



**POLITECHNIKA ŚLĄSKA**

**WYDZIAŁ AUTOMATYKI, ELEKTRONIKI I INFORMATYKI**

Praca dyplomowa inżynierska

Analiza zachowań ludzkich na bazie aktywności rejestrowanych w aplikacjach mobilnych z wykorzystaniem metod uczenia maszynowego

autor: Bartłomiej Gładys

kierujący pracą: dr inż. Jakub Nalepa

Gliwice, grudzień 2019



## Oświadczenie

Wyrażam zgodę / Nie wyrażam zgody\* na udostępnienie mojej pracy dyplomowej / rozprawy doktorskiej\*.

Gliwice, dnia 27 grudnia 2019

.....  
(podpis)

.....  
(poświadczenie wiarygodności  
podpisu przez Dziekanat)

\* podkreślić właściwe



## Oświadczenie promotora

Oświadczam, że praca „ Analiza zachowań ludzkich na bazie aktywności rejestrowanych w aplikacjach mobilnych z wykorzystaniem metod uczenia maszynowego ” spełnia wymagania formalne pracy dyplomowej inżynierskiej.

Gliwice, dnia 27 grudnia 2019

.....  
(podpis promotora)



# Spis treści

1	Wstęp i cel pracy	1
2	Analiza tematu	3
3	Wymagania i narzędzia	5
4	Specyfikacja zewnętrzna	7
5	Specyfikacja wewnętrzna	13
6	Weryfikacja i walidacja	25
7	Wnioski i napotkane problemy	33





# Rozdział 1

## Wstęp i cel pracy

Ludzie często nie wiedzą dlaczego ich samopoczucie bywa gorsze lub lepsze. Niewątpliwie można stwierdzić, że wpływa na to wiele czynników. Przedstawiony tutaj projekt inżynierski skupia się na czynniku technologicznym, który otacza nas wszystkich często nie zważając na moment skupienia, czy też wyciszenia. Praca ma więc za zadanie pomóc użytkownikowi w odszukaniu nawyków związanych z używaniem aplikacji mobilnych, które wpływają na niego negatywnie, oraz uświadomić jak często oraz w jakich porach dnia dane aplikacje są przez niego uruchamiane. Celem pracy jest implementacja narzędzia, które pozwoli zbierać dane od użytkowników i klasyfikować ich aktywności jako takie, które mają pozytywny wpływ, oraz takie które wpływają na niego w niepożądany sposób. Dzięki takiemu narzędziu osoba, która będzie je posiadać może zniwelować czas jaki poświęca na aplikacje, które podświadomie są przez nią źle odbierane. Może także dowiedzieć się w jaki sposób wpływa na nią spędzanie czasu podczas korzystania z technologii mobilnych w trakcie pewnego etapu dnia. Zakres pracy obejmuje stworzenie aplikacji mobilnej która będzie w stanie pobierać z urządzenia dane historyczne wszystkich aktywności oraz udostępniać użytkownikowi opcje oceny jego samopoczucia oraz produktywności w ciągu dnia. Analiza wpływu aplikacji i korelacja między nimi a czasem w którym użytkownik z nich korzysta będzie należało do stworzonego systemu. Kolejnym punktem objętym w pracy jest stworzenie pary serwerów, które będą w stanie zbierać uzyskane dane z wielu urządzeń mobilnych, przekształcać je do formatu łatwiejszego do analizy, umożliwiającego optymalną

klasyfikację. Finalnie, zakres pracy obejmuje również wybór algorytmu do klasyfikacji oraz opis analizy i eksperymentów które zostały przeprowadzone aby wybrać model spełniający wszystkie wymagania i posiadający relatywnie wysoką skuteczność. W kolejnym rozdziale zostanie przedstawiona analiza dotycząca aktualnych rozwiązań opisanego wyżej problemu oraz sposób w jaki dane będą grupowane w stosunku do oceny wystawionej przez użytkownika. Następnie zostaną przedstawione wymagania sprzętowe i systemowe, które pozwolą na włączenie stworzonej aplikacji mobilnej i narzędzia jakie były wykorzystane w trakcie implementacji wszystkich serwisów. W kolejnym rozdziale zostaną przedstawione scenariusze i ścieżki z jakimi użytkownik może się spotkać posługując się aplikacją. Następny rozdział przekaże informacje o technologiach i szczegółach implementacji każdego z modułów. Kolejnym wątkiem będzie szczegółowa analiza tego w jaki sposób aplikacja była testowana oraz w jaki sposób został dobrany model dokonujący klasyfikacji danych. Na samym końcu zostaną przedstawione wnioski oraz największe wyzwania i problemy w trakcie tworzenia projektu.

## Rozdział 2

### Analiza tematu

Na rynku aktualnie istnieją systemy, które zwracają uwagę na przedstawiony problem i pomagają użytkownikowi śledzić jego aktywności. Jednym z nich jest RescueTime, system wieloplatformowy, który jest złożony z aplikacji mobilnej, desktopowej oraz rozszerzenia do przeglądarki Google Chrome. Niestety system ten nie posiada funkcji, która analizuje dane a jedynie dostarcza statystki, a użytkownik sam decyduje o tym jak będzie je interpretował. Inną aplikacją tego typu jest TimeDoctor[8], która to wprowadza rozszerzenie dla firm i zespołów, co pozwala na generowanie raportów na temat każdego pracownika i przesłanie ich do lidera danego zespołu. Aplikacja wprowadza też funkcjonalność automatycznego tworzenia zrzutu ekranu. Ten system, tak jak i wcześniejszy nie posiada jednak wbudowanej analizy danych, która przewidywałaby jakie aplikacje działają pozytywnie, a jakie negatywnie. Przedstawiona tutaj aplikacja nie posiada wielu funkcji wcześniej opisanych systemów, ale wprowadza sposób pozyskiwania danych dla każdej aktywności poprzez użycie klasyfikacji i modelu regresji logistycznej. Aby umożliwić klasyfikację danych na bazie aktywności użytkownika i ocen, które podaje, wymagane było zastosowanie pewnego grupowania wyglądającego następująco. Na samym początku została wykorzystana dyskretyzacja, w celu zamiany godziny danej aktywności na porę dnia, np. aktywność, która odbywała się o godzinie 19:00 została przyporządkowana do nominalnej klasy "Wieczór", a aktywność z godziny 12:00 do klasy "Południe". Dzięki takiej zamianie można było w pewien sposób przetestować algorytm oraz uzyskać wyniki lepsze w kontekście interpretacji. Następną kwestią

time_in_sec	name	day_part
10459.658	youtube	night
8932.143	youtube	afternoon
8860.36	com.facebook.orca	night



Tablica 2.1: Oryginalne dane historyczne

time_in_sec	youtube	facebook	day_part_night	day_part_afternoon
28252.161	19391.801	8860.36	19320.018	8932.143

Tablica 2.2: Dane historyczne po transformacji






było zagadnienie dotyczące klasyfikacji wszystkich aktywności z jednego dnia do jednego rekordu posiadającego ocenę w tym dniu. Z tego względu oryginalne dane widoczne w tabeli 2.1 zostały przekształcone do wektora danych który opisuje sumę czasu w sekundach dla każdej aplikacji oraz każdej pory dnia i jest widoczny w tabeli 2.2. Dodatkowo widać, że w procesie transformacji danych użyto kodowanie "1 z n"[10] zarówno dla nominalnych danych odnośnie pory dnia jak i nazw aplikacji. W ten właśnie sposób było możliwe przygotowanie danych historycznych w jednym wierszu i przypisane ich do oceny którą dany użytkownik udzielił. Następnie użyto regresji logistycznej w celu uzyskania wag dla każdego z atrybutów bazując na ocenie. Klasyfikator oceniał dany przykład jako pozytywny jeżeli prognozowana ocena była równa lub większa średniej wszystkich ocen dotychczasowo wystawionych przez użytkownika.




# Rozdział 3


## Wymagania i narzędzia




### Wymagania funkcjonalne

- Użytkownik autoryzuje się w aplikacji mobilnej poprzez wybranie konta  ogólnie 
- Użytkownik ma możliwość przejrzania w aplikacji historii wszystkich aktywności z  przed ostatnich dwudziestu czterech godzin
- Użytkownik otrzymuje każdego dnia powiadomienie z możliwością oceny aktualnego dnia. 
- Użytkownik ma możliwość przejrzania raportu informującego które aplikacje wpływają  negatywnie, a które pozytywnie.
- Program w tle cyklicznie przesyła dane na temat czasu jaki użytkownik spędził na aplikacjach do serwisu zewnętrznego bezpośrednio podłączonego do bazy danych.

### Wymagania нефunkcjonalne

- System: Android 
- Język programowania dla aplikacji mobilnej: Kotlin
- Użycie biblioteki do uwierzytelnienia poprzez konto google: play-services-auth


- Użycie mechanizmów Alarmu w celu cyklicznych zachowań w aplikacji mobilnej
- Dystrybucja aplikacji w Google play[2]
- Użycie natywnych mechanizmów systemu Android w celu pozyskania historii aktywności
- Stworzenie serwisu zarządzającego bazą aktywności
- Użycie klasy ktora w celu interpretacji historii aktywności


 Kod źródłowy projektu zarządzany był z użyciem rozproszonego systemu kontroli wersji Git, korzystając z hostingowego serwisu internetowego GitHub. Podczas implementacji aplikacji mobilnej używane było zintegrowane środowisko Android Studio. W celu testowania i debugowania jakości informacji w bazie danych zostało użyte narzędzie DataGrip przeznaczone  do pracy z różnego rodzaju bazami SQL. Podczas tworzenia serwisu połączonego z aplikacją mobilną używano edytora programistycznego Visual Studio Code. Natomiast w trakcie implementacji serwisu generującego raport oraz podczas analizy danych i różnych klasyfikatorów użyto zintegrowane środowisko Pycharm przeznaczone dla języka Python. Obserwacja i zarządzanie komunikacją między poszczególnymi jednostkami odbywała się za pomocą serwisów wewnątrz AV 


# Rozdział 4

## Specyfikacja zewnętrzna

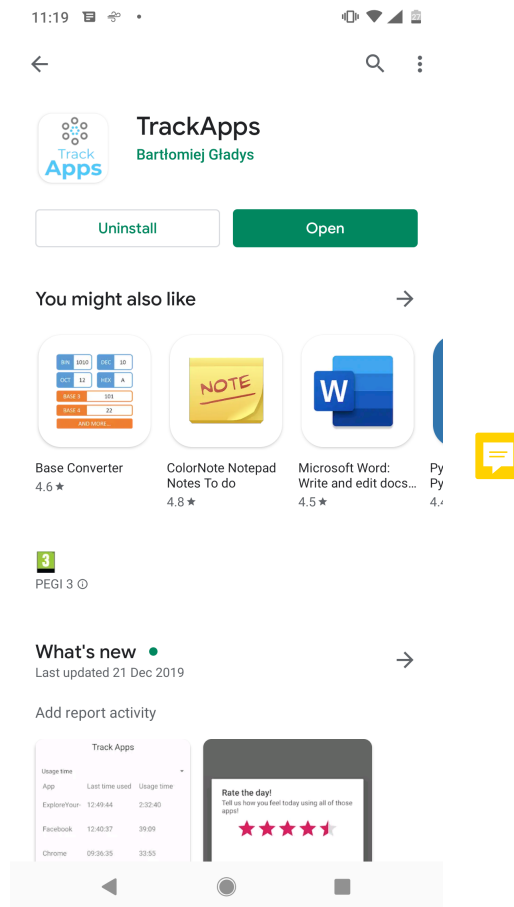


 wymagania sprzętowe, programowe i instalacja

Aby zainstalować aplikację wymagane jest posiadanie urządzenia mobilnego z systemem Android w wersji przynajmniej 6.0 (Android Marshmallow). Ze względu na komunikację z serwisami zewnętrznymi, urządzenie musi również posiadać stały dostęp do internetu. W celu pozyskania programu na urządzenie mobilne wystarczy włączyć Play Store i wyszukać aplikację wpisując TrackApps, a następnie wybrać z listy element  órego szczegóły widoczne są na zamieszczonym obrazku 4.1.

Po pomyślnej instalacji użytkownik jest przekierowany do widoku zezwoleń, w którym świadomie musi ustawić aplikacji mobilnej prawo pobierania historii aktywności. Widok procesu znajduje się na zamieszczonym zrzucie ekranu  4.2.

Następnie wyświetlona zostaje informacja o wymaganej autoryzacji poprzez wybranie konta Google prezentująca się tak jak na obrazku 4.3. Jeżeli użytkownik pozytywnie przejdzie przez wymienione wyżej dwa etapy, na ekranie pojawi się lista z ostatnio odwiedzionymi aplikacjami oraz ilością czasu jaki spędził na danych aktywnościach. Dzięki autoryzacji przez konto Google, użytkownik nie musi wpisywać hasła ani uzupełniać ręcznie swoich danych do logowania, co powoduje, że żadne poufne informacje nie są w stanie wycieknąć przez tę aplikację. Kolejną cechą autoryzacji w ten sposób jest to, że nawet w przypadku gdy użytkownik zmieni urządzenie mobilne to jego



Rysunek 4.1: Szczegóły aplikacji w Google Play.<sup>1</sup>

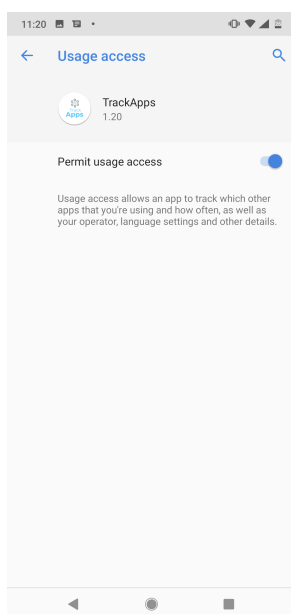
wszystkie wpisy historyczne i oceny będą zapisane wewnątrz systemu.

#### Scenariusze korzystania z systemu

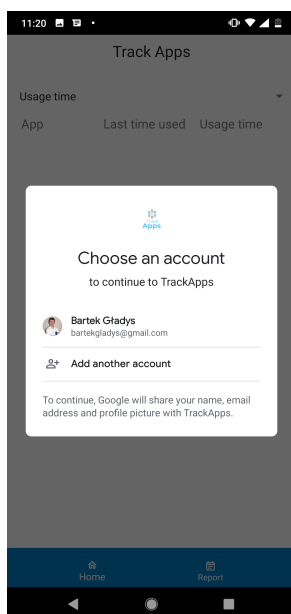
Jedyną aktywnością wymaganą do poprawnego zbierania i klasyfikacji danych jest wpisywanie ocen w zakresie od 0 do 5. Proces ten jest widoczny na zrzucie ekranu 4.4.

Widok ten jest wyświetlony cyklicznie raz każdego dnia i nie można go wywołać ręcznie poprzez nawigację w programie. Aplikacja ukazuje dwie standardowe aktywności po jakich użytkownik może się poruszać. Pierwszą z nich jest widoczna na zrzucie ekranu 4.5 lista, która może być sortowana po trzech atrybutach: nazwie programu, spędzonym na aplikacji czasie oraz




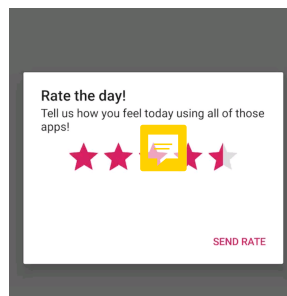
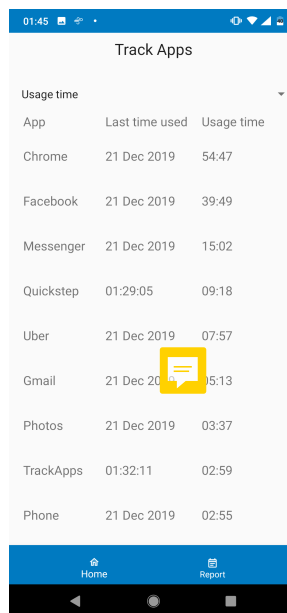



Rysunek 4.2: Ustawianie zezwoleń do śledzenia historii aktywności.<sup>2</sup>

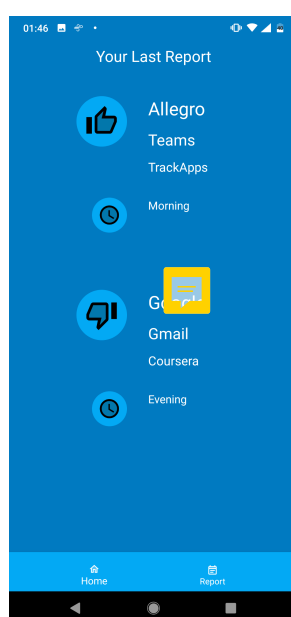


Rysunek 4.3: Autoryzacja poprzez korzystanie z usługi Google Play Services.<sup>3</sup>

bazując na tym  która aplikacja była używana ostatnio. Kolejną aktywnością jest widok raportu 4.6 przedstawiający trzy aplikacje, które uzyskały największe wagi podczas procesu klasyfikacji oraz trzy, które uzyskały naj-

Rysunek 4.4: Widok oceny dnia.<sup>4</sup>Rysunek 4.5: Widok raportu.<sup>5</sup>

mniejsze. Dodatkowo, widnieje tam informacja o tym, jaka pora dnia jest najodpowiedniejsza  oraz przeciwnie o jakich godzinach używanie telefonu wpływa na użytkownika negatywnie.



Rysunek 4.6: Widok raportu.<sup>6</sup>





# Rozdział 5

## Specyfikacja wewnętrzna

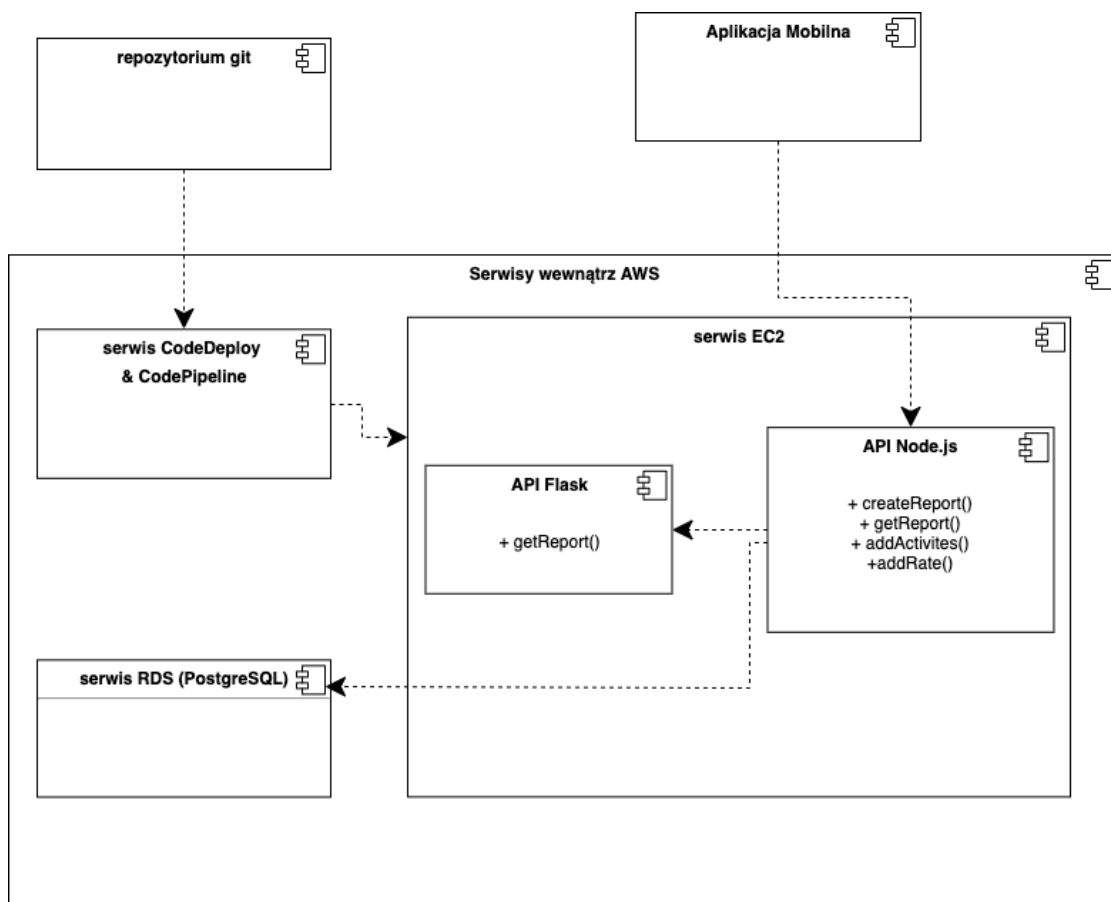


### Struktura projektu

Projekt od strony implementacji został podzielony na trzy części. Jest to aplikacja mobilna napisana w języku Kotlin dla systemów Android, system napisany w środowisku Node.js połączony z bazą danych PostgreSQL w oparciu o infrastrukturę zbudowaną wewnątrz serwisów dostawcy chmurowego AWS (Amazon  Web Services), oraz aplikacja analizująca dane i generująca raporty napisana w języku Python. Na diagramie 5.1 przedstawiono komunikację  pomiędzy poszczególnymi elementami systemu. Wszystkie komponenty poza aplikacją mobilną znajdują się wewnątrz AWS. Aplikacje bezpośrednio komunikujące się z bazą danych są uruchomione wewnątrz klastra EC2 (Amazon Elastic Compute Cloud)[6], czyli serwisie www, który dostarcza skalowalną moc obliczeniową w chmurze. Interfejs wystawiony przez aplikację napisaną w środowisku Node.js jest głównym źródłem komunikacji dla aplikacji mobilnej. Natomiast serwis napisany w języku Python posiada interfejs, z którego korzysta jedynie wcześniej opisany serwis na potrzeby tworzenia i zapisywania w bazie danych raportów. Dodatkowo, warto zwrócić uwagę na serwis CodeDeploy & CodePipeline, który to ułatwia pracę programiście poprzez wprowadzanie automatycznej budowy aplikacji. W momencie kiedy pojawia się nowa zmiana na serwerze git, natychmiastowo zostaje o tym poinformowany owy serwis i wykonuje wszystkie niezbędne skrypty po stronie klastra EC2, które umożliwiają ściągnięcie zależności, zainstalowanie



pakietów i zbudowanie aplikacji. Inną zaletą skorzystania z serwisów AWS i klastra EC2, jest łatwość obsługi i połączenia systemu z bazą danych, oraz możliwość zarządzania otwartymi i zamkniętymi portami, dzięki czemu aplikacja pozostaje bezpieczna udostępniając w sieci tylko niezbędne zasoby.



Rysunek 5.1: Infrastruktura projektu.<sup>1</sup>

### Aplikacja Mobilna

Aplikacja została napisana w języku Kotlin przy użyciu środowiska Android Studio i wydana na platformie Google Play, aby ułatwić zbieranie danych i instalowanie aplikacji przez potencjalnych użytkowników. Na diagramie 5.2 widać, że aplikacja posiada trzy aktywności (widoki na jakie użytkownik może się natknąć). Pierwszym z nich jest *MainActivity*, gdzie użytkownik ma dostęp do listy z historią wszystkich aplikacji jakich używał podczas ostatnich

dwudziestu czterech godzin. Na potrzeby wyświetlania listy została stworzona klasa *UsageStatsAdapter*, która rozszerza natywną klasę *BaseAdapter* dodając nowe funkcje, takie jak: sortowanie po nazwie aplikacji, sortowanie po ostatniej dacie uruchomienia i sortowanie po czasie spędzonym na danej aplikacji. W celu implementacji sortowania w aplikacji stworzono trzy klasy pochodzące z istniejących wewnątrz klasy *UsageStatsComparator* rozszerzające funkcjonalność generycznej klasy *Comparator*. Klasa ta również implementuje metodę *getView*, która wyświetla wszystkie niezbędne informacje z listy aktywności i przekazuje je do widoku listy, która zostanie wygenerowana. Dodatkowo, następuje tutaj pozyskanie nazwy aplikacji bazując na nazwie modułu, który został zapisany w bazie danych dla konkretnego programu, oraz formatowanie daty z postaci zapisanej w milisekundach do postaci łatwej do przeczytania i rozpoznania. Pobranie historii wszystkich aplikacji zajmuje się klasa *UsageStatsUtil*, która konwertuje dane w taki sposób aby były łatwiejsze do wyświetlenia, czy też przesłania do innych serwisów. Implementacja logiki zajmującej się konwersją danych znajduje się w metodzie *fetchStatsDataForExternal*. Metoda ta pobiera listę zdarzeń, które odbyły się na urządzeniu mobilnym w przekazanym przez parametr okresie czasu, wykorzystując przy tym natywną klasę *UsageStats*. Następnie odbywa się filtrowanie przekazujące dalej jedynie te zdarzenia, które miały wpływ na użytkownika i są one opisane flagami *MoveToForeground* i *MoveToBackground*. Gdy zdarzenia są już pobrane i przefiltrowane, następuje sortowanie i konwersja do typu listy, gdzie każdy element składa się z dwóch zdarzeń o różnych flagach, które zostały opisane wcześniej. Dodatkowymi funkcjami tej aktywności jest sprawdzenie czy użytkownik nadał aplikacji mobilnej prawo do pobrania historii statystyk oraz weryfikacja stanu uwierzytelnienia z serwisem Google Auth[3]. Jeżeli użytkownik wchodzi do aplikacji po raz pierwszy to pojawi się okno z wyborem konta Google przez które chciałby się zalogować do aplikacji, następnie zostanie on przekierowany do widoku ustawień urządzenia mobilnego, gdzie będzie musiał zaznaczyć odpowiednie ustawienia dla tej aplikacji. Mechanizm uwierzytelnienia został napisany w klasie *ApiHan-*

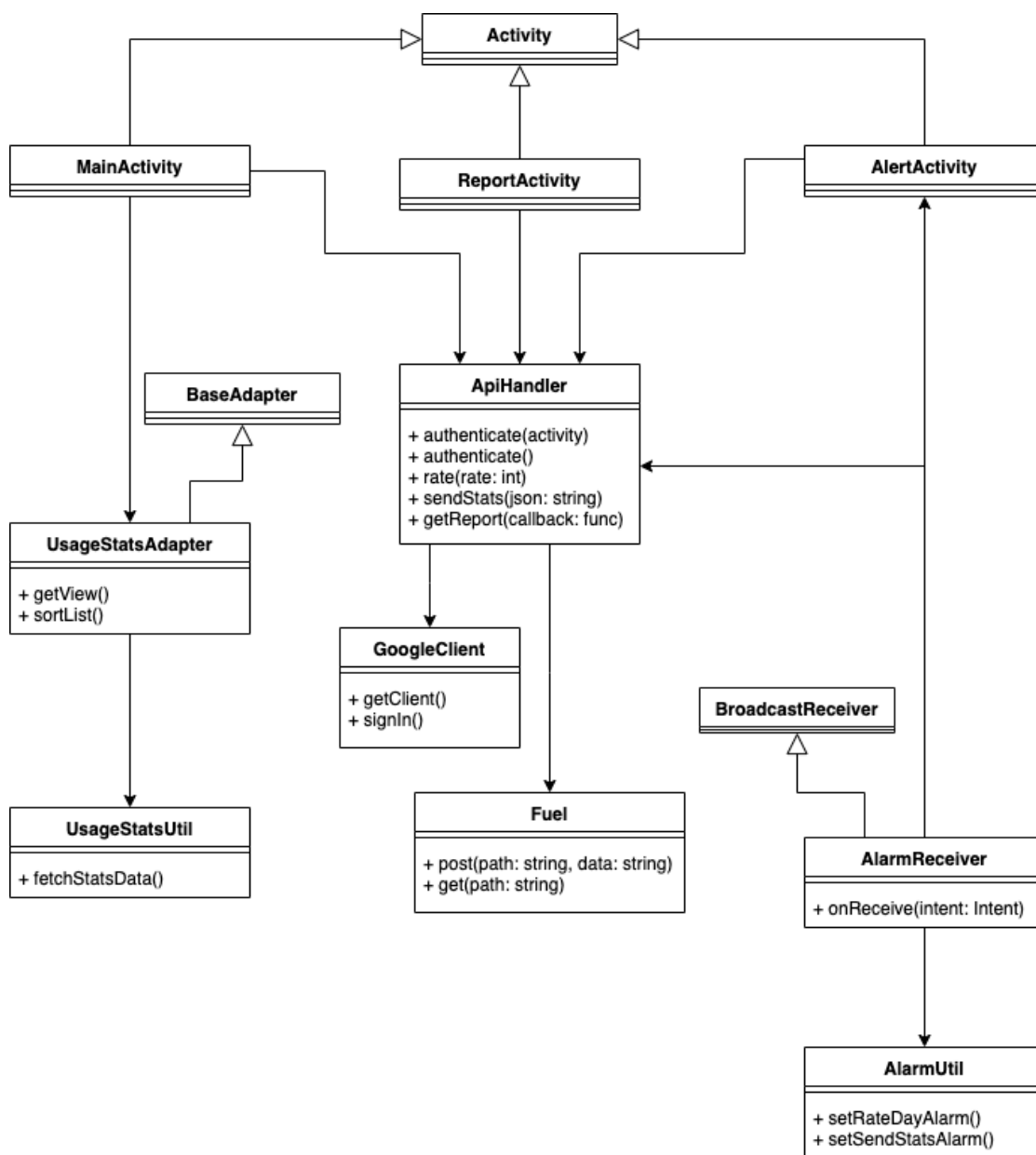
bler i bazuje na bibliotece play-services-auth. Natomiast mechanizm, który powoduje przejście do widoku ustawień został stworzony za pośrednictwem natywnej klasy *UsageStatsManager*[1] i mechanizmu intencji, dzięki czemu istnieje możliwość przekierowania użytkownika z aplikacji do widoku ustawień. Za kolejny widok w aplikacji mobilnej odpowiada klasa *ReportActivity*. Tak samo jak we wcześniejszym przypadku, używane są zależności z klasy *ApiHandler*. Natomiast w tym momencie poza metodą *authenticate*, używana jest również metoda *getReport*, która pozwala na pobranie przypisanego do konta ostatniego raportu w bazie danych i możliwość wyświetlenia go w aktywności. W raporcie widnieje informacja o tym jakie aplikacje działały na użytkownika pozytywnie, a jakie negatywnie. Znajduje się tu także informacja o tym jaka pora dnia powinna być bardziej sprzyjająca w kontekście użytkownika urządzenia mobilnego. Metoda *getReport* jak i metody, które opisane zostaną w dalszej części (*rate*, *sendStats*) zostały zaimplementowane przy użyciu biblioteki Fuel, która znacznie ułatwia wykonywanie żądań do serwisów zewnętrznych za pomocą protokołu HTTP. Warto dodać, że biblioteka pozwala również na łatwe skorzystanie z wzorca projektowego jakim jest Interceptor w celu dodanie dodatkowych funkcjonalności do każdego żądania. W przypadku tej aplikacji mobilnej Interceptor ma za zadanie wstrzykiwać odpowiedni nagłówek HTTP za każdym razem gdy następuje komunikacja z serwisem zewnętrznym, dzięki czemu zostaje przesłana informacja na temat aktualnie zalogowanego użytkownika. Za ostatni widok w aplikacji jest odpowiedzialna klasa *AlertActivity*, ta aktywność nie jest dostępna z widoku menu w aplikacji, natomiast cyklicznie każdego dnia pojawia się na urządzeniu mobilnym użytkownika z zapytaniem o ocenę produktowności. Do komunikacji z serwisem zewnętrznym w celu przesłania wybranej przez użytkownika oceny służy po raz kolejny klasa *ApiHandler* oraz metoda *rate*. Codzienne wywołanie komunikatu w postaci aktywności zostało zaimplementowane w klasie *AlarmReceiver*, która rozszerza natywny *BroadcastReceiver*[4] wprowadzając funkcjonalność nasłuchiwanie na wywołanie intencji z systemu android. Wyzwalacz akcji bazuje na natywnym mechanizmie alarmów, pozwala on o wybranej godzinie uruchomić wybraną funkcję. Implementacja tego rozwiązania opiera się o nadpisanie bazowej metody *onReceive* dodając funkcjonalność,



która nasłuchuje na dany typ intencji. Jeśli takowa intencja się pojawi, zostanie stworzona nowa, do której przypisana będzie flaga informująca o tym, że powinno zostać stworzone nowe zadanie i finalnie nowo stworzona intencja zostanie przesłana jako argument do metody *startActivity* w kontekście uzyskanym z parametru *onReceive*. Poza widokami w aplikacji mobilnej warto też wyszczególnić mechanizm, który cyklicznie wysyła dane na temat historii użytkownika do serwisu zewnętrznego. Mechanizm ten korzysta także z funkcji natywnego alarmu oraz w znacznej mierze z wcześniej opisanych metod *authenticate* i *sendStats* z klasy *ApiHandler*. W celu autoryzacji z serwisem zewnętrznym gdy aplikacja działa w tle i przesłania niezbędnych informacji. Implementacja tego rozwiązania również opiera się o dodanie warunku nasłuchującego na konkretną intencję. Gdy takowa się pojawi, w pierwszym kroku zostanie utworzona nowa instancja typu *UsageStatsManager*. Następnie, obiekt ten zostanie przesłany jako źródło informacji razem z pewnym domyślnym odcinkiem czasu do wcześniej wspomnianej metody *fetchStatsDataForExternal*. Przed samym wysłaniem żądania uzyskane dane zostaną przekształcone do formatu JSON używając przy tym biblioteki Gson, aby w ostatnim kroku użyta została statyczna metoda *sendStats* na obiekcie *API*, która wyśle żądanie HTTP POST pod adresem *activites* serwisu zewnętrznego.

## Baza danych

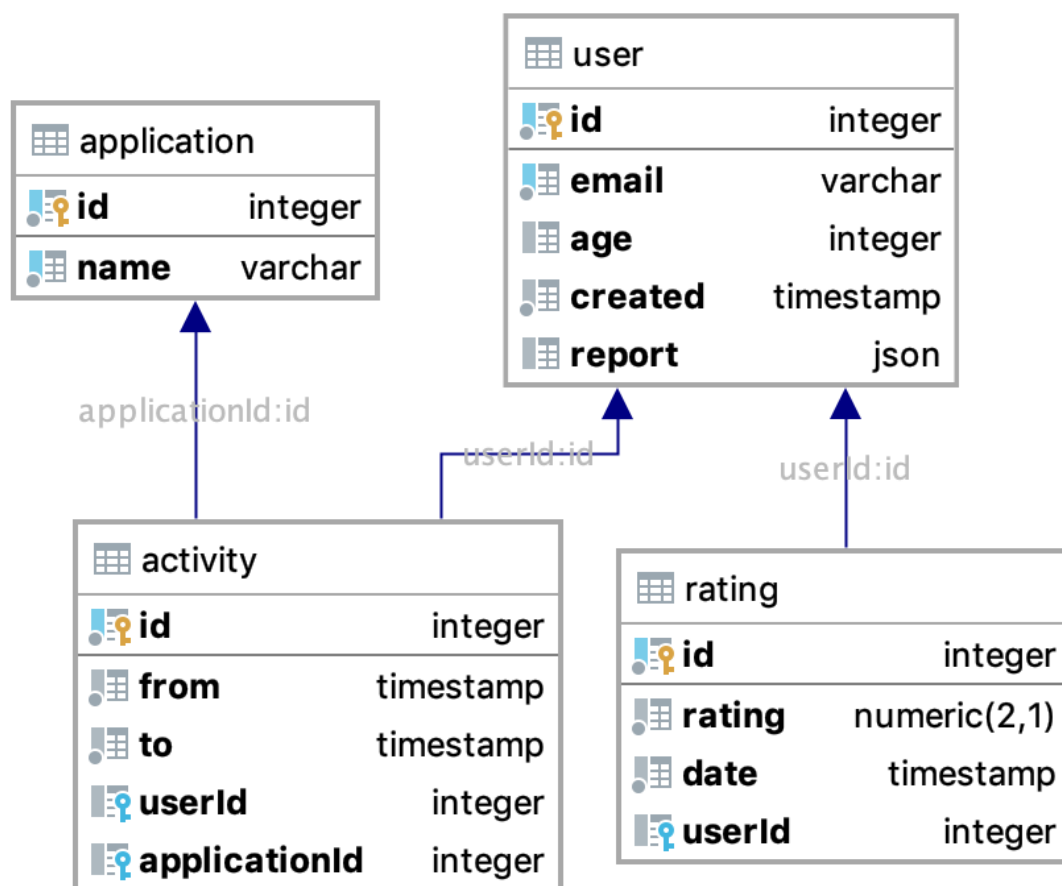
Silnikiem bazodanowym, który został wybrany jest już wcześniej napomniany PostgreSQL. Głównym atutem tego narzędzia jest ogromna ilość dostępnych typów danych i funkcji pomagających w pewien sposób te dane konwertować. Na diagramie 5.3 widać podział na cztery tabele. Pierwsza z nich, *application*, opisuje zbiór wszystkich aplikacji rozpoznawanych w systemie. Dzięki takiej reprezentacji danych bezproblemowo można sprawdzać, jak często dana aplikacja była uruchamiana przez użytkownika oraz z technicznego punktu widzenia zachować wysoki poziom normalizacji bazy danych. Kolejna z nich, *user*, przechowuje informacje na temat adresu email, wieku, punktu w czasie kiedy to dany użytkownik został stworzony w bazie danych oraz raportu przechowywanego jako typ JSON (JavaScript Object Notation), czyli



Rysunek 5.2: Diagram przedstawiający zależności między klasami w aplikacji mobilnej.<sup>3</sup>

obiektu który posiada swoje własne atrybuty i jest dowolnie zagnieżdżony. Można by pomyśleć, że wybór takiego typu danych ogranicza funkcjonalność i elastyczność tabeli, natomiast w tym przypadku tworzeniem struktury ca-

tego atrybutu raport zajmuje się inny system, a baza danych pełni jedynie rolę, w której przyporządkuje ostatni raport do użytkownika, dzięki czemu może być on wyświetlony w aplikacji mobilnej.



Rysunek 5.3: Diagram przedstawiający schemat bazy danych.<sup>4</sup>

System zarządzający bazą danych i obsługą żądań z aplikacji mobilnych

Aplikacja została napisana przy pomocy środowiska Node.js oraz języka TypeScript, który jest kompilowany prosto do postaci zgodnej z standardem JavaScript. Wybór ten został podjęty ze względu na możliwość używania silnego typowania, które nie jest dostępne w natywnej wersji JavaScript. Dodatkowo TypeScript wspiera wiele rozszerzeń nie uwzględnionych w specyfikacji JavaScript np. interfejsy, czy też wyliczeniowe typy danych, które pomagają

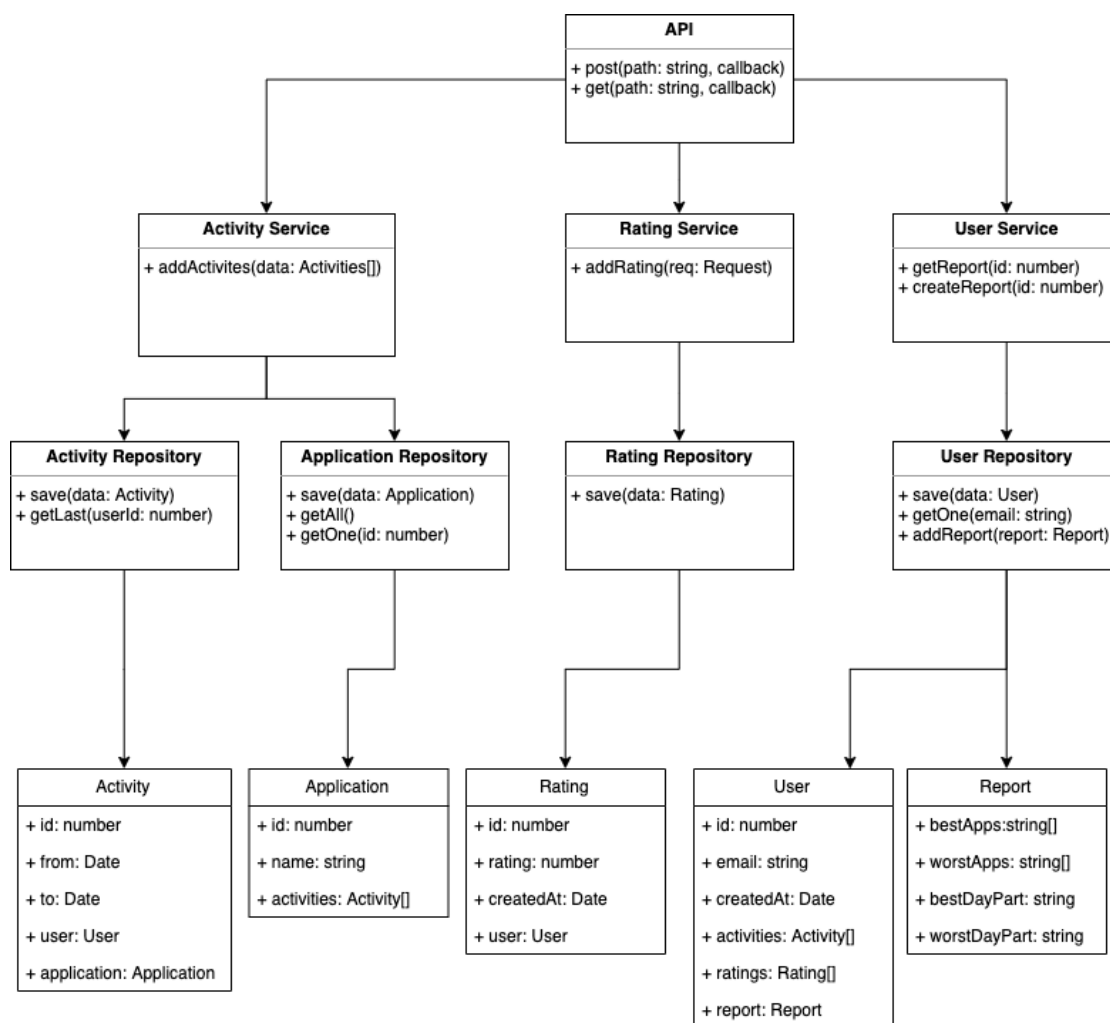
opisać implementację zgodnie domeną problemu. Jediną wadą wspomaganie się silnym typowaniem w środowisku Node jest przymus do kompilowania kodu po każdej zmianie, ale istnieją narzędzia, które ułatwiają ten proces i pozwalają na dynamiczne wyprodukowanie wersji wynikowej w bardzo krótkim czasie. Dodatkowo, podczas implementacji projektu zostały użyte biblioteki pomocnicze. Jedną z nich jest `typeorm`[11] i jak nazwa wskazuje zapewnia ona mapowanie obiektowo-relacyjne, co powoduje, że obsługa bazy danych po stronie serwisu jest w większym stopniu odseparowana od silnika bazodanowego. Przewagą `typeorm` nad innymi rozszerzeniami tego typu jest bardzo dobre wsparcie dla języka TypeScript, dzięki czemu statyczna analiza kodu źródłowego może być w pełni wykorzystana. Inną, niemniej ważną zaletą tej biblioteki jest wsparcie dla wielu paradygmatów programowania, które można użyć w różnych przypadkach i przy innej poziomie trudności projektu. Jednym z nich jest wzorzec obiektowy aktywnego rekordu, dzięki któremu podstawowe metody do zarządzania danymi mogą być automatycznie wygenerowane. Niestety wzorzec ten jest ściśle zintegrowany z strukturą bazy danych i praktycznie zaciera ślady granicy między warstwą przechowującą dane, a logiką domenową, dlatego też podczas implementacji użyto wzorca Repozytorium, który również jest bardzo dobrze wspierany przez `typeorm` i umożliwia separację między wymienionymi powyżej warstwami. Inną kluczową biblioteką jest `google-auth-library`, która konwertuje i weryfikuje nagłówki przychodzących żądań z aplikacji mobilnych dzięki czemu możliwe jest przypisanie użytkownika do danych historycznych. Na samym końcu, przy samej obsłudze wymiany informacji z klientami mobilnymi oraz innymi serwisami w całym procesie pomocne są biblioteki takie jak `axios` oraz `express`. Pierwsza z nich umożliwia wysyłanie żądań HTTP do innych aplikacji, natomiast druga sprawia, że w łatwy sposób można utworzyć REST API (interfejs pozwalający zarządzać zasobami serwisu) oraz wstrzykiwać zależności, które będą wykonywać pewne transformacje na danym żądaniu i przekazywać je dalej. Na diagramie 5.4 widać podział ze względu na serwisy, które zajmują się pewną częścią logiczną systemu i spotykają się dopiero w miejscu gdzie aplikacja tworzy interfejs, z którego mogą skorzystać inni klienci. Zaczynając od lewej strony, pierwszy widnieje serwis *Activity*, który



umożliwia dodanie danych historycznych przez aplikację mobilną. Korzysta on z dwóch repozytoriów *Activity* oraz *Application*, ponieważ dane są te bezpośrednio połączone. Serwis zapisuje ile razy i jak długo użytkownik korzystał z danej aplikacji więc istnieje tu relacja jeden do wielu, gdzie dla jednego obiektu typu *Application* przydzielonych jest wiele obiektów typu *Activity*. Wyżej opisane repozytoria korzystają z klas typu *Entity*, które odzwierciedlają postać tabel w bazie danych. Widać tu jednak bardzo istotną różnicę w reprezentacji relacji, ponieważ obiekt typu *Application* posiada referencję do listy aktywności co nie miałyby miejsca w modelu czysto relacyjnym. Kolejną gałąź diagramu przedstawia serwis *Rating*, który jest odpowiedzialny za obsługę ocen przychodzących z urządzeń mobilnych. Tak samo jak we wcześniejszym przypadku jest podział na serwis, repozytorium oraz strukturę odpowiadającą tabeli *Rating* z bazy danych. Ostatnią częścią aplikacji jest funkcjonalność odpowiadająca za tworzenie spersonalizowanych raportów. W tym miejscu główny serwis przed stworzeniem raportu w bazie danych używa wcześniej wymienionej biblioteki *requests* w celu utworzenia żądania HTTP i wysłania go do systemu zajmującego się przetworzeniem danych. Jako rezultat operacji zwrócony zostanie model, będący w stanie przedstawić aplikację wpływającą na użytkownika pozytywnie i negatywnie.

Serwis odpowiedzialny za generowanie raportów

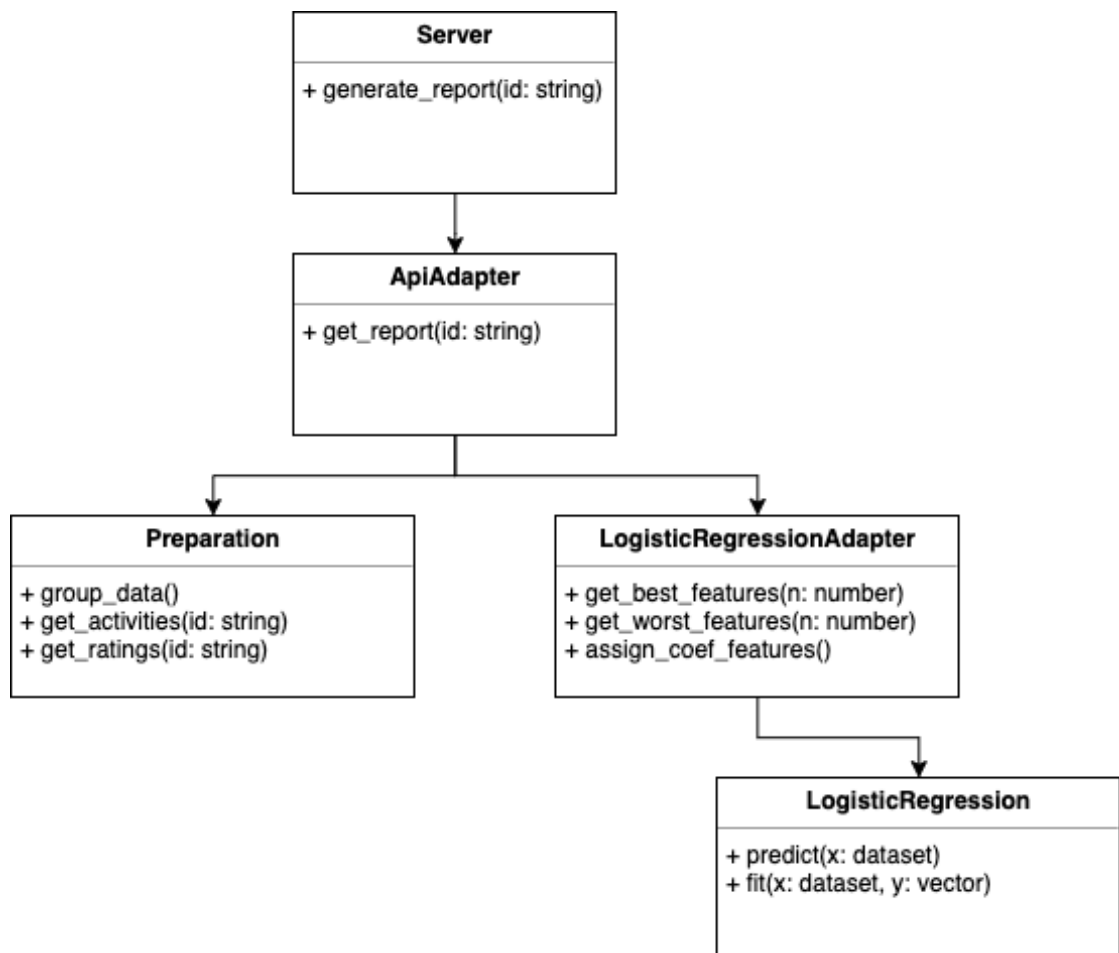
Aplikacja została napisana w języku Python ze względu na dużą ilość gotowych do użycia narzędzi wspomagających transformację danych oraz uczenie maszynowe. Wśród wykorzystanych bibliotek znajdują się biblioteki takie jak *numpy* oraz *pandas*. Pozwalają one w łatwy sposób operować na danych dostarczając funkcje umożliwiające operacje na macierzach, czy też funkcje działające na wektorach. Do komunikacji z bazą danych nie zostało użyte rozszerzenie mapujące relacje na obiekty, a zwykły adapter zgodny z silnikiem PostgreSQL o nazwie *psycopg2*. W celu stworzenia modelu treningowego oraz testowego, który będzie mógł przetworzyć statystyki na predykcję i odpowiednio je zinterpretować została użyta biblioteka *sklearn*. Aby umożliwić drugiemu serwisowi łatwy dostęp do tworzenia raportów, użyto również bibliotekę *Flask* w celu udostępnienia interfejsu będącego w stanie przyjąć



Rysunek 5.4: Diagram przedstawiający zależności między elementami w aplikacji Node.js.<sup>6</sup>

żądania HTTP. Diagram 5.5 przedstawia zależności między częściami tego systemu. Na górze znajduje się obiekt mający na celu jedynie udostępnienie wcześniej opisanego interfejsu pod zadaną ścieżką. Po otrzymaniu identyfikatora użytkownika, przesyła on go dalej do adaptera mającego kontakt z obiektami odpowiedzialnymi za pobranie i wstępne przetworzenie danych oraz zwrócenie niezbędnych danych dla raportu. Obiekt *Preparation* jest bezpośrednio połączony z bazą przez opisany wcześniej adapter *psycopg2*. Wywołuje on zapytanie SQL zwracające aktywności użytkownika o podanym

identyfikatorze, a następnie modyfikuje pobrane dane tworząc dodatkowe atrybuty w celu dyskretyzacji. Dalej tworzone jest kolejne zapytanie SQL w celu pobrania wszystkich ocen jakie dany użytkownik podał wewnątrz aplikacji mobilnej. Na samym końcu wywoływana jest funkcja mająca za zadanie pogrupować dane w odpowiednim formacie, zgodnym z interfejsem biblioteki pandas. Po pobraniu i przekształceniu wierszy z bazy danych użyty jest obiekt *LogisticRegressionAdapter* odpowiadający za pobieranie nazw aplikacji wpływających na użytkownika pozytywnie i negatywnie. Implementacja ta opiera się o klasyfikator regresji logistycznej importowany z wcześniej wspomnianej biblioteki sklearn[9]. W trakcie inwencji metody *predict* i *fit*, algorytm przyporządkuje specjalne wagi. Na ich podstawie można określić jak na użytkownika działa dana aplikacja, biorąc pod uwagę oceny, które wybrał.



Rysunek 5.5: Diagram przedstawiający zależności między elementami w aplikacji napisanej w języku Python.<sup>8</sup>



# Rozdział 6

## Weryfikacja i walidacja



### Proces testowania serwisów

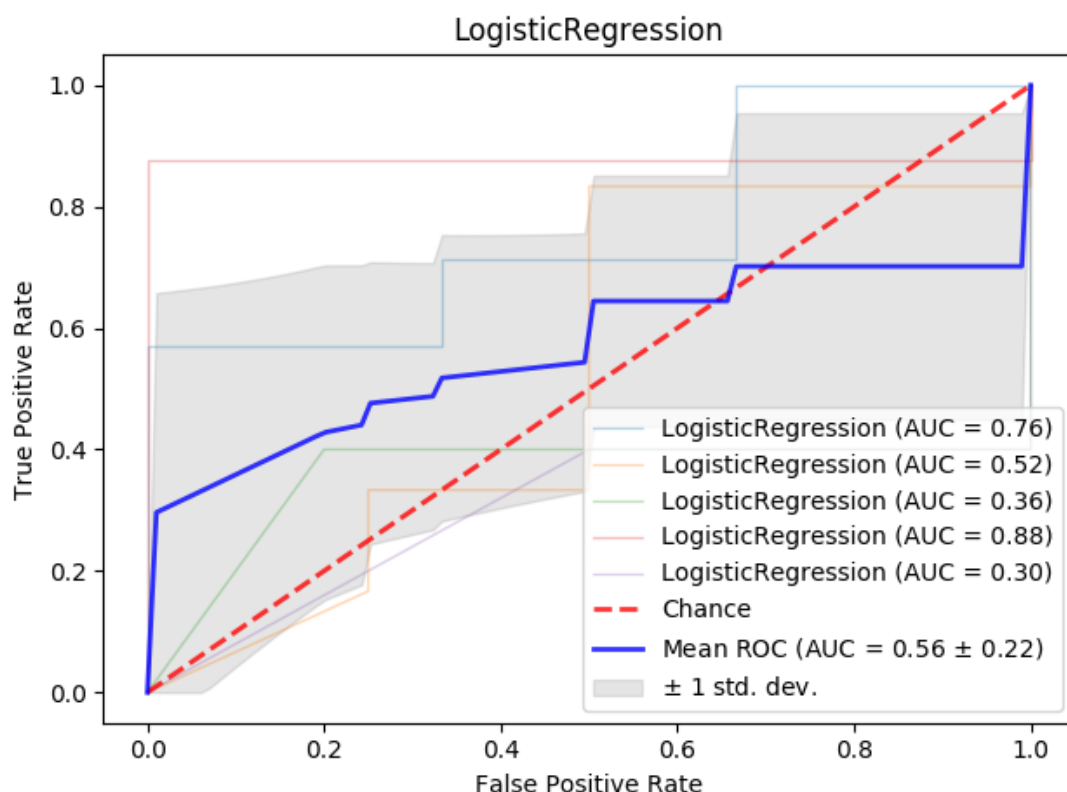
Testowanie wszystkich serwisów połączonych z bazą danych w znacznym stopniu było powiązane z serwisami AWS. Przez cały czas trwania projektu silnik bazy danych nie był zainstalowany lokalnie. Wszystkie dane były przechowywane i testowane wewnątrz serwisu R<sup>2</sup>, a na urządzeniu lokalnym używano jedynie zintegrowanego środowiska DataGrip, aby uzyskać zdalne połączenie z bazą. W momencie implementacji każdego z serwisów, pozostałe serwisy działały zawsze w obszarze AWS. Dodatkowo jak wspomniano wcześniej podczas opisu struktury systemu, używano podejścia CD (ang. *Continuous Delivery*). Pozwala ono na automatyczne budowanie aplikacji. Przyczyniło się to do szybkiej weryfikacji czy oprogramowanie działa poprawnie. Natychmiastowo po każdej zmianie w serwisie, który udostępnia API widoczna była różnica w zachowaniu aplikacji mobilnej.

### Porównanie i wybór algorytmu użytego do klasyfikacji

Biorąc pod uwagę ilość danych w innych eksperymentach z wykorzystaniem uczenia maszynowego można stwierdzić, że w tym przypadku danych było stosunkowo mało (w ciągu jednego miesiąca dla jednego użytkownika można było uzyskać maksymalnie 30 ocen, czyli 30 wierszy w bazie danych). W celu wytrenowania modelu oraz sprawdzenia jaki klasyfikator sprawdza się najlepiej użyto walidacji krzyżowej dla  $K=5$ . Podczas tego procesu każdy z podzbiórów raz pełni rolę podzbioru testowego i  $K$  razy rolę podzbioru



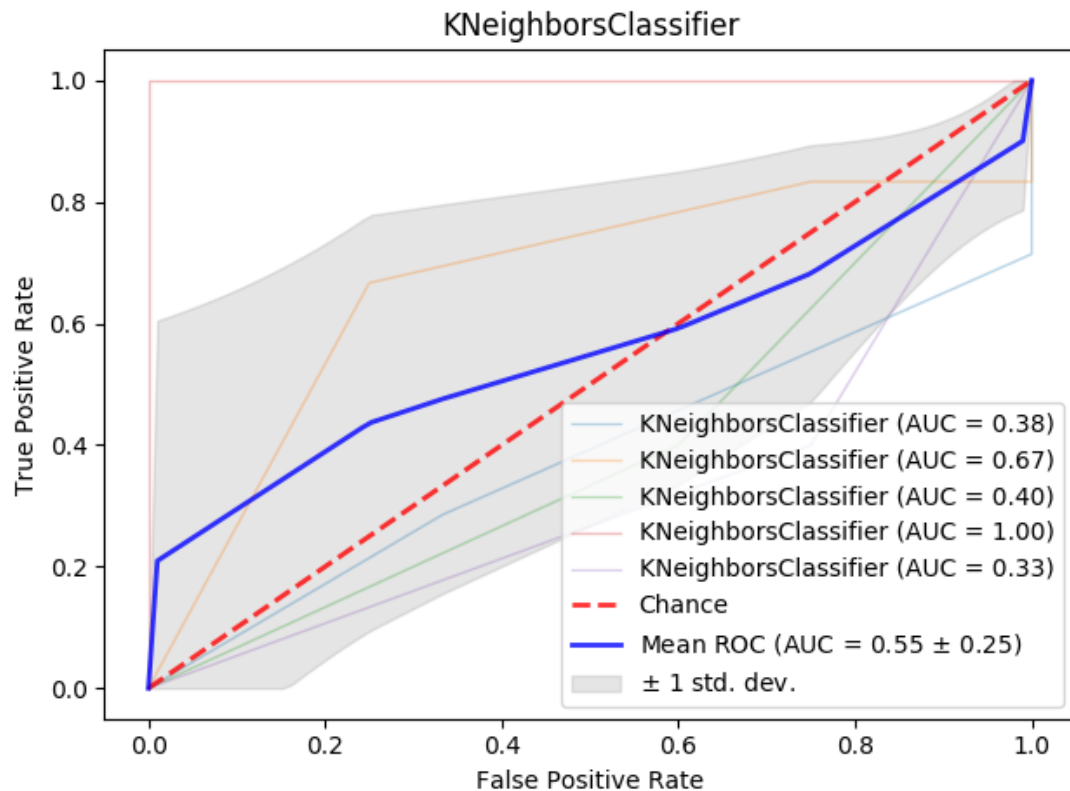
treningowego. Ze względu na to, że nie należy się spodziewać zbalansowanych danych (czasem może być więcej przykładów pozytywnych, a czasem więcej negatywnych), sugerujemy nie został objęty taki wskaźnik jak trafność (z ang. *accuracy*). Wskaźnik ten działa w ten sposób, że jeżeli większość przykładów jest pozytywna (lub negatywna), to bezwzględne ustawienie wartości pozytywnej (lub negatywnej) podczas prognozy fałszywie zwróci dobry wynik. Z tego względu jako parametr określający jakość danego klasyfikatora użyto rozmiar pola (AUC – z ang. *Area under the curve*) pod krzywą ROC (z ang. *Receiver Operating Characteristic*), która jest funkcją odcięcia przedstawiającą TPR (z ang. *True Positive Rate* – miara pokrycia) w zależności od FPR (z ang. *False Positive Rate* – poziom błędu na podstawie przykładów fałszywie przewidzianych jako prawdziwe)[5]. Eksperymentom zostały poddane 4 klasyfikatory: Regresja Logistyczna, K-NN (z ang. *k nearest neighbours* – najbliższych sąsiadów), drzewo decyzyjne oraz MLP (z ang. *Multilayer Perceptron* – najpopularniejszy typ sztucznej sieci neuronowej). Wszystkie wykresy dotyczące eksperymentu posiadają tę samą strukturę. Każdy z nich składa się pięciu składowych, które powstały podczas walidacji krzyżowej, jednej wynikowej będącej średnią wszystkich elementów oraz linii diagonalnej pomocnej podczas wizualnego określania jaką jakość posiada dany model. Dodatkowo wykresy przedstawiają informacje na temat AUC oraz odchylenia standardowego. W implementacji części programu zajmującej się generowaniem raportu wybrano Regresję Logistyczną widoczną na wykresie 6.1 ze względu na relatywnie wysoką wartość średniej ROC AUC względem innych algorytmów oraz łatwość interpretacji modelu. Regresja Logistyczna bazuje na funkcji sigmoid wizualnie prezentującej się jako krzywa przypominająca literę “S” zdefiniowaną wzorem:  $\frac{1}{1+e^{-y}}$ , gdzie  $y$  to wynik funkcji typu  $b_0 + b_1 x_1 + b_2 x_2 + \dots + b_n x_n$ , natomiast  $b$  opisuje wagi dla poszczególnych wartości  $x$ . Dla danych opisywanych w tym projekcie  $n$  jest zdefiniowane jako suma wszystkich aplikacji, z których pewny użytkownik korzysta plus liczba etykiet, które opisują porę dnia zależną od godziny używania aplikacji. We wzorze znajduje się jeden wolny element  $b_0$  (z ang. *bias*), który definiuje pewną wartość początkową i w przypadku regresji liniowej dla zerowych wartości  $x$  rezultatem będzie właśnie ta składowa.



Rysunek 6.1: ROC AUC dla regresji logistycznej.<sup>1</sup>

Podobny wynik uzyskano używając modelu K-NN widocznego na wykresie 6.2, jednak interpretowalność wyników, będąca kluczową jednostką w tym projekcie była zdecydowanie lepsza dla regresji. K-NN jest nieparametryczną metodą klasyfikacji. Bazuje ona na przydzielaniu pewnego elementu do klasy, do której należy większość z jego sąsiadów. Do prawidłowego działania algorytmu trzeba zdefiniować pewną liczbę  $k$ , która przedstawia ilość sąsiadów wymaganą by przydzielić dany obiekt do danej klasy. Przedstawiony tutaj model opierał się o hiperparametr  $k = 3$ . Dlaczego więc algorytm K-NN jest nazwany metodą nieparametryczną, jeżeli trzeba zdefiniować parametr  $k$ ? Otóż, podczas uczenia maszynowego występuje rozróżnienie na parametry i hiperparametry. Te pierwsze są dynamiczne i zmieniają się za każdym razem, gdy model jest trenowany i testowany na nowych danych (np. wagi przyporządkowane we wcześniej opisanym modelu regresji logistycznej). Hi-

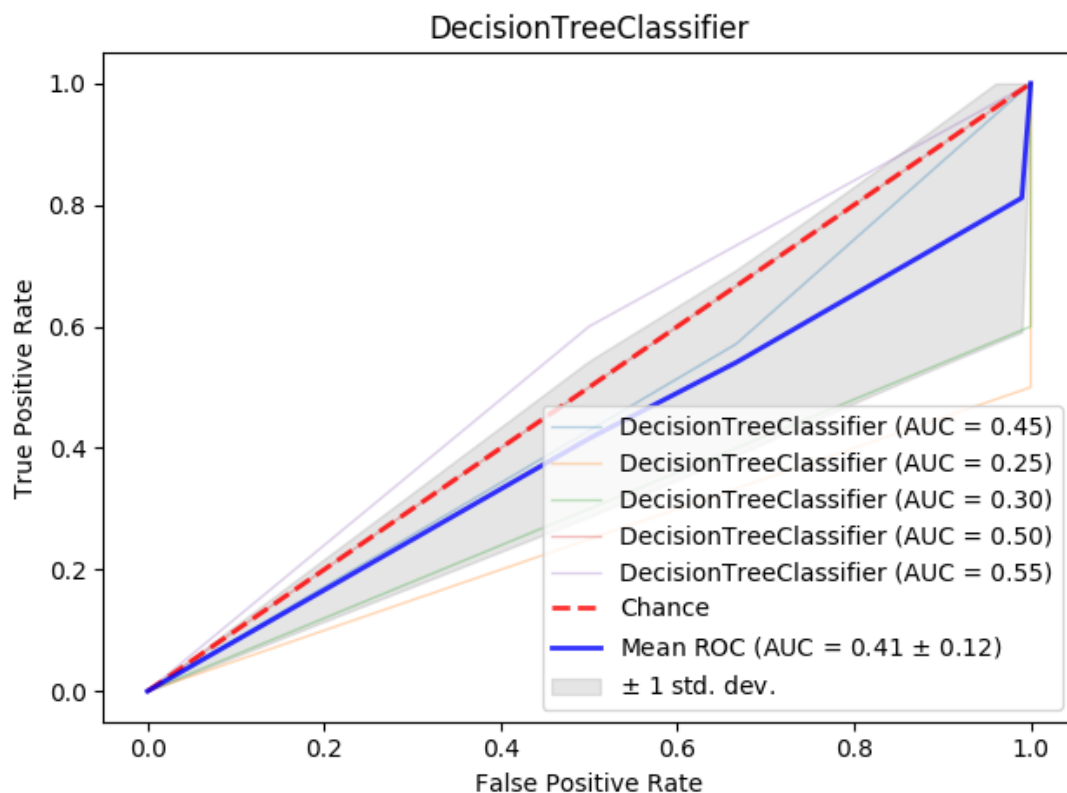
parametry natomiast są ustawiane na początku algorytmu i nie zmieniają się dynamicznie podczas całego procesu. Z faktu, że KNN jest metodą nieparametryczną wynika więc, że model ten nie uczy się wraz z prognozowaniem nowych danych, a za każdym razem tworzony jest nowy model nie mający żadnego pokrewieństwa z wcześniejszym.



Rysunek 6.2: ROC AUC dla KNN.<sup>2</sup>

Przez wymienioną wcześniej interpretowalność wyników rozumie się to, w jaki sposób dany algorytm dotarł do pewnych rezultatów. Jako, że Regresja Logistyczna jest modelowana jako liniowa funkcja, to z łatwością można uzyskać znaczenie danego atrybutu w całym procesie i dowiedzieć się, czy wysoka wartość tego atrybutu wpłynie na pozytywną lub negatywną prognozę (atrybuty posiadające przy sobie wagi negatywne będą sugerowały negatywną prognozę, a wagi pozytywne odwrotnie).

Kolejnym algorytmem posiadającym bardzo dobrą interpretowalność jest drzewo decyzyjne którego wyniki pokazane są na rysunku 6.3. AUC tego

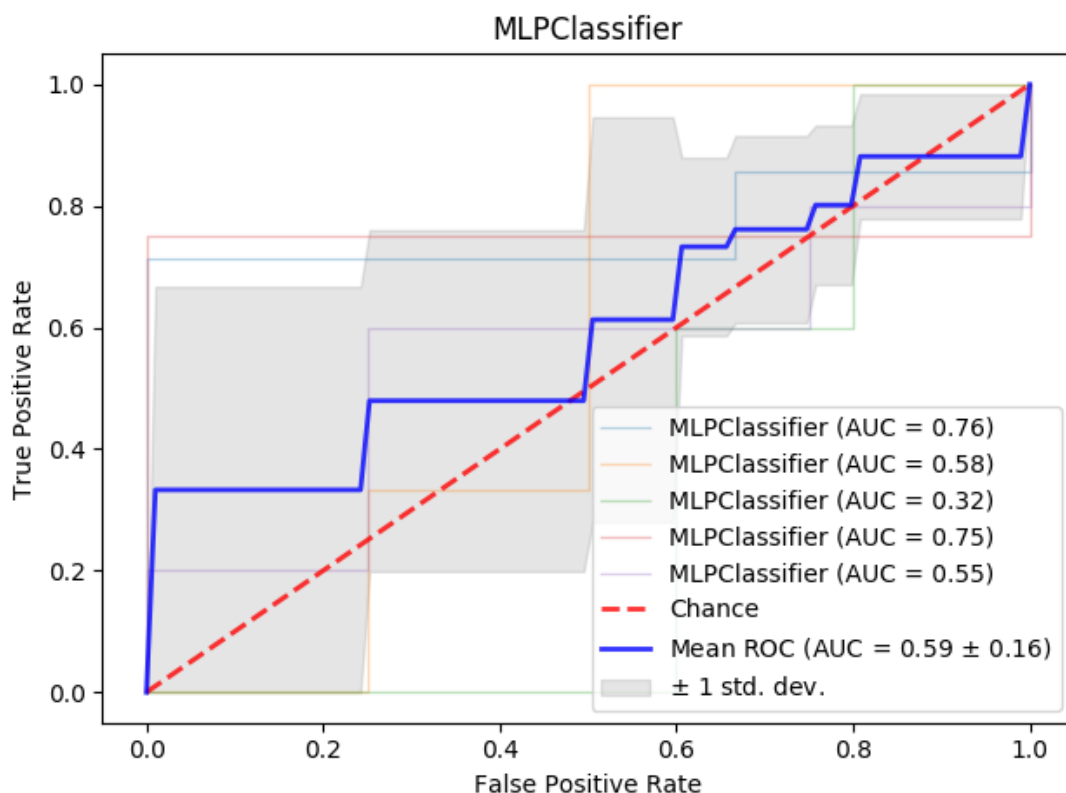


Rysunek 6.3: ROC AUC dla drzewa decyzyjnego.<sup>3</sup>

klasyfikatora wynosi mniej niż 0.5 co oznacza, że model częściej wskazywał klasę **nie** właściwą. Drzewo decyzyjne tak samo jak **KNN** jest modelem nieparametrycznym, który tworzy pewne **regule** za pomocą danych treningowych. Klasyfikator jest tworzony w taki sposób, że atrybuty mające największe znaczenie są bliżej korzenia względem tych, których znacznie jest znikome podczas prognozy klasy wynikowej. Każdy element posiadający rozgałęzienie przedstawia predykat bazujący na atrybucie, a liście drzewa przedstawiają wynik prognozy. Ze względu na to, że atrybuty nie przedstawiają nominalnych danych (na początku była wykonana dyskretyzacja, ale następnie wartości zostały przemnożone przez czas, którego jednostką były sekundy), to jako metoda do wyboru atrybutów na poszczególnych rozgałęzieniach wy-

brany został współczynnik gini (z ang. *Gini index*) przedstawiony wzorem:  $1 - \sum_{k=1}^K \left(\frac{|C_k|}{|D|}\right)^2$ , gdzie  $D$  to liczba wszystkich przykładów, a  $C_k$  to liczba tych przykładów, które należą do klasy  $k$ . Ze wzoru można zauważyć, że czym współczynnik gini jest mniejszy, tym rozgałęzienie jest lepsze. Prawdopodobnie powodem tego, że AUC było tak niskie jest mała ilość danych treningowych.

Ostatnim modelem poddanym eksperymentowi jest MLP widoczny na rysunku 6.4. Model ten uzyskał największą średnią ROCAUC, ale jego interpretacja jest dużo trudniejsza niż w przypadku regresji logistycznej. Sieć neuronowa na ogół składa się z jednej warstwy do wprowadzania danych (z ang. *input layer*), jednej warstwy do wyprowadzania danych (z ang. *output layer*) oraz jednej lub więcej warstw ukrytych (z ang. *hidden layer*).



Rysunek 6.4: ROC AUC dla MLP.<sup>4</sup>

W warstwie ukrytej znajdują się tzw. neurony, które otrzymują dane z warstw poprzednich pomnożone przez pewne wagi ustalane podczas procesu tworzenia sieci, czyli modelu. Neurony po otrzymaniu danych używają z góry zdefiniowanych funkcji aktywacyjnych. Przekształcają one informacje wejściowe i przesyłają je dalej. Cały proces składa się z wielu iteracji, gdzie za jedną iterację uważa się przejście danych przez wszystkie warstwy ukryte aż do warstwy wyjściowej (z ang. *forward propagation*) i etap "uczenia" korygujący wagi pomiędzy warstwami, podczas którego zmiany propagują się od strony wyjściowej do wejściowej (z ang. *back propagation*). Dla sieci użytej w projekcie zostały użyte pewne empirycznie wybrane hiperparametry. Maksymalna ilość iteracji została ustawiona na 1000. Jako funkcję aktywacyjną wybrano "relu" opisany wzorem  $\max(0, x)$ . Rozmiar sieci neuronowej to jedna warstwa, w której zawiera się 25 neuronów.







# Rozdział 7

## Wnioski i napotkane problemy



Analiza danych mogła zostać w pełni przeprowadzona dzięki kompleksowemu API dostarczonemu przez system Android w urządzeniach mobilnych. Przed i w trakcie implementacji pojawiło się jednak wiele wątpliwości i problemów technicznych, które musiały zostać rozwiązane. Jednym z nich było niecodzienne zachowanie natywnego mechanizmu do pobierania historii aktywności. API to nie posiada wbudowanego rozwiązania do powiadomienia o tym, czy dany użytkownik aktualnie zezwolił na pobieranie tych informacji. Mechanizm ten nie zwraca także wyjątku w momencie gdy aplikacja próbuje pobrać dane historyczne nie mając na to pozwolenia, a jedynie pustą listę. Trzeba było więc wykorzystać ten fakt i w momencie gdy zapytanie zwracało pustą listę aktywności zaimplementowano przejście bazujące na mechanizmie intencji do widoku ustawień. W widoku tym użytkownik świadomie musi zaznaczyć prawidłową opcję. Kolejnym problemem związanym z wspomnianym API była pewna niekonsekwencja odnośnie zwracania danych historycznych. Specjalny mechanizm przeznaczony do pobierania listy aktywności bazujący na pewnym przedziale czasowym omijał niektóre elementy z listy i dlatego też pojawiła się potrzeba użycia bardziej imperatywnej funkcjonalności tego API. Działa ona w ten sposób, że można pobrać pewne zdarzenia i przefiltrować je po pewnych flagach (flaga przedstawia rodzaj zdarzenia np. użytkownik wszedł do aplikacji lub "użytkownik wszedł z aplikacji"). W taki też sposób zostało zaimplementowane pobieranie aktywności. Zdarzenia były pobierane, filtrowane i następnie chronologicznie segregowane aby uzyskać ciągły przebieg wszystkich ak-

tywności odbywających się na urządzeniu mobilnym dla konkretnego użytkownika. **T**ym problemem jaki musiał zostać rozwiązany było nieprawidłowe uwierzytelnianie użytkownika podczas cyklicznych prób pobierania historii aktywności i wysyłania ich do serwisu zewnętrznego, który mógł je potem bezpiecznie zapisać w bazie danych. Rozwiązaniem tego problemu było użycie innej metody dostępnej w bibliotece `play-services-auth` do autoryzacji, która działa na podstawie zapisanych w pamięci podręcznej ustawień związanych z wcześniej zalogowanym użytkownikiem. **K**olejnym problemem napotkanym podczas tworzenia aplikacji mobilnej była kwestia niepoprawnie działającej autoryzacji, gdy aplikacja była pobrana z sklepu Play. Podczas instalowania biblioteki zapewniającej uwierzytelnianie dzięki kontu Google wymagane było podanie klucza, który służył do podpisywania programu wynikowego ze względu na bezpieczeństwo. Okazało się, że aplikacja w momencie pojawienia się w sklepie Play jest podpisana jeszcze raz innym kluczem dostępnym z panelu konsoli Play i w tym też momencie moduł odpowiedzialny za autoryzację przestał działać. Rozwiązaniem tego problemu było stworzenie kolejnego klienta dla biblioteki `play-services-auth` i warunkowa zmiana identyfikatora zależnie od tego czy aplikacja była budowana i instalowana przez plik `.apk`, czy też instalowana prosto z sklepu Play. **W**racając do samej analizy danych, a konkretnie klasyfikacji i jej jakości można było się spodziewać dość niskiego wyniku z tego powodu, że zbierane dane pochodzą jedynie z jednego źródła. Aby takie badania były przeprowadzone dokładniej, dany użytkownik mógłby udostępnić również informacje na temat tego ile czasu przeznacza na korzystanie z aplikacji komputerowych oraz stron internetowych. W kolejnym etapie można by było pokusić się o dokładniejszą ocenę samopoczucia i zdrowia użytkownika względem przeznaczonego czasu na tego typu aktywności, np. poprzez krótki test logiczny lub zręcznościowy badający jego skupienie, czy też czas reakcji. Program mógłby również ulec rozwinięciu poprzez dodanie notyfikacji, które bazowałyby na klasyfikacji danych. W tle badawczy by one czy aktualny czas jaki użytkownik przeznacza na daną aplikację może na niego działać z niepożądanym efektem. Analiza i przekształcenie danych poprzez dyskretyzację również mogłoby się odbyć w nieco inny sposób. W tym momencie zachowana jest informacja, o tym ile użytkownik spędził czasu na danej aplikacji w ciągu dnia oraz ile i jak długo używał urządzenia mobilnego o danej porze. Nie jest utrzymany związek między tymi dwoma składowymi, a mogłoby to pozytywnie

wpłynąć na ogólną jakość klasyfikatora.



# Dodatki



# Spis skrótów i symboli

AWS Zbiór gotowych do użycia serwisów udostępnionych przez przedsiębiorstwo Amazon (ang. *Amazon Web Services*)

EC2 Serwis WWW dostarczający skalowalną moc obliczeniową w chmurze obliczeniowej (ang. *Amazon Elastic Compute Cloud* )

API Interfejs programowania aplikacji, interfejs programistyczny aplikacji, API (ang. *application programming interface*)

Node.js Wieloplatformowe środowisko uruchomieniowe o otwartym kodzie do tworzenia aplikacji typu server-side napisanych w języku JavaScript.

JSON (ang. *Java Script Object Notation*) Lekki format wymiany danych komputerowych. JSON jest formatem tekstowym, bazującym na podzbiorze języka JavaScript.

HTTP (ang. *Hypertext Transfer Protocol*) Protokół przesyłania dokumentów hipertekstowych to protokół sieci WWW.

Aktywność (ang. *Activity*) Komponentów systemu Android odpowiedzialny za interakcję z użytkownikiem.

PostgreSQL Jeden z trzech najpopularniejszych otwartych systemów zarządzania relacyjnymi bazami danych.

AUC (ang. *Area under the curve*) Pole powierzchni pod krzywą







# Bibliografia



- [1] *API Usage Stats*  
Dokumentacja Android  
<https://developer.android.com/reference/android/app/usage/UsageStats>
  
- [2] *Publikacja Aplikacji*  
Dokumentacja Android  
<https://developer.android.com/studio/publish>
  
- [3] *API Google Auth*  
Dokumentacja Google  
<https://developers.google.com/android/reference/com/google/android/gms/auth>
  
- [4] *API Alarm*  
Dokumentacja Google  
<https://developers.google.com/android/reference/com/google/android/gms/auth>
  
- [5] *Understanding AUC - ROC Curve*  
Sarang Narkhede, 26 czerwiec 2018  
<https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5>
  
- [6] *Deploying a Node App on Amazon EC2*  
Kunal Yadav, 16 listopad 2018  
<https://hackernoon.com/deploying-a-node-app-on-amazon-ec2-d2fb9a6757eb>
  
- [7] *Opis RescueTime*  
Jory MacKay, 30 sierpień 2018  
<https://blog.rescuetime.com/rescuetime-work-hours/>

- [8] *Opis TimeDoctor*  
TimeDoctor, 30 listopad 2019  
<https://www.timedoctor.com/features.html>
- [9] *Building A Logistic Regression in Python, Step by Step*  
Susan Li, 29 wrzesień 2017  
<https://towardsdatascience.com/building-a-logistic-regression-in-python-step>
- [10] *One Hot Encoding in Python*  
Susan Li, 10 wrzesień 2018  
<https://blog.cambridgespark.com/robust-one-hot-encoding-in-python-3e29bfcec7>
- [11] *TypeORM - TypeScript guide*  
David Herron, 18 lipiec 2019  
<https://levelup.gitconnected.com/complete-guide-to-using-typeorm-and-typescr>
- [12] *Naive Bayes Classifier*  
Rohith Gandhi, 5 maj 2018  
<https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c>

# Zawartość dołączonej płyty

Do pracy dołączona jest płyta CD z następującą zawartością:

- praca (źródła  $\text{\LaTeX}$ owe i końcowa wersja w pdf),
- źródła programu,
- dane testowe.



# Spis rysunków

4.1	Szczegóły aplikacji w Google Play. <sup>1</sup>	8
4.2	Ustawianie zezwoleń do śledzenia historii aktywności. <sup>2</sup>	9
4.3	Autoryzacja poprzez korzystanie z usługi Google Play Services. <sup>3</sup>	9
4.4	Widok oceny dnia. <sup>4</sup>	10
4.5	Widok raportu. <sup>5</sup>	10
4.6	Widok raportu. <sup>6</sup>	11
5.1	Infrastruktura projektu. <sup>7</sup>	14
5.2	Diagram przedstawiający zależności między klasami w aplikacji mobilnej. <sup>8</sup>	18
5.3	Diagram przedstawiający schemat bazy danych. <sup>9</sup>	19
5.4	Diagram przedstawiający zależności między elementami w aplikacji Node.js. <sup>10</sup>	22
5.5	Diagram przedstawiający zależności między elementami w aplikacji napisanej w języku Python. <sup>11</sup>	24
6.1	ROC AUC dla regresji logistycznej. <sup>12</sup>	27
6.2	ROC AUC dla KNN. <sup>13</sup>	28
6.3	ROC AUC dla drzewa decyzyjnego. <sup>14</sup>	29
6.4	ROC AUC dla MLP. <sup>15</sup>	30



# Spis tablic

2.1	Oryginalne dane historyczne . . . . .	4
2.2	Dane historyczne po transformacji . . . . .	4