

**POLITECHNIKA BIAŁOSTOCKA**  
**WYDZIAŁ INFORMATYKI**

**PRACA DYPLOMOWA INŻYNIERSKA**

**TEMAT: APLIKACJA INTERNETOWA DO OBSŁUGI  
SYSTEMU GDT W TECHNOLOGII  
PYTHON/DJANGO.**

**WYKONAWCA: MATEUSZ PERNAL**

.....  
podpis

**PROMOTOR: DR INŻ. KRZYSZTOF JURCZUK**

.....  
podpis

**BIAŁYSTOK 2020 r.**

## Karta dyplomowa

Politechnika Białostocka Wydział Informatyki  Katedra Oprogramowania	Studia stacjonarne studia I stopnia	Numer albumu studenta: 101420  Rok akademicki 2019/2020  Kierunek studiów: informatyka Specjalność: Brak
---	--	---

**Mateusz Pernal**

**TEMAT PRACY DYPLOMOWEJ:** Aplikacja internetowa do obsługi systemu GDT w technologii Python/Django.

Zakres pracy:

1. Zapoznanie z systemem GDT
2. Analiza wymagań aplikacji
3. Projekt i implementacja aplikacji
4. Testy oraz wdrożenie aplikacji.

..... Imię i nazwisko promotora - podpis	..... Imię i nazwisko kierownika katedry - podpis	
..... Data wydania tematu pracy dyplomowej - podpis promotora	..... Regulaminowy termin złożenia pracy dyplomowej	..... Data złożenia pracy dyplomowej - potwierdzenie dziekanatu
..... Ocena promotora		..... Podpis promotora
..... Imię i nazwisko recenzenta	..... Ocena recenzenta	..... Podpis recenzenta

Subject of diploma thesis: Internet application to support the GDT system in Python/Django technology.

## Summary

The aim of this thesis was to design, implement and introduce a web application to support GDT (*Global Decision Trees*) system using Python and Django technologies. In the basic version, GDT is a console program for creating decision trees. The requirement of the project was to create a web application allowing to create, delegate and manage the tasks launched by using the GDT system. The developed tool will also provide graphical representation of the obtained results.

The thesis consists of four chapters. First chapter describes the problem and presents the decision trees. It also shows the most popular existing solutions. Second chapter contains an analysis of project requirements and a overview of the technologies that have been used. Third chapter is intended to present the system architecture. It illustrates most important mechanisms and solutions created for the application. It also shows the database schema with a short description of tables. Fourth chapter contains a overview of particular views of the application, but also an example how to use them. In addition, it presents the results of load and manual tests.

# **Spis treści**

<b>Streszczenie</b>	<b>3</b>
<b>Wprowadzenie</b>	<b>5</b>
<b>1 Przedstawienie problemu</b>	<b>8</b>
1.1 Drzewa decyzyjne . . . . .	8
1.2 Uczenie maszynowe . . . . .	8
1.3 Drzewa decyzyjne w technikach uczenia maszynowego . . . . .	9
1.4 Istniejące rozwiązania . . . . .	10
<b>2 Wizja aplikacji</b>	<b>15</b>
2.1 Wymagania funkcjonalne . . . . .	15
2.2 Wymagania niefunkcjonalne . . . . .	20
2.3 Wykorzystane technologie . . . . .	20
<b>3 Architektura rozwiązania</b>	<b>25</b>
3.1 Architektura aplikacji . . . . .	25
3.2 Przechowywanie danych . . . . .	26
3.3 Wdrożenie aplikacji . . . . .	28
<b>4 Prezentacja aplikacji oraz testy</b>	<b>34</b>
4.1 Przedstawienie aplikacji . . . . .	34
4.2 Testy aplikacji . . . . .	38
<b>Podsumowanie</b>	<b>46</b>
<b>Bibliografia</b>	<b>48</b>
<b>Spis tabel</b>	<b>49</b>
<b>Spis rysункów</b>	<b>51</b>
<b>Spis listingów</b>	<b>52</b>

# Wprowadzenie

Proces myślowy człowieka jest w dużej mierze oparty o pewien schemat podejmowania decyzji. Podejmowane decyzje mają kluczowy wpływ na jego aktualne życie i przyszłość. Wybór najbardziej optymalnego rozwiązania danego problemu wymaga dokładnej analizy dostępnych informacji. Posiadając wystarczającą ilość danych możemy wykorzystać różne algorytmy, które mogą pomóc podjąć właściwy wybór. Mechanizm podejmowania decyzji bezpośrednio dotyczy nie tylko człowieka, a wszystkiego co znajduje się w jego otoczeniu.

Szybki rozwój technologii w XX i XXI wieku prowadzi do produkowania i gromadzenia coraz większej ilości informacji. Firmy starają się wyciągnąć ze zgromadzonych danych możliwe jak najlepsze wnioski. Poddając analizie tak duże zbiory informacji wymagane jest zastosowanie narzędzi uproszczających i przyśpieszających uzyskanie wyników. Prowadzi to do tworzenia algorytmów oraz mechanizmów zarówno obróbki danych, jak i ich analizy w celu osiągnięcia zadowalających rezultatów. W przeciągu ostatnich kilkunastu lat entuzjazm związany z technikami komputerowymi wzrósł gwałtownie i zdominował przemysł. Uczenie maszynowe wraz z analizą danych stanowi bardzo ważny element rozwiązań produkowanych przez firmy. Wspomaga takie technologie, jak rozpoznawanie mowy, pisma czy też autonomiczne samochody i roboty sprzątające. Wszystkie te rozwiązania wymagają przetwarzania ogromnych ilości informacji, w jak najkrótszym czasie oraz podjęcie wystarczająco dobrej decyzji.

W pracy tej rozwijany będzie system do uczenia maszynowego GDT (*Global Decision trees*)<sup>[1]</sup> tworzony przez pracowników Politechniki Białostockiej. System ten służy do generowania drzew decyzyjnych na podstawie zbioru uczącego. Drzewa generowane są z wykorzystaniem algorytmów ewolucyjnych (metoda alternatywna do algorytmów zachłannych typu *top-down*). Aplikacja GDT jest programem konsolowym. W celu ułatwienia dostępu do platformy GDT większemu gronu użytkowników w niniejszej pracy zostanie zaprojektowana, zaimplementowana oraz wdrożona aplikacja do obsługi systemu GDT z poziomu przeglądarki internetowej.

## **Cel pracy**

Celem pracy jest stworzenie aplikacji webowej umożliwiającej obsługę systemu GDT. Aplikacja ta będzie umożliwiać tworzenie, zlecanie oraz zarządzanie zadaniami uruchamianymi przy pomocy systemu. Podczas tworzenia zadań użytkownik powinien móc ustawić opcje dotyczące, np. wybranego algorytmu oraz jego parametrów. Aplikacja powinna także udostępniać opcje związane z wyświetleniem drzewa wynikowego w postaci graficznej, jego eksport do pliku oraz wgląd do pozostałych wyników uruchomianego algorytmu.

## **Zakres pracy**

Zakres pracy obejmuje:

- Zapoznanie z systemem GDT,
- Analiza wymagań aplikacji,
- Projekt i implementacja aplikacji,
- Testy oraz wdrożenie aplikacji.

## **Organizacja pracy**

Praca została podzielona na cztery główne części. Pierwsze dwa rozdziały zawierają przedstawienie problemu, analizę wymagań i wykorzystane technologie. W dalszej części pracy została omówiona architektura aplikacji wraz z zastosowanymi rozwiązaniami. Natomiast prezentacja stworzonej aplikacji oraz opis testów został przedstawiony w rozdziale 4.

Rozdział 1 zawiera opis mechanizmu tworzenia drzew decyzyjnych. Przedstawia również zagadnienia związane z systemem GDT. Porusza też temat podobnych aplikacji dostępnych w internecie.

Rozdział 2 przedstawia analizę wymagań funkcjonalnych i niefunkcjonalnych tworzonych aplikacji. Został w nim zamieszczony diagram przypadków użycia wraz z ich opisem oraz diagram czynności. Następnie zaprezentowany został schemat rozwiązania. Rozdział kończy przedstawienie użytych technologii podczas tworzenia aplikacji.

Rozdział 3 przedstawia architekturę tworzonej aplikacji. Prezentuje najważniejsze mechanizmy oraz rozwiązania stworzone na potrzeby aplikacji. Przedstawia także schemat bazy danych wraz z krótkim opisem najważniejszych tabel.

Rozdział 4 przedstawia stworzoną aplikację. Zawiera on opis poszczególnych widoków aplikacji, ale także przykładowy sposób ich użycia. Ponadto przedstawia wyniki testów obciążeniowych i manualnych przeprowadzonych przez studentów Wydziału Informatyki Politechniki Białostockiej.

# 1. Przedstawienie problemu

## 1.1 Drzewa decyzyjne

Podejmowanie decyzji jest procesem nietrywialnym. Już od początku istnienia ludzkości dobrze podjęte decyzje pozwalały przeżyć wybranym grupom ludzi czy też zwierząt. Wpływ na optymalną decyzję mają informacje, które zostaną poddane analizie, ale także sama metoda analizy. Racjonalny wybór może być wspomagany różnymi algorytmami, czy też wizualną reprezentacją możliwych decyzji. Jedną z form graficznych jest drzewo decyzyjne.

Podstawowymi elementami drzewa są węzły oraz gałęzie. Korzeń drzewa to pierwszy węzeł od którego rozpoczyna się budowa całej struktury zawierającej poszczególne węzły odpowiadające za sprawdzenie pewnego warunku. Natomiast gałęzie pełnią rolę połączenia pomiędzy węzłami na kolejnych poziomach drzewa [2]. Liście są końcowymi wierzchołkami drzewa i zawierają decyzje. Aby otrzymać decyzję konieczne jest przejście całego drzewa od samego korzenia do wynikowego liścia. Rezultatem takiej operacji będzie klasa decyzyjna (w przypadku drzew klasyfikacyjna) lub np. model regresyjny (w przypadku drzew regresyjnych).

## 1.2 Uczenie maszynowe

W otaczającym nas świecie ilość generowanych oraz gromadzonych informacji nadal przewyższa ilość danych, które można przeanalizować z użyciem obecnych zasobów. Aby analizować tak duże ilości informacji wykorzystywane są najnowsze rozwiązania technologiczne, zarówno na poziomie sprzętu komputerowego oraz oprogramowania. Dzięki zastosowaniu różnych algorytmów przetwarzania danych, klasyfikacji oraz predykcji programy komputerowe posiadają możliwość uczenia się. Tzn. uczenie maszynowe (ang. *machine learning*) jest obszarem sztucznej inteligencji, który zajmuje się wykorzystywaniem komputerowego wspomagania lub podejmowania decyzji. Uczenie maszynowe w przeciągu ostatniej dekady stało się tak popularne, iż w dużej mierze zdominowało przemysł sztucznej inteligencji oraz przyczyniło się do jej rozwoju [3]. Uczenie maszynowe stanowi trzon wielu usług, serwisów i aplikacji. Pod względem algorytmicznym odpowiada za wyniki wyszukiwania w przeglądarkach oraz za rozpoznawanie mowy przez nasze telefony. Jest

także wykorzystywane w dużo trudniejszych zadaniach, jak sterowanie autonomicznymi samochodami, czy też wspomaganie lotów kosmicznych i operacji chirurgicznych w medycynie.

### **1.3 Drzewa decyzyjne w technikach uczenia maszynowego**

Drzewa decyzyjne stanowią jeden z najbardziej rozpowszechnianych mechanizmów w obszarze uczenia maszynowego. Z jednej strony mogą być wykorzystywane w zadaniach z zakresu klasyfikacji, a z drugiej strony również odgrywają ważną rolę w regresji [3]. Mechanizm drzew decyzyjnych pozwala na budowanie modeli na podstawie ogromnych zbiorów uczących. Dodatkowym atutem drzew jest możliwość wizualnego przedstawienia sposobu dojścia do rozwiązania, które będzie zrozumiałe dla osób nie mających do czynienia z uczeniem maszynowym i statystyką.

#### **1.3.1 System GDT**

Pracownicy Wydziału Informatyki Politechniki Białostockiej od ponad 20 lat tworzą i rozwijają narzędzie do indukowania drzew decyzyjnych, nazwane GDT (*Global Decision Trees*). Narzędzie te zostało wykorzystane w pracy inżynierskiej. GDT służy do generowania drzew decyzyjnych na podstawie zbiorów wejściowych [1]. System ten jest zaimplementowany w języku C++. Podstawowa jego wersja jest programem konsolowym. Całe rozwiązanie jest unikalne, a głównym założeniem jest wykorzystanie algorytmów ewolucyjnych w procesie indukcji drzew decyzyjnych. Używając algorytmów ewolucyjnych budowane drzewa są bardziej globalne (trudniej wpaść w minimum lokalne) niż w klasycznym podejściu. Skutkuje to możliwością osiągnięcia dokładniejszych i lepszych wyników [4]. Algorytmy ewolucyjne wzorują się na ewolucji biologicznej [5]. Podczas inicjalizacji parametrów algorytmu należy podać takie parametry jak wielkość populacji, prawdopodobieństwo mutacji czy też krzyżowania się danych osobników. Wartości parametrów algorytmu są określone w pliku konfiguracyjnym opartym o strukturę XML (ang. *Extensible Markup Language*), który jest zarazem plikiem wejściowym do aplikacji GDT. Oprócz pliku z opcjami należy także określić pliki ze zbiorem danych:

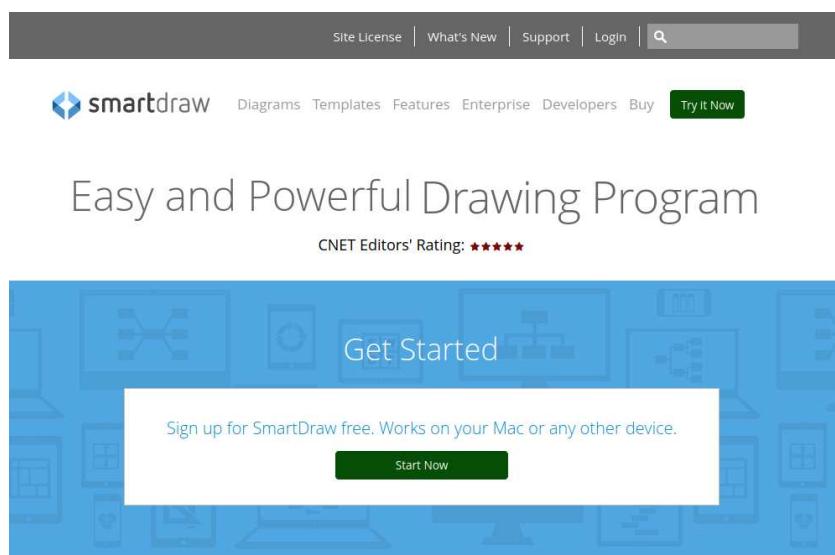
- \*.data - plik zawierający dane treningowe,
- \*.test - plik zawierający dane testowe,

- \*.names - plik określający nazwy klas oraz rodzaj zmiennych.

Wykorzystując określony zbiór trenujący oraz wartości parametrów algorytmu w pliku XML, system GDT indukuje drzewa decyzyjne. Aplikacja zapisuje do plików tekstowych statystyki wynikowe drzewa oraz użyte ustawienia parametrów.

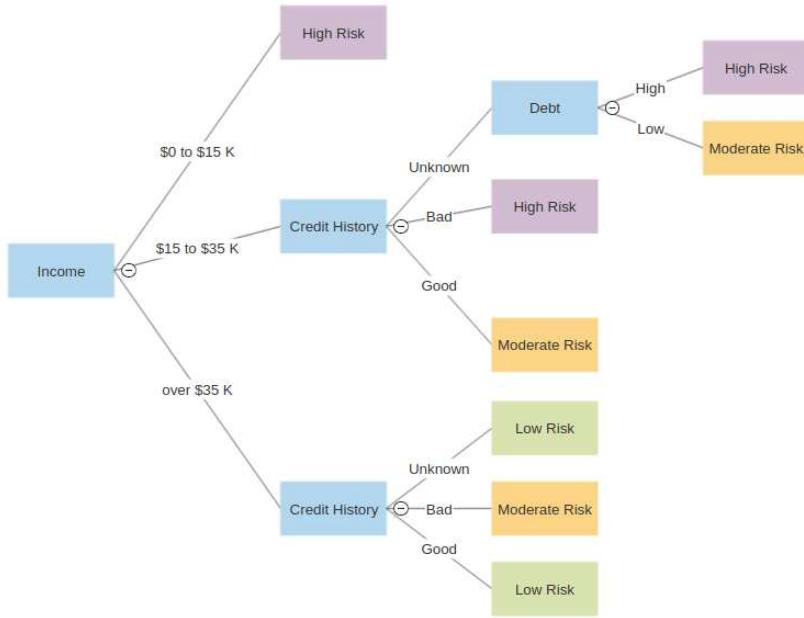
## 1.4 Istniejące rozwiązania

Aktualnie na rynku można znaleźć różne aplikacje pozwalające na budowanie drzew decyzyjnych. Aplikacje te różnią się między sobą zakresem funkcjonalności. Rozwiązania internetowe głównie są nastawione na zarobek, ale oferują też darmowe wersje z pewnymi ograniczeniami. Istnieją też liczne rozwiązania dla programistów w postaci np. bibliotek dla wielu popularnych języków programowania. Takie biblioteki umożliwiają poprzez wykorzystanie modułów stworzenie podstawowych modeli uczenia maszynowego w tym drzew decyzyjnych. Niestety ich wykorzystanie wymaga przynajmniej podstawowej wiedzy z zakresu programowania. Dodatkowo chcąc osiągnąć bardzo wydajne rozwiązania zazwyczaj należy znać szczegóły implementacji biblioteki. Możemy także spotkać aplikacje desktopowe. Wiele takich programów jest rozwijanych przez zespoły naukowe na uniwersytetach. Do wykonania obliczeń wymagają dobrej jakości sprzętu komputerowego, który zapewni odpowiednią moc obliczeniową.



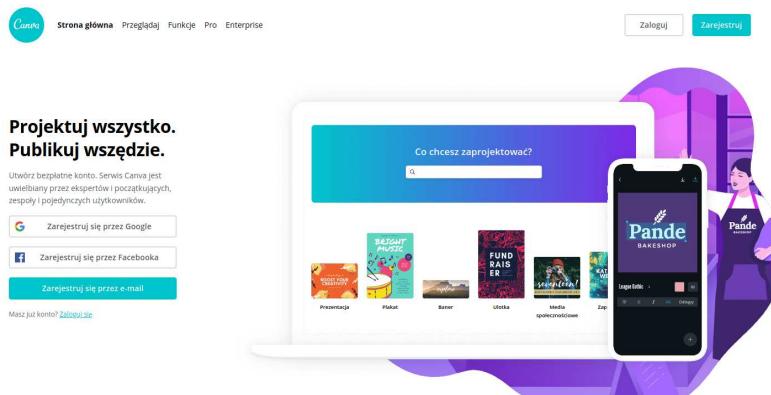
Rysunek 1.1: Strona główna aplikacji *SmartDraw* [6].

Aplikacja internetowa *SmartDraw* jest według autora pracy jedną z wygodniejszych platform do tworzenia drzew decyzyjnych [6]. Użytkownik może za darmo założyć

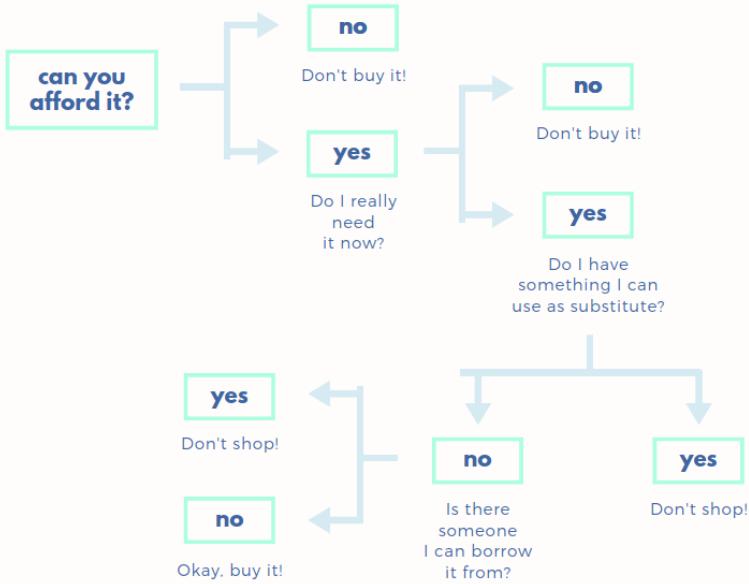


Rysunek 1.2: Drzewo stworzone w aplikacji *SmartDraw*, źródło: opracowanie własne.

konto oraz używać aplikacji bez abonamentu przez okres próbny. Widok strony głównej został przedstawiony na Rys. 1.1. Platforma ponadto udostępnia funkcjonalność tworzenia innych rodzajów diagramów. Proces budowy diagramu (np. w postaci drzewa) polega na przeciąganiu i łączeniu bloków. Istnieje również możliwość wczytania struktury drzewa z pliku o rozszerzeniu \*csv (ang. *comma-separated values*). Diagramy są prezentowane w czytelny i przejrzysty sposób (Rys. 1.2). Ukończony diagram można wyeksportować do pliku graficznego lub dokumentu programu MS Word. Aplikacja niestety nie umożliwia budowy drzewa używając metody uczenia, jedynie pozwala na graficzne przedstawienie wcześniejszej przygotowanej struktury.



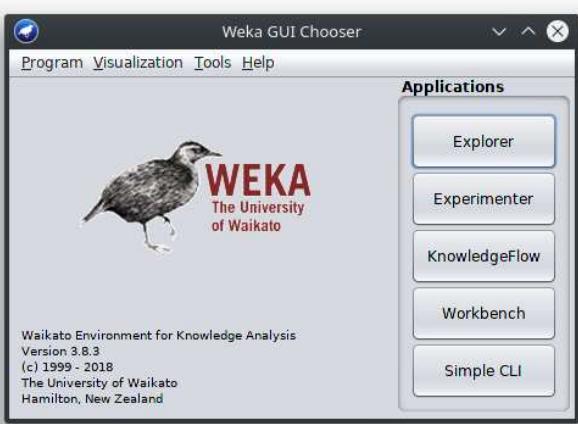
Rysunek 1.3: Strona główna aplikacji *Canva* [7].



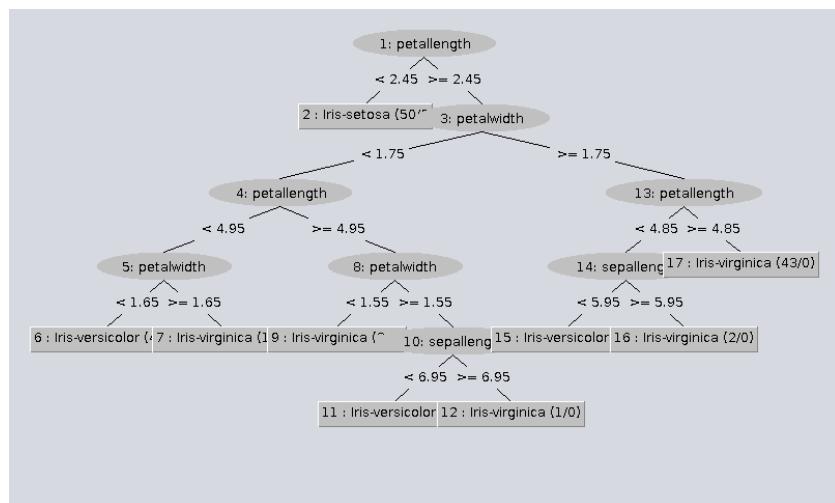
Rysunek 1.4: Drzewo stworzone w aplikacji *Canva*, źródło: opracowanie własne.

*Canva* to kolejne narzędzie umożliwiające tworzenie drzew decyzyjnych przy pomocy przeglądarki internetowej [7]. Dostęp do platformy wymaga założenia konta. Aplikacja także posiada rozszerzony, płatny pakiet funkcjonalności dla firm oraz osób prywatnych. Strona główna aplikacji została przedstawiona na Rys. 1.3. Użytkownik ma możliwość wizualnego przedstawienia metodą przeciągnij i upuść (ang. *drag'n drop*). W aplikacji jednak brakuje opcji indukcji drzewa decyzyjnego z wykorzystaniem zbioru uczącego. Tworzone drzewa można wzbogacić o liczne walory wizualne i dostępne gotowe motywów (Rys. 1.4). Głównymi odbiorcami aplikacji są reklamodawcy oraz osoby prowadzące rozbudowaną działalność w serwisach społecznościowych.

*Weka* jest aplikacją desktopową stworzoną przez naukowców zajmujących się tematami uczenia maszynowego na Uniwersytecie Waikato w Nowej Zelandii. Aplikacja została zaimplementowana w technologii JAVA SE. Pozwala to na jej uruchomienie na różnych systemach operacyjnych. Główne okno aplikacji zostało zaprezentowane na Rys. 1.5. Ponadto naukowcy zaimplementowali liczne tzw. nakładki, umożliwiające korzystanie ze stworzonych mechanizmów za pośrednictwem innych języków programowania. W aplikacji użytkownik ma dostęp do dużej ilości gotowych algorytmów uczenia maszynowego, łącznie z algorytmami indukcji drzew decyzyjnych. Tworzenie nowego eksperymentu zaczyna się od wybrania zestawu danych. Użytkownik poczynający może skorzystać z przykładowych



Rysunek 1.5: Okno główne aplikacji Weka, źródło: opracowanie własne.



Rysunek 1.6: Drzewo stworzone w aplikacji Weka, źródło: opracowanie własne.

plików. Po wczytaniu zbioru wejściowego istnieje możliwość wizualizacji danych oraz ich wstępnej obróbki. W kolejnym kroku użytkownik wybiera algorytm budowy klasyfikatora. Do wyboru jest kilka różnych algorytmów związanych z tworzeniem drzew decyzyjnych. Czas obliczeń zależy od ilości danych, wybranego algorytmu oraz posiadanych zasobów obliczeniowych. Rezultaty eksperymentu są przedstawiane w postaci tekstowej. Istnieje także możliwość wyświetlenia drzewa graficznie. Przykładowe drzewo decyzyjne stworzone za pośrednictwem programu zostało przedstawione na Rys. 1.6.

Aplikacja tworzona w ramach pracy dyplomowej jest aplikacją oryginalną. Po pierwsze ze względu na wykorzystany system GDT. Po drugie, będzie to aplikacja internetowa, która umożliwi indukowanie drzew decyzyjnych z wykorzystaniem algorytmów

ewolucyjnych. To co będzie ją odróżniać to moduł do zarządzania zadaniami, które będą uruchamiać się zdalnie, przez co użytkownik końcowy nie będzie musiał posiadać znaczących zasobów pamięciowych i obliczeniowych. Budowane narzędzie pozwoli także na graficzną i interaktywną reprezentację otrzymanych wyników.

## **2. Wizja aplikacji**

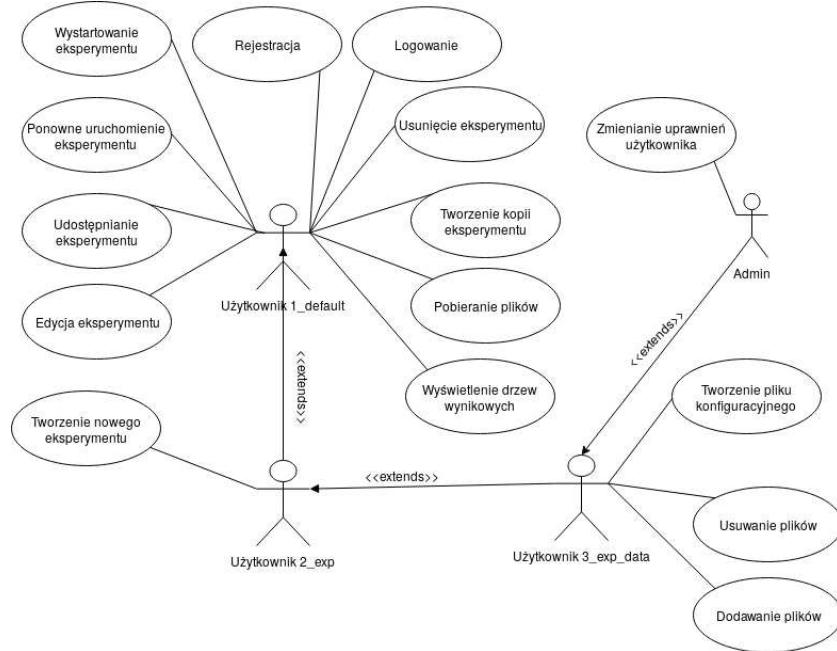
### **2.1 Wymagania funkcjonalne**

Tworzenie aplikacji należało zacząć od nakreślenia zakresu funkcjonalności, które aplikacja będzie udostępniać użytkownikom. Podstawowym zadaniem, budowanej aplikacji jest możliwość przeprowadzania eksperymentów przy pomocy systemu GDT z poziomu aplikacji internetowej. Kolejnym ważnym aspektem jest wariant zarządzania, wyświetlania, udostępniania oraz edycji poszczególnych eksperymentów (zadań). Każdy z użytkowników powinien widzieć poszczególne zadania, które zostały ukończone, są w trakcie wykonywania lub czekają w kolejce. Aplikacja powinna również w przejrzysty sposób wyświetlać wyniki, zarówno wynikowe drzewo decyzyjne oraz statystyki obliczeń. Po uruchomieniu zadania, użytkownikowi zostanie wyświetlony pasek postępu oraz oszacowana długość trwania całego zadania. Zadanie będzie można anulować w dowolnym momencie. Użytkownik będzie posiadać możliwość zarządzania plikami wejściowymi do zadania oraz plikami z wynikami. Dla użytkowników poczynających zostanie stworzona opcja budowy podstawowych plików konfiguracyjnych, bez względania się w bardziej zaawansowane parametry eksperymentu. Dostęp do funkcji aplikacji będzie wymagał założenia konta. Nowo założone konto będzie miało domyślnie ograniczone możliwości. Natomiast możliwość rejestracji oraz logowania będzie ogólnodostępna.

Użytkownik będzie mógł posiadać jedną z dostępnych ролей. Role będą definiowały dostęp do poszczególnych funkcjonalności aplikacji. Zarządzanie tymi uprawnieniami będzie się odbywać poprzez panel administratora. Administrator aplikacji dodatkowo może modyfikować oraz usuwać konta użytkowników. Co więcej z interfejsu admina będzie istniała możliwość edycji rekordów bazy danych oraz edycja uprawnień do poszczególnych eksperymentów.

Biorąc pod uwagę perspektywę udostępniania przez użytkownika eksperymentów innemu użytkownikowi, ważnym aspektem będzie możliwość ograniczenia części akcji wykonywanych na eksperymencie. Aplikacja nie pozwoli na zablokowanie wyświetlania drzewa wraz z wynikami. Natomiast reszta funkcjonalności możliwych do wykonania na zadaniu, takich jak uruchamianie, kopowanie, edycja, usuwanie czy też pobieranie plików wejściowych lub wyjściowych może zostać ograniczona. Użytkownik posiadający

udostępniony eksperyment z pewnymi ograniczeniami, może udostępnić go dalej jeśli posiada nadane prawa do udostępniania. Przy czym nie może rozszerzyć uprawnień uprzednio zablokowanych.



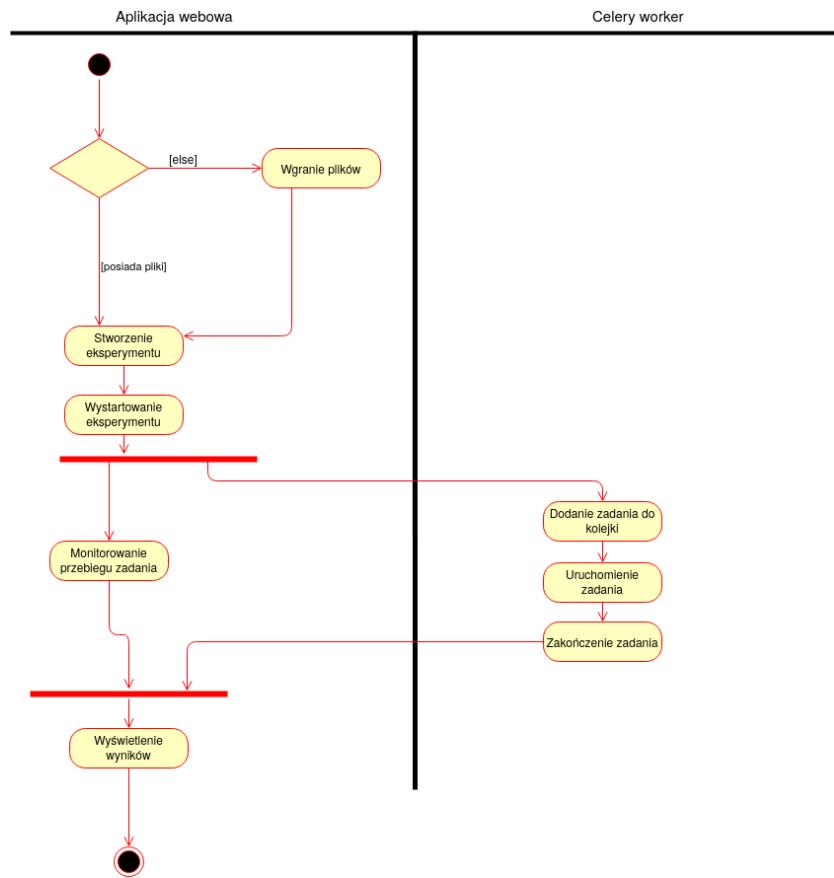
Rysunek 2.1: Diagram przypadków użycia, źródło: opracowanie własne.

Na rysunku 2.1 przedstawiono funkcjonalności w postaci diagramu przypadków użycia. W aplikacji zostały wyszczególnione trzy role dostępne do uzyskania dla użytkownika oraz rola administratora systemu. Wszystkie przypadki użycia oprócz logowania i rejestracji są dostępne tylko dla użytkowników zalogowanych. Każdy nowy użytkownik musi założyć konto, aby mieć dostęp do aplikacji. Nowo powstałe konta otrzymują uprawnienia na domyślnym poziomie „1\_default”, a wyższe poziomy uprawnień mogą zostać nadane przez administratora. Kolejne role rozszerzają możliwości użytkownika pod względem ilości akcji do wykonania. Poziom „2\_exp” pozwala na tworzenie nowych eksperymentów, przy czym tylko najwyższy poziom uprawnień „3\_exp\_data” może autoryzować do wgrywania plików do aplikacji. Przebieg czynności związanych ze stworzeniem nowego eksperymentu oraz wyświetlaniem wyników został przedstawiony na Rys. 2.2. W dalszej części tego podrozdziału przedstawione zostały opisy trzech wybranych przypadków użycia.

Opis przypadku użycia „Tworzenie nowego eksperymentu”:

### 1. Aktor

- Użytkownik.



Rysunek 2.2: Diagram czynności tworzenia i uruchomiania eksperymentu, źródło: opracowanie własne.

## 2. Warunki początkowe

- Aktor jest zalogowany oraz posiada uprawnienia przynajmniej na poziomie „2\_exp”.

## 3. Zdarzenie inicjujące

- Naciśnięcie przycisku „New experiment” nad listą wszystkich eksperymentów użytkownika.

## 4. Przebieg w krokach

- Aplikacja przechodzi do formularza tworzenia nowego eksperymentu,
- Użytkownik wypełnia i zatwierdza formularz.

## 5. Przebiegi alternatywne

- Użytkownik nie uzupełnia wszystkich pól formularza, aplikacja wyświetla powiadomienie o pustych polach.

## 6. Sytuacje wyjątkowe

- Użytkownik nie posiada żadnych plików wgranych do aplikacji. Powoduje to, że pola formularza zawierające pliki są puste. Uniemożliwia to stworzenie nowego eksperymentu, a aplikacja wyświetla powiadomienie o pustych polach przy podjętej próbie zatwierdzenia.

## 7. Warunki końcowe

- System przekierowuje użytkownika do listy z eksperymentami, a na liście znajduje się nowo utworzony eksperiment.

## 8. Zależności czasowe

- Częstotliwość wykonywania: Około 20 razy dziennie na każdego użytkownika,
- Typowy czas realizacji: 8 sekund.

Opis przypadku użycia „Wystartowanie eksperymentu”:

### 1. Aktor

- Użytkownik.

### 2. Warunki początkowe

- Aktor jest zalogowany oraz posiada stworzony eksperiment.

### 3. Zdarzenie inicjujące

- Naciśnięcie przycisku „Show” w liście eksperymentów na elemencie, którego status to „Created”.

### 4. Przebieg w krokach

- Aplikacja przechodzi do podglądu szczegółów wybranego eksperymentu,
- Użytkownik kliką przycisk „Start” znajdujący się na pasku możliwych czynności,
- Aplikacja przekierowuje użytkownika do listy eksperymentów.

## 5. Przebiegi alternatywne

- Po wystartowaniu eksperymentu nastąpił błąd i jest to sygnalizowane zmianą statusu na „Error”, a w szczegółach eksperymentu można podejrzeć wiadomość z błędem.

## 6. Sytuacje wyjątkowe

- Użytkownikowi nie posiada praw do wystartowania konkretnego eksperymentu i w panelu akcji nie wyświetla się przycisk „Start”.

## 7. Warunki końcowe

- Eksperiment zmienił swój status na „In queue” lub „Running”, a po przejściu do szczegółów wyświetla się pasek postępu oraz szacowany czas oczekiwania na zakończenie.

## 8. Zależności czasowe

- Częstotliwość wykonywania: Około 20 razy dziennie na każdego użytkownika,
- Typowy czas realizacji: 8 sekund.

Opis przypadku użycia „Wyświetlenie drzewa wynikowego”:

### 1. Aktor

- Użytkownik.

### 2. Warunki początkowe

- Aktor jest zalogowany oraz posiada ukończony eksperiment.

### 3. Zdarzenie inicjujące

- Naciśnięcie przycisku „Show” w liście eksperymentów na elemencie, którego status to „Finished”.

### 4. Przebieg w krokach

- Aplikacja przechodzi do podglądu szczegółów wybranego eksperymentu, a na samym dole karty wyświetlają się linki do drzew decyzyjnych,

- Użytkownik kliką w link do drzewa decyzyjnego.

## 5. Przebiegi alternatywne

- Brak.

## 6. Sytuacje wyjątkowe

- Brak.

## 7. Warunki końcowe

- Aplikacja wyświetliła drzewo decyzyjne wraz ze statystykami.

## 8. Zależności czasowe

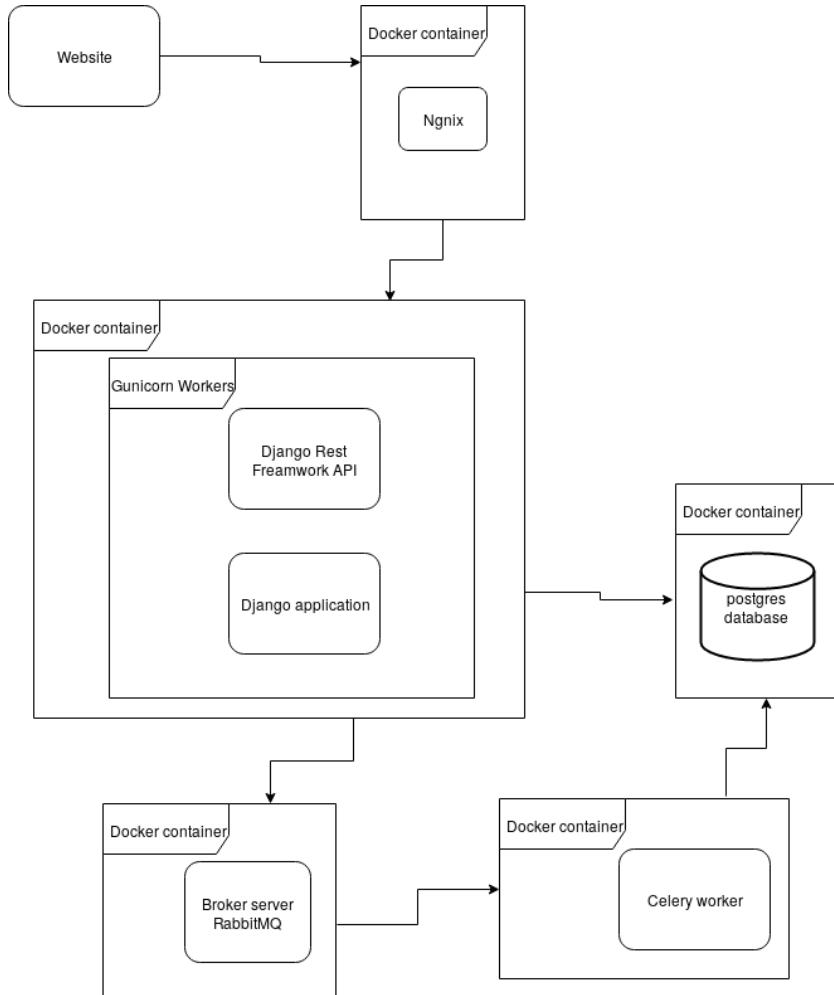
- Częstotliwość wykonywania: Około 30 razy dziennie na każdego użytkownika,
- Typowy czas realizacji: 10 sekund.

## 2.2 Wymagania niefunkcjonalne

Aplikacja zostanie podzielona na dwa oddzielne komponenty: jeden odpowiadający za aplikację internetową, drugi natomiast za zarządzanie zadaniami na serwerze: kolejkowanie i uruchamianie zadań w systemie GDT. Stworzone oprogramowanie powinno pozwolić na zarządzanie całością aplikacji z poziomu przeglądarki internetowej. Poszczególne elementy części serwerowej aplikacji powinny być odporne na błędy, umożliwiać łatwy mechanizm wdrożenia oraz restartu modułów w przypadku takiej potrzeby. Zostanie to zapewnione poprzez konteneryzację aplikacji. Zależności pomiędzy konkretnymi kontenerami utworzonymi przy pomocy oprogramowania Docker zostały przedstawione na Rys. 2.3. Dzięki takiemu rozwiązaniu wdrożenie aplikacji, czy też zmiana któregoś z komponentów na przykład serwera bazy danych, wymaga małego pokładu pracy i może zostać wykonana półautomatycznie. Aby uruchomić aplikację na serwerze niezbędna będzie platforma Docker. Wykorzystane technologie zostaną opisane w kolejnym podrozdziale.

## 2.3 Wykorzystane technologie

Aplikacja zostanie zaimplementowana z użyciem języka programowania Python (wersja 3.7.2), który jest rozpowszechnioną technologią wśród aplikacji internetowych.



Rysunek 2.3: Architektura systemu, źródło: opracowanie własne.

Wspiera on kilka różnych paradygmatów programowania takich jak programowanie funkcyjne, proceduralne i obiektowe [8]. Po stronie serwera będzie udostępniony interfejs programistyczny (ang. *API*) w architekturze REST (*Representational State Transfer*) z wykorzystaniem Django oraz Django REST Framework. Natomiast interfejs graficzny strony będzie zaprojektowany przy pomocy JavaScriptu [9] oraz biblioteki ReactJS [10].

Jedną z najważniejszych cech języka Python jest interpretowalność kodu źródłowego zamiast jego komplikacji [11]. Umożliwia on pisanie zarówno skryptów systemowych, jak i pełnoprawnych programów. Kolejną jego charakterystyczną cechą jest dynamiczne typowanie czyli tak zwany *duck typing*. Określa to sposób przypisywania typów do wartości przechowywanych w zmiennych. Typy te są określone dynamicznie podczas działania programu, w odróżnieniu od typowania statycznego, gdy wartości poszczególnych zmiennych muszą być jasno podane przed procesem komplikacji. Takie podejście do przypisywania rodzajów na pewno przyśpiesza pracę, ale może też powodować pewne

problemy. W trakcie implementacji, programista musi sam pamiętać jaki typ w danym momencie ma zmienna. Proces ten ułatwia biblioteka wbudowana w język o nazwie „Typing”. Umożliwia ona w prosty sposób wprowadzenie namiastki typowania statycznego w postaci sprawdzania i podpowiedzi.

Do implementacji części biznesowej aplikacji (czyli stronie serwera) zostanie wykorzystany framework Django (wersja 2.2.6), który jest jednym z najbardziej popularnych rozwiązań webowych w języku Python. Charakteryzuje się on prostotą implementacji (przynajmniej w początkowej fazie aplikacji) oraz podziałem aplikacji na komponenty. Zarazem narzuca on pewną strukturę projektu, która zapewnia czystość kodu oraz możliwość ponownego użycia wcześniejszych opracowanych modułów [12]. Społeczność zebrana wokół tej platformy, dynamicznie rozwija nowe rozwiązania, które są udostępniane dla szerszego grona odbiorców. Dzięki temu istnieje łatwy dostęp do wysokiej jakości modułów bezpieczeństwa, czy też wsparcie techniczne przy występujących problemach. Kolejnym argumentem, który przemawiał za wybraniem tej technologii był wbudowany panel administratora, dostarczany wraz z całą platformą. Po konfiguracji umożliwia on nie tylko zarządzanie użytkownikami, ale także edycję rekordów w bazie danych z poziomu przeglądarki internetowej.

W celu udostępnienia REST API (ang. *Representational State Transfer Application Programming Interface*) dla interfejsu graficznego został użyty Django REST Framework (*DRF*), który wraz z podstawową wersją Django stanowi trzon aplikacji. DRF jest narzędziem używanym oraz rozwijanym przez takie rozpoznawalne marki jak Mozilla, Red Hat, Heroku [13]. Świadczy to nie tylko o popularności tego rozwiązania, ale też o jakości jego wykonania. Cały framework skupia się na dostarczeniu programistycznie zestawu narzędzi do budowy interfejsu restowego. W skład takiej paczki wchodzą serializatory (*serializers*), widoki (*views*), rutery (*router*) oraz wiele innych pomocniczych obiektów. Komunikacja z takim interfejsem programistycznym odbywa się za pomocą metod protokołu HTTP (*Hypertext Transfer Protocol*). Kolejność kroków pracy API możemy określić w następujący sposób:

1. Klient tworzy zapytanie i uzupełnia je o potrzebne dane,
2. Następnie wysłane jest zapytanie pod konkretny adres,
3. Serwer przetwarza żądanie klienta oraz wysyła odpowiedź,

#### 4. Klient otrzymuje rezultat.

Do zapisu informacji związanych z działaniem aplikacji zostanie wykorzystana relacyjna baza danych PostgreSQL (wersja 10.3). Jest to rozwiązanie pod licencją *Open Source* i szeroko stosowane również z aplikacjami opartymi o technologie Django. Bazę danych można w łatwy sposób podpiąć pod panel administratora, ale także uzyskać dostęp z poziomu kodu aplikacji. W bazie danych będą gromadzone konta użytkowników oraz informacje o stworzonych eksperymentach wraz ze ścieżkami do plików.

Biblioteka ReactJS łącznie z JavaScriptem pozwoli na zaimplementowanie funkcjonalności po stronie klienta. Zapewnia one strukturę projektu, która oferuje czystość kodu, czytelność oraz wygodę użytkowania. Umożliwia one na wstawianie fragmentów kodu hmtl do kodu JavaScript za pośrednictwem języka JSX. Początkowo biblioteka ReactJS została stworzona dla potrzeb wewnętrznych firmy Facebook, ale z czasem została udostępniona innym twórcom [10]. ReactJS jest aktualnie wykorzystywany przez wielkie korporacje takie jak Netflix czy Uber, a sam projekt jest czwartym najpopularniejszym repozytorium na GitHub [14]. W celu połączenia interfejsu graficznego z logiką biznesową zostanie wykorzystana biblioteka Axios [15]. Cechuje się kompatybilnością wsteczną ze starszymi wersjami przeglądarki internetowych. Umożliwia ona budowę zapytań http oraz ich realizację. Aktualnie jest to najpopularniejsze rozwiązanie w języku JavaScript.

W celu uzyskania pełnej asynchroniczności podczas uruchamiania eksperymentów w systemie GDT, do obsługi kolejki zadań zostanie wykorzystana biblioteka Celery [16]. Takie rozwiązanie jest łatwe w integracji z innymi platformami programistycznymi. Głównym założeniem działania tej biblioteki polega na stworzeniu kolejki z zadaniami, które następnie zostaną przypisane i wykonane przez wolnego „robotnika”. Liczba „robotników” działających w aplikacji może zostać określona jako jeden z parametrów uruchomienia. Komunikacja pomiędzy Celery, a aplikacją w technologii Django odbywa się przy pomocy brokera wiadomości (ang. *message broker*), który odpowiada za przesył informacji pomiędzy dwoma komponentami. Przykładem takiego oprogramowania może być broker RabbitMQ i on zostanie wykorzystany podczas implementacji aplikacji w ramach pracy dyplomowej [17]. Oprogramowanie te osiąga bardzo dobre wyniki wydajnościowe w porównaniu z produktami konkurencyjnymi.

Gunicorn jest serwerem HTTP aplikacji Django, umożliwiającym zdefiniowanie liczby „robotników” realizujących zapytania. Jego głównym założeniem jest realizacja

wszystkiego co się dzieje pomiędzy serwerem, a aplikacją webową. Dodatkowym atutem jest minimalna ilość zasobów, które zużywa oraz duża szybkość działania [18]. Będzie on wykorzystywany jako serwer produkcyjny. Gunicorn dobrze współpracuje z Nginx i umożliwi łatwe wdrożenie aplikacji. Nginx jest serwerem WWW i posłuży do przekierowania przychodzących pod adres domeny „decisiontree.pl” żądań prosto do aplikacji działającej pod serwerem Gunicorn.

Jedną z najważniejszych części całej architektury jest konteneryzacja poszczególnych elementów. W tym celu zostaną wykorzystane kontenery stworzone przy pomocy platformy Docker. Jest to oprogramowanie umożliwiający wirtualizacje oraz podział projektu na oddzielne komponenty [19]. Stosując takie rozwiązanie w dowolnej chwili można podmieniać kontenery aplikacji, zmieniając dynamicznie ich wersje oraz łatwo restartować tylko te elementy, które tego wymagają. Cały system zostanie podzielony na pięć pracujących obok siebie instancji tworzących wspólnie całość. Podział ten jest przedstawiony na Rys. 2.3.

### **3. Architektura rozwiązania**

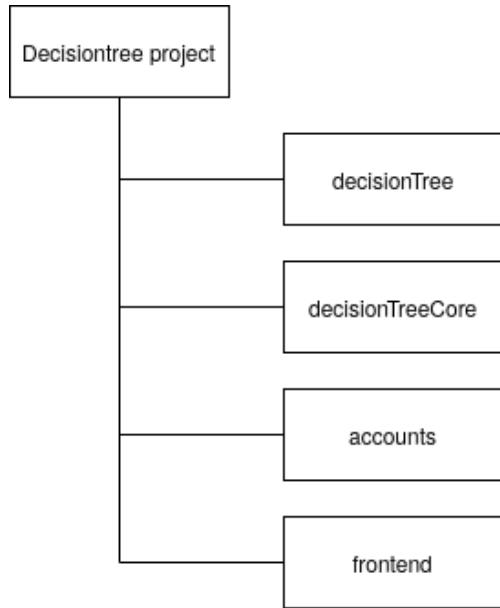
#### **3.1 Architektura aplikacji**

Struktura aplikacji jest oparta na podziale narzuconym przez wykorzystany framework Django. Zgodnie z tym założeniem projekt dzieli się na poszczególne moduły, które nazywane są paczkami. W aplikacji można wyróżnić cztery podstawowe elementy kolejno odpowiadające za zarządzanie eksperymentami i użytkownikami, graficzny interfejs użytkownika oraz moduł, który scalą te elementy w całość. Schemat modułów jest przedstawiony na Rys. 3.1.

Aplikacja została zaimplementowana zgodnie ze stylem architektonicznym REST (*Representational State Transfer*). Do komunikacji pomiędzy widokami aplikacji, a logiką biznesową znajdującą się na serwerze, wykorzystywany będzie protokół http. W tym celu do każdej poszczególnej funkcjonalności działającej w projekcie musiał zostać wystawiony punkt końcowy. Taki rodzaj rozwiązania zapewnia jasny podział na poszczególne warstwy, które są od siebie niezależne. Dzięki temu część serwerowa aplikacji może działać nie ingerując w część interfejsu graficznego. Mapowaniem poszczególnych modułów aplikacji na endpointy zajmuje się główna aplikacja „decisionTree”.

Dane użytkowników będą przechowywane w katalogu „users” i odpowiednim podkatalogu o nazwie loginu danego użytkownika. Każdy użytkownik może wgrywać dowolną ilość plików z danymi (o rozszerzeniach „\*.xml”, „\*.data”, „\*.test” oraz „\*.names” do swojego folderu). Podczas tworzenia nowego eksperymentu tworzony jest dla niego katalog o nazwie składającej się z pola „id” i „name”. Tam są przechowywane wszystkie pliki związane z zadaniem. Użytkownik będzie miał możliwość pobrania całego folderu z plikami eksperymentu w postaci archiwum o rozszerzeniu „\*.zip”. Dodatkowy wariant w aplikacji pozwala na zarządzanie katalogiem głównym użytkownika. Udostępnione są opcje do zmiany nazwy, usunięcia czy też pobrania pliku.

Użytkownik po uruchomieniu zadania będzie widział procent zaawansowania tego zadania, jak i średni czas, który pozostał do końca. Zostało to rozwiązane w aplikacji po przez tabelę w bazie pośredniczącą w wymianie informacji o zaawansowaniu danego eksperymentu. Program uruchomiony w robotniku Celery, wypisuje informacje na standardowe wyjście. Z tych informacji pobierany jest numer wykonanej iteracji oraz



Rysunek 3.1: Podział projektu na moduły, źródło: opracowanie własne

średni jej czas. Stosując połączenie za pośrednictwem tzn. PIPE, proces robotnika Celery może przeanalizować te informacje. W celu ograniczenia obciążenia co dziesiąta linia jest poddawana analizie. Na podstawie zawartych tam danych oraz ilości uruchomień algorytmu uzyskanej z pliku konfiguracyjnego można wyliczyć średni czas pozostały do końca obliczeń. Natomiast pasek postępu zostaje wyznaczony poprzez określenie numeru ostatniej iteracji i stwierdzeniu ile wykonień algorytmu jeszcze pozostało.

Autoryzacja użytkowników jest nieodłącznym elementem aplikacji internetowej. W tym celu został wykorzystany mechanizm tokenów. Taki token zostaje przyznawany użytkownikowi po zalogowaniu lub samej rejestracji. Umożliwia on autoryzowany dostęp do strony internetowej i dołącza się go w każdym zapytaniu. Po stronie wizualnej aplikacji jest przechowywany w *local storage* przeglądarki internetowej. Każdy token posiada swój czas aktywności, a po jego wygaśnięciu lub przy dowolnym problemie z jego uwierzytelnieniem użytkownik zostanie przekierowany na stronę główną.

## 3.2 Przechowywanie danych

W celu przechowywania danych została wykorzystana relacyjna baza danych PostgreSQL. Całość działa na oddzielnym kontenerze w celu uzyskania większej stabilności i nie zależności od głównego modułu aplikacji. Schemat struktury wszystkich tabel przedstawiono na Rys. 3.2. Większość tabel wynika z samego zastosowania framework'a

Django i DjangoRestFramework. Wykorzystując gotowe moduły zostały zapewnione takie modele encji jak „auth\_user” odpowiadający za zapisywanie informacji o użytkownikach, czy też „authtoken\_token” mający na celu przetrzymywanie tokenów autoryzacji. Do całej struktury zostały dodane dodatkowe tabele:

- „decisionTreeCore\_experiment” przetrzymująca dane o eksperymentach,
- „decisionTreeCore\_permissions” zawierająca informacje o prawach dostępowych do eksperymentu,
- „decisionTreeCore\_progress” składa się z pól określających postęp wykonywania doświadczenia.

Relacja pomiędzy nowo stworzonymi tabelami, a użytkownikiem została przedstawiona na Rys. 3.3. Tabela „decisionTreeCore\_experiment” zawiera informacje o pojedynczym eksperymencie użytkownika, w Tab.3.1 znajduje się opis jej pól. W celu śledzenia postępu danego doświadczenia została stworzona tabela „decisionTreeCore\_progress”. Poszczególne pola zostały rozpisane w Tab.3.2. Dane do tej tabeli są wpisywane prosto z robotnika Celery, przy czym brana jest pod uwagę tylko co dziesiąta iteracja systemu GDT. Zakres uprawnień do eksperymentu określa ostatnia tabela „decisionTreeCore\_permissions”. Schemat pól encji został rozpisany w Tab.3.3. Prawa dostępowe do doświadczenia są definiowane przez użytkownika przed samym udostępnieniem. Kolejny właściciel nie może rozszerzyć uprawnień uprzednio zablokowanych.

### **3.2.1 Mechanizm kontroli wersji eksperymentu**

Nierzadko zdarza się tak, że użytkownik będzie chciał powtórzyć zadanie z tymi samymi parametrami albo zechce je trochę zmienić. Innym przypadkiem wymagającym częstego ponownego uruchamiania eksperymentu może być zmiana plików z danymi wejściowymi. Oba ta przypadki wymagają ingerencji w plikach doświadczenia co może powodować problem, z traceniem poprzednich wyników. Rozwiążaniem tej kwestii może być na przykład tworzeni kopii eksperymentu za każdym razem, gdy następuje zmiana danych. Niestety wiąże się to ze zwiększeniem rozmiaru listy eksperymentów. W związku z tym został zaimplementowany mechanizm przechowywania poprzednich wersji eksperymentu. Przy każdej zmianie dowolnego pliku wejściowego, stare pliki zostaną zachowane

poprzez dodanie na początku nazwy „\_old”. W przypadku, gdy plik o takiej nazwie już istnieje na koniec nazwy jest dołączany kolejny numer porządkowy w nawiasach. Dzięki temu użytkownik w dowolnym momencie może pobrać archiwum z plikami i zobaczyć stare ustawienia.

Należało założyć, że jedno doświadczenie może być przeprowadzane kilka razy, co powoduje kolejny problem z nadpisywaniem plików wyjściowych. W aplikacji zostało to rozwiązane po przez dodanie do mechanizmu kontroli wersji eksperymentu, dodatkowych funkcjonalności. Z każdym ponownym uruchomieniem doświadczenia automatycznie robi się kopia zapasowa poprzedniego folderu wyjściowego. Do nazwy katalogu jest dodawany na końcu przedrostek „old\_”, w przypadku gdy już taka nazwa istnieje w systemie plików zostaje dodany numer porządkowy w nawiasach. Tak samo jak w poprzednim przypadku użytkownik po pobraniu plików, może obejrzeć poprzednie pliki wyjściowe.

Dodatkowym mechanizmem zaimplementowanym w aplikacji jest tworzenie pliku „readme.txt”. Zabezpiecza on przed straceniem informacji o eksperymencie w przypadku awarii bazy danych. Przechowuje on informacje takie jak nazwy plików wejściowych czy też nazwę eksperymentu. Na tej podstawie w dowolnej chwili istnieje możliwość odtworzenia struktury w bazie danych. Przykład zawartości takiego pliku jest widoczny w List. 3.1.

```
Information about created experiment:  
Experiment id: 4669  
Experiment name: testowy_eksperiment  
Files used in experiment:  
- config: test.xml,  
- data: chess3x3x10000.data,  
- test: chess3x3x10000.test,  
- names: chess3x3x10000(1).names
```

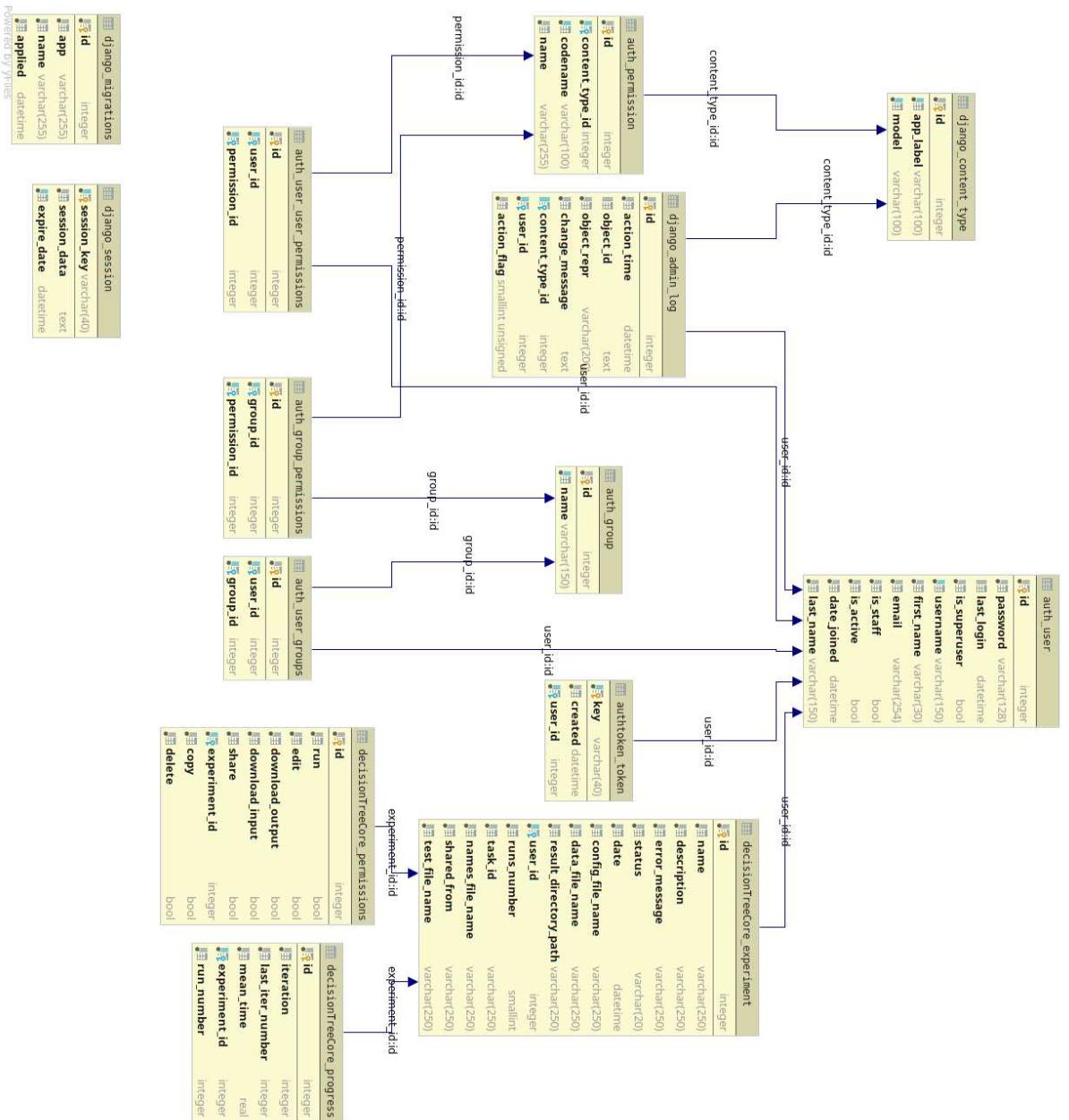
Listing 3.1: Przykład zawartości pliku readme

### 3.3 Wdrożenie aplikacji

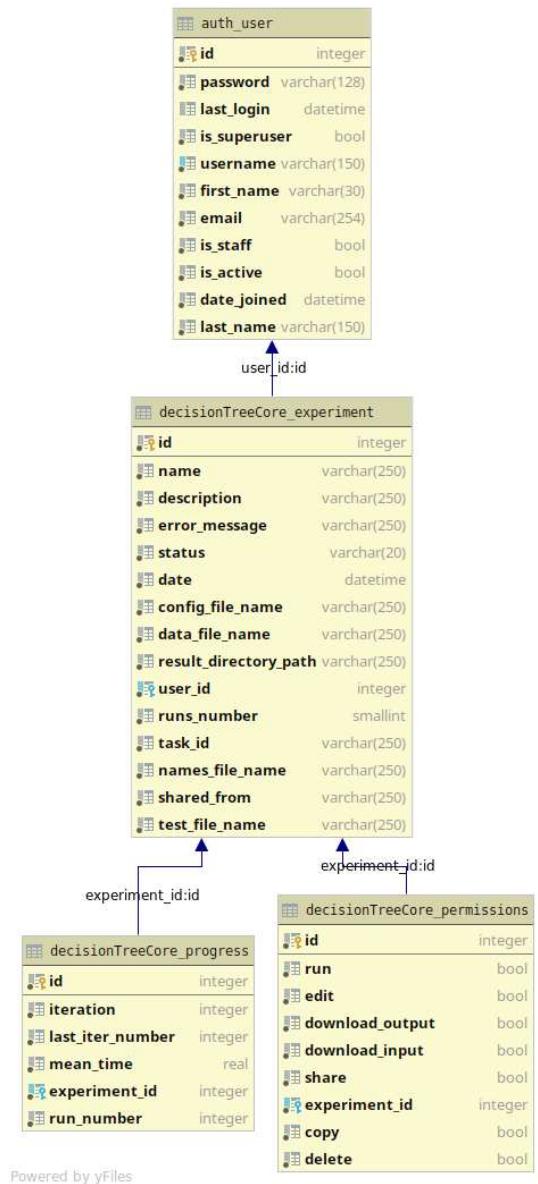
Aplikacja została wdrożona na serwer posiadający 2 GB RAM, procesor o częstotliwości taktowania 2 GHz oraz dysk SSD. Spełnia on podstawowe wymagania pod względem sprzętowym, przy założeniu niedużego obciążenia aplikacji. Do uruchomienia projektu jest potrzebna zainstalowana instancja platformy Docker. Dzięki konteneryzacji proces wdrożenia projektu przebiega bardzo prosto i ogranicza się do pojedynczej komendy aplikacji docker-compose. Po uruchomieniu oprogramowanie działa w tle i świadczy usługi użytkownikom. Dostęp do aplikacji jest możliwy przez dowolną przeglądarkę internetową

poprzez przejście pod adres „<http://decisiontree.pl>”. Przykładowymi danymi do logowania są:

- nazwa użytkownika: „test\_default”,
- hasło: „test\_default”.



Rysunek 3.2: Schemat bazy danych, źródło: opracowanie własne



Rysunek 3.3: Schemat tabel dodatkowych, źródło: opracowanie własne

Tabela 3.1: Opis pól encji „decisionTreeCore\_experiment”

Nazwa pola	Typ	Opis
id	integer	Klucz główny tabeli
name	varchar(50)	Nazwa eksperymentu
description	varchar(250)	Opis eksperymentu
error_message	varchar(250)	Wiadomość o błędzie, który wystąpił podczas uruchomienia eksperymentu
status	varchar(15)	Pole określające w jakim statusie znajduje się eksperyment. Możliwe wartości to: „Created”, „In queue”, „Running”, „Finished”, „Error”. Zmiana statusów następuje w trakcie przechodzenia eksperymentu przez kolejne etapy
data	datetime	Data stworzenia eksperymentu przez użytkownika
config_file_name	varchar(50)	Nazwa pliku konfiguracyjnego użytego w eksperymencie
data_file_name	varchar(50)	Nazwa pliku zawierającego zbiór uczący
test_file_name	varchar(50)	Nazwa pliku zawierającego zbiór testowy
names_file_name	varchar(50)	Nazwa pliku określającego nazwy klas oraz rodzaj zmiennych
result_directory_path	varchar(50)	Ścieżka do folderu z wynikami eksperymentu
user_id	integer	ID użytkownika, który jest właścicielem eksperymentu
runs_number	smallint	Pole określające ile przebiegów algorytmu ma się odbyć podczas uruchomienia eksperymentu w systemie GDT. Pełni ważną rolę przy tworzeniu paska postępu oraz określeniu liczby wyświetlanych drzew
task_id	varchar(250)	Zawiera id zadania, które trafiło do robotnika Celery. Umożliwia zarządzanie danym zadaniem np. usunięcie z kolejki, lub anulowanie w trakcie trwania
shared_from	varchar(250)	Pole pełni rolę zapisu informacji o poprzednich właścicielach. W wartością pola są nazwy użytkowników. Podczas kolejnych udostępnień do pola są dodawane po przecinku następne loginy

Tabela 3.2: Opis pól encji „decisionTreeCore\_progress”

Nazwa pola	Typ	Opis
id	integer	Klucz główny tabeli
iteration	integer	Liczba iteracji eksperymentu
last_iter_number	integer	Numer ostatniej iteracji
mean_time	real	Wiadomość o błędzie, który wystąpił podczas uruchomienia eksperymentu
experiment_id	integer	Numer id eksperymentu, do którego jest przypisany postęp
run_number	integer	Numer określający, które uruchomienie algorytmu właśnie trwa

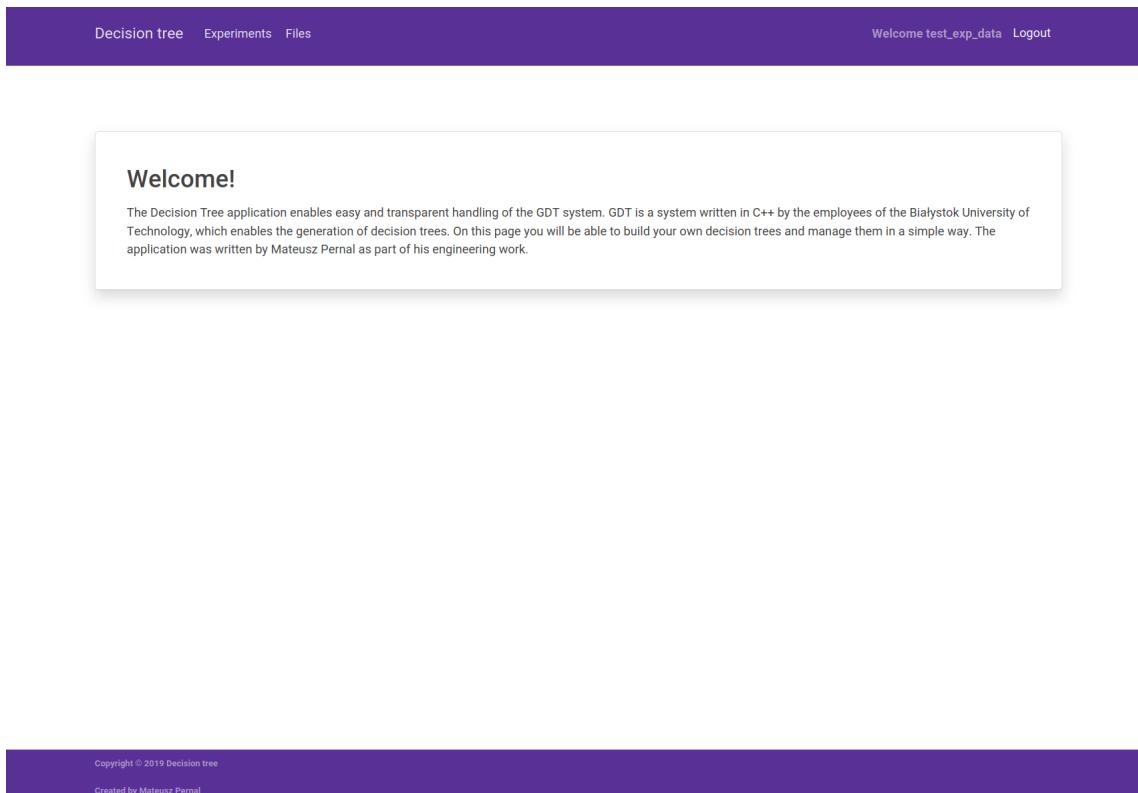
Tabela 3.3: Opis pól encji „decisionTreeCore\_permissions”

<b>Nazwa pola</b>	<b>Typ</b>	<b>Opis</b>
id	integer	Klucz główny tabeli
run	bool	Prawo do uruchamiania eksperymentu
edit	bool	Prawo do edycji
download_out	bool	Prawo dające możliwość pobierania plików wyjściowych
download_in	bool	Prawo dające możliwość pobierania plików wejściowych
share	bool	Pozwolenie do dalszego udostępniania eksperymentu
copy	bool	Prawo do tworzenia kopii
delete	bool	Prawo do usunięcia eksperymentu
experiment_id	integer	Numer id eksperymentu, do którego są przypisane prawa dostępowe

## 4. Prezentacja aplikacji oraz testy

### 4.1 Przedstawienie aplikacji

Użytkownik przechodząc pod adres „<http://decisiontree.pl>” w przeglądarce zostanie przekierowany na stronę główną projektu (Rys. 4.1). Zobaczy tam krótki opis aplikacji. W celu korzystania z systemu wymagana jest autoryzacja użytkownika. Osoba nowa może założyć konto przy pomocy formularza rejestracji (Rys. 4.2). Wypełniając pola użytkownik musi zaakceptować pole „*recaptcha*”, pełniącą rolę zabezpieczenie przed botami w internecie. Po zatwierdzeniu formularza osoba zostanie automatycznie zalogowana. Użytkownicy, którzy już posiadają konto w aplikacji mogą skorzystać z pola logowania przedstawionego na Rys. 4.3.



Rysunek 4.1: Strona główna aplikacji, źródło: opracowanie własne.

Uwierzytelniony użytkownik w zależności od nadanych mu uprawnień będzie miał możliwość wykonania innych akcji. Funkcje do których nie będzie miał dostępu nie będą wyświetlane w interfejsie graficznym, na przykład mając uprawnienia z poziomu „1\_default” na pasku nawigacji nie będzie zakładki „Files”. Widok przedstawiający wygląd strony dla użytkownika ze wszystkimi prawami został pokazany na Rys. 4.1. Przechodząc do zakładki

The form is titled "Register". It contains four input fields: "Username", "Email", "Password", and "Confirm Password". Below these fields is a reCAPTCHA verification box containing a checkbox labeled "Nie jestem robotem", the reCAPTCHA logo, and a link "Prywatność - Warunki". At the bottom left is a purple "Register" button, and at the bottom right is a link "Already have an account? [Login](#)".

Rysunek 4.2: Formularz rejestracji nowego konta, źródło: opracowanie własne.

„Experiments” zostanie wyświetlona lista wszystkich eksperymentów wraz z przyciskiem do tworzenia nowego doświadczenia lub podglądem już istniejącego (Rys. 4.4).

Kolejnym elementem na pasku nawigacji jest odnośnik do strony zarządzania plikami użytkownika. Na tym widoku znajduję się strefa wgrywania umożliwiająca wrzucanie plików niezbędnych do przeprowadzenia doświadczenia (Rys. 4.12). W dowolnym momencie użytkownik może zmienić nazwę pliku (Rys. 4.14), usunąć lub też pobrać dowolny plik. Podczas próby wgrania pliku o rozszerzeniu innym niż znajdujących się na liście dozwolonych dla użytkownika pojawi się ostrzeżenie o błędzie. Dodatkową funkcjonalnością jest przycisk kierujący do formularza tworzenia nowego pliku konfiguracyjnego. Formularz zawiera podstawowe pola, które są uzupełnione wartościami domyślnymi (Rys. 4.13). Stanowi to ułatwienie dla nowych użytkowników, którzy chcieliby zrobić swój pierwszy eksperiment bez zagłębiania się w zaawansowane parametry algorytmu.

Proces tworzenia nowego eksperymentu rozpoczyna się od wypełnienia formularza informacjami oraz wybraniu plików wejściowych (zarówno ze zbiorami danych oraz ustawieniami algorytmu) (Rys. 4.5). W przypadku niewypełnienia, któregoś pola zostanie

The image shows a simple login interface. At the top center is the word "Login". Below it is a "Username" field with a placeholder. Underneath is a "Password" field with a placeholder. A purple "Login" button is centered below the password field. At the bottom, there is a link "Don't have an account? [Register](#)".

Rysunek 4.3: Formularz logowania, źródło: opracowanie własne.

## Experiments:

Experiments:				
Name	Description	Status	Date	Option
test 2	Test desc	Created	17.12.2019 12:34:30	Show
test 3	Test desc	Created	17.12.2019 12:34:35	Show
Testowy 1	Test desc	Canceled	17.12.2019 12:33:29	Show
test experiment	test desc	Created	17.12.2019 12:34:04	Show
test 4	Test desc	Finished	17.12.2019 12:34:36	Show
test 2 (copy)	Test desc	Created	17.12.2019 12:34:37	Show

Rysunek 4.4: Widok listy eksperymentów, źródło: opracowanie własne.

wyświetlony komunikat o błędzie. Jeżeli wszystkie pola będą wypełnione, po zatwierdzeniu wyświetli się informacja o sukcesie oraz nastąpi przekierowanie do listy eksperymentów. Nowo stworzone doświadczenia na liście posiadają status „Created”. Klikając w przycisk „Show” użytkownik może wyświetlić szczegóły eksperymentu, wraz z listą akcji możliwych do wykonania. Po uruchomieniu doświadczenie trafia do kolejki i przyjmuje status „In queue”. Natomiast zaraz po zaczęciu obliczeń przechodzi w stan „Running” i użytkownik ma możliwość zobaczenia postępu zadania w zakładce szczegółów eksperymentu (Rys. 4.9). W dowolnym czasie trwania obliczeń istnieje możliwość wysłania sygnału przerwania, wtedy status zmieni się na „Cancel”. Kiedy podczas uruchomienia wystąpi błąd oznaczenie ustawi się na wartość „Error”. W pełni ukończony eksperiment otrzymuje status „Finished”,

The form consists of several input fields and dropdown menus:

- Experiment name:** test nam
- Description:** Test description
- Config file:** to test.xml
- Data file:** chess3x3x10000.data
- Test file:** chess3x3x10000.test
- Names file:** chess3x3x10000.names
- Submit** button (highlighted in purple)

Rysunek 4.5: Formularz tworzenia nowego eksperymentu, źródło: opracowanie własne.

a w widoku szczegółów pojawią się odnośniki do powstałych drzew decyzyjnych (Rys.4.6). W zależności od stanu doświadczenia ramka karty przyjmie inne kolory informując o rezultacie ostatnich akcji. Przechodząc do widoku z wynikami otrzymamy tabele ze statystykami takimi jak rozmiar drzewa, czas wykonania, ilość danych treningowych i testowych poddanych ewaluacji oraz wartości procentowe rezultatów. Rysunek wynikowego drzewa można skalować przy pomocy kółka myszki, a poszczególne węzły można zwijać i rozwijać. Istnieje dodatkowa opcja wydrukowania wizualizacji wyników.

Użytkownik może wykonywać dodatkowe operacje po przez panel wyświetlenia szczegółów doświadczenia, na przykład edycję eksperymentu za pomocą formularza (Rys. 4.7). Kolejną funkcjonalnością jest udostępnianie eksperymentu innym osobom. W formularzu wyświetlają się kontrolki do nadawania praw, które zostały przedstawione na Rys. 4.8. Eksperyment po udostępnieniu pojawi się na liście wszystkich doświadczeń u drugiej osoby. Natomiast na końcu nazwy w nawiasach zostanie dodana nazwa właściciela, który udostępnił eksperyment.

Aplikacja udostępnia również panel administratora dostępny pod adresem „decisontree.pl\admin”. Po autoryzacji, administrator może zarządzać modelami w bazie, ale także zmieniać role użytkowników. Również w dowolnym momencie może modyfikować

Experiment with name: test 4

Description: Test desc

Date: 17.12.2019 12:34:36

Status: Finished

Config file: test.xml

Dataset name: chess3x3x10000

- Tree from run number 1
- Tree from run number 2

Rysunek 4.6: Strona przedstawiający szczegóły eksperymentu, źródło: opracowanie własne.

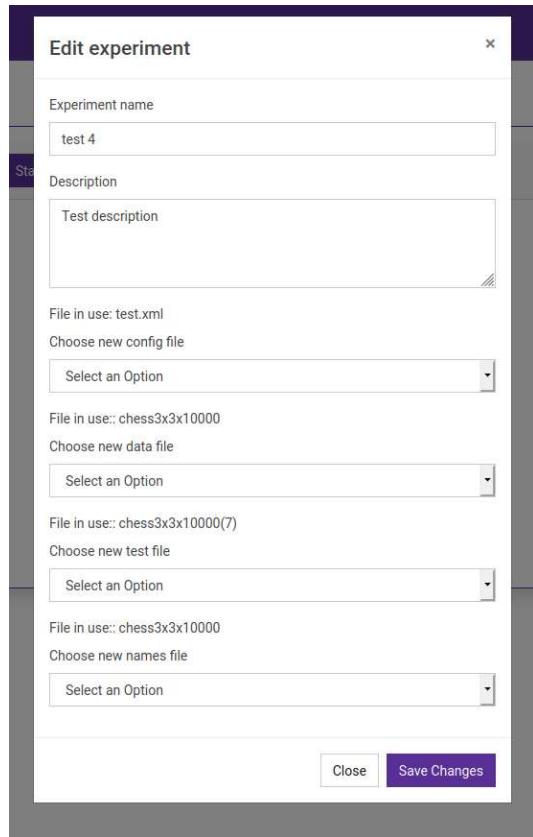
ograniczenia dostępu do eksperymentu, przez co może ograniczyć lub nadać prawa. Widok zarządzania eksperymentem został przedstawiony na Rys. 4.15.

## 4.2 Testy aplikacji

Nieodłącznym elementem związanym z tworzeniem aplikacji jest przeprowadzenie testów w celu sprawdzenia poprawności jej działania. Testowanie ma za zadanie zweryfikować czy dany produkt jest zgodny ze specyfikacją oraz wykryć błędy w kodzie. W projekcie aplikacji zostały wykorzystane testy wydajnościowe oraz manualne na grupie kontrolnej.

### 4.2.1 Testy wydajnościowe

Oprogramowaniem użytym do określenia wydajności poszczególnych punktów końcowych jest framework Locust. Narzędzie te służy do testowania obciążenia aplikacji internetowej. Głównym jego celem jest określenie maksymalnej liczby osób, które w danym czasie mogą używać aplikacji bez problemów wydajnościowych. Podczas uruchamiania zestawu testowego należy podać co ile sekund pojawi się nowy użytkownik oraz ich łączną liczbę. Na potrzeby testów aplikacji przyjęte wartości to 10, 100, 500 jednocześnie korzystających użytkowników oraz częstotliwość tworzenia konta 10 na sekundę. Dodatkowo ten sam zestaw został uruchomiony podczas obciążenia serwera obliczeniami w systemie GDT. Wyniki poszczególnych testów zostały przedstawione w Tab. 4.1.

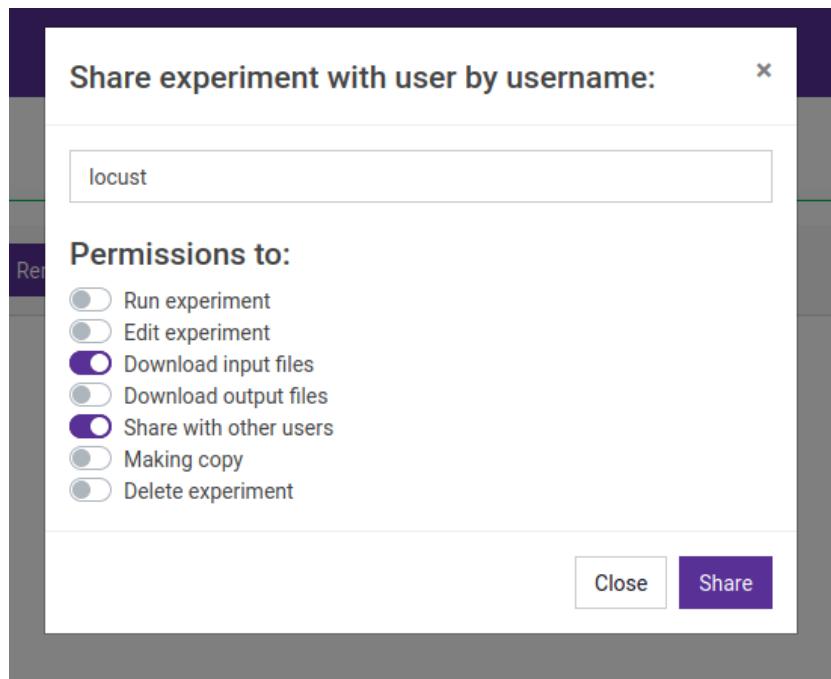


Rysunek 4.7: Formularz edycji eksperymentu, źródło: opracowanie własne.

Analizując wartości otrzymane w testach możemy znaleźć zależność, że im większa liczba użytkowników tym dłużej serwer potrzebuje czasu na realizację zapytania. Przy liczbie kont równej 500 pewna część żądań do aplikacji kończy się błędem oznaczającym zresetowaniem połączenia. Na podstawie wyników można określić, że liczba 100 użytkowników na raz może w dość nieograniczający sposób korzystać z aplikacji. Ograniczenia wydajnościowe są spowodowane niewielką mocą serwera. Poprawę wyników można osiągnąć poprzez zwiększenie pojemność pamięci RAM i zwiększenie mocy procesora. Jednoczesna praca nad obliczeniami powoduje spadek wydajności realizowania zapytań. Rozwiązaniem tego problemu może być opcja przeniesienia części kontenerów na oddzielne serwery. Dzięki temu moduły nie będą konkurować o moc obliczeniową.

#### 4.2.2 Testy manualne

Testy manualne przeprowadzane zgodnie ze scenariuszami testowymi sa ważnym aspektem podczas przygotowania się do wdrożenia aplikacji. Za grupę kontrolną zostali wybrani studenci Politechniki Białostockiej. Łączna liczba osób, która wzięła udział



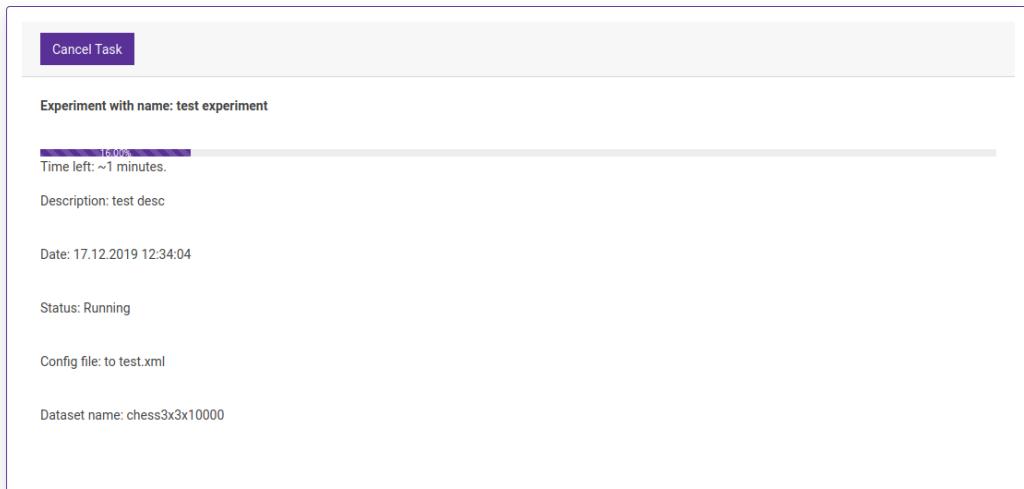
Rysunek 4.8: Formularz udostępniania eksperymentu, źródło: opracowanie własne.

Tabela 4.1: Wyniki testów wydajnościowych z użyciem Locust, źródło: opracowanie własne.

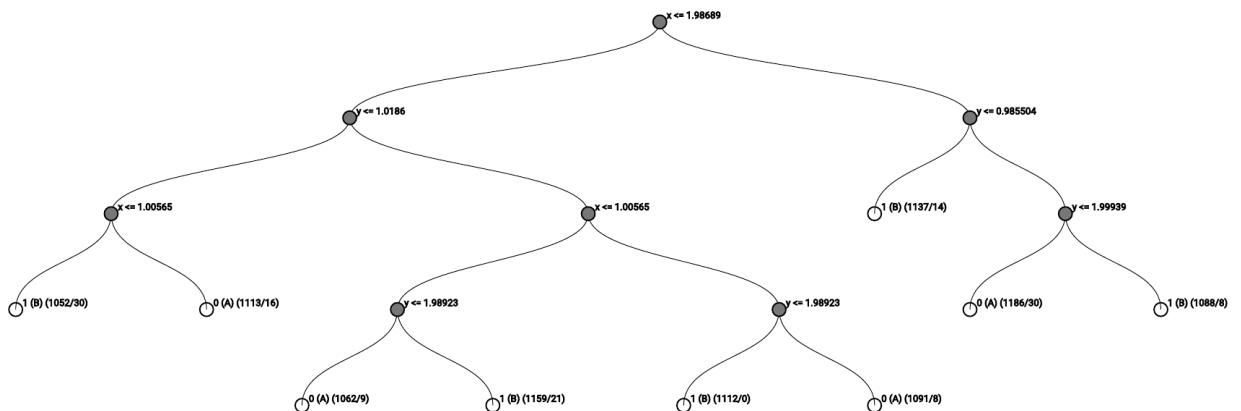
Liczba użytkowników	Pod obciążeniem	Zapytania na sekundę	Średni czas odpowiedzi
10	Nie	1.5	50 ms
100	Nie	15.5	200 ms
500	Nie	6.5	25000 ms
10	Tak	1.3	60 ms
100	Tak	8.5	11000 ms
500	Tak	5.0	35000 ms

w testach wynosiła 20. W celu przetestowania najważniejszych funkcjonalności zostały stworzone dwa zestawy testowe opisujące kolejne kroki i oczekiwane rezultaty. Przykładowy uzupełniony zestaw testowy został przedstawiony na Rys. 4.16. Jeden z nich miał za zadanie sprawdzenia następujących funkcjonalności:

- Rejestracja,
- Logowanie,
- Uruchomienie eksperymentu,
- Zatrzymanie doświadczenia,
- Ponowne uruchomienie eksperymentu,
- Wyświetlenie wyników.



Rysunek 4.9: Widok uruchomionego eksperymentu, źródło: opracowanie własne.



Rysunek 4.10: Widok stworzonego drzewa decyzyjnego, źródło: opracowanie własne.

Natomiast drugi zestaw testowy wymagał pracy w parach, a jego celem była weryfikacja:

- Udostępnienie eksperymentu,
- Uruchomienie otrzymanego doświadczenia i wyświetlenie wyników.

W początkowej fazie testów zostały wykryte problemy z mechanizmem kolejkowania zadań dla robotnika Celery, które zostały rozwiązane po przez zmianę konfiguracji w metodzie zadania. W kolejnych etapach nie zlokalizowano poważniejszych błędów, tylko kilka mniejszych związanych z zachowaniem interfejsu graficznego. Wszystkie problemy, które były opisane i zweryfikowane przez grupę kontrolną zostały rozwiązane.

Parameter	Value
Tree Size	17
Time	86
Evaluation on training data	10000 items
Result on training data	98.64%
Evaluation on test data	1000 items
Result on test data	98.1%

Rysunek 4.11: Tabela ze statystykami algorytmu, źródło: opracowanie własne.

Decision tree Experiments Files Welcome test\_exp\_data Logout

Drag 'n' drop some files here, or click to select files  
(Only \*.xml, \*.data, \*.names, \*.test files)

**Config xml files:**  
(Files with \*.xml extension)

Name	Options
krotki(1).xml	Delete Rename Download File
to test.xml	Delete Rename Download File

**Experiment files:**  
(Files with \*.data, \*.test, \*.names extension)

Name	Options
chess3x3x10000.data	Delete Rename Download File
chess3x3x10000.names	Delete Rename Download File
chess3x3x10000.test	Delete Rename Download File

Copyright © 2019 Decision tree  
Created by Mateusz Pernal

Rysunek 4.12: Widok strony do zarządzania plikami, źródło: opracowanie własne.

**Create new config file**

Config file name

Runs of experiments

Mutation OMP

Mutation OMP

Mutation OMP

Mutation OMP

Size of population

Maximum iterations number

Minimum iterations number

Probability of mutation

Probability of crossover

Selection pressure

Rysunek 4.13: Formularz tworzenia nowego pliku konfiguracyjnego, źródło: opracowanie własne.

**New name form**

New file name:

Rysunek 4.14: Formularz edycji nazwy pliku, źródło: opracowanie własne.

Change experiment

Name:	<input type="text" value="test 4"/>	HISTORY																
Description:	<input type="text" value="Test desc"/>																	
Error message:	<input type="text"/>																	
Status:	<input type="button" value="Finished"/>																	
User:	<input type="text" value="test_exp_data"/>																	
Config file name:	<input type="text" value="test.xml"/>																	
Data file name:	<input type="text" value="chess3x310000"/>																	
Test file name:	<input type="text" value="chess3x310000"/>																	
Names file name:	<input type="text" value="chess3x310000"/>																	
Result directory path:	<input type="text" value="users/test_exp_data/282_test 4"/>																	
Runs number:	<input type="text" value="2"/>																	
Task id:	<input type="text" value="b7d2461e-64dc-4a1a-8dec-61a257048cd6"/>																	
Shared from:	<input type="text"/>																	
<b>PERMISSIONS</b> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>RUN</th> <th>EDIT</th> <th>DOWNLOAD OUTPUT</th> <th>DOWNLOAD INPUT</th> <th>SHARE</th> <th>COPY</th> <th>DELETE</th> <th>DELETE?</th> </tr> </thead> <tbody> <tr> <td><input checked="" type="checkbox"/></td> </tr> </tbody> </table>			RUN	EDIT	DOWNLOAD OUTPUT	DOWNLOAD INPUT	SHARE	COPY	DELETE	DELETE?	<input checked="" type="checkbox"/>							
RUN	EDIT	DOWNLOAD OUTPUT	DOWNLOAD INPUT	SHARE	COPY	DELETE	DELETE?											
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>											
<input type="button" value="Delete"/> <input type="button" value="Save and add another"/> <input type="button" value="Save and continue editing"/> <input type="button" value="SAVE"/>																		

Rysunek 4.15: Panel modyfikacji praw dostępowych do eksperymentu, źródło: opracowanie własne.

Web application test			
Test case	Expected result	OK/NOK	Additional information
TC1.1 Go to page decisiontree.pl 1. Login failure	Log in form displayed.	TC S 1	
TC1.2 Sign up to application with wrong credentials	Displayed alert message. The user is still on the login page	V	
TC1.3 Go to page decisiontree.pl 2. Login successfully	Log in form displayed.	TC S 1	
TC2.1 Go to page decisiontree.pl 2. Sign up to application with correct credentials	Redirected to the main page.	V	
TC2.2 Go to page decisiontree.pl 3. Registration failure	Log in form displayed.	TC S 1	
TC3.1 Go to page decisiontree.pl	Registration form displayed.	V	
TC3.2 Click on Registration menu item at menu bar	Registration form displayed.	V	
TC3.3 Fill form with wrong data	Displayed right alert message. The user is still on the registration page.	V	
TC4.1 Go to page decisiontree.pl 4. Registration successfully	Log in form displayed.	TC S 1	OK
TC4.2 Click on Registration menu item at menu bar	Registration form displayed.	V	
TC4.3 Fill form with right data	Redirected to the main page.	V	
TC5.1 Go to page decisiontree.pl and log in TC5.2 Click experiments tab.	Redirected to the experiment page with list.	OK	
TC5.3 Choose one of experiment with status "created"	Experiment details page displayed with start button.	V	
TC5.4 Click on start button.	Redirected to the main page with a table of experiments.	OK	
TC5.5 Choose experiment which was started and click show button	Experiment has changed the status to "Running".	V	
TC5.6 Wait till experiment will be finished.	Experiment details page displayed. Every ~10 sec progress bar refresh.	OK	
TC5.7 Choose one of decision tree and click 6. Rerun experiment	At the bottom of card view links to decision trees displayed. Page with image of tree display.	V	
TC6.1 Go to page decisiontree.pl and log in TC6.2 Click experiments tab.	Redirected to the main page.	OK	
TC6.3 Choose one of experiment with status "finished"	Redirected to the experiment page with list.	OK	
TC6.4 Click rerun button.	Experiment details page displayed with rerun button.	V	
TC6.5 Choose experiment which was started and click show button	Redirected to the main page with a table of experiments.	OK	
TC6.6 Click rerun button.	Experiment has changed the status to "Running".	V	
TC7.1 Go to page decisiontree.pl and log in 7. Cancel experiment	Experiment details page displayed. Every ~10 sec progress bar refresh.	OK	Test in queue of 10 min
TC7.2 Click experiments tab.	Redirected to the main page.	V	
TC7.3 Choose one of experiment with status "Running". If not present start new one with previous steps.	Redirected to the experiment page with list.	V	
TC7.4 Click cancel button.	Experiment has changed the status to "Cancelled".	V	

Rysunek 4.16: Przykładowy wypełniony zestaw testowy, źródło: opracowanie własne.

## **Podsumowanie**

Założeniem pracy dyplomowej było stworzenie aplikacji internetowej do obsługi systemu GDT. W tym celu został wykorzystany język Python wraz z frameworkm Django i Django REST Framework. Stworzona aplikacja umożliwia w łatwy sposób zarządzania eksperymentami i danymi. Wyniki eksperymentu są przedstawione w formie graficznej oraz tekstuowej. Dodatkowo został stworzony specjalny zestaw ról możliwych do przypisania dla użytkownika. Dzięki temu aplikacja może mieć większe grono użytkowników o różnym stopniu zaawansowania. Zarządzanie uprawnieniami użytkowników jest możliwe poprzez panel administratora. Ponadto administrator ma możliwość edytowania modeli zapisanych w bazie danych na przykład zmiany praw dostępu do akcji eksperymentu.

Stworzona aplikacja internetowa cechuje się wysoką intuicyjnością korzystania, a zarazem łatwością zarządzania. Stosując logiczne rozdzielenie poszczególnych elementów aplikacji w celu zwiększenia stabilności zastosowano konteneryzacje z użyciem oprogramowania Docker. Cały system został podzielony na pięć instancji tworzących wspólnie całość. Wykorzystanie takiego rozwiązania umożliwia w dowolnym momencie wymiany kontenerów aplikacji na przykład na ich nowszą wersję. Podział na elementy pozwalała wdrożyć aplikacje na kilka serwerów, gdzie każdy będzie odpowiadał za jednąinstancję.

Rozwiązanie stworzone w ramach pracy dyplomowej zostało wdrożone i udostępnione użytkownikom pod adresem „<http://decisiontree.pl>”. Dostęp do systemu jest możliwy za pośrednictwem dowolnej przeglądarki internetowej. Mechanizm zakładania kont i logowania jest udostępniony dla każdego. Bardziej zaawansowane zagadnienia wymagają uprzedniej autoryzacji. Wraz ze wzrostem zainteresowania istnieje możliwość rozbudowania zaimplementowanego oprogramowania o dodatkowe metody uczenia maszynowego, czy też mechanizm predykcji oraz bardziej zaawansowane drzewa decyzyjne (jak np. modelowe).

## Bibliografia

- [1] Kretowski M. *Evolutionary Decision Trees in Large-Scale Data Mining*. Studies in Big Data, vol. 59, Springer.
- [2] Lucid Software Inc. Lucidchart. <https://www.lucidchart.com/pages/decision-tree>, stan z 25.11.2019 r.
- [3] Aurélien Géron. *Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow*. Helion S.A., 2018.
- [4] Kretowski M. Jurczuk K., Czajkowski M. *Evolutionary Induction of Decision Tree for Large Scale Data*. Soft Computing, vol. 21: 7363-79.
- [5] Prateek Joshi. *Artificial Intelligence with Python*. Packt Publishing, 2017.
- [6] SmartDraw. Smartdraw. <https://www.smartdraw.com/>, stan z 17.12.2019 r.
- [7] Canva. Canva. <https://www.canva.com/>, stan z 17.12.2019 r.
- [8] Mark Lutz. *Python. Wprowadzenie. Wydanie IV*. Helion S.A., 2010.
- [9] Mozilla and individual contributors. Javascript. <https://developer.mozilla.org/pl/docs/Web/JavaScript>, stan z 10.12.2019 r.
- [10] Facebook Inc. Reactjs. <https://reactjs.org/>, stan z 10.12.2019 r.
- [11] Python Software Foundation. Python. <https://www.python.org/>, stan z 08.12.2019 r.
- [12] Django Software Foundation. Django. <https://www.djangoproject.com/>, stan z 10.12.2019 r.
- [13] Collaboratively funded project. Django rest framework. <https://www.django-rest-framework.org/>, stan z 10.12.2019 r.
- [14] Inc. GitHub. Github. <https://github.com/search?o=desc&q=stars%3A%3E1&s=stars&type=Repositories>, stan z 10.12.2019 r.

- [15] Axios. Axios. <https://github.com/axios/axios>, stan z 10.12.2019 r.
- [16] Ask Solem and contributors. Celery project. <http://www.celeryproject.org/>, stan z 10.12.2019 r.
- [17] Inc. Pivotal Software. Rabbitmq. <https://www.rabbitmq.com/>, stan z 10.12.2019 r.
- [18] Benoit Chesneau and contributors. Gunicorn. <https://gunicorn.org/>, stan z 10.12.2019 r.
- [19] Docker Inc. Docker. <https://www.docker.com/>, stan z 10.12.2019 r.
- [20] The PostgreSQL Global Development Group. Postgresql. <https://www.postgresql.org/>, stan z 10.12.2019 r.
- [21] Inc. NGINX. Nginx. <https://www.nginx.com/>, stan z 10.12.2019 r.
- [22] Locustio. Locust. <https://locust.io/>, stan z 17.12.2019 r.
- [23] Machine Learning Group at the University of Waikato. Weka. <https://www.cs.waikato.ac.nz/ml/weka/>, stan z 17.12.2019 r.

## **Spis tabel**

Tablica 3.1	Opis pól tabeli „decisionTreeCore_experiment” . . . . .	32
Tablica 3.2	Opis pól tabeli „decisionTreeCore_progress” . . . . .	32
Tablica 3.3	Opis pól tabeli „decisionTreeCore_experiment” . . . . .	33
Tablica 4.1	Wyniki testów wydajnościowych z użyciem Locust, źródło: opracowanie własne. . . . .	40

# Spis rysunków

Rysunek 1.1 Strona główna aplikacji <i>SmartDraw</i> [6]. . . . .	10
Rysunek 1.2 Drzewo stworzone w aplikacji <i>SmartDraw</i> , źródło: opracowanie własne. . . . .	11
Rysunek 1.3 Strona główna aplikacji <i>Canva</i> [7]. . . . .	11
Rysunek 1.4 Drzewo stworzone w aplikacji <i>Canva</i> , źródło: opracowanie własne. . . . .	12
Rysunek 1.5 Okno główne aplikacji <i>Weka</i> , źródło: opracowanie własne. . . . .	13
Rysunek 1.6 Drzewo stworzone w aplikacji <i>Weka</i> , źródło: opracowanie własne. . . . .	13
Rysunek 2.1 Diagram przypadków użycia, źródło: opracowanie własne. . . . .	16
Rysunek 2.2 Diagram czynności tworzenia i uruchomiania eksperymentu, źródło: opracowanie własne. . . . .	17
Rysunek 2.3 Architektura systemu, źródło: opracowanie własne. . . . .	21
Rysunek 3.1 Podział projektu na moduły, źródło: opracowanie własne . . . . .	26
Rysunek 3.2 Schemat bazy danych, źródło: opracowanie własne . . . . .	30
Rysunek 3.3 Schemat tabel dodatkowych, źródło: opracowanie własne . . . . .	31
Rysunek 4.1 Strona główna aplikacji, źródło: opracowanie własne. . . . .	34
Rysunek 4.2 Formularz rejestracji nowego konta, źródło: opracowanie własne. . . . .	35
Rysunek 4.3 Formularz logowania, źródło: opracowanie własne. . . . .	36
Rysunek 4.4 Widok listy eksperymentów, źródło: opracowanie własne. . . . .	36
Rysunek 4.5 Formularz tworzenia nowego eksperymentu, źródło: opracowanie własne. . . . .	37
Rysunek 4.6 Strona przedstawiający szczegóły eksperymentu, źródło: opracowanie własne. . . . .	38
Rysunek 4.7 Formularz edycji eksperymentu, źródło: opracowanie własne. . . . .	39
Rysunek 4.8 Formularz udostępniania eksperymentu, źródło: opracowanie własne. . . . .	40
Rysunek 4.9 Widok uruchomionego eksperymentu, źródło: opracowanie własne. . . . .	41
Rysunek 4.10 Widok stworzonego drzewa decyzyjnego, źródło: opracowanie własne. . . . .	41
Rysunek 4.11 Tabela ze statystykami algorytmu, źródło: opracowanie własne. . . . .	42

Rysunek 4.12 Widok strony do zarządzania plikami, źródło: opracowanie własne.	42
Rysunek 4.13 Formularz tworzenia nowego pliku konfiguracyjnego, źródło: opracowanie własne. . . . .	43
Rysunek 4.14 Formularz edycji nazwy pliku, źródło: opracowanie własne. . . . .	43
Rysunek 4.15 Panel modyfikacji praw dostępowych do eksperimentu, źródło: opracowanie własne. . . . .	44
Rysunek 4.16 Przykładowy wypełniony zestaw testowy, źródło: opracowanie własne. . . . .	45

## **Spis listingów**

3.1 Przykład zawartości pliku readme . . . . .	28
--	----