlony komponent wyboru pliku z dysku. Po zaimportowaniu pliku, należy wybrać przycisk *Next* aby przejść do następnego kroku. W przypadku importu pliku z repozytorium wyświetlona zostaje tabela plików w repozytorium wraz z jednym przyciskiem akcji *Prepare*. Po wciśnięciu

tego przycisku dla danego pliku zostaniemy przeniesieni do kolejnego kroku, jakim jest krok edycji danych. Komponenty w tym kroku prezentują się następująco:



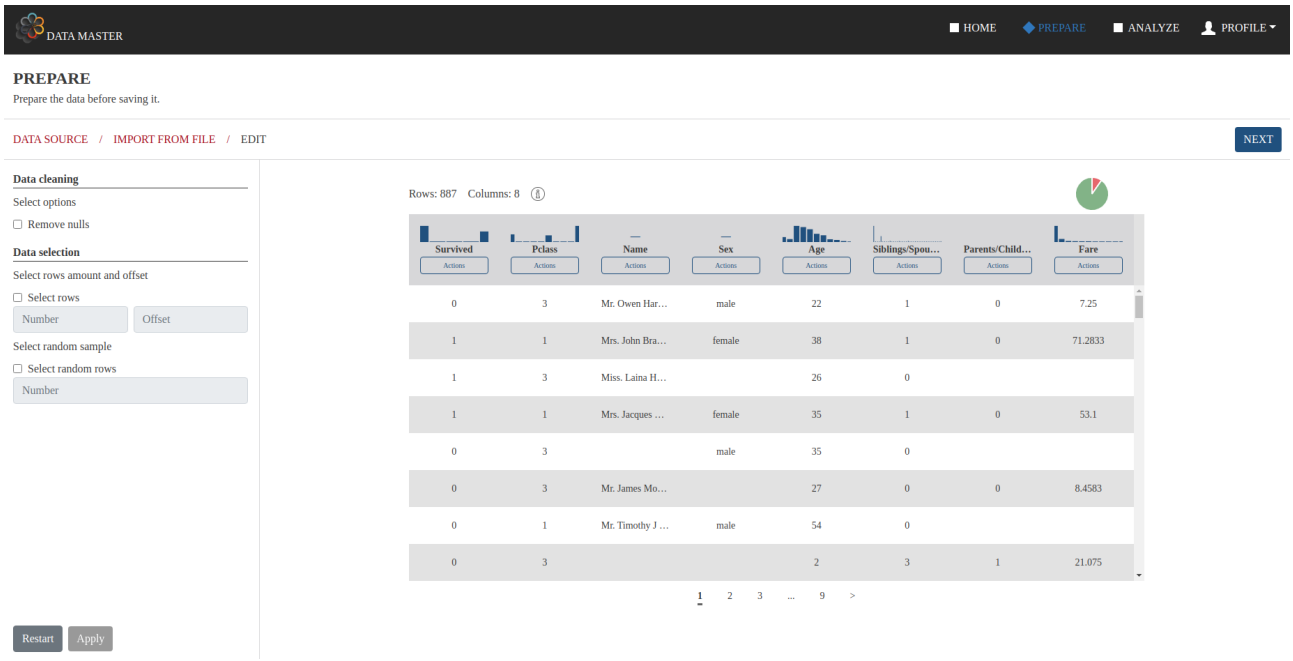
Rysunek 35: Komponent importu pliku do analizy z dysku komputera.



Rysunek 36: Komponent importu danych z repozytorium serwisu.

7.3.3 Krok edycji danych - *Edit*

Krok edycji danych stanowi najważniejszy krok sekcji *Prepare*. Tutaj odbywa się właściwa obróbka danych po ich zaimportowaniu, a komponent to umożliwiający jest najbardziej rozbudowany. Na ekranie można wyróżnić dwa główne obszary. Po lewej stronie znajduje się panel, z którego użytkownik wybiera, w jaki sposób chciałby edytować dane. Po prawej stronie znajduje się tabela wyświetlająca dane wraz z ich szczegółami.

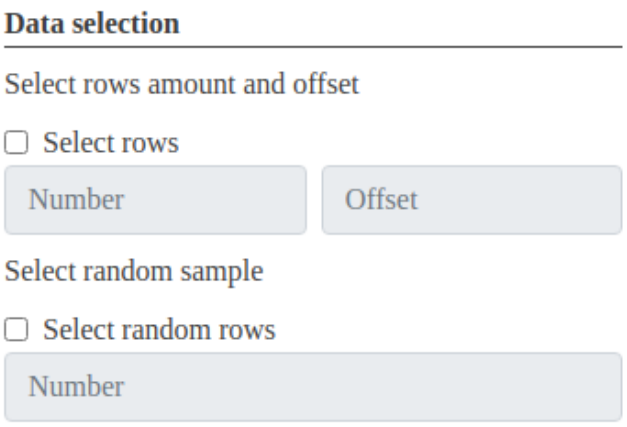


Rysunek 37: Strona edycji danych w sekcji *Prepare*.

Panel po lewej stronie składa się z opcji do wyboru oraz przycisków *Reset* i *Apply*. Po wybraniu poszczególnych opcji użytkownik musi wcisnąć przycisk *Apply*, aby zastosować zaznaczone opcje. Przycisk *Reset* służy do przywrócenia danych do pierwotnej formy.
Opcje pogrupowane są w zależności od zastosowania. Pierwsze opcje dotyczą czyszczenia danych i zawierają jedną funkcjonalność tj. usuwanie danych brakujących. Jako dane brakujące aplikacja interpretuje puste znaki oraz myślniki, czyli " " oraz "-". Kiedy dane zawierają tylko kompletne rekordy, wówczas taki zbiór traktowany jest jako pełny. Wykres przedstawiający stosunek rekordów niepełnych do pełnych znajduje się nad tabelą. Kolorem czerwonym oznaczone są rekordy zawierające wartości brakujące, a zielonym rekordy kompletne.

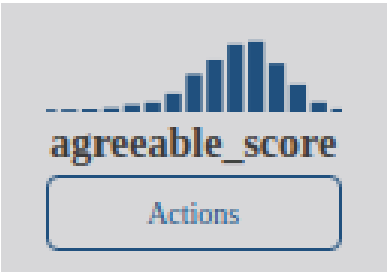
Rysunek 38: Pierwsza grupa opcji w kroku edycji danych w sekcji *Prepare*.

Druga grupa opcji to grupa selekcji danych, czyli wyboru podzbioru ze zbioru początkowego. Może się to odbyć na dwa sposoby: wybór n losowych wierszy lub wybór n wierszy zaczynając od wiersza k. Obie wartości są walidowane i jeśli użytkownik poda liczby, które są niemożliwe do zastosowania, wówczas zostanie o tym poinformowany.



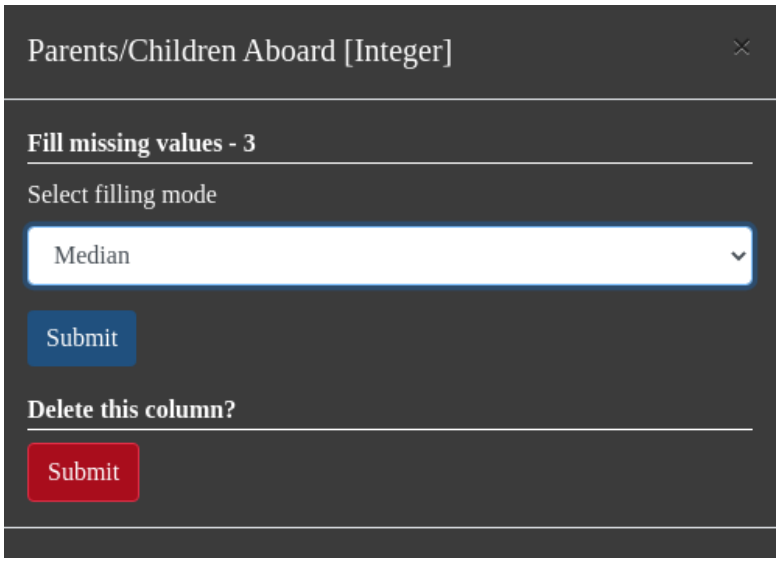
Rysunek 39: Druga grupa opcji w kroku edycji danych w sekcji *Prepare*.

Oprócz edycji za pomocą menu po lewej stronie, która odnosi się do całego zbioru danych, użytkownik może też edytować zawartości poszczególnych kolumn, za pomocą menu akcji. Menu to pojawi się po przyciśnięciu przycisku *Actions* znajdującego się pod nazwą danej kolumny.



Rysunek 40: Nagłówek kolumny wraz z przyciskiem *Actions*.

Umożliwia nam ono dwie akcje: usunięcie danej kolumny oraz wypełnienie brakujących wartości w kolumnach. Wypełnienie brakujących wartości może odbyć się przy pomocy wartości średniej, mediany oraz mody dla danej kolumny. W przypadku danych kategorycznych, dla których wyliczenie mediany i średniej jest niemożliwe, możemy wybrać wypełnienie jedynie modą. Opcja uzupełnienia danych niekompletnych będzie widoczna jedynie wtedy, gdy kolumna będzie zawierała dane interpretowane przez aplikację jako brakujące.

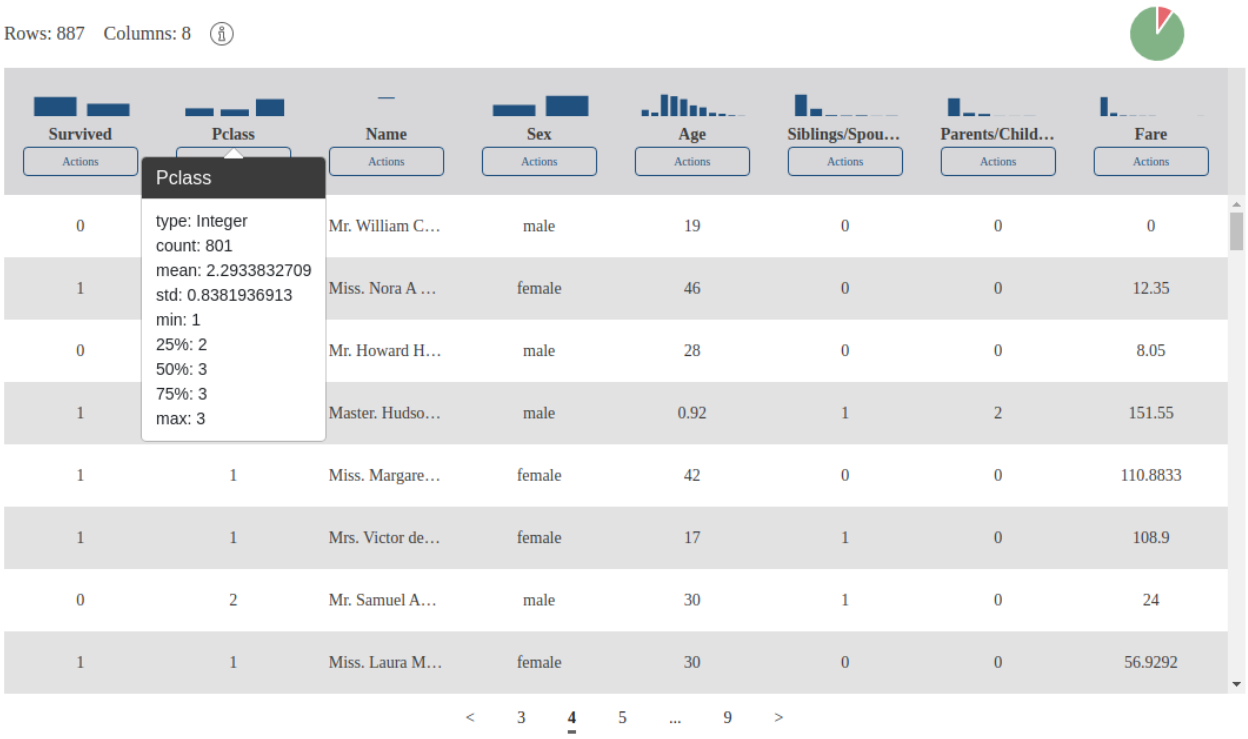


Rysunek 41: Moduł edycji kolumny w sekcji *Prepare*.

Dane wprowadzane do pól tekstowych w kroku *Prepare* są walidowane, a informacja o błędzie wyświetlana jest w prawym górnym rogu ekranu. Informacja pojawia się w momencie użycia przycisku *Apply*, gdy wprowadzone dane nie spełniają nałożonych na nie ograniczeń.

Rysunek 42: Walidacja ustawień w postaci komponentu typu *Toast*.

Poza opcjami do edycji danych mamy do dyspozycji również komponenty opisujące dane. Nad tabelą w prawym górnym rogu znajduje się informacja o ilości wierszy i kolumn w tabeli. W prawym górnym rogu znajduje się wykres kołowy przedstawiający stosunek ilości wierszy niekompletnych do kompletnych. Szczegóły dotyczące danej kolumny dostępne są po kliknięciu nazwy kolumny, a nad tą nazwą znajduje się histogram danej kolumny.

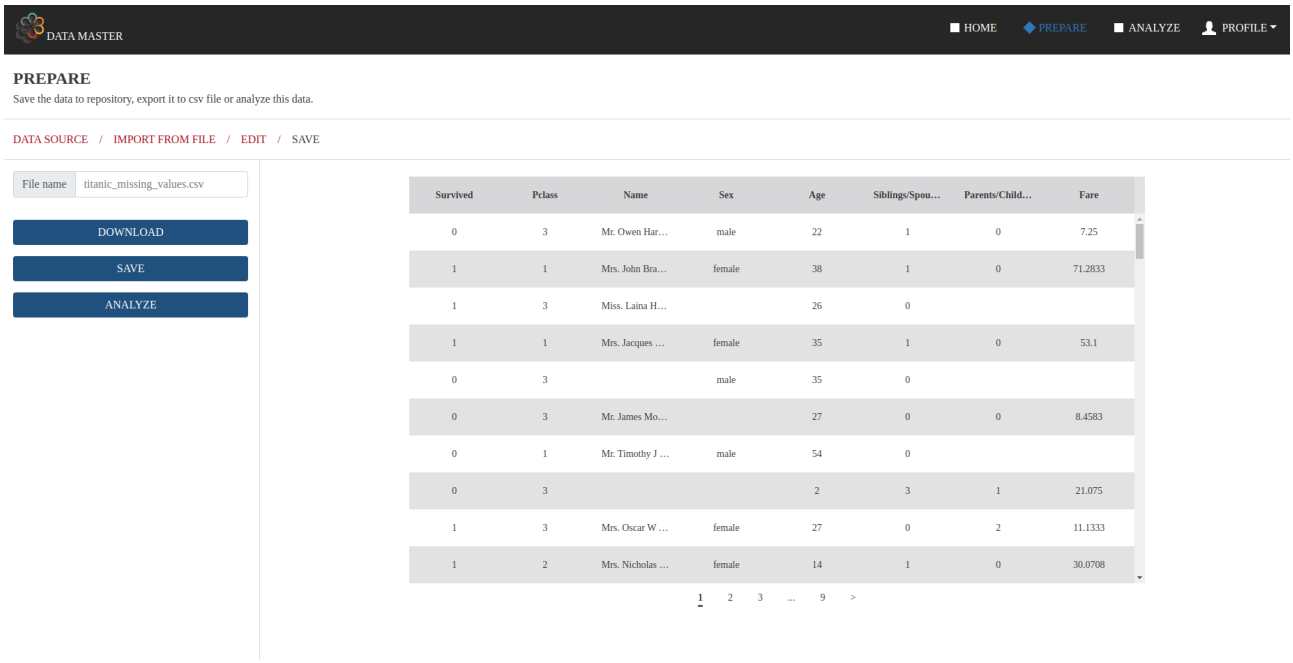


Rysunek 43: Tabela z danymi.

Po kliknięciu przycisku *Next* znajdującego się w prawym górnym rogu ekranu na pasku nawigacji przechodzimy do sekcji zapisu danych.

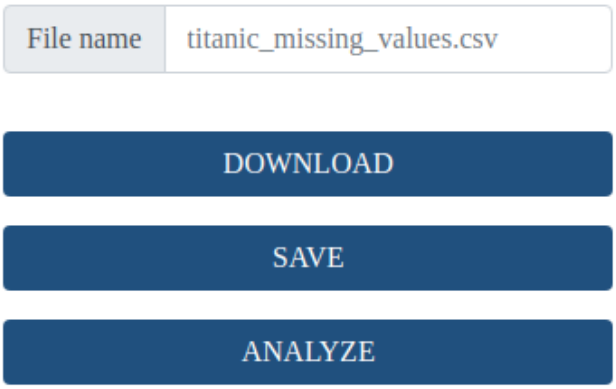
7.3.4 Krok zapisu danych - *Save*

Krok ten stanowi ostatni krok kończący akcję przygotowania danych. W nim mamy do dyspozycji podgląd danych w tabeli oraz panel akcji po lewej stronie widoku.



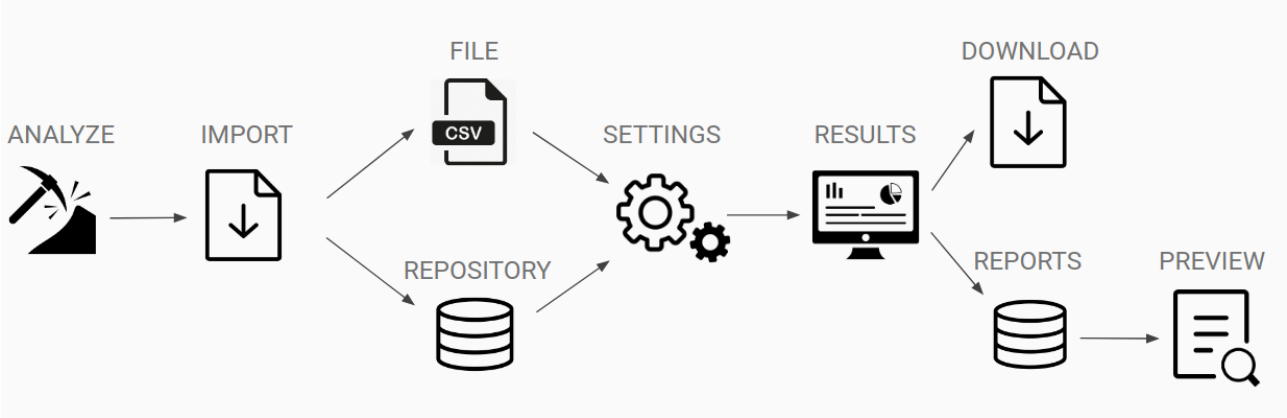
Rysunek 44: Strona kroku zapisu danych w sekcji *Prepare*.

Akcje pozwalają nam kolejno na: zapis danych na dysku - pod przyciskiem *Download*, zapis danych do repozytorium - pod przyciskiem *Save* oraz przejście do analizy bieżących danych - pod przyciskiem *Analyze*. Każdy przycisk posiada krótki opis, który uwidacznia się po przytrzymaniu nad nim kursora. Nazwę pliku ustalamy, wpisując ją w pole tekstowe nad przyciskami. Pole to jest walidowane poprzez sprawdzenie rozszerzenia pliku podane przez użytkownika. Walidacja ustawiona jest tak, że akceptowane są tylko nazwy kończące się na ".csv". Domyślna nazwa pliku to początkowa nazwa zaimportowanego pliku.



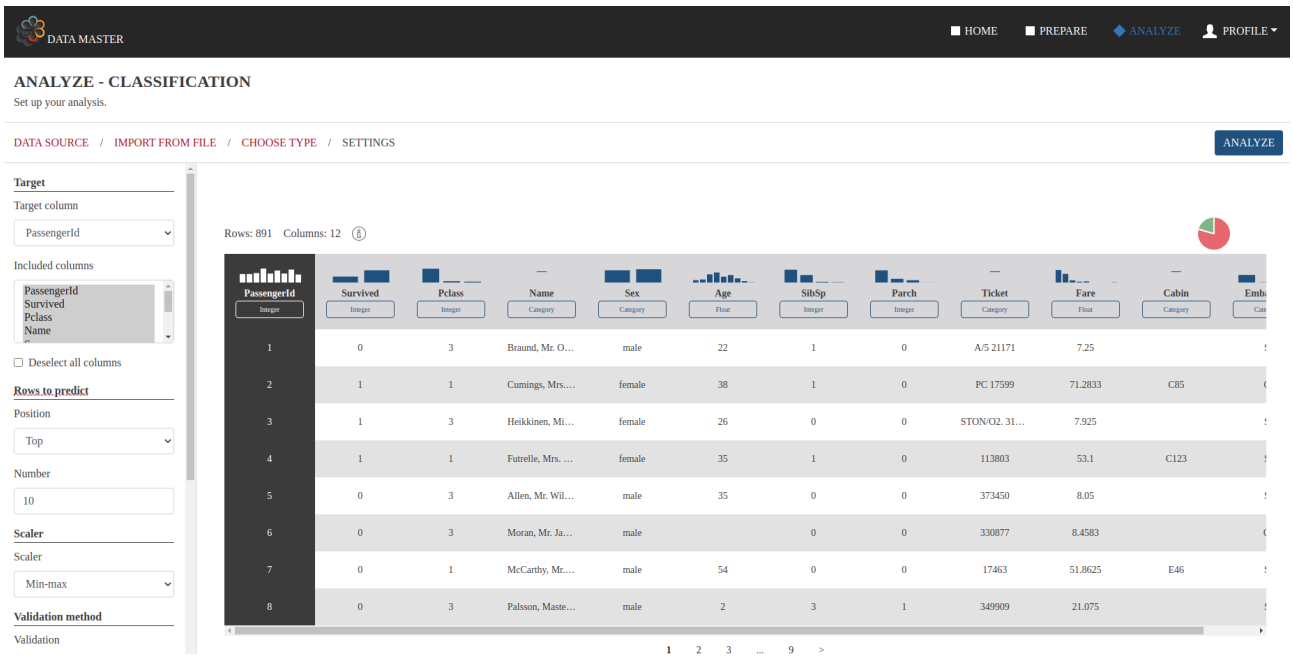
Rysunek 45: Przyciski akcji w kroku zapisu danych w sekcji *Prepare*.

7.4 Sekcja analizy danych - Analyze



Rysunek 46: Schemat procesu analizy danych w sekcji Analyze.
Źródło: opracowanie własne.

Sekcja *Analyze* znajduje się pod linkiem `https://edufront.pl/analysis`. Sekcja ta stanowi główną funkcjonalność aplikacji. Pozwala przewidywać wartości pewnych cech, klasyfikować rekordy, grupować rekordy w klastry oraz analizować współzależność zmiennych. Podobnie jak w przypadku sekcji *Prepare*, w górnej części widoku znajduje się nagłówek z nazwą sekcji oraz jej krótkim opisem. Nawigacja po sekcji odbywa się za pomocą komponentu nawigacji, znajdującego się pod nagłówkiem tak jak w sekcji *Prepare*.



Rysunek 47: Strona sekcji *Analyze* w kroku konfiguracji analizy.

Do sekcji można przejść, klikając przycisk *Analyze* na pasku nawigacyjnym.

7.4.1 Krok wyboru źródła danych - *Data source*

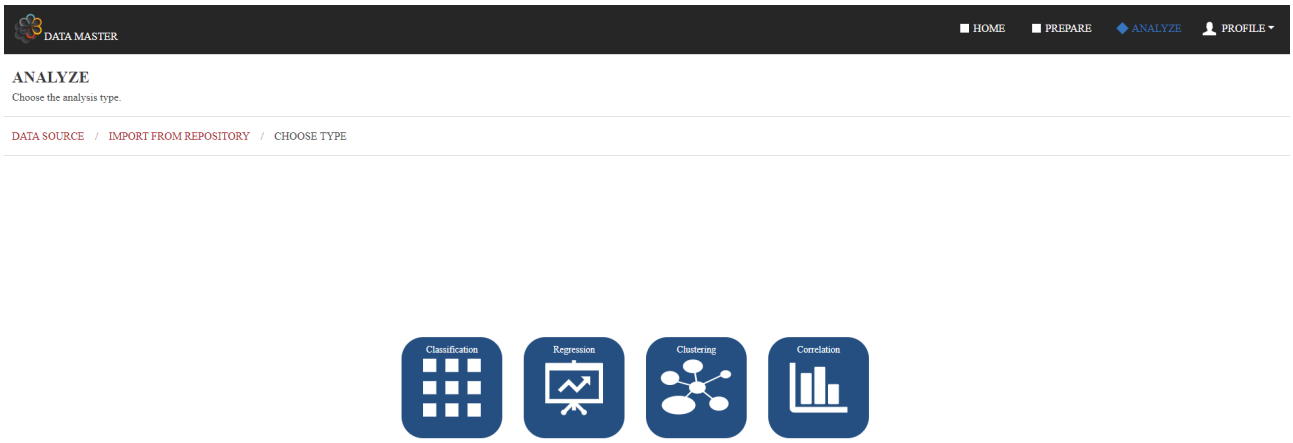
Krok wyboru źródła danych występuje pod tą samą postacią zarówno w sekcji *Analyze* jak i w sekcji *Prepare*. Został on opisany w punkcie 7.3.1.

7.4.2 Krok importu danych - *Import from file/repository*

Krok importu danych również przebiega w taki sam sposób jak w sekcji przygotowania danych. Został on opisany w punkcie 7.3.2.

7.4.3 Krok wyboru analizy - *Choose type*

W tym kroku użytkownik ma do wyboru jeden z 4 typów analiz. Są to klasyfikacja, regresja, klasteryzacja i korelacja.

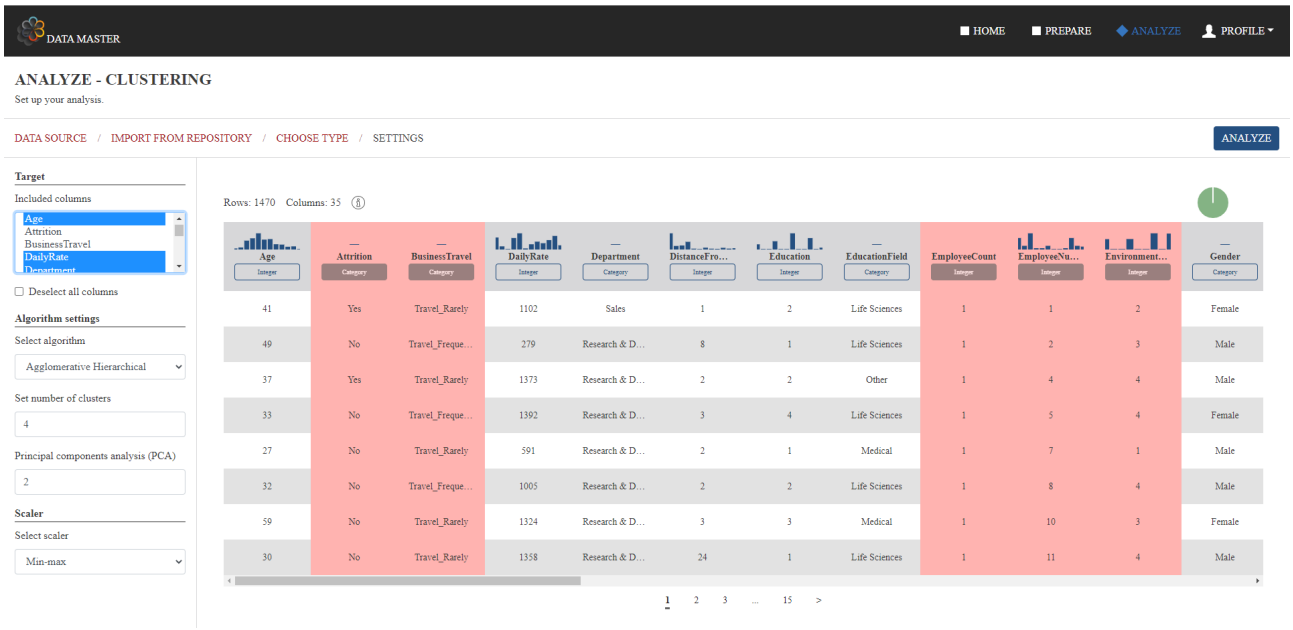


Rysunek 48: Komponent wyboru typu analizy danych.

Po kliknięciu jednego z czterech bloków użytkownik zostanie przeniesiony do kolejnego kroku, jakim jest konfiguracja wybranej analizy.

7.4.4 Krok konfiguracji analizy - *Settings*

Użytkownik po dokonaniu wyboru rodzaju analizy zostanie przekierowany do widoku jej konfiguracji.



Rysunek 49: Widok konfiguracji analizy.

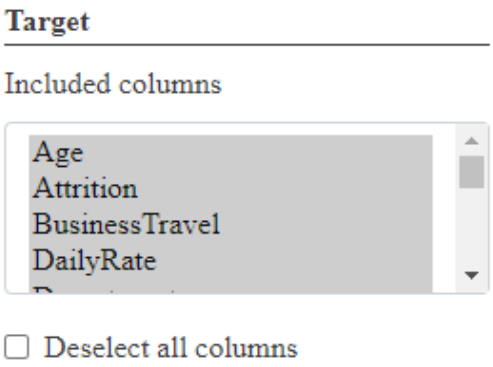
Widoki dla poszczególnych analiz różnią się od siebie pod względem zawartości panelu ustawień po prawej stronie widoku. Mają one jednak elementy wspólne, które są opisane w punkcie 7.4.4.1. W kolejnych punktach omówione są ustawienia unikalne dla danych typów analiz.

7.4.4.1 Wspólne ustawienia analiz

Chcąc dokonać analizy danych, musimy określić np. jakie kolumny będą włączone do analizy lub jak będziemy skalować dane. Tego typu ustawienia są wspólne dla wszystkich analiz i będą omówione w tym punkcie.

7.4.4.1.1 Włączone kolumny do analizy

Istnieje możliwość użycia do analizy tylko części kolumn ze zbioru danych. Wybór kolumn do analizy obsługiwany jest przez komponent znajdujący się w panelu po lewej stronie. Kolor kolumny w tabeli wskazuje na to, czy będzie ona analizowana, czy też nie. Kolorem czerwonym oznaczone są kolumny wyłączone z analizy.



Rysunek 50: Komponent do wyboru kolumn, które zostaną włączone do analizy.

Komponent ten posiada pole wielokrotnego wyboru, w którym użytkownik zaznacza nazwy kolumn, jakie będą włączone do analizy. Pod polem wielokrotnego wyboru znajduje się przycisk umożliwiający zaznaczenie lub odznaczenie wszystkich kolumn.

7.4.4.1.2 Skalowanie danych

Do wyboru sposobu skalowania danych służy rozwijalna lista w panelu konfiguracyjnym. Posiada ona 4 opcje: *Min-max*, *Standard*, *Robust* oraz *None*. Po zaznaczeniu jednej z 3 pierwszych opcji, przed zastosowaniem algorytmów analiz, dane zostaną przeskalowane. Opcja *None* nie wprowadzi żadnego skalowania.

Rysunek 51: Komponent skalowania danych.

Konieczność skalowania danych może wynikać np. z występowania danych w różnych jednostkach. W algorytmach bazujących na odległościach między rekordami jest to szczególny problem, gdyż wartości powinny być sprowadzone do jednej skali by miały taki sam wpływ na końcowy wynik.

Poniżej zamieszczono dostępne opcje skalowania z krótkim opisem:

- *Min-max*

  Przekształca cechy tak, by mieściły się w zadanym przedziale. W aplikacji ustawiono ten przedział na od 0 do 1. Dobre w przypadku kiedy algorytmy nie zakładają rozkładu zmiennych *Gaussa* [14].

- *Standard*

  Przekształca cechy tak, aby miały wartość średnią 0 oraz odchylenie standardowe 1. Dobre w przypadku kiedy algorytmy zakładają rozkład zmiennych *Gaussa* [15].

- *Robust*

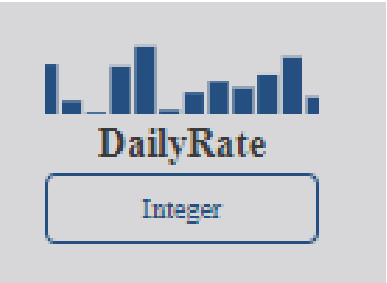
  Skaluje cechy za pomocą statystyk odpornych na wartości odstające. Zarówno normalizacja *Min-max* jak i standaryzacja *Standard* są wrażliwe na wartości odstające. Kiedy mamy do czynienia z takimi danymi, lepiej powinien spisać się *Robust scaler* [16].

- *None*

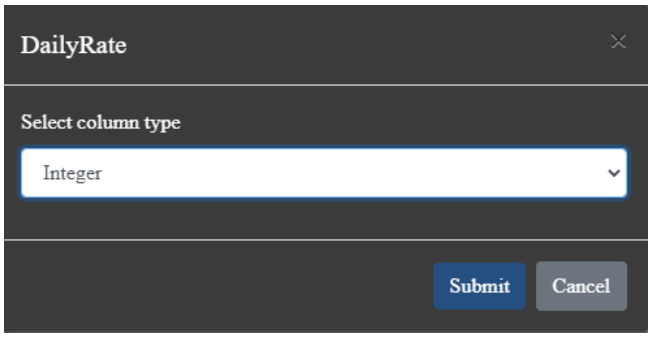
  Zastosowanie tej opcji nie wprowadzi skalowania danych. Dobre w przypadku algorytmów niestosujących metryk odległości takich jak algorytm *drzewa decyzyjnego* lub *losowego lasu*.

7.4.4.1.3 Wybór typu kolumny

W tabeli istnieje możliwość wyboru typu danych w kolumnie. Początkowy typ jest wnioskowany na podstawie danych zawartych w kolumnach. Jeśli wszystkie wartości w danej kolumnie są liczbami całkowitymi, wówczas typ kolumny zostanie ustawiony jako *Integer*. Jeśli jakakolwiek wartość w kolumnie nie może być interpretowana jako liczba, wówczas typem kolumny będzie typ *Category*. W przypadku gdy wszystkie wartości w kolumnie mogą być interpretowane jako liczba, ale przynajmniej jedna wartość jest typu zmiennoprzecinkowego, wtedy taka kolumna będzie miała typ *Float*.
Naciśnięcie przycisku znajdującego się pod nagłówkiem kolumny w tabeli z danymi, spowoduje wyświetlenie modułu wyboru typu danych.



Rysunek 52: Nagłówek kolumny wraz z przyciskiem zmiany typu danych.



Rysunek 53: Moduł zmiany typu danych wybranej kolumny.

Użytkownik ma do wyboru 4 typy danych tj. *Integer*, *Float*, *Category* oraz *Ordinal*. Typy danych można zmieniać w ograniczony sposób. Poniżej w punktach wymieniono dozwolone zmiany typów danych wraz z opisem. Po dwukropku umieszczono nazwy typów, na które można zamienić typ przed dwukropkiem.

- *Float*: *Integer*

  Typ liczby zmiennoprzecinkowej.

- *Integer*: *Category*, *Float*

  Typ liczby całkowitej. Można go zamienić na typ *Category*, co spowoduje, że algorytm nie będzie brał pod uwagę zmiennych jako wartość, a każdą unikalną wartość będzie traktował jako osobną kategorię.

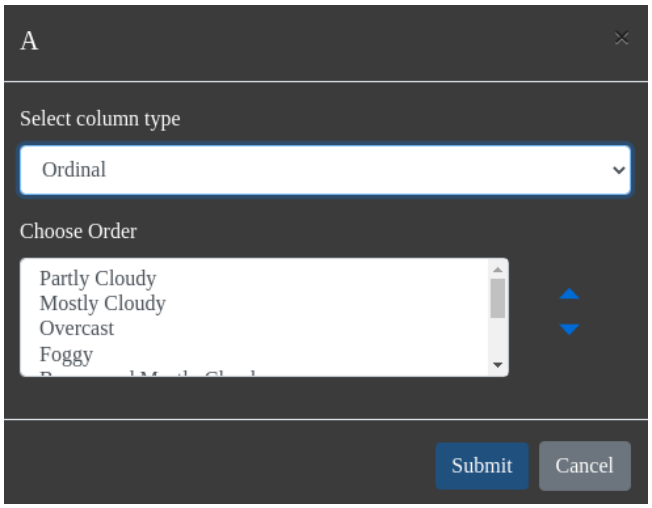
- *Category*: *Ordinal*

  Typ kategoryczny nie posiada uporządkowania. Dobrym przykładem może być kolumna z różnymi kolorami, dlatego, że nie różnią się one w sensie wielkościowym.

- *Ordinal*: *Category*

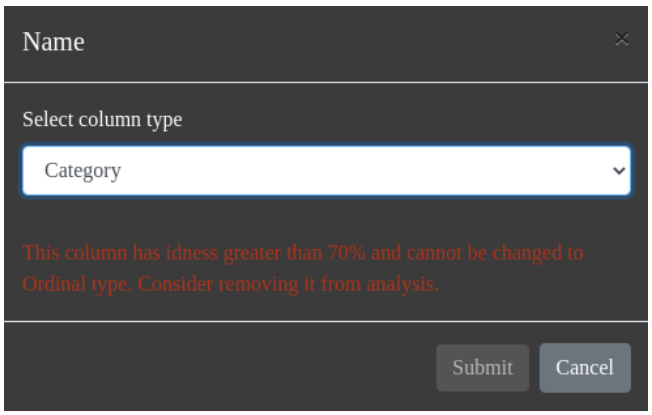
  Typ *Ordinal* wprowadza uporządkowanie do zmiennych typu *Category*, w postaci kolejnych liczb naturalnych zaczynając od 1.

Do ustawienia uporządkowania w typie *Ordinal* służy komponent pojawiający się w module zmiany typu danych, po wyborze typu *Ordianl*. W celu zmiany uporządkowania wartości należy zaznaczyć daną wartość, a następnie użyć strzałek góra-dół znajdujących się po prawej stronie listy.



Rysunek 54: Moduł zmiany typu danych dla typu *Ordinal*.

Zmiana typu kategorycznego na porządkowy (*Ordinal*) nie będzie możliwa w przypadku, gdy stosunek ilości unikalnych wartości do liczebności zbioru przekracza 70 procent. Ręczne uporządkowywanie tylu zmiennych nie miałoby sensu, więc usunięto taką możliwość. Użytkownikowi zostanie również wyświetlony komunikat informujący o tym, że dana kolumna posiada bardzo dużo unikalnych wartości kategorycznych.



Rysunek 55: Moduł wyboru typu danych wraz z powiadomieniem o wielu unikalnych wartościach kategorycznych w kolumnie.

Podczas obliczeń wszystkie wartości w kolumnach o typie kategorycznym są zamieniane za pomocą procesu *kodowania gorącojedynkowego*[54]. Jest to wymagane ze względu na to, że biblioteka *Sklearn* w wielu algorytmach nie akceptuje cech kategorycznych.

7.4.4.2 Ustawienia klasyfikacji i regresji

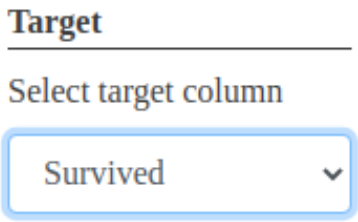
Obie metody mają na celu przewidzieć wartość pewnej cechy rekordu na podstawie pozostałych

[54]kodowanie gorącojedynkowe (ang. One Hot Encoding) - Metoda zastępowania każdej z klas typu kategorycznego, cechą binarną o wartościach 0 w wierszach nieposiadających tej klasy i wartościach 1 w wierszach posiadających daną klasę [17].

cech. Dlatego ustawienia dla obydwu metod nie różnią się znacznie. Różnica polega głównie na innym zestawie algorytmów do rozwiązywania problemów klasyfikacji i regresji.

7.4.4.2.1 Kolumna docelowa

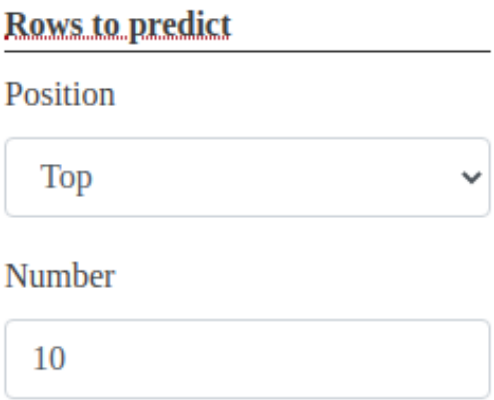
Kolumna docelowa to taka, dla której chcemy przewidzieć wartości na podstawie wartości w pozostałych kolumnach. Analizy klasyfikacji i regresji posiadają ustawienie kolumny docelowej, którą można wybrać z listy rozwijalnej. Na liście znajdują się jedynie kolumny, które nie zostały wyłączone z analizy. Domyślnie zaznaczona jest pierwsza kolumna.



Rysunek 56: Komponent wyboru kolumny docelowej.

7.4.4.2.2 Wiersze do przewidzenia

Chcąc przewidzieć wartości w kolumnie docelowej dla wybranych wierszy, należy umieścić takie wiersze na dole lub na górze zbioru danych w pliku csv. W kolumnie docelowej powinna znajdować się dowolna wartość o typie danych zgodnym z tą kolumną. Wiersze nie powinny posiadać wartości pustych, gdyż nie będzie wtedy możliwe wykonanie analizy na takich danych. Następnie w panelu po prawej stronie w części *Rows to predict* należy wybrać z listy rozwijalnej o nazwie *Position* to gdzie dane do przewidzenia zostały umieszczone. Mamy tam do wyboru opcje *Bottom* lub *Top*. Po wyborze pozycji określamy ilość wierszy, dla którch chcemy przewidzieć wartość cechy docelowej.



Rysunek 57: Komponent ustawienia wierszy do przewidzenia.

Takie ustawienia spowodują, że w kolejnym kroku tj. kroku rezultatów analizy, zostanie wygenerowana tabela z wartościami cech docelowych, obliczonych przez wybrany algorytm, na

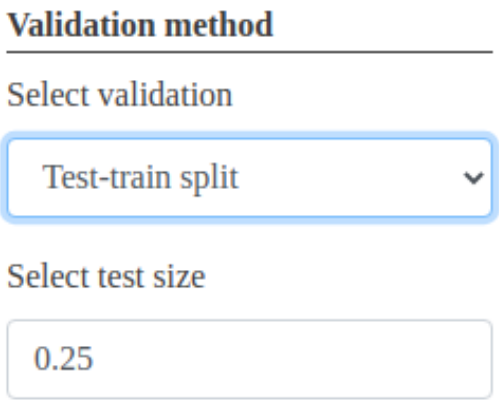
podstawie zaznaczonych wcześniej do przewidzenia wierszy. Domyślnie pozycja wierszy do przewidzenia to *Top*, a ich ilość to 10.

7.4.4.2.3 Metoda walidacji

Metoda walidacji określa sposób dzielenia danych na zbiory trenujące i testujące. Do wyboru mamy dwa typy walidacji:

- *Test-train split*

  Metoda ta polega na prostym podziale zbioru danych na dane uczące i dane testujące. Użytkownik może zdecydować o wielkości zbioru testującego. Domyślną wartością jest 0.25, czyli 25 procent całego zbioru. Wybór tej metody spowoduje, że szybciej otrzymamy rezultaty, natomiast mogą one nie być tak dokładne jak w przypadku metody *Cross validation*. Innym problemem tej metody jest zagadnienie niedopasowania. Może się zdarzyć, że dzieląc dane na zbiory uczące i testujące, pozbawimy zbiór danych uczących pewnych istotnych wzorców występujących w zbiorze danych [18].

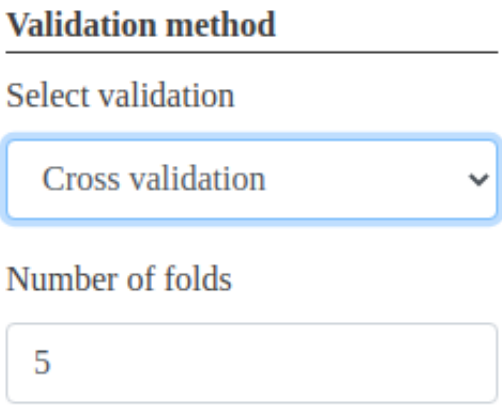


Rysunek 58: Komponent wyboru metody walidacji dla *Test-train split*.

  Dokładny opis algorytmu można znaleźć pod linkiem `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html`.

- *K-fold cross validation*

  Ta metoda niweluje problemy poprzedniej metody kosztem dłuższego wykonywania obliczeń. Zbiór danych dzielony jest na k zbiorów, po czym model trenowany jest na zbiorach k - *1*, a testowany na pozostałym zbiorze. Obliczenia wykonywane są do momentu, aż wyczerpią się wszystkie kombinacje takiego podziału danych. Wynikiem dokładności algorytmu jest średnia z wyników dla każdego podziału. Użytkownik może określić wartość k w przedziale od 4 do 10 [19].



Rysunek 59: Komponent wyboru metody walidacji dla *Cross validation*.

Dokładny opis algorytmu można znaleźć pod linkiem `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html`.

7.4.4.2.4 Algorytmy klasyfikacji

W metodzie klasyfikacji do dyspozycji mamy trzy algorytmy:

- *K-nearest neighbors*

  W tym algorytmie dany rekord klasyfikowany jest na podstawie jego najbliższych sąsiadów. Kluczową rolę odgrywa fakt, że algorytm oblicza odległości pomiędzy rekordami, więc konieczne jest zastosowanie skalowania danych w przypadku gdy dane tego wymagają [20].

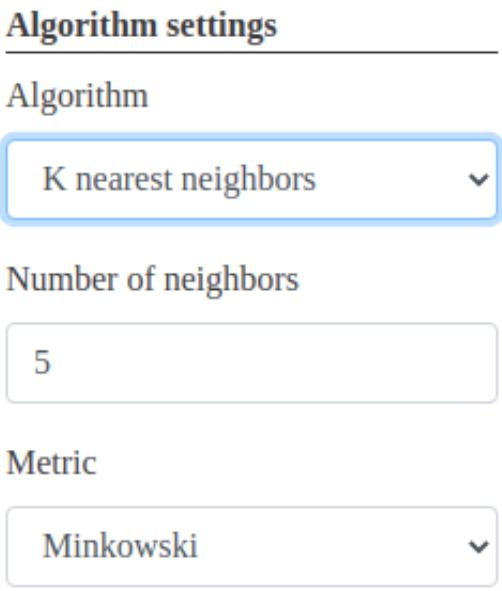
  Użytkownik może ustawić następujące parametry:

  – *Number of neighbors*

    Liczba najbliższych sąsiadów branych pod uwagę w obliczeniach. Jest to liczba naturalna w zakresie od 1 do 100. Wartością domyślną jest 5.

  – *Metric*

    Metryka odległościowa zastosowana w algorytmie. Dostępne metryki to *Minkowski*, *Chebyshev*, *Manhattan*, *Euclidean*



Rysunek 60: Ustawienia dla algorytmu *K-nearest neighbors*.

Dokładny opis algorytmu można znaleźć pod linkiem `https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html`.

- *Random forest*

  Podstawą tego algorytmu jest drzewo decyzyjne, złożone z serii reguł decyzyjnych. Każda reguła występuje w tak zwanym węźle decyzyjnym, a reguły tworzą gałęzie prowadzące do nowych węzłów. Gałąź bez reguły na końcu nosi nazwę liścia. Ten algorytm nie wymaga skalowania danych, gdyż nie jest oparty na metrykach odległości [21] [22].

  Użytkownik może ustawić następujące parametry:

  – *Number of trees*

    Liczba drzew decyzyjnych do zastosowania w algorytmie. Jest to liczba naturalna w zakresie od 1 do 1000. Wartością domyślną jest 100.

  – *Bootstrap*

    Określa czy podczas tworzenia drzewa wykorzystywane są podzbiory. Ustawienie tej wartości na *True* może zmniejszyć wariancję wyniku [55].

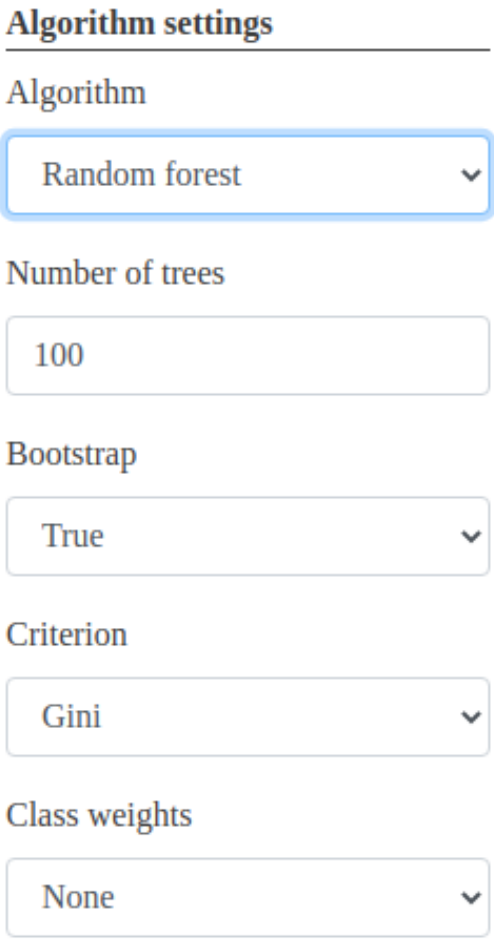
  – *Criterion*

    Określa jakie kryterium zostanie wykorzystane do dzielenia węzłów w drzewie decyzyjnym. Użytkownik ma do wyboru dwa kryteria: *Gini* oraz *Entropy*. Domyślną wartością jest *Gini*.

[55] *Wariancja* - określa jak, zmienne są wyniki przewidywań, dla zastosowania danego modelu dla różnych zbiorów danych

– *Class weights*

Określa czy wprowadzić współczynniki do przewidywanych klas. Aby wyrównać wpływ poszczególnych klas na obliczenia należy wybrać opcję *balanced*. Dla klas zrównoważonych nie zachodzi potrzeba stosowania wag klas i można zastosować opcję *None*. Domyślną opcją tego ustawienia jest *None*. Klasy można określić mianem *niezbalansowanych*, gdy nie występują w takich samych proporcjach w zbiorze wykorzystanym do trenowania modelu.



Rysunek 61: Ustawienia dla algorytmu *Random forest*.

Dokładny opis algorytmu można znaleźć pod linkiem `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html`.

- *Logistic regression*

Jest to technika klasyfikacji. Podstawowa jej wersja stanowi klasyfikator binarny tj. taki który klasyfikuje rekord do jednej z dwóch grup. Posiada ona jednak rozszerzenia pozwalające zastosować ją do problemu wieloklasowego, czyli takiego, w którym występuje więcej niż dwie klasy, do których klasyfikujemy rekord. Jednym z takich rozszerzeń jest *multinominal logistic regression*, czyli wielomianowa regresja logistyczna [23] [24].

Użytkownik może ustawić następujące parametry:

- *Max iterations*

  Maksymalna liczba wykonanych iteracji. Jest to liczba naturalna w przedziale od 1 do 10000. Wartością domyślną jest 100.

- *Penalty*

  Ustawienie penalizacji. Zastosowanie opcji *l2* zastosuje regularyzację[56] celem zmniejszenia wariancji wyniku. Opcja *None* nie wprowadzi regularyzacji. Opcją domyślną jest *l2*.

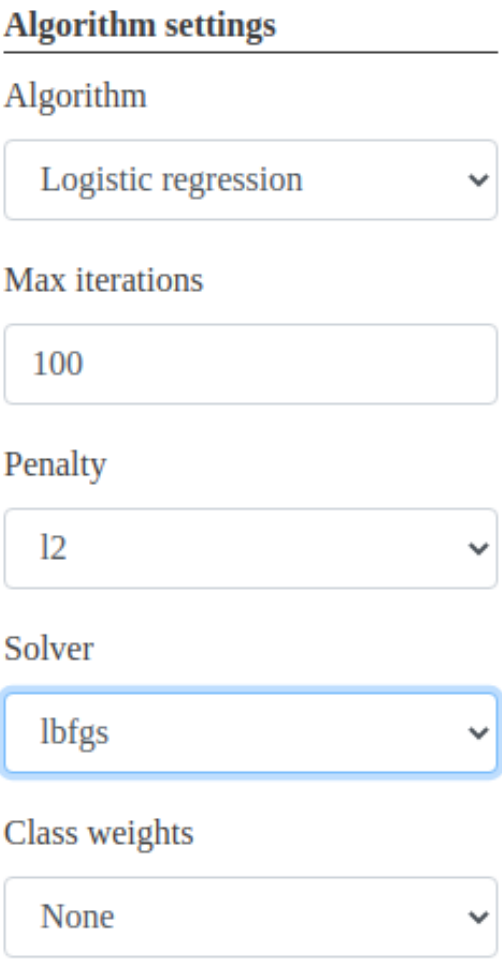
- *Solver*

  Algorytm do użycia. Opcją domyślną jest *lbfgs*. W przypadku dużych zbiorów danych należy zastosować opcję *sag*. Jest ona jednak bardzo wrażliwa na skalę danych, dlatego niezbędne jest przeskalowanie danych.

- *Class weights*

  Określa czy wprowadzić współczynniki do przewidywanych klas celem ich zrównoważenia. Należy zaznaczyć tą opcję jeśli przewidywane klasy występują w różnych proporcjach w zbiorach uczących. Domyślną opcją jest *None*

[56] regularyzacja - proces uogulnienia modelu stosowany w celu uniknięcia nadmiernego dopasowania.



Rysunek 62: Ustawienia dla algorytmu *Logistic regression*.

Dokładny opis algorytmu można znaleźć pod linkiem `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html`.

7.4.4.2.5 Algorytmy regresji

W metodzie regresji mamy do dyspozycji dwa algorytmy:

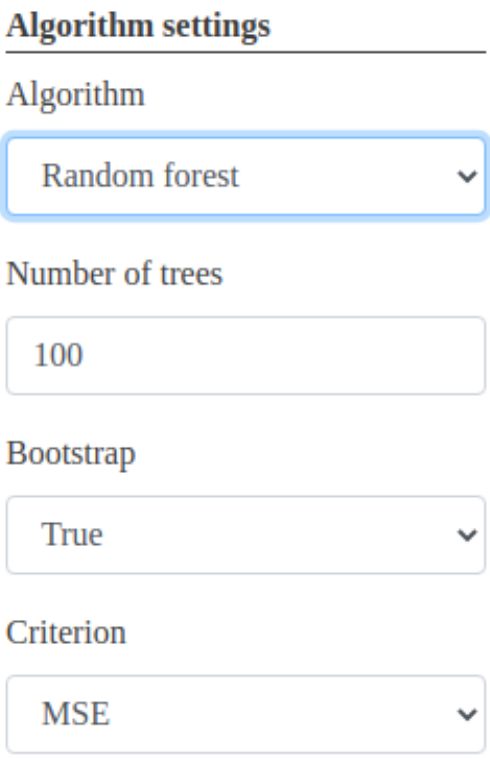
- *Random forest*

  Algorytm losowego lasu w wersji dla problemu regresji. Krótki opis algorytmu można znaleźć w punkcie 6.4.4.2.

  Użytkownik może ustawić następujące parametry:

  - *Number of trees* Liczba drzew decyzyjnych w algorytmie losowego lasu. Jest to liczba naturalna w zakresie 1 do 1000. Wartością domyślną jest 100.
  - *Bootstrap* Określa czy podczas tworzenia drzewa wykorzystywane są podzbiory. Ustawienie tej wartości na *True* może zmniejszyć wariancję wyniku. Wartością domyślną jest *True*.

– *Criterion* Funkcja pomiaru jakości podziału drzewa. Użytkownik ma do wyboru dwie opcje: *MSE*[57] oraz *MAE*[58].



Rysunek 63: Ustawienia dla algorytmu *Random forest regressor*.

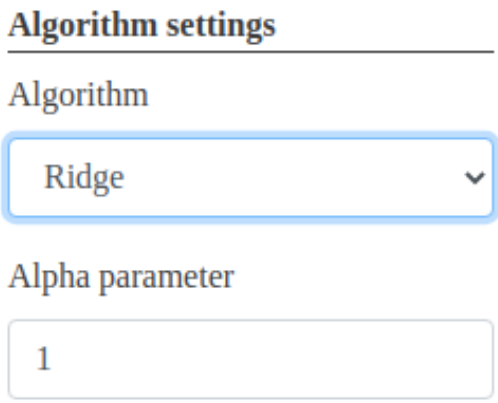
Dokładny opis algorytmu można znaleźć pod linkiem `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html`.

- *Ridge*

  Algorytm liniowych najmniejszych kwadratów z regularyzacją *l2* [26].

  Użytkownik może ustawić moc regularyzacji ustawiając *Alpha parameter*. Wartość tego parametru musi być liczbą dodatnią.

[57]MSE (ang. Mean square error) - błąd średniokwadratowy
[58]MAE (ang. Mean absolute error) - średni błąd bezwzględny



Rysunek 64: Ustawienia dla algorytmu *Ridge*.

Dokładny opis algorytmu można znaleźć pod linkiem `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html`

7.4.4.3 Ustawienia klasteryzacji

Celem klasteryzacji inaczej *analizy skupień* jest podział elementów na względnie jednorodne klasy. Podstawą grupowania algorytmu jest podobieństwo pomiędzy elementami wyrażone przy pomocy funkcji (*metryki*) podobieństwa. Grupowanie jest techniką klasyfikacji bez nadzoru (*klasyfikacja bezwzorcowa*[59]) i jest pojęciem z zakresu eksploracji oraz uczenia maszynowego [27].

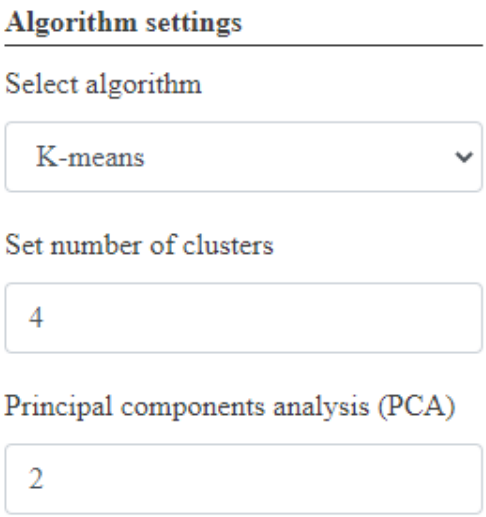
7.4.4.3.1 Algorytm

Najważniejszym elementem klasteryzacji jest odpowiedni wybór algorytmu oraz jego parametrów. Ustawienia nie różnią się znacznie w zależności od wybranego algorytmu, dlatego każdy będzie je wykorzystywał do swoich obliczeń. Aplikacja udostępnia możliwość wybrania dwóch głównych algorytmów klasteryzacji: *k-średnich (k-means)* [28] oraz *hierarchicznej aglomeracyjnej (Agglomerative Hierarchical)* [29]. Różnice między tymi dwoma algorytmami są znaczące:

- *Klastrowanie hierarchiczne* nie radzi sobie dobrze z dużymi danymi, ale *klastrowanie k-średnich* już tak. Wynika to z faktu, że złożoność czasowa k-średnich jest *liniowa*, tj. $O(n)$, natomiast złożoność hierarchiczna jest *kwadratowa*, tj. $O(n^2)$.
- W przypadku *klasteryzacji k-średnich* zaczynamy od losowego wyboru klastrów, wyniki uzyskane przez wielokrotne uruchomienie algorytmu mogą się różnić. Podczas gdy wyniki w *klasteryzacji hierarchicznej* są odtwarzalne.
- Stwierdzono, że algorytm *k-średnich* działa dobrze, gdy kształt skupisk jest *hipersferyczny (jak koło w 2D, kula w 3D)*.

[59] *Klasyfikacja bezwzorcowa (tzw. taksonomia)* - stosowana w sytuacji, kiedy strukturę klas należy dopiero co odkryć.

- Grupowanie *k-średnich* wymaga wcześniejszej znajomości ilości klastrów, na które dane mają być podzielone. Natomiast w klastrowaniu *hierarchicznym* istnieje możliwość zatrzymania się na dowolnej liczbie klastrów, które zostaną uznane za odpowiednie interpretując *dendrogram*[60].



Rysunek 65: Komponent konfiguracji algorytmu klasteryzacji.

Po wyborze algorytmu użytkownik musi zdecydować się na liczbę klastrów (*klas*) znajdującą się pod polem oznaczonym *Set number of clusters*. Parametr ten odpowiada za utworzenie pogrupowań podobnych do siebie obiektów w obszarze danego zbioru. Domyślnie jest ono ustawione na 3. Natomiast użytkownik może zmienić jego wartość w przedziale od 1 do 50.

Ostatnim ważnym parametrem w wyniku klasteryzacji jest *Principal components analysis (PCA)*. Analiza głównych składowych jest uważana za jeden z najpopularniejszych algorytmów redukcji wymiarów. W skrócie polega on na rzutowaniu danych do przestrzeni o mniejszej liczbie wymiarów tak, aby jak najlepiej zachować strukturę danych. Jej głównym zastosowaniem jest redukcja zmiennych opisujących dane zjawisko oraz odkrycie ewentualnych prawidłowości między cechami. Pozwala ona również wskazać zmienne początkowe mające ogromny wpływ na wygląd poszczególnych składowych głównych, czyli tych które tworzą grupę jednorodną. Oprócz redukcji liczby zmiennych, wykrywania struktur w związkach oraz prawidłowości i powiązań między nimi pozwala ona na klasyfikację obiektów w nowych przestrzeniach, co znacząco ułatwia proces wizualizacji wyników algorytmu klasteryzacji. Dlatego wartościami, jakimi użytkownik może się posłużyć wyznaczając *PCA* jest liczba 2 lub 3, z których można stworzyć model 2D, bądź 3D [30] [31].

[60] *Dendrogram* - diagram w kształcie drzewa przedstawiający hierarchiczne relacje między obiektami. Jest najczęściej tworzony jako wynik z hierarchicznego grupowania. Głównym zastosowaniem dendrogramu jest znalezienie najlepszego sposobu przydzielania obiektów do klastrów.

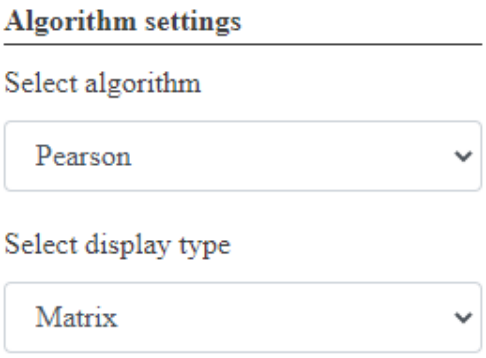
7.4.4.4 Ustawienia korelacji

Analiza korelacji w statystyce polega na zbadaniu czy dwie zmienne są ze sobą istotnie statystycznie powiązane. Innymi słowy, sprawdza czy jakiekolwiek dwie cechy, atrybuty lub własności (wyrażone liczbowo) współwystępują ze sobą. Obliczany współczynnik zawsze waha się od -1 do 1 [32].

Aplikacja ogranicza analizę korelacji do maksymalnie 10 cech. Spowodowane jest to faktem, że dla większej ilości cech jakość generowanego obrazu jest niewystarczająca do prawidłowego odczytu.

7.4.4.4.1 Algorytm

Konfiguracja analizy korelacji nie jest złożonym procesem. Wymaga jedynie wyznaczenia jednego z trzech współczynników, do których zaliczamy: *Pearson*[61], *Kendall*[62] oraz *Spearman*[63][33]. Zaprezentowane są one w formie rozwijalnej listy. Dodatkowo użytkownik musi określić, w jakiej formie będą wyświetlone dane w rezultacie analizy. Wyniki mogą zostać utworzone w dwóch postaciach. Pierwsza z nich to *macierz (Matrix)*, a druga wizualizowana jest za pomocą *mapy cieplnej (Heatmap)*.



Rysunek 66: Komponent konfiguracji algorytmu korelacji.

7.4.4.5 Sprawdzenie ustawień

[61] *Współczynnik korelacji liniowej Pearsona* - współczynnik określający poziom zależności liniowej między zmiennymi losowymi.

[62] *Tau Kendalla* – statystyka będąca jedną z miar monotonicznej zależności dwóch zmiennych losowych. Służy w praktyce do opisu korelacji między zmiennymi porządkowymi. Tau Kendalla jako metoda rangowa jest odporne na obserwacje odstające, w przeciwieństwie do współczynnika korelacji.

[63] *Współczynnik korelacji rang Spearmana* - jedna z nieparametrycznych miar monotonicznej zależności statystyczne między zmiennymi losowymi. Współczynnik ten jest wykorzystywany do opisu siły korelacji dwóch cech, gdy są one mierzalne, badana zbiorowość jest nieliczna oraz mają charakter jakościowy i istnieje możliwość ich uporządkowania. Miarę tę stosuję się również do badania zależności między cechami ilościowymi w przypadku niewielkiej liczby obserwacji.

Ustawienia w sekcji *Analyze* są sprawdzane, a informacja o błędnie wprowadzonych danych wyświetlana jest w prawym górnym rogu. Informacja ta pojawia się w momencie użycia przycisku *Analyze*, gdy wprowadzone dane nie spełniają nałożonych na nie ograniczeń. Ograniczenia te dotyczą ustawień niemożliwych do zastosowania z punktu widzenia zastosowanych algorytmów. Wprowadzono również ograniczenie minimalnej ilości rekordów do analizy równe 100 w celu uzyskiwania dokładniejszych wyników.

Rysunek 67: Walidacja ustawień w postaci komponentu typu *Toast*.

7.4.5 Krok wyników analizy - *Results*

Otrzymany wynik prezentowany jest w formie dokumentu, jak zobrazowano na poniższym rysunku.

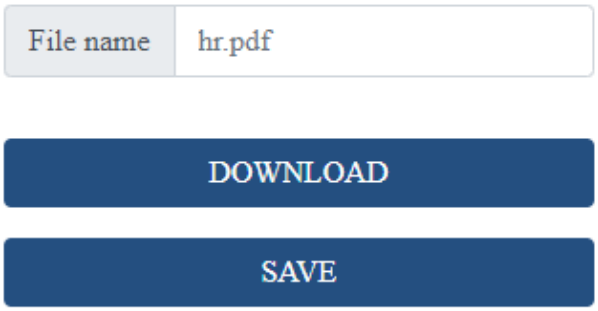
Rysunek 68: Widok rezultatu analizy.

Po lewej stronie widoku znajduje się panel zarządzania wynikiem analizy, a w centralnej jego części usytuowany jest widoczny dokument.

7.4.5.1 Panel zarządzania wynikami analiz

Panel zarządzania wynikami analiz powstał w celu pobrania lub zapisania dokumentu o dowolnej

nazwie wprowadzonej przez użytkownika z zachowaniem formatu pdf. Zapisane wyniki trafiają do magazynu raportów *Reports*, gdzie użytkownik może je przeglądać oraz nimi zarządzać.



Rysunek 69: Panel zarządzania wynikami analiz.

7.4.5.2 Wspólne elementy wyników analiz

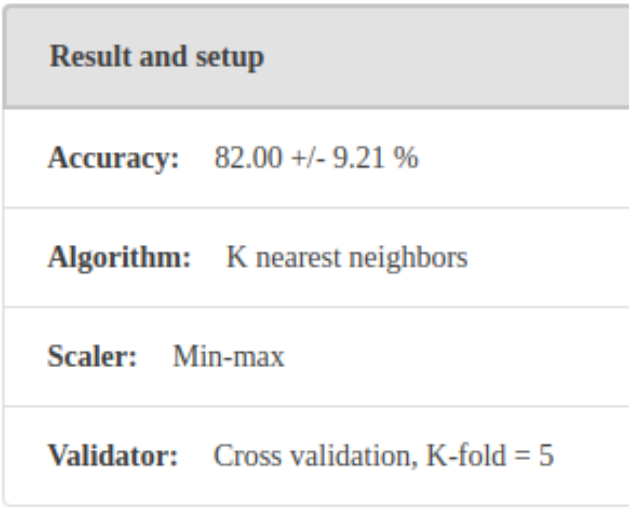
Wyniki wszystkich analiz zaprezentowane są w formie dokumentu. Tytuł dokumentu zawiera informacje dotyczące wybranej metody oraz nazwę pliku wprowadzoną przez użytkownika. Dodatkowo każda z analiz przekazuje parametry użyte w algorytmach do sekcji *Result and setup*, *Algorithm parameters* oraz *PCA parameters*.

7.4.5.2.1 Rezultat klasyfikacji i regresji

Rezultaty dla tych analiz są do siebie zbliżone pod względem formy. Dokument z rezultatami podzielony jest na 3 części: *Result and setup*, *Algorithm parameters* oraz *Predicted n values in column m*, gdzie n to liczebność przewidywanych wierszy, a m to nazwa kolumny docelowej.
W części *Result and setup* wartości zaliczane do wyników analizy są unikalne dla klasyfikacji i regresji.

- Klasyfikacja
  - *Accuracy*

    Dokładność wyniku klasyfikacji. Jest to stosunek liczby poprawnie przewidzianych klas cechy docelowej i całkowitej liczby przewidywań. W przypadku metody *Test-train split* wartość ta obliczana jest na podstawie zbioru testowego. W przypadku metody *Cross validation* dodatkowo wyświetlona jest dokładność przewidywania określana jako +/- 2 odchylenia standardowe obliczone na podstawie dokładności z podzbiorów testowych (w metodzie *Cross validation* jest kilka zbiorów testowych).



Rysunek 70: Pierwsza część rezultatów klasyfikacji.

- Regresja
  - R^2

    Współczynnik determinacji stanowiący miarę dopasowania modelu (jak model pasuje do próby). Obliczona jest również dokładność wyznaczonego współczynnika na podstawie zbiorów testowych o wartości +/- 2 odchylenia standardowe. [34]
  - *RSME*

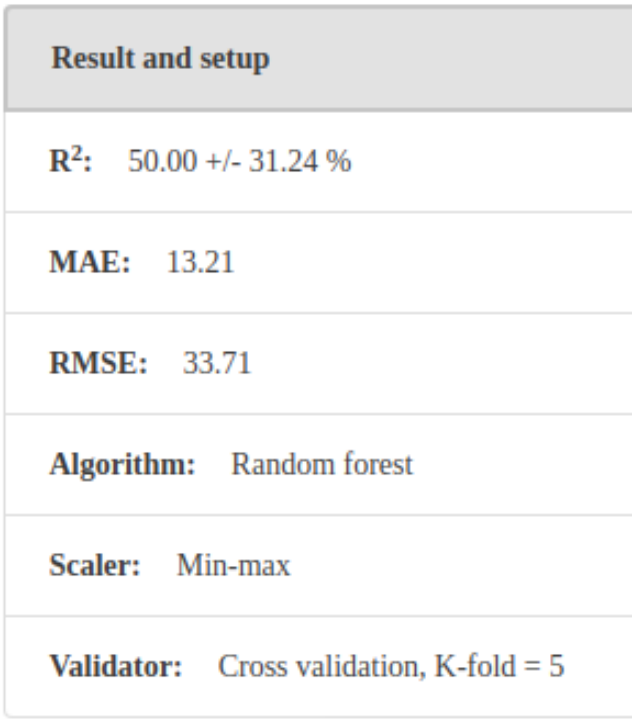
    Przedstawia średni błąd przewidywania na podstawie zbioru testowego.

$$RSME = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

  - *MAE*

    Tak jak *RSME* jest to miara błędu przewidywania na podstawie zbioru testowego.

$$MAE = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$



Rysunek 71: Pierwsza część rezultatów regresji.

W części *Result and setup* wartości zaliczane do ustawień (*setup*) są takie same dla obu typów analiz i są to:

- *Algorithm*

  Ustawiony algorytm.

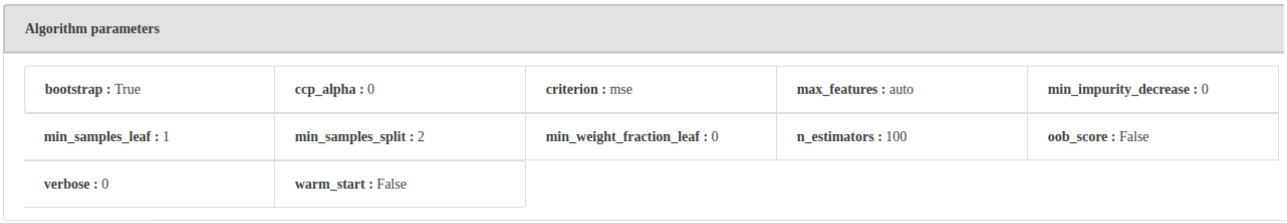
- *Scaler*

  Sposób skalowania danych.

- *Validator*

  Sposób walidowania danych.

Drugą częścią rezultatów są parametry algorytmu zastosowanego do analizy. Tylko część z tych parametrów można ustawiać w aplikacji. Pozostałe podane są w celach informacyjnych.



Rysunek 72: Parametry algorytmu w kroku wyników analiz.

Ostatnią częścią rezultatów jest tabela z danymi. Zawiera ona wiersze, dla których użytkownik ustawił przewidywanie wartości w kolumnie docelowej.

Predicted 10 values in column Survived

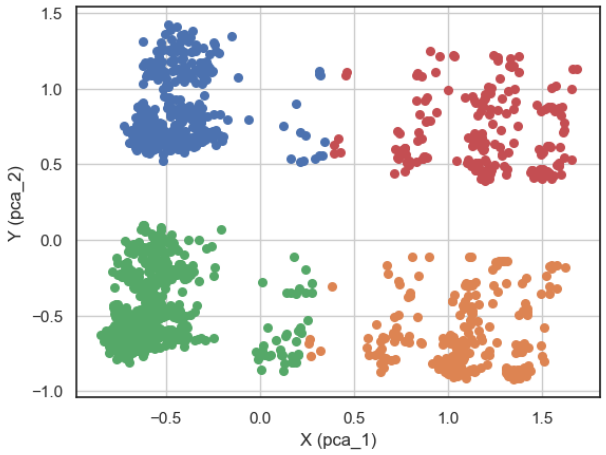
Survived	Probability [%]	index	Pclass	Name	Sex	Age	Siblings/Spou…	Parents/Child…	Fare
0	60	0	3	Mr. Owen Har…	male	22	1	0	7.25
1	100	1	1	Mrs. John Bra…	female	38	1	0	71.2833
1	60	2	3	Miss. Laina H…	female	26	0	0	7.925
1	100	3	1	Mrs. Jacques …	female	35	1	0	53.1
0	100	4	3	Mr. William H…	male	35	0	0	8.05
0	60	5	3	Mr. James Mo…	male	27	0	0	8.4583
0	60	6	1	Mr. Timothy J …	male	54	0	0	51.8625
0	100	7	3	Master. Gosta …	male	2	3	1	21.075
1	60	8	3	Mrs. Oscar W …	female	27	0	2	11.1333
1	100	9	2	Mrs. Nicholas …	female	14	1	0	30.0708

Rysunek 73: Tabela z wynikami klasyfikacji.

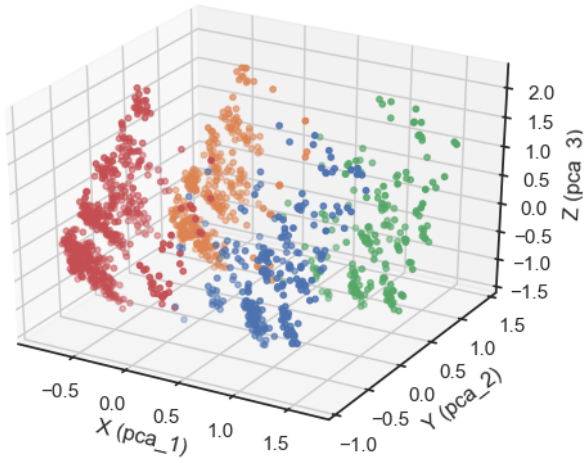
W powyższej tabeli widać, że pierwszą kolumną jest teraz kolumna docelowa. Zawiera ona obliczone przez algorytm wartości cechy docelowej. Następną kolumną jest prawdopodobieństwo poprawnego przewidzenia tej wartości. Oprócz tych dwóch kolumn dodana jest również kolumna index. Pozostałe kolumny są kolumnami ze zbioru danych wprowadzonego do analizy. W przypadku analizy regresji nie występuje kolumna z wynikiem prawdopodobieństwa.

7.4.5.2.2 Rezultat klasteryzacji

Wynik klasteryzacji w zależności od algorytmu zaprezentowany jest w dwóch formach. Pierwsza z nich dla algorytmu *k-średnich* tworzy wykres ukazujący podział elementów na względnie jednorodne klasy. W zależności od parametru *PCA*, czyli analizy głównych składowych rozmiar danych statystycznych zaprezentowany jest w wymiarze 2D lub 3D.

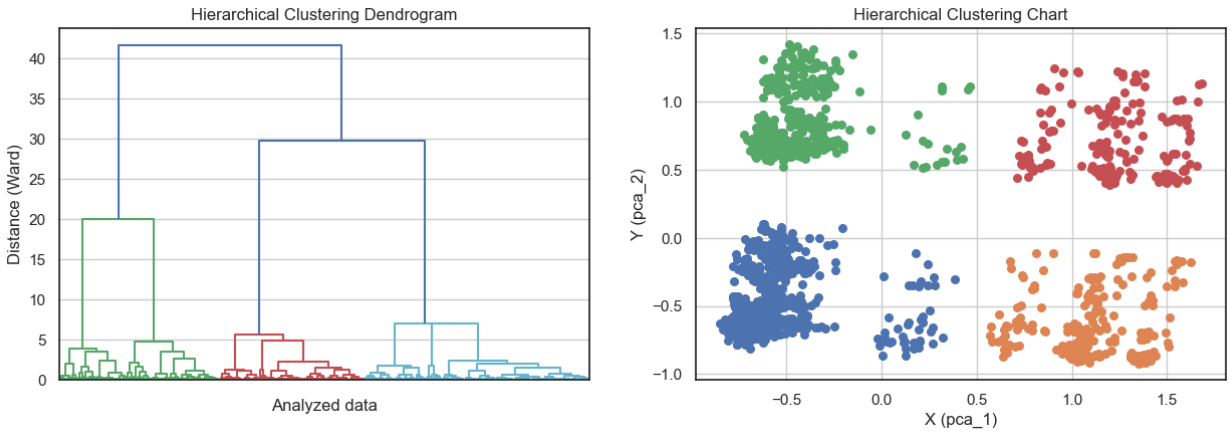


Rysunek 74: Wynik klasteryzacji metodą k-średnich w wymiarze 2D.



Rysunek 75: Wynik klasteryzacji metodą k-średnich w wymiarze 3D.

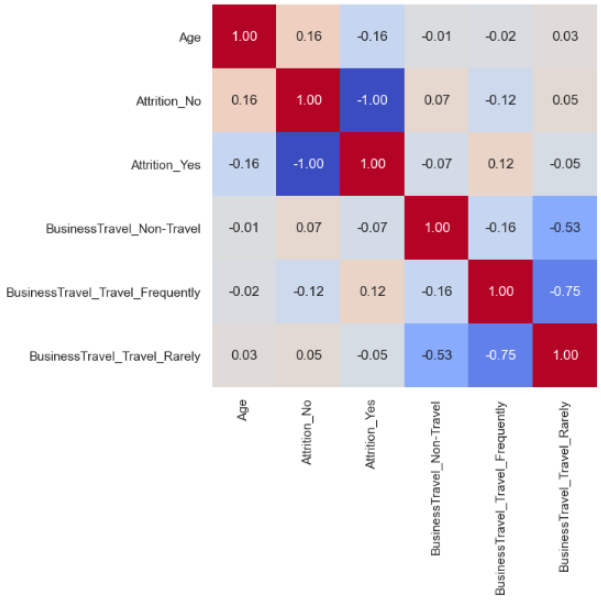
Druga forma wyniku dla algorytmu *hierarchicznego* tworzy *dendrogram* wraz z wykresem. Dendrogram jest diagramem w kształcie drzewa ukazującym związki pomiędzy wybranymi elementami na podstawie przyjętego kryterium.



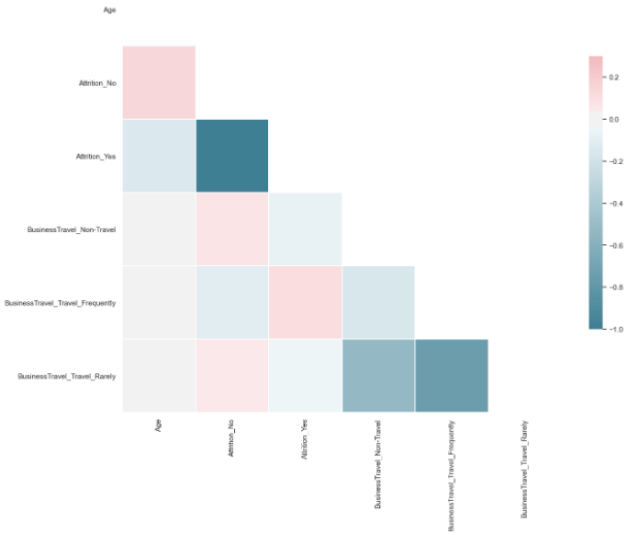
Rysunek 76: Wynik klasteryzacji metodą aglomeracyjno hierarchiczną w wymiarze 2D.

7.4.5.2.3 Rezultat korelacji

Uzyskane w procesie korelacji wyniki mogą zostać zaprezentowane za pomocą macierzy lub mapy cieplnej.



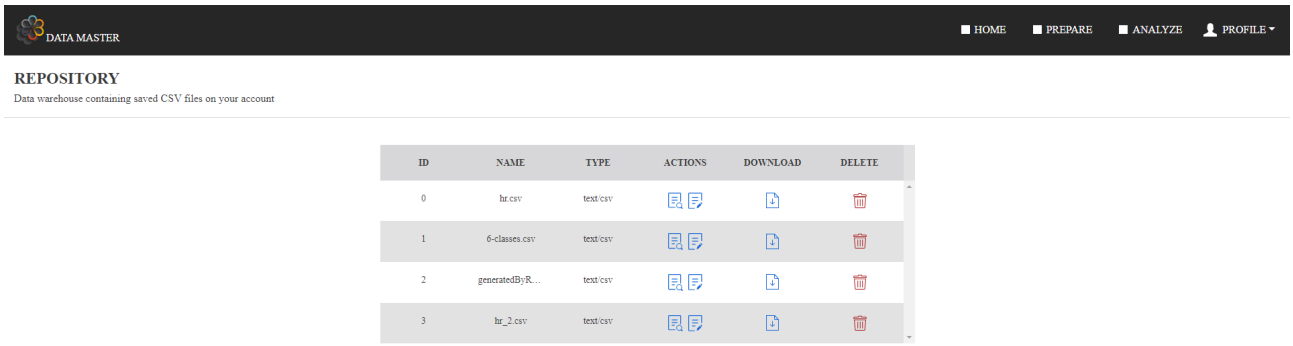
Rysunek 77: Prezentacja wyniku korelacji (postać macierzowa).



Rysunek 78: Prezentacja wyniku korelacji (mapa cieplna).

7.5 Sekcja magazynu danych - Repository

Aplikacja oprócz podstawowych funkcjonalności, jakimi są analiza i eksploracja danych, daje użytkownikowi możliwość magazynowania danych. Przechowywane pliki danych aktualnie zalogowanego użytkownika znajdują się w sekcji *Repository*, do której użytkownik może się dostać z panelu nawigacyjnego po rozwinięciu przycisku *Profile*, bądź bezpośrednio przez link `https://edufront.pl/repository`.



Rysunek 79: Sekcja Repository.

W repozytorium znajdują się dane wcześniej przygotowane w kroku *Prepare* oraz zapisane w formacie *text/csv*. Dostęp do magazynu plików użytkownika posiada tylko właściciel konta. Po wejściu do repozytorium widzimy tabelę. Zawiera ona zbiór wszystkich zapisanych przez użytkownika plików. Wśród najważniejszych informacji o danym pliku wyróżniamy:

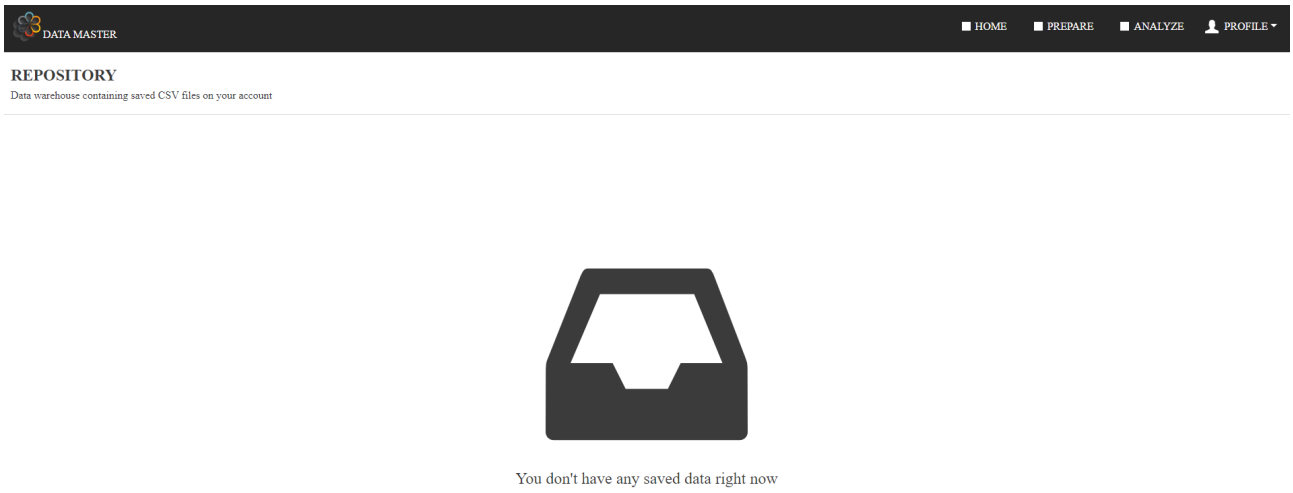
- nazwę pliku

- format pliku - pliki tworzone w sekcji *Prepare* zawsze będą miały format *text/csv*, natomiast aplikacja jest w stanie przechowywać pliki również innych formatów

Dodatkowo użytkownik może wykonać określone operacje na zapisanych plikach, takie jak:

- przejście do sekcji *Analyze* na wybranym zbiorze
- przejście do sekcji *Prepare* na wybranym zbiorze, w celu ponownej modyfikacji
- pobranie pliku z magazynu na dysk
- usunięcie pliku z magazynu

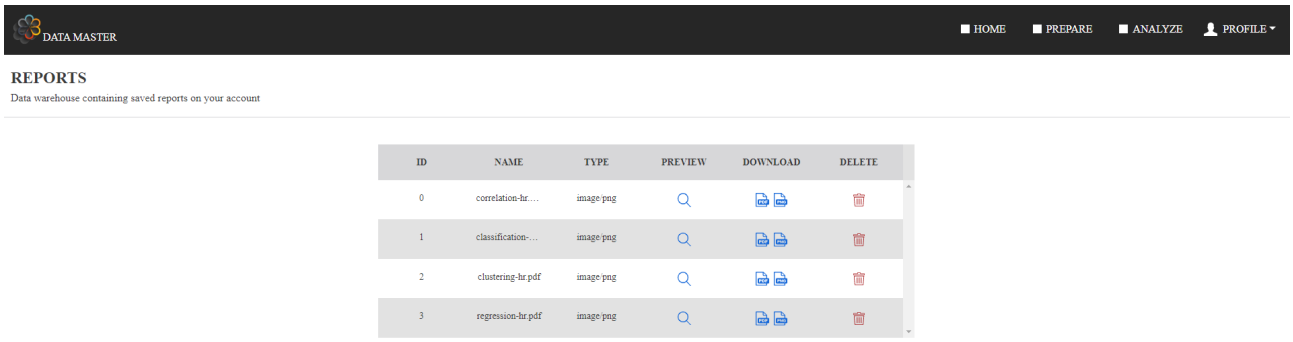
W przypadku gdy użytkownik nie posiada żadnych zapisanych plików bądź wszystkie zostały usunięte, tabela zamieniana jest na poniższy obrazek.



Rysunek 80: Pusty magazyn danych użytkownika.

7.6 Sekcja magazynu raportów - Reports

Dodatkową funkcjonalnością korzystną dla użytkownika jest sekcja magazynowania raportów pod nazwą *Reports*. Dostęp do niej można uzyskać z panelu nawigacyjnego po rozszerzeniu *Profile* bądź przez link `https://edufront.pl/reports`.

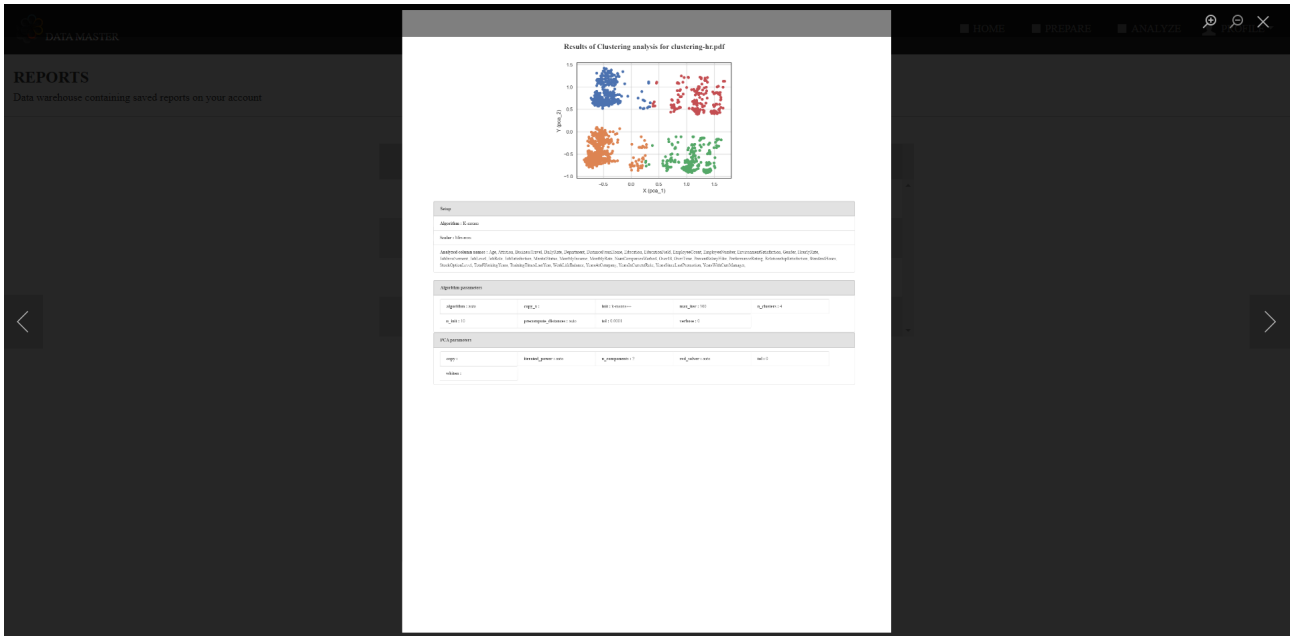


Rysunek 81: Strona magazynu raportów - Reports.

Magazyn raportów wygląda bardzo podobnie do magazynu danych sekcji *Repository*. Natomiast kluczowym elementem odróżniającym wspomniane sekcje są wykonywane operacje na przechowywanych zasobach.

Raporty zapisane w bazie danych są formatu *image/png*. Dzięki temu rozszerzeniu aplikacja przy użyciu języka *Javascript* wykona konwersję obrazka do typu *image/jpeg* lub dokumentu *pdf*, które możemy pobrać na własny dysk, przyciskając odpowiednie ikony w kolumnie *Download*.

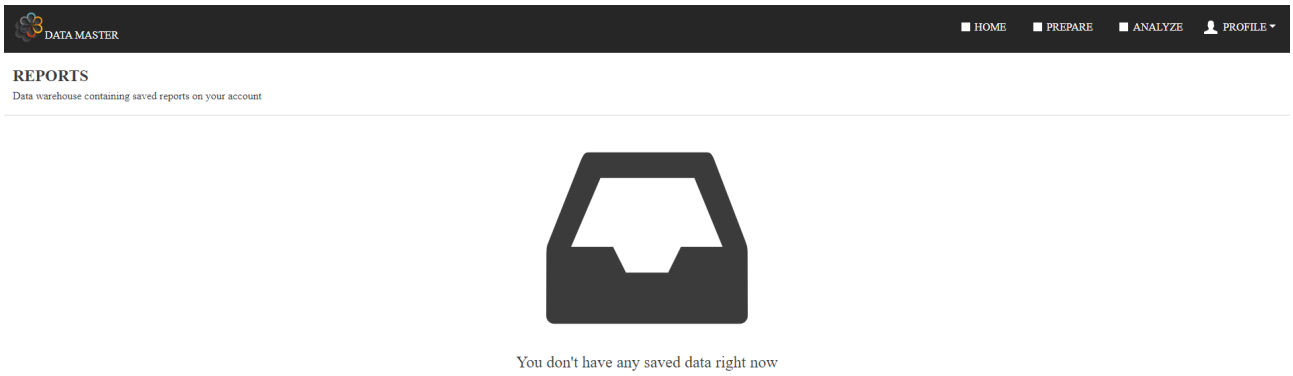
Aplikacja udostępnia również użytkownikowi możliwość przeglądania zapisanych raportów. Powstała ona z korzyścią dla właściciela konta, gdyż nie musi on pobierać pliku, mimo to ma dostęp do aktualnych danych. Wystarczy, że użytkownik kliknie ikonę lupy znajdującą się w kolumnie *Preview*, która otworzy wybrany dokument w formie galerii.



Rysunek 82: Komponent galerii do przeglądania raportów analiz.

Po otwarciu galerii użytkownik może poruszać się pomiędzy zapisanymi dokumentami strzałkami z prawej oraz lewej strony ekranu. Dodatkowo w prawym górnym rogu znajduje się ikona lupy

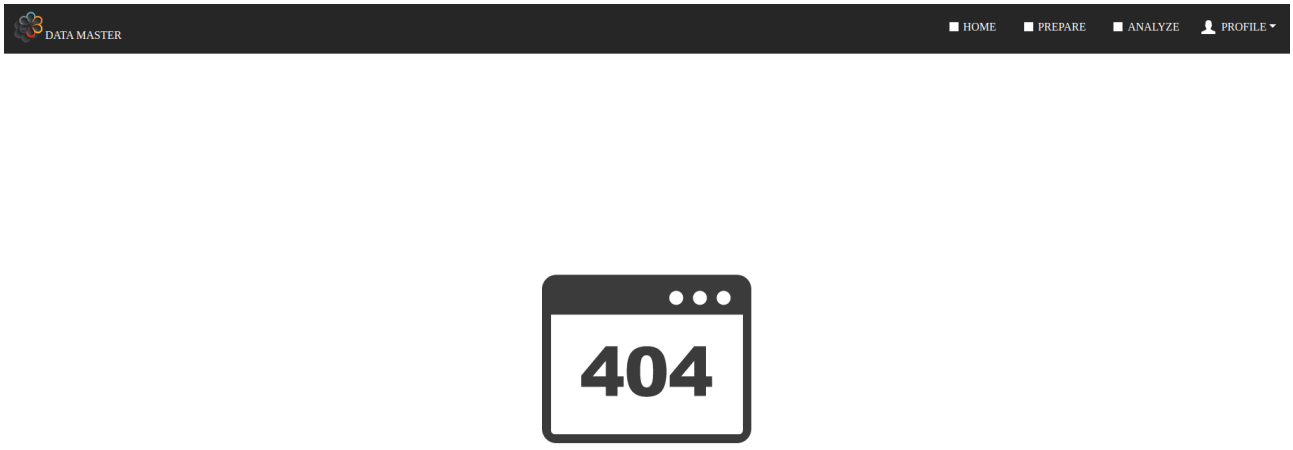
przybliżającej bądź oddalającej dokument. Przyciśnięcie ikony X powoduje zamknięcie galerii. Galeria posiada raporty aktualnie zapisane na koncie użytkownika, dlatego każde dodanie oraz usunięcie pliku spowoduje odświeżenie liczby zapisanych plików analiz.
Tak samo, jak w sekcji *Repository* użytkownik nieposiadający zapisanych dokumentów, otrzyma komunikat w postaci pustego zbioru dokumentów wraz z opisem poświadczającym *ang. "You don't have any saved data right now"*.



Rysunek 83: Pusty magazyn raportów użytkownika.

7.7 Podstrona nieistniejąca

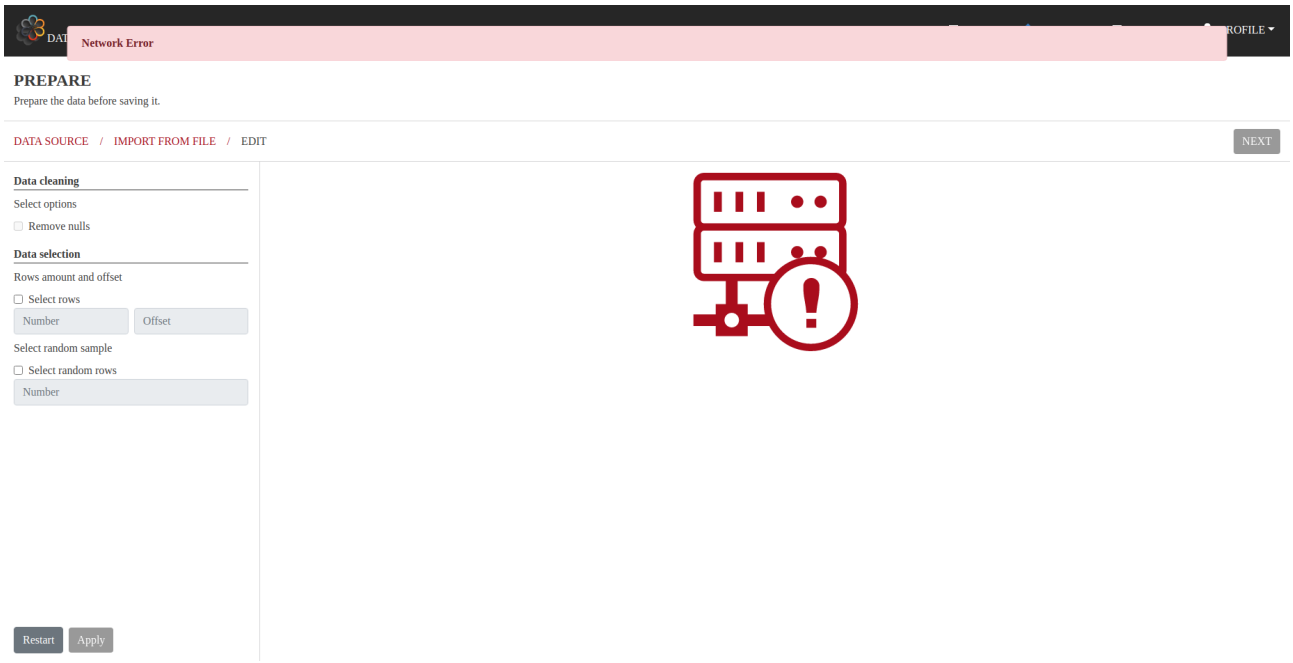
W przypadku wprowadzenia przez użytkownika adresu podstrony, która nie istnieje w obrębie serwisu, zostanie wyświetlona strona z grafiką 404.



Rysunek 84: Widok strony pod adresem niezawierającym zażdnej zawartości.

7.8 Obsługa błędów serwera

Błędy po stronie serwera są obsługiwane przez notyfikacje w całej aplikacji. Takie błędy mogą wystąpić np. gdy dane wprowadzone przez użytkownika nie spełniają wymagań algorytmów, gdy dane występujące w kolumnach są typów mieszanych, serwer przestanie działać i w innych tego typu sytuacjach.



Rysunek 85: Notyfikacja o błędzie sieci w kroku *Prepare*.

Na powyższym zrzucie ekranu widzimy notyfikację o błędzie po stronie serwera oraz grafikę błędu.

8 Podsumowanie i wnioski

Początkowa koncepcja projektu była bardzo ogólna. Aplikacja miała dotyczyć analizy i eksploracji danych, jednak nie sprecyzowano szczegółowego planu jej realizacji. Zaczęto od postawienia ogólnych wymagań funkcjonalnych, a forma tworzonej aplikacji kształtowała się podczas implementacji oraz na podstawie napotkanych problemów. Wraz z poznawaniem tematyki uczenia maszynowego testowano różne rozwiązania w celu znalezienia optymalnego. Udało się sprostać postawionym wymaganiom funkcjonalnym oraz rozszerzono je. W obecnym stanie aplikacja pozwala na przygotowanie danych do analizy, przeprowadzenie analizy oraz magazynowanie zarówno wyników analiz, jak i samych danych.

W sekcji przygotowania danych użytkownik ma możliwość obsługi wartości brakujących w zbiorze, uzupełniając je na podstawie pozostałych danych, lub po prostu usuwając je. W tym samym procesie można również zaznaczyć podzbiór danych lub usunąć kolumny. Po

przeprowadzeniu tego procesu, dane można zapisać na dysk komputera lub do repozytorium aplikacji.

W sekcji analizy danych użytkownik może wybrać jeden z czterech typów analiz oraz ustawić ich argumenty, aby najlepiej pasowały do wybranego zbioru danych. Po przeprowadzeniu analizy wyświetlone są wyniki, które również można pobrać na dysk własnego komputera bądź zapisać w repozytorium aplikacji.

Oprócz funkcjonalności pozwalających operować na danych aplikacja posiada również dwie sekcje przeznaczone do ich magazynowania i tak repozytorium aplikacji magazynuje pliki *csv*, a sekcja raportów zbiera wyniki analiz i udostępnia możliwość dynamicznego przeglądania ich.

W kwestii możliwych usprawnień aplikacji można wspomnieć wydajność oraz obsługę wielu zapytań jednocześnie. W obecnym stanie aplikacja jest umieszczona na serwerze o bardzo niewielkich zasobach, co mocno ogranicza jej możliwości obsługi dużego obciążenia serwera. Rozwiązaniem tego problemu może być zastosowanie serwera o lepszych parametrach oraz balansowanie zapytań. Dodatkowo w kwestii wykonywanych analiz, bardzo dobrą poprawą funkcjonalności byłaby możliwość zapisywania stworzonych modeli do późniejszych analiz. Obecna implementacja wymaga każdorazowego przejścia przez krok analizy z celu otrzymania rezultatów.

Aplikacja dzięki zastosowanej architekturze mikroserwisów w pełni wykorzystuje potencjał każdej użytej technologi. W ten sposób framework React oraz biblioteka Redux pozwoliła stworzyć w pełni skalowalny, oraz przejrzysty interfejs graficzny serwisu *Datamaster-ui*. Framework Spring oraz Hibernate zapewniły doskonałe zaplecze dla aplikacji *Datamaster-service* odpowiedzialnej za zabezpieczenie, oraz zarządzanie zbiorami danych analiz i ich wyników. Natomiast język Python z REST-owym frameworkiem Flask i bibliotekami Scikit-learn oraz Pandas pozwoliły wykorzystać algorytmy niezbędne do eksploracji danych w serwisie *Datamaster-engine*.

9 Załączniki

1. Kod źródłowy datamaster-service: `https://github.com/ArekMich/datamaster-service`.
2. Kod źródłowy datamaster-ui: `https://github.com/erykjarocki/datamaster-ui`.
3. Kod źródłowy datamaster-engine: `https://github.com/erykjarocki/datamaster-engine`.
4. Dokumentacja kodu źródłowego datamaster-service: `https://edufront.pl/docs/service`.
5. Dokumentacja kodu źródłowego datamaster-ui: `https://edufront.pl/docs/ui`.
6. Dokumentacja kodu źródłowego datamaster-engine: `https://edufront.pl/docs/engine`.

Bibliografia

[1] *Spring.io, Spring Framework*: https://spring.io/projects/spring-framework

[2] *Tutorialspoint, Spring Boot*: https://www.tutorialspoint.com/spring_boot/index.htm

[3] *Spring.io, Spring Boot*: https://spring.io/projects/spring-boot

[4] *Baeldung, Spring Security*: https://www.baeldung.com/security-spring

[5] *Michał Sajdak, Securitum, Bezpieczeństwo aplikacji webowych, rozdz."Niebezpieczeństwa JSON Web Token (JWT)" str. 607, Wydanie 1, 2019*

[6] *RESTful, REST API Tutorial*: https://restfulapi.net/

[7] *Swagger, Best Practices in API Design*: https://swagger.io/resources/articles/best-practices-in-api-design/

[8] *Link do zdjęcia działania narzędzia Webpack*: https://webpack.js.org/

[9] *The Benefits of Server Side Rendering Over Client Side Rendering - Alex Grigoryan*: https://medium.com/walmartglobaltech/the-benefits-of-server-side-rendering-over-client-side-rendering-5d07ff2cefe8

[10] *Wikipedia, Domain name system*: https://pl.wikipedia.org/wiki/Domain_Name_System

[11] *Oracle, How to Write Doc Comments for the Javadoc Tool*: https://www.oracle.com/technical-resources/articles/java/javadoc-tool.html

[12] *Sphinx, Python Documentation Generator*: https://www.sphinx-doc.org/en/master/usage/quickstart.html

[13] *Rinohtype, The Python Document Processor*: https://rinohtype.readthedocs.io/en/latest/

[14] *Min-max* scaler w bibliotece *Sklearn*: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

[15] *Standard* scaler w bibliotece *Sklearn*: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

[16] *Robust* scaler w bibliotece *Sklearn*: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html

[17] *Uczenie maszynowe w Pythonie, Chris Albon* - Rozdział 5, Obsługa danych kategoryzujących.

[18] *Test-train split* w bibliotece *Sklearn*: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

[19] *K-fold cross validation* w bibliotece *Sklearn*: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html#sklearn.model_selection.KFold

[20] *K-nearest neighbors* w bibliotece *Sklearn*: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

[21] *Random forest classifier* w bibliotece *Sklearn*: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[22] *Uczenie maszynowe w Pythonie, Chris Albon* - Rozdział 14, Drzewa i lasy.

[23] *Uczenie maszynowe w Pythonie, Chris Albon* - Rozdział 16, Regresja logistyczna.

[24] *Logistic regression* w bibliotece *Sklearn*: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

[25] *Random forest regressor* w bibliotece *Sklearn*: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

[26] *Ridge* w bibliotece *Sklearn*: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html

[27] *Wikipedia, Analiza skupień*: https://pl.wikipedia.org/wiki/Analiza_skupień

[28] Metoda klasteryzacji *K-means* w bibliotece *Sklearn*: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

[29] Metoda klasteryzacji *Agglomerative Hierarchical* w bibliotece *Sklearn*: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html

[30] Metoda analizy głównych składowych *PCA* w bibliotece *Sklearn*: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

[31] *Victor Powell, Principal Component Analysis*: https://setosa.io/ev/principal-component-analysis/

[32] *Korelacja - Pogotowie Statystyczne, Analiza Korelacji*: https://pogotowiestatystyczne.pl/slowniczek/korelacja/

[33] Metoda *korelacji* w bibliotece *Pandas*: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.corr.html

[34] Wikipedia, współczynnik determinacji: https://en.wikipedia.org/wiki/Coefficient_of_determination

Spis rysunków