

POLITECHNIKA BIAŁOSTOCKA

WYDZIAŁ INFORMATYKI

KATEDRA OPROGRAMOWANIA

PRACA DYPLOMOWA INŻYNIERSKA

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

WYKONAWCA: MATEUSZ PERNAL

.....

podpis

PROMOTOR: DR INŻ. KRZYSZTOF JURCZUK

.....

podpis

BIAŁYSTOK 2019 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
<b>Mateusz Pernal</b>  <b>TEMAT PRACY DYPLOMOWEJ: Aplikacja internetowa do obsługi system gdt w technologii Python/Django.</b>  Zakres pracy: 1. Zakres 1 2. Zakres 2 3. Zakres 3		
<div style="display: flex; justify-content: space-between; margin-top: 20px;"> <div style="width: 45%;">                 .....                  Imię i nazwisko promotora - podpis             </div> <div style="width: 45%;">                 .....                  Imię i nazwisko kierownika katedry - podpis             </div> </div>		
<div style="display: flex; justify-content: space-between; margin-top: 20px;"> <div style="width: 30%;">                 .....                  Data wydania tematu pracy dyplomowej - podpis promotora             </div> <div style="width: 35%;">                 .....                  Regulaminowy termin złożenia pracy dyplomowej             </div> <div style="width: 30%;">                 .....                  Data złożenia pracy dyplomowej - potwierdzenie dziekanatu             </div> </div>		
<div style="display: flex; justify-content: space-between; margin-top: 20px;"> <div style="width: 45%;">                 .....                  Ocena promotora             </div> <div style="width: 45%;">                 .....                  Podpis promotora             </div> </div>		
<div style="display: flex; justify-content: space-between; margin-top: 20px;"> <div style="width: 30%;">                 .....                  Imię i nazwisko recenzenta             </div> <div style="width: 35%;">                 .....                  Ocena recenzenta             </div> <div style="width: 30%;">                 .....                  Podpis recenzenta             </div> </div>		

Thesis topic:

Temat po angielsku.

## **Summary**

Abstrakt po angielsku

# Spis treści

<b>Streszczenie</b>	<b>3</b>
<b>Wprowadzenie</b>	<b>5</b>
<b>1 Przedstawienie problemu</b>	<b>6</b>
1.1 Drzewa decyzyjne . . . . .	6
1.2 Uczenie maszynowe . . . . .	6
1.3 Drzewa decyzyjne w technikach uczenia maszynowego . . . . .	7
1.4 Istniejące rozwiązania . . . . .	8
<b>2 Wizja aplikacji</b>	<b>9</b>
2.1 Wymagania funkcjonalne . . . . .	9
2.2 Wymagania нефункционалне . . . . .	14
2.3 Wykorzystane technologie . . . . .	14
<b>3 Architektura rozwiązania</b>	<b>19</b>
3.1 Architektura aplikacji . . . . .	19
3.2 Przechowywanie danych . . . . .	21
3.3 Wdrożenie aplikacji . . . . .	23
<b>4 Przedstawienie aplikacji oraz testy</b>	<b>28</b>
<b>Podsumowanie</b>	<b>29</b>
<b>Bibliografia</b>	<b>31</b>
<b>Spis tabel</b>	<b>32</b>
<b>Spis rysunków</b>	<b>33</b>
<b>Spis listingów</b>	<b>34</b>
<b>Spis algorytmów</b>	<b>35</b>

# **Wprowadzenie**

Tu będzie wstęp

## **Cel pracy**

Celem pracy jest stworzenie aplikacji webowej umożliwiającej obsługę systemu GDT (*Global Decision trees*) służącego do generowania drzew decyzyjnych. Strona internetowa umożliwi tworzenie oraz zarządzanie zadaniami uruchamianymi przy pomocy systemu. Jedną z ważniejszych cech programu powinno być danie użytkownikowi możliwość ustawienia parametrów konfiguracyjnych przed wystartowaniem zadania. Aplikacja internetowa powinna także udostępniać opcje związane z wyświetleniem drzewa w postaci graficznej oraz przedstawieniem wyników eksperymentu.

## **Zakres pracy**

Zakres pracy obejmuje:

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

## **Organizacja pracy**

Tu będzie organizacja pracy/ zawartość pracy

# **1. Przedstawienie problemu**

## **1.1 Drzewa decyzyjne**

Podjęcie decyzji jest procesem myślowym, który od początku istnienia ludzkości stwarza pewne trudności, a polega on na wybraniu najlepszego rozwiązania z dostępnych. 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 w postaci diagramu. Sam diagram może przybrać formę graficzną drzewa decyzyjnego.

Podstawowymi elementami drzewa są korzeń, gałęzie, węzły oraz liście. Korzeniem jest decyzja od którego rozpoczyna się budowa całej struktury zawierającej poszczególne węzły odpowiadające za sprawdzenie pewnego warunku. Natomiast gałęzie pełnią rolę połączenia wszystkich elementów [11]. Liście są końcowymi wierzchołkami drzewa i określają wybraną decyzję. Podczas próby określenia decyzji, należy poddać klasyfikacji posiadane dane, aby to osiągnąć konieczne jest przejście całego drzewa od samego korzenia do wynikowego liścia. Rezultatem takiej operacji będzie klasa definiująca decyzję.

## **1.2 Uczenie maszynowe**

W otaczającym nas świecie ilość informacji produkowanych przez otoczenie oraz zbieranych przez firmy czy instytucje nadal przewyższa ilość danych, które można przeanalizować z użyciem obecnych zasobów. W celu wyciągnięcia wniosków z takiej ilości danych wykorzystuje się liczne rozwiązania technologiczne. Dzięki zastosowaniu różnych algorytmów przetwarzania danych, klasyfikacji oraz predykcji programy komputerowe posiadają możliwość uczenia się. Kierunek nauki, który zajmuje się tą dziedziną nazywamy uczeniem maszynowym. W ciągu ostatnich dziesięciu lat entuzjazm związany z wykorzystywaniem tej technologii wzrósł gwałtownie i w dużej mierze zdominował przemysł, ale również przyczynił się do jej rozwoju [8]. Uczenie maszynowe stanowi trzon wielu usług, serwisów i aplikacji. Pod względem technologicznym odpowiada za wyniki wyszukiwania w przeglądarkach, za rozpoznawanie mowy przez nasze telefony, ale także jest odpowiedzialne za prowadzenie autonomicznych samochodów.

### 1.3 Drzewa decyzyjne w technikach uczenia maszynowego

Drzewa decyzyjne stanowią jedno z bardziej wszechstronnych algorytmów w dziedzinie uczenia maszynowego. Z jednej strony mogą być wykorzystywane w zadaniach z zakresu klasyfikacji, a z drugiej strony również odgrywają ważną rolę w regresji [8]. Z ich pomocą możemy uzyskać potężne modele i narzędzia zdolne do uczenia się ze złożonych zbiorów danych. Dodatkowym atutem drzew jest możliwość wizualnego przedstawienia rozwiązania, które będzie zrozumiałe dla osób nie mających do czynienia z uczeniem maszynowym lub ze statystyką. Z racji wzrostu popularności tej technologii zwiększyły się nakłady pracy naukowej w celu osiągnięcia coraz to lepszych i bardziej optymalnych algorytmów pod względem wydajnościowym.

#### 1.3.1 System GDT

Pracownicy Politechniki Białostockiej również mają wkład w budowę takich rozwiązań. Autorski system GDT (*Global Decision Trees*), który jest wykorzystywany w aplikacji inżynierskiej, służy do generowania modelu drzewa decyzyjnego na podstawie zbiorów wejściowych. Ten system jest zaimplementowany w języku c++ oraz jest skompilowany do pliku wykonywalnego, aby umożliwić jego uruchomienie z poziomu konsoli systemu operacyjnego. Całe rozwiązanie jest unikalne, a głównym założeniem jest wykorzystanie algorytmów genetycznych. Z ich pomocą przestrzeń rozwiązań danego problemu jest większa niż w klasycznym podejściu, co skutkuje możliwością osiągnięcia dokładniejszych i lepszych wyników. Metody pracy algorytmów genetycznych w dużej mierze odwzorowują działania samej natury [13]. Podczas definiowania pracy algorytmu należy podać takie parametry jak wielkość populacji, prawdopodobieństwo mutacji czy też krzyżowania się danych osobników. Wartości tych parametrów i innych są określane w pliku konfiguracyjnym opartym o strukturę XML, który jest zarazem plikiem wejściowym do aplikacji GDT. System oprócz tego pliku wykorzystuje pliki z konkretnymi rozszerzeniami:

- \*.data - plik zawierający dane treningowe,
- \*.test - plik zawierający dane testowe,
- \*.names - plik określających nazwy klas oraz rodzaj zmiennych.

Na podstawie tych danych aplikacja GDT może stworzyć model drzewa decyzyjnego, którego przedstawienie jest zapisywane w pliku tekstowym.

#### **1.4 Istniejące rozwiązania**



## **2. Wizja aplikacji**

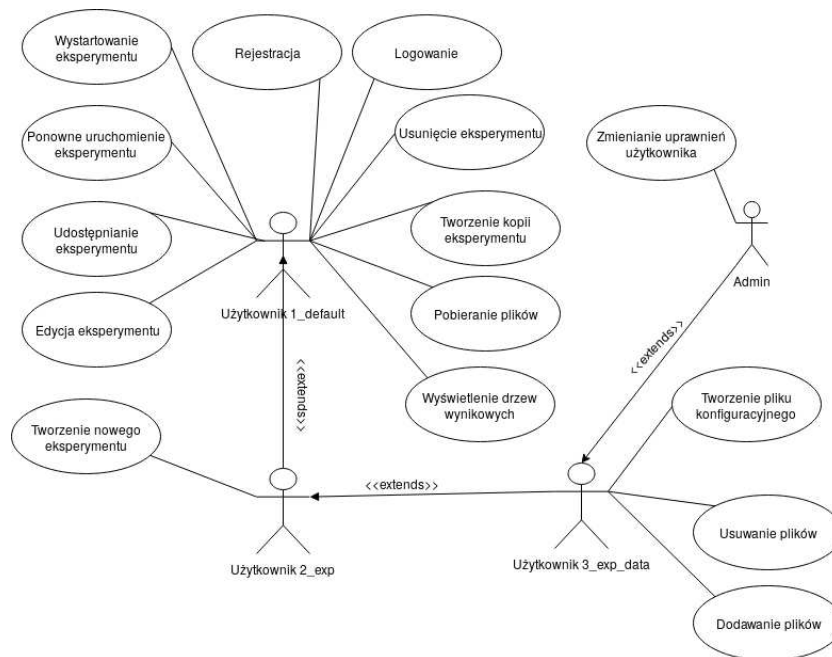
### **2.1 Wymagania funkcjonalne**

Tworzenie aplikacji należało zacząć od nakreślenia zakresu funkcjonalności, które aplikacja będzie udostępniać użytkownikom. Podstawowym zadaniem, jakie powinna spełniać jest możliwość przeprowadzania eksperymentów przy pomocy systemu GDT. Kolejnym ważnym aspektem jest wariant zarządzania, wyświetlania, udostępniania oraz edycji poszczególnych eksperymentów. Każdy z użytkowników powinien widzieć poszczególne doświadczenia, które zostały ukończone, są w trakcie lub czekają w kolejce do obliczenia. Aplikacja powinna również przede wszystkim w przejrzysty sposób wyświetlać wyniki doświadczenia w postaci wygenerowanego drzewa decyzyjnego i poszczególnych statystyk. Podczas uruchomienia nowego zadania do obliczenia, użytkownikowi zostanie wyświetlony pasek postępu oraz oszacowana długość trwania całego zadania, przy czym w dowolnym momencie będzie mógł anulować polecenie. Wraz z możliwością tworzenia eksperymentu nie odłącznym elementem będzie funkcjonalność zarządzania plikami wejściowymi oraz wyjściowymi. Dla użytkowników początkujących zostanie przedstawiona opcja tworzenia podstawowych plików konfiguracyjnych, bez wgłębiania się w bardziej zaawansowane parametry eksperymentu. Dostęp do całej platformy wymaga założenia konta użytkownika. Natomiast możliwość rejestracji oraz logowania będzie ogólnodostępna.

System kont użytkowników powinien wyróżniać różne role, które definiowałyby 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 z bazy danych oraz edycja uprawnień do poszczególnych eksperymentów.

Biorąc pod uwagę perspektywę udostępniania przez użytkownika doświadczeń innemu użytkownikowi, ważnym aspektem będzie umożliwienie ograniczenia części akcji wykonywanych na eksperymencie. Aplikacja nie pozwoli na zablokowanie wyświetlania wraz ze statystykami. Natomiast reszta funkcjonalności możliwych do wykonania na doświadczeniu, takich jak uruchamianie, kopiowanie, 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.



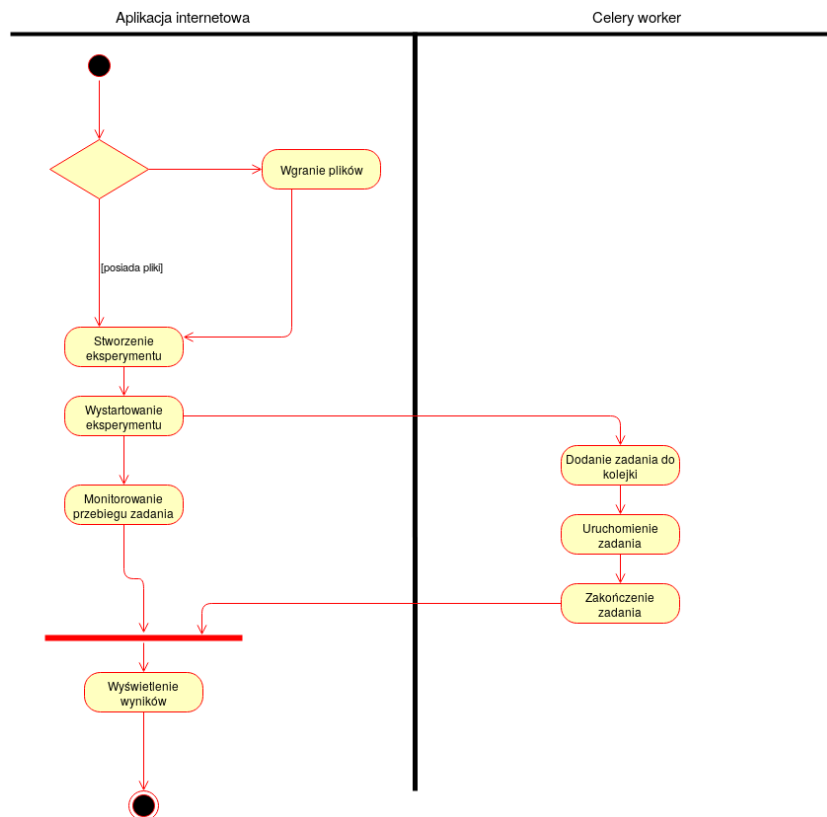
Rysunek 2.1: Diagram przypadków użycia, źródło: opracowanie własne

Na rysunku 2.1 przedstawiono wszystkie funkcjonalności w postaci diagramu przypadków użycia. W aplikacji zostały wyszczególnione trzy role dostępne do uzyskania dla każdego użytkownika oraz rola administratora całego 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 wyświetleniem wyników został przedstawiony na Rys. 2.2.

Opis przypadku użycia „Tworzenie nowego eksperymentu”:

#### 1. Aktor

- Użytkownik.



Rysunek 2.2: Diagram czynności tworzenia i uruchamiania 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 eksperyment.

## 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 eksperyment.

### 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 klika 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 podejrzec 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

- Eksperyment 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 eksperyment.

##### 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 klika 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

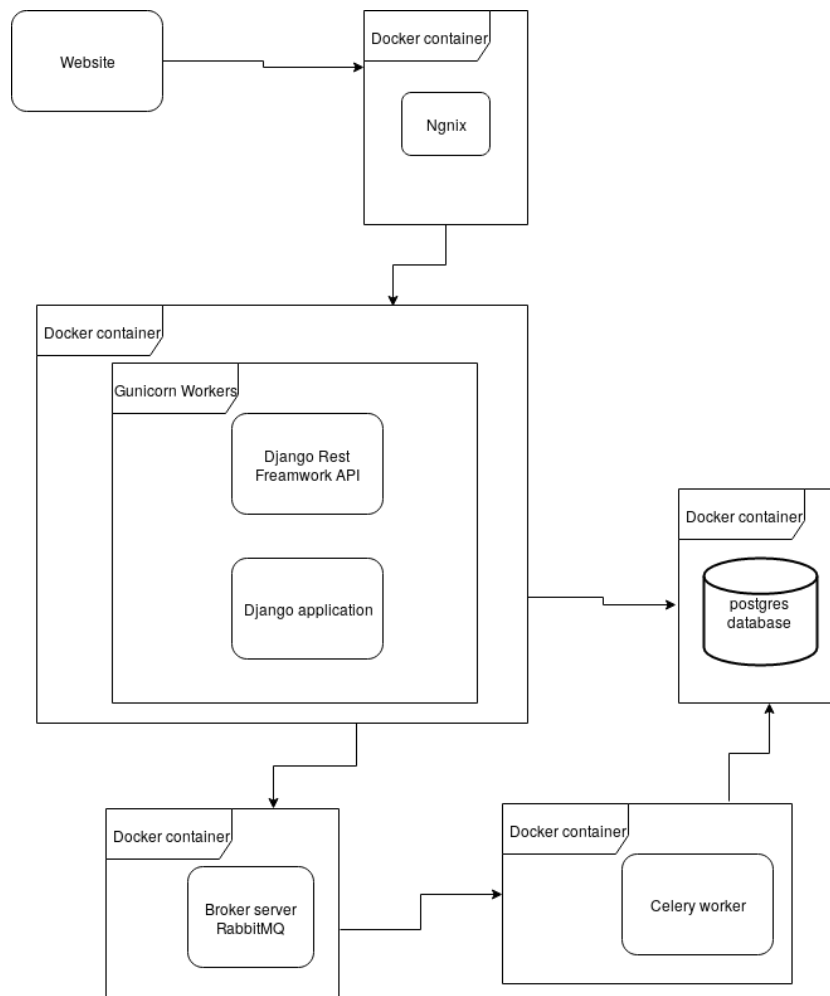
- Częstotliwość wykonywania: Około 30 razy dziennie na każdego użytkownika,
- Typowy czas realizacji: 10 sekund.

## 2.2 Wymagania niefunkcjonalne

Projekt aplikacji zostanie podzielony na dwa oddzielne komponenty jeden odpowiadający za stronę internetową, drugi natomiast za kolejkowanie i uruchamianie zadań w systemie GDT. Kompletne oprogramowanie pozwala na zarządzanie całością aplikacji w dość przejrzysty sposób. Poszczególne elementy strony serwerowej 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, jest możliwa w sposób natychmiastowy i mało inwazyjny. Jedyną rzeczą niezbędną na serwerze, aby uruchomić aplikację jest platforma Docker.

## 2.3 Wykorzystane technologie

Aplikacja zostanie zaimplementowana z użyciem języka programowania Python, który zyskuje coraz większą popularność wśród programistów. Charakteryzuje się on łatwym progim wejścia oraz wspieraniem kilku różnych paradygmatów programowania takich jak programowanie funkcyjne, proceduralne i obiektowe [14]. Po stronie serwera będzie



Rysunek 2.3: Architektura systemu, źródło: opracowanie własne

wystawione API w architekturze REST (*Representational State Transfer*) z wykorzystaniem Django oraz Django REST Framework. Natomiast interfejs graficzny strony będzie zaprojektowany przy pomocy JavaScriptu oraz biblioteki ReactJS.

Jedną z najważniejszych cech języka Python jest interpretowalność kodu źródłowego zamiast jego kompilacja [4]. Umożliwia on pisanie zarówno skryptów jak i zaawansowane programy. Kolejnym jego atrybutem jest dynamiczne typowanie czyli tak zwany *duck typing*. Określa to sposób przypisywania typów do wartości przechowywanych w zmiennych. Typy te są określane dynamicznie podczas działania programu, w odróżnieniu od typowania statycznego, gdy wartości poszczególnych zmiennych muszą być jasno podane przed procesem kompilacji. Takie podejście do przypisywania rodzajów na pewno przyspiesza pracę, ale wiąże się z pewnymi wadami. W trakcie implementacji, programista musi sam pamiętać jaki typ w danym momencie ma zmienna. Z pomocą przychodzi 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 zostanie wykorzystany framework Django, który jest jednym z najbardziej popularnych rozwiązań webowych w języku Python. Charakteryzuje się on dużą szybkością implementacji oraz jasnym podziałem aplikacji na konkretne komponenty. Zarazem narzuca on pewną strukturę projektu, która zapewnia czystość kodu oraz możliwość ponownego używania wcześniej opracowanych modułów [3]. Społeczność zebrana wokół tej platformy, czynnie 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 edycje rekordów w bazie danych.

W celu wystawienia REST API(*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 [5]. Świadczy to nie tylko o popularności tego rozwiązania, ale też o jakości jego wykonania. Cały framework skupia się na wystawieniu dla programisty zestawu narzędzi do budowy interfejsu restowego. W skład takiej paczki wchodzi serializatory(*serializers*), widoki(*views*), rutery(*router*) oraz wiele innych pomocniczych obiektów. Komunikacja z takim interfejsem programistycznym odbywa się za pomocą metod protokołu http. Kolejność kroków pracy API możemy określić w następujący sposób:

1. Klient tworzy zapytanie i uzupełnia je o potrzebne dane,
2. Następnie następuje wysłanie zapytania 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. Jest to rozwiązanie pod licencją *Open Source* i szeroko



stosowane również z aplikacjami opartymi o technologie Django. Bazę można w łatwy sposób podpiąć pod panel administratora, ale także uzyskać dostęp z poziomu kodu aplikacji. W bazie danych będą zapisywane konta użytkowników oraz informacje o stworzonych eksperymentach wraz ze ścieżkami do plików.

Biblioteka ReactJS łącznie z JavaScriptem pozwoli na zaimplementowanie funkcjonalnego interfejsu graficznego strony internetowej. Zapewnia ona pewną strukturę projektu, która oferuje czystość kodu, czytelność oraz wygodę użytkowania. Pozwala na wstawianie wstawek kodu html do kodu JavaScriptowego za pośrednictwem języka JSX. Początkowo biblioteka została stworzona dla potrzeb wewnętrznych firmy Facebook, ale z czasem została udostępniona innym twórcom [10]. Programiści na całym świecie tak docenili te rozwiązanie, że ReactJS jest aktualnie wykorzystywany przez wielkie korporacje takie jak Netflix czy Uber, a sam projekt jest czwartym najpopularniejszym repozytorium na GitHub [6]. W celu połączenia interfejsu graficznego z logiką biznesową wystawioną za pomocą REST API została wykorzystana biblioteka Axios [1]. Cechuje się implementacją klienta http działająca po stronie strony internetowej. 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 [17]. 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 (*message broker*), który odpowiada za przesył informacji pomiędzy dwoma komponentami. Przykładem takiego oprogramowania może być broker RabbitMQ i on zostanie wykorzystany podczas implementacji aplikacji dyplomowej [16]. Oprogramowanie do wymiany wiadomości od Pivotal Software 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 [2]. W projekcie strony

internetowej zostanie wykorzystany zaraz obok serwera www o nazwie Nginx pełniącego rolę pośrednika zapytań. Serwer www 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 umożliwiające łatwą wirtualizację oraz podział projektu na oddzielne komponenty [9]. Stosując takie rozwiązanie w dowolnej chwili można podmieniać ze sobą 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

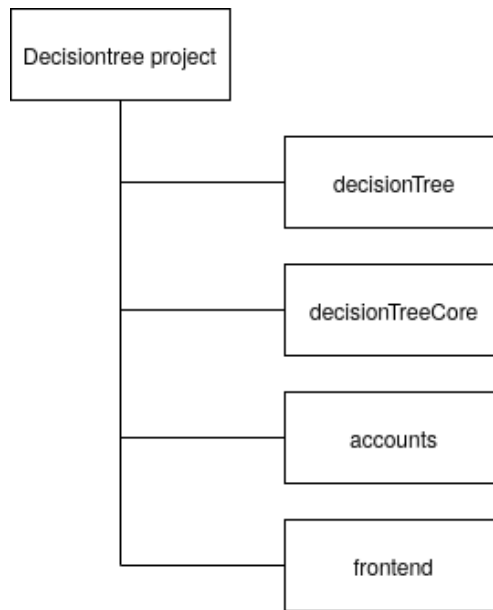
### 3.1 Architektura aplikacji

Struktura aplikacji jest oparta na podziale wynikającym z wykorzystania frameworka Django. Zgodnie z tym założeniem projekt dzieli się na poszczególne moduły, które w rozumieniu platformy nazywane są paczkami. W aplikacji można wyróżnić cztery podstawowe elementy kolejno odpowiadające za zarządzanie eksperymentami i użytkownikami, interfejs graficzny oraz aplikacja zbierająca wszystko w całość. Schemat modułów jest przedstawiony na Rys. 3.1.

Aplikacja jest zaimplementowana zgodnie ze stylem architektonicznym REST (*Representational State Transfer*). Architektura ta jest bardzo popularna przy tworzeniu aplikacji internetowych ze względu na swoją lekkość oraz prostotę. Do komunikacji pomiędzy widokami aplikacji, a logiką biznesową znajdującą się na serwerze, wykorzystywany jest protokół http. W tym celu do każdej poszczególnej funkcjonalności działającej w projekcie musiał zostać wystawiony endpoint. Zbiór wszystkich wystawionych serwisów przyjmuje nazwę API (*Application Programming Interface*). 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”.

Obok plików aplikacji można wyróżnić folder „users” zawierający foldery użytkowników. Nazwy tych katalogów są tworzone na bazie loginu. Każdy użytkownik może wgrywać dowolną ilość plików 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 doświadczeniem. Użytkownik ma możliwość pobrania całego folderu z plikami eksperymentu w postaci archiwum o rozszerzeniu „\*.zip”. Dodatkowo w aplikacji jest dostępna opcja zarządzania katalogiem głównym użytkownika. Udostępnione są opcje do zmiany nazwy, usunięcia czy też pobrania pliku.

Użytkownik uruchamiając nowy eksperyment, powinien widzieć jego postęp, jak i średni czas, który pozostał do końca. Zostało to rozwiązane w aplikacji po przez



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

tabelę w bazie pośredniczącą wymianę informacji o progresie doświadczenia. Program uruchomiony w robotniku Celery, wypisuje informacje na standardowe wyjście (stdin). W tych danych znajduje się numer iteracji wykonanej oraz średni jej czas. Stosując połączenie za pośrednictwem PIPE, robotnik 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 wykonań algorytmu jeszcze pozostało.

Autoryzacja użytkowników jest nieodłącznym elementem aplikacji internetowej. W tym celu został wykorzystany mechanizm tokenów. Taki token jest przyznawany dla użytkownika po zalogowaniu lub samej rejestracji. Umożliwia on autoryzowany dostęp do strony internetowej i jest przekazywany 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 wszelkich informacji została wykorzystana relacyjna baza danych PostgreSQL. Platforma ta jest postawiona na oddzielnym kontenerze w celu uzyskania większej stabilności i nie zależności od głównego modułu aplikacji. Schemat struktury wszystkich tabel został przedstawiony na Rys. 3.2. Większość tabel wynika z samego zastosowania frameworka Django i DjangoRestFramework. Wykorzystując gotowe rozwiązania zostały zapewnione takie modele jak „auth\_user” odpowiadający za zapisywanie informacji o użytkownikach, czy też „authtoken\_token” mająca 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. Ostatnią dodaną tabelą na potrzeby aplikacji jest tabela „decisionTreeCore\_permissions” określająca zakres uprawnień do danego eksperymentu. 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

Użytkownik robiąc doświadczenie będzie chciał otrzymać jak najlepsze wyniki. W tym celu dość często będzie zmieniać parametry w pliku konfiguracyjnym. Innym przypadkiem wymagającym częstego ponownego uruchamiania eksperymentu może być

zmiana plików z danymi wejściowymi. Oba te przypadki wymagają ingerencji w plikach doświadczenia co powoduje problem, z traceniem poprzednich danych. Rozwiązaniem tej kwestii może być na przykład tworzenie kopii eksperymentu za każdym razem, gdy następuje zmiana czegokolwiek. Niesie to niestety ze sobą pewne wady, ponieważ lista eksperymentów rozrośnie się do dużych rozmiarów. 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.

Należy przypuścić, 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 jest robiona 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.

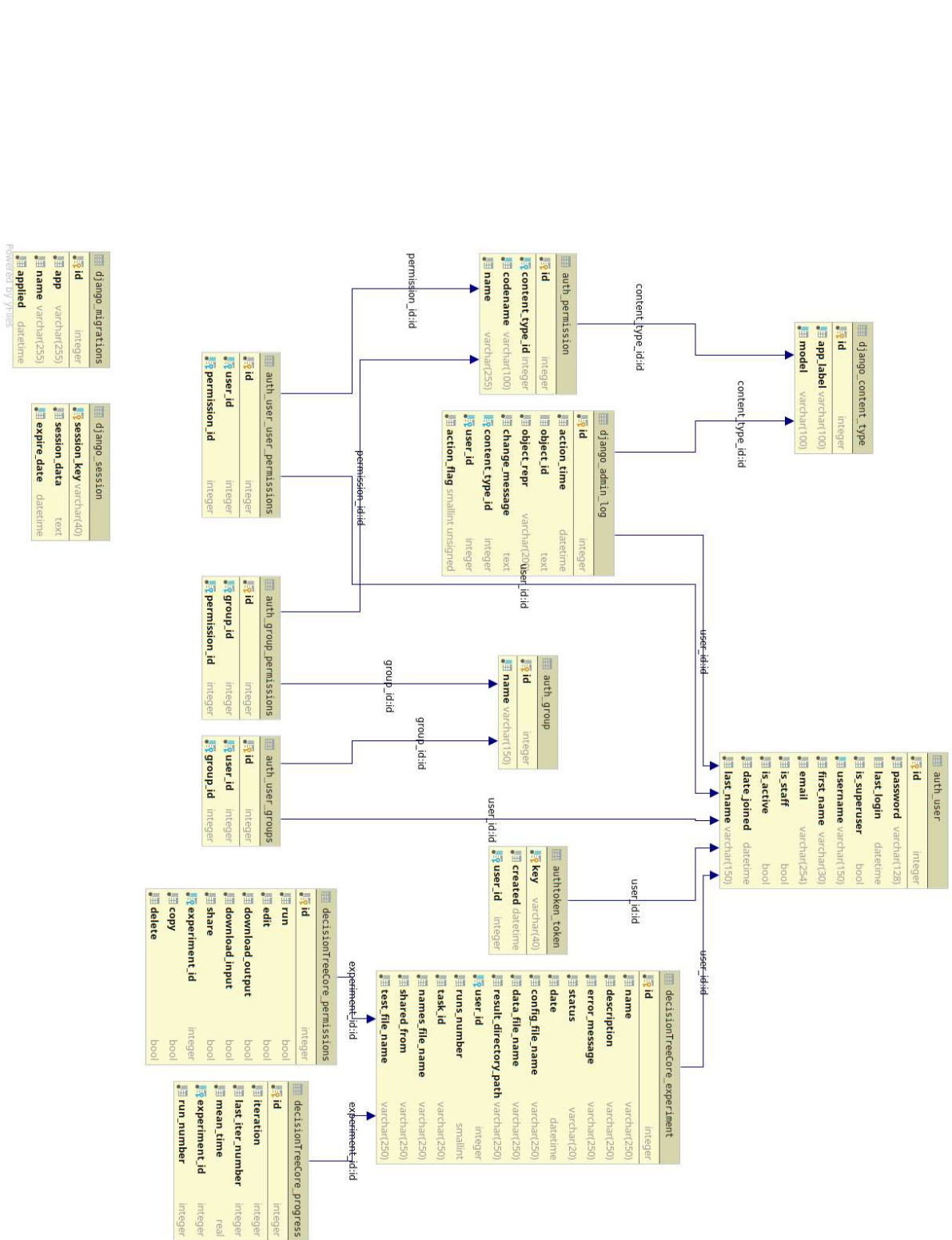
Dodatkowym mechanizmem zaimplementowanym w celu dodania niezawodności aplikacji jest tworzenie pliku „readme.txt”. Zabezpiecza to przed straceniem informacji o eksperymencie w przypadku awarii bazy danych. Przechowuje on informacje takie jak nazwy plików wejściowych czy też nazwę eksperymentu. Na tej podstawie w dowolnej chwili istnieje opcja 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_eksperyment
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 prywatny serwer posiadający 2 GB RAM, procesor o częstotliwości taktowania 2 GHz oraz dysk SSD. Spełnia to podstawowe wymagania pod względem sprzętowym, przy założeniu na początku mniejszego ruchu na stronie. Do uruchomienia projektu jest potrzebna uprzednio zainstalowana instancja platformy Docker. Dzięki konteneryzacji proces wdrożenia projektu jest bardzo prosty i ogranicza się do pojedynczej komendy aplikacji docker-compose. Po uruchomieniu oprogramowanie działa w tle i jest dostępne dla użytkowników. Dostęp do aplikacji jest możliwy przez dowolną przeglądarkę internetową poprzez przejście pod adres „[decisontree.pl](https://decisontree.pl)”.



Rysunek 3.2: Schemat bazy danych, źródło: opracowanie własne



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 eksperyment
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 wartości pola są nazwy użytkowników. Podczas kolejnych udostępnień do pola są dodawane po przecinku następne loginy

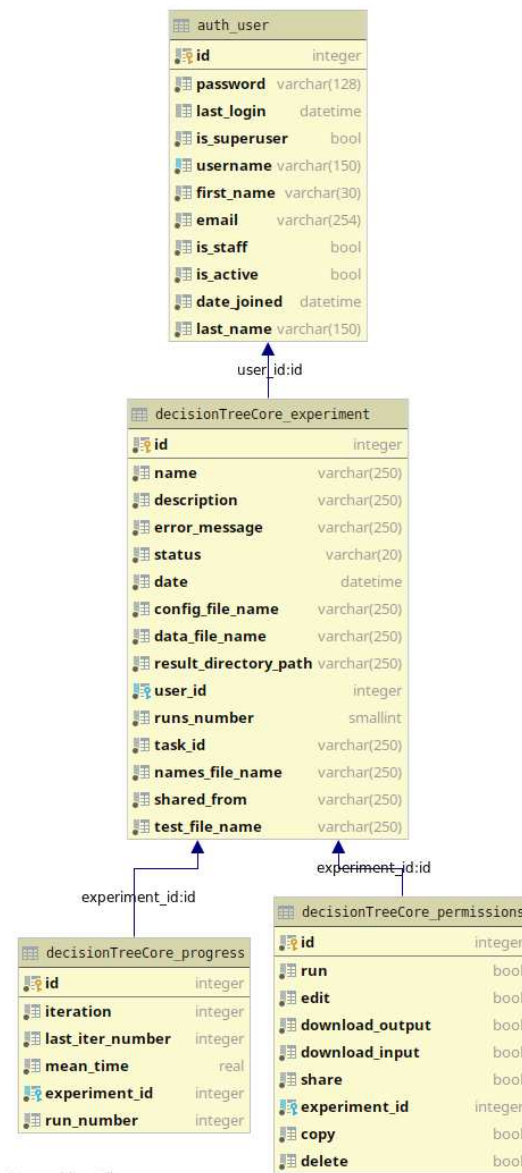
Tabela 3.1: Opis pól encji „decisionTreeCore\_experiment”

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 eksperyment
experiment_id	integer	Numer id eksperymentu, do którego jest przypisany postęp
run_number	integer	Numer określający, które uruchomienie algorytmu właśnie trwa

Tabela 3.2: Opis pól encji „decisionTreeCore\_progress”

<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

Tabela 3.3: Opis pól encji „decisionTreeCore\_permissions”



Rysunek 3.3: Schemat tabel dodatkowych, źródło: opracowanie własne

## **4. Przedstawienie aplikacji oraz testy**

## **Podsumowanie**

Tutaj będzie podsumowanie.

## Bibliografia

- [1] Axios. Axios. <https://github.com/axios/axios>, stan z 10.12.2019 r.
- [2] Benoit Chesneau and contributors. Gunicorn. <https://gunicorn.org/>, stan z 10.12.2019 r.
- [3] Django Software Foundation. Django. <https://www.djangoproject.com/>, stan z 10.12.2019 r.
- [4] Python Software Foundation. Python. <https://www.python.org/>, stan z 08.12.2019 r.
- [5] Collaboratively funded project. Django rest framework. <https://www.django-rest-framework.org/>, stan z 10.12.2019 r.
- [6] Inc. GitHub. Github. <https://github.com/search?o=desc&q=stars%3A%3E1&s=stars&type=Repositories>, stan z 10.12.2019 r.
- [7] The PostgreSQL Global Development Group. Postgresql. <https://www.postgresql.org/>, stan z 10.12.2019 r.
- [8] Aurélien Géron. *Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow*. Helion S.A., 2018.
- [9] Docker Inc. Docker. <https://www.docker.com/>, stan z 10.12.2019 r.
- [10] Facebook Inc. Reactjs. <https://reactjs.org/>, stan z 10.12.2019 r.
- [11] Lucid Software Inc. Lucidchart. <https://www.lucidchart.com/pages/decision-tree>, stan z 25.11.2019 r.
- [12] B. Inny. Tytuł<sup>3</sup> publikacji. In *Tytuł książki*, pages 5–32, Feb 2011.
- [13] Prateek Joshi. *Artificial Intelligence with Python*. Packt Publishing, 2017.
- [14] Mark Lutz. *Python. Wprowadzenie. Wydanie IV*. Helion S.A., 2010.
- [15] Inc. NGINX. Nginx. <https://www.nginx.com/>, stan z 10.12.2019 r.

- [16] Inc. Pivotal Software. Rabbitmq. <https://www.rabbitmq.com/>, stan z 10.12.2019 r.
- [17] Ask Solem and contributors. Celery project. <http://www.celeryproject.org/>, stan z 10.12.2019 r.

## Spis tabel

Tablica 3.1	Opis pól encji „decisionTreeCore_experiment” . . . . .	25
Tablica 3.2	Opis pól encji „decisionTreeCore_progress” . . . . .	25
Tablica 3.3	Opis pól encji „decisionTreeCore_experiment” . . . . .	26



## Spis rysunków

Rysunek 2.1	Diagram przypadków użycia, źródło: opracowanie własne . . . . .	10
Rysunek 2.2	Diagram czynności tworzenia i uruchomienia eksperymentu, źródło: opracowanie wł	
Rysunek 2.3	Architektura systemu, źródło: opracowanie własne . . . . .	15
Rysunek 3.1	Podział projektu na moduły, źródło: opracowanie własne . . . . .	20
Rysunek 3.2	Schemat bazy danych, źródło: opracowanie własne . . . . .	24
Rysunek 3.3	Schemat tabel dodatkowych, źródło: opracowanie własne . . . . .	27

## Spis listingów

3.1	Przykład zawartości pliku readme . . . . .	22
-----	--	----

## **Spis algorytmów**