



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, styczeń 2020

Oświadczenie

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

Gliwice, dnia 9 stycznia 2020

.....
(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 9 stycznia 2020

.....
(podpis promotora)

Spis treści

1	Wstęp i cel pracy	1
2	Analiza tematu	3
3	Wymagania i narzędzia	5
3.1	Wymagania funkcjonalne	5
3.2	Wymagania niefunkcjonalne	7
3.3	Narzędzia	8
4	Specyfikacja zewnętrzna	9
4.1	Wymagania sprzętowe, programowe i instalacja	9
4.2	Scenariusze korzystania z systemu	12
5	Specyfikacja wewnętrzna	17
5.1	Przygotowanie danych	17
5.2	Opis algorytmów klasyfikujących	19
5.3	Struktura projektu	21
5.4	Aplikacja mobilna	22
5.5	Baza Danych	25
5.6	Serwis zarządzający bazą danych i obsługą żądań z aplikacji mobilnych	27
5.7	Serwis odpowiedzialny za analizę danych i generowanie raportów . .	31
6	Weryfikacja i walidacja	35
6.1	Proces testowania serwisów	35
6.2	Porównanie i wybór algorytmu użytego do klasyfikacji	36

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 tj. stres, bieżące wydarzenia, czy też uwarunkowania społeczne. Niniejszy projekt inżynierski skupia się na czynniku technologicznym, który otacza nas wszystkich często nie pozwalając 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 odbywać się wewnątrz zaimplementowanego systemu w ramach pracy dyplomowej. Kolejnym punktem objętym w pracy jest stworzenie serwisów (elementów mogących dzia-

łać w separacji i komunikować się między sobą przez określony protokół), które będą w stanie zbierać uzyskane dane z wielu urządzeń mobilnych, przekształcać je do formatu łatwiejszego do analizy, umożliwiającą klasyfikację. Finalnie, zakres pracy obejmuje również wybór algorytmu do klasyfikacji oraz opis analizy i eksperymentów które zostały przeprowadzone aby wybrać model posiadający wysoką skuteczność.

Niniejsza praca składa się z siedmiu rozdziałów. W kolejnym rozdziale zostanie przedstawiona analiza dotycząca aktualnych rozwiązań opisanego wyżej problemu oraz sposób w jaki dane będą grupowane. Proces grupowania polegał na scaleniu danych w taki sposób, aby umożliwić klasyfikację w stosunku do oceny wystawionej przez użytkownika. W rozdziale 3 zostaną przedstawione wymagania sprzętowe i systemowe, które pozwolą na uruchomienie stworzonej aplikacji mobilnej i narzędzia jakie były wykorzystane w trakcie implementacji wszystkich serwisów. W rozdziale 4 zostaną przedstawione scenariusze i ścieżki z jakimi użytkownik może się spotkać posługując się aplikacją. W rozdziale 5 przekazane zostaną informacje o technologiach i szczegółach implementacji każdego z modułów. W rozdziale 6 opisana będzie szczegółowa analiza tego w jaki sposób aplikacja była testowana oraz w jaki sposób został dobrany model dokonujący klasyfikacji danych. W rozdziale 7 zostaną przedstawione wnioski oraz największe wyzwania i problemy w trakcie tworzenia projektu.

Rozdział 2

Analiza tematu

Na rynku aktualnie istnieją systemy, które pomagają użytkownikowi śledzić jego aktywności zwracając uwagę na to ile czasu spędza on korzystając z pewnej aplikacji. Cechują się one doskonałą analizą i raportami, z których użytkownik może odczytać czas spędzony na danej kategorii, gdzie kategoria to np. gry lub media społecznościowe, oraz odczytać wskaźnik przedstawiający procentową wartość czasu spędzonego „produktywnie”. Wskaźnik ten bazuje często na domyślnych, z góry narzuconych etykietach np. aplikacja o kategorii media społecznościowe będzie posiadała etykietę negatywną, a pewien znany serwis popularnonaukowy etykietę pozytywną. Takie rozwiązanie jest dokładne i pozwala w łatwy, aczkolwiek pracochłonny (etykiety muszą zostać dodane dla każdej aplikacji) sposób udostępnić użytkownikowi informację o tym, w jaki sposób spędził on czas korzystając z danego urządzenia.

Jednym z narzędzi, działającym tak jak opisano powyżej jest RescueTime [7], system wieloplatformowy, który jest złożony z aplikacji mobilnej, desktopowej oraz rozszerzenia do przeglądarki Google Chrome. System ten umożliwia ręczne przypisywanie etykiet, więc jeżeli użytkownik nie zgadza się, że np. aplikacja o kategorii media społecznościowe wpływa na niego niekorzystnie, to zawsze może ustalić inną etykietę.

Inną aplikacją tego typu jest TimeDoctor [9], 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 również posiada wbudowaną analizę danych, opierającą się na podstawowych kategoriach oraz etykietach.

Kolejną aplikacją pomagającą w śledzeniu aktywności jest QualityTime [8]. System ten ze względów technologicznych wydaje się być najbardziej zbliżony do aplikacji wykonywanej w trakcie pracy dyplomowej, ponieważ tak samo skupia się na systemie Android. Oferuje on jednak wiele dodatkowych funkcji np. powiadomienia odnośnie zbyt częstego korzystania z danej aplikacji, mogące świadczyć o uzależnieniu.

Przedstawiona w pracy inżynierskiej aplikacja nie posiada wielu funkcji wcześniej opisanych systemów tj. możliwości ręcznego przyznawania etykiet do danych aplikacji, czy też powiadomień świadczących o nadmiernym korzystaniu z danego programu. Wprowadza natomiast automatyczne przyznawanie etykiet do danych aplikacji dzięki dopasowaniu wyników historycznych do ocen codziennie wystawianych przez użytkownika. Przyznawanie etykiet odbywa się podczas procesu klasyfikacji, który zostanie omówiony w rozdziale 5.

Rozdział 3

Wymagania i narzędzia

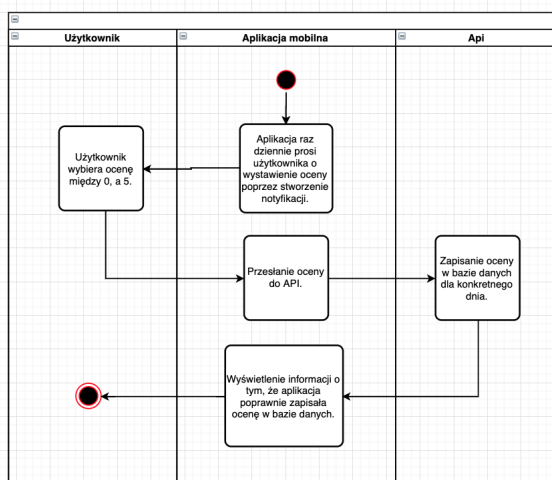
W tym rozdziale znajdują się informacje na temat głównych części pracy inżynierskiej. Czytelnik może się dowiedzieć w jaki sposób aplikacja działa od strony użytkownika, oraz jakie są główne mechanizmy i podmioty zastosowane od strony implementacji. Wyszczególnione są tu również narzędzia jakie zostały wykorzystane podczas pracy nad projektem.

3.1 Wymagania funkcjonalne

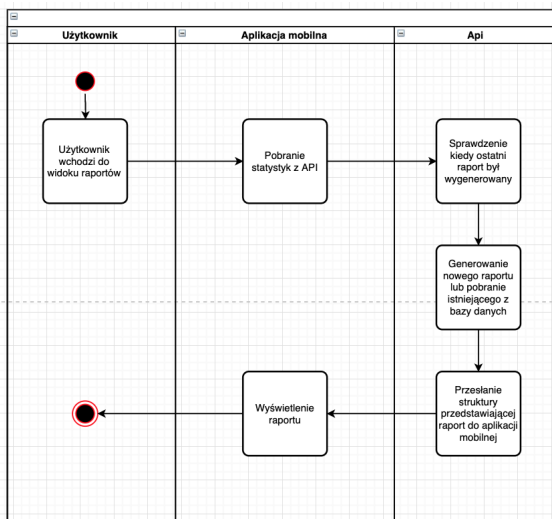
W tej sekcji znajduje się lista funkcjonalności, które bezpośrednio dotyczą użytkownika podczas pracy ze stworzonym projektem. Wszystkie elementy zostały w pełni zaimplementowane w aplikacji mobilnej.

- Użytkownik autoryzuje się w aplikacji mobilnej poprzez wybranie konta Google.
- Użytkownik ma możliwość przejrzania w aplikacji historii wszystkich aktywności sprzed ostatnich dwudziestu czterech godzin.
- Użytkownik otrzymuje każdego dnia powiadomienie z możliwością oceny aktualnego dnia. Proces ten widoczny jest na rysunku 3.1.
- Użytkownik ma możliwość przejrzania raportu informującego które aplikacje wpływają negatywnie, a które pozytywnie. Proces ten widoczny jest na rysunku 3.2.

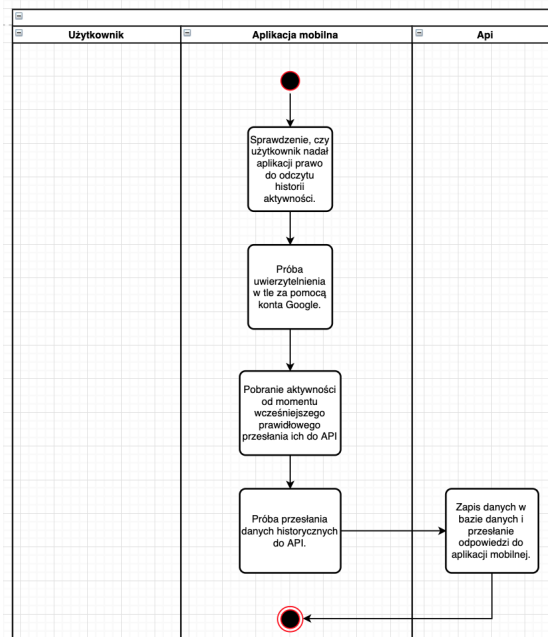
- Program w tle cyklicznie przesyła dane na temat czasu jaki użytkownik spędził na aplikacjach do serwisu zewnętrznego bezpośrednio podłączonego do bazy danych. Proces ten widoczny jest na rysunku 3.3.



Rysunek 3.1: Diagram przedstawiający codzienny proces „oceny dnia” przez użytkownika.



Rysunek 3.2: Diagram przedstawiający proces odczytywania raportu w aplikacji mobilnej.



Rysunek 3.3: Diagram przedstawiający cykliczny proces przesyłania danych historyczny do serwisu zewnętrznego.

3.2 Wymagania niefunkcjonalne

Sekcja ta przedstawia te elementy z których składa się projekt, a nie są widoczne z perspektywy użytkownika.

- 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.

- Wybór oraz użycie algorytmu uczenia maszynowego w celu interpretacji historii aktywności.

3.3 Narzędzia

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 AWS (z ang. *Amazon Web Services*).

Rozdział 4

Specyfikacja zewnętrzna

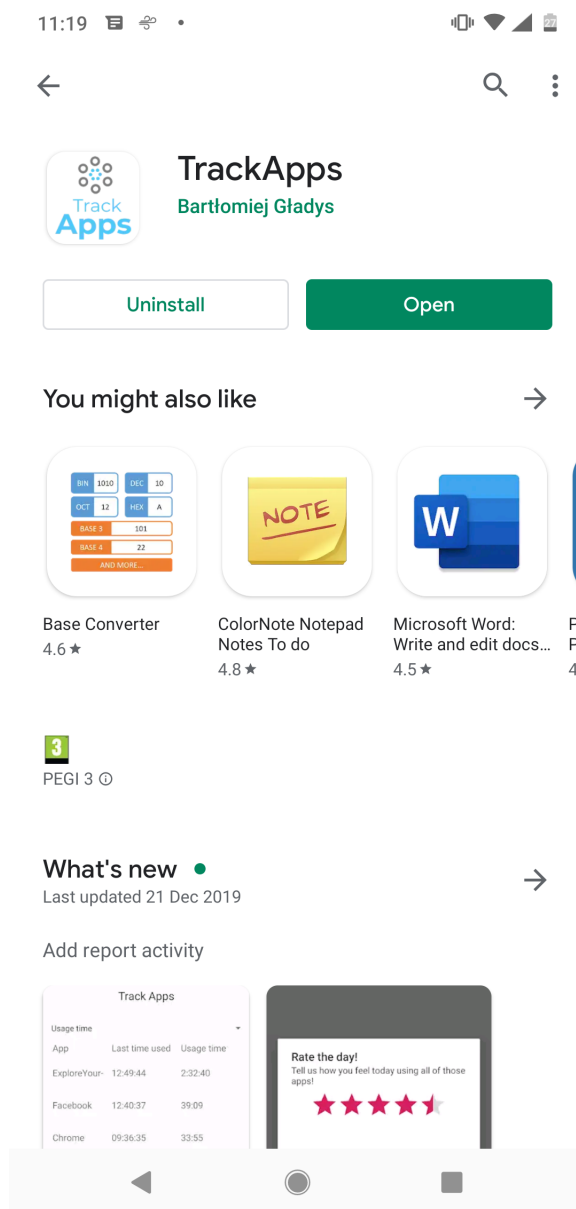
Rozdział ten przedstawia podstawowe wymagania niezbędne do uruchomienia aplikacji mobilnej stworzonej podczas pracy inżynierskiej. Znajduje się tu również informacja o wszystkich krokach niezbędnych do instalacji i uruchomienia tej aplikacji. Użytkownik może też zapoznać się z tym w jaki sposób korzystać z systemu wybierając jeden z scenariuszy.

4.1 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, którego szczegóły widoczne są na rysunku 4.1.

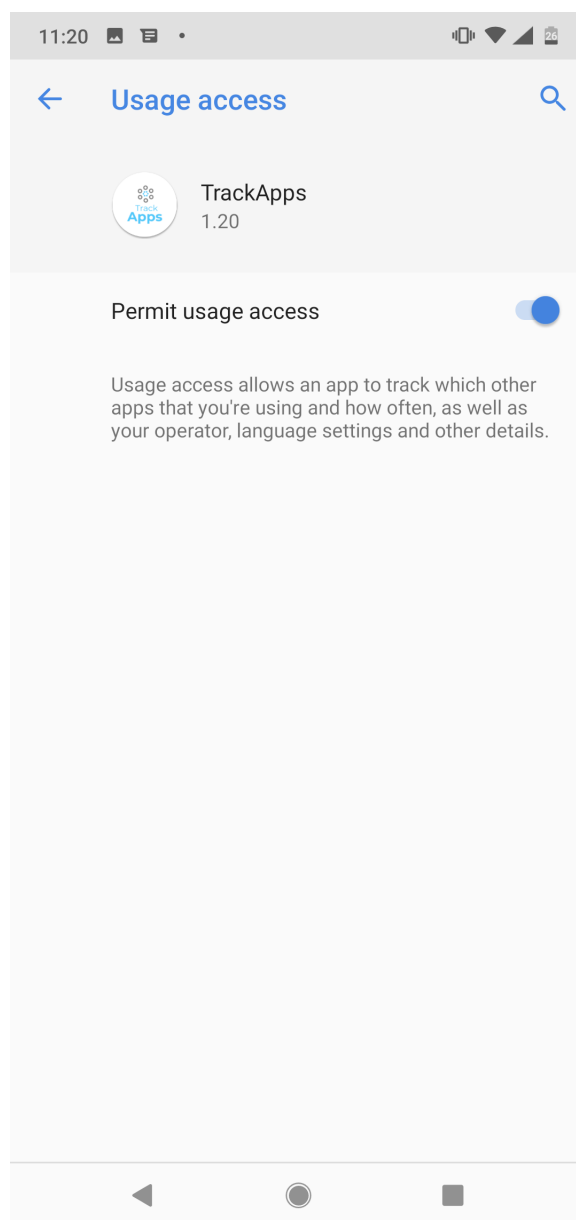
Po pomyślnej instalacji użytkownik jest przekierowany do widoku zezwoleń, w którym świadomie musi ustawić aplikacji mobilnej prawo pobierania historii aktywności. Widok procesu znajduje się na przedstawionym rysunku 4.2.

Następnie wyświetlona zostaje informacja o wymaganej autoryzacji poprzez wybranie konta Google prezentująca się tak jak na rysunku 4.3. Jeżeli użytkownik pozytywnie przejdzie przez wymienione wyżej dwa etapy, na ekranie pojawi się



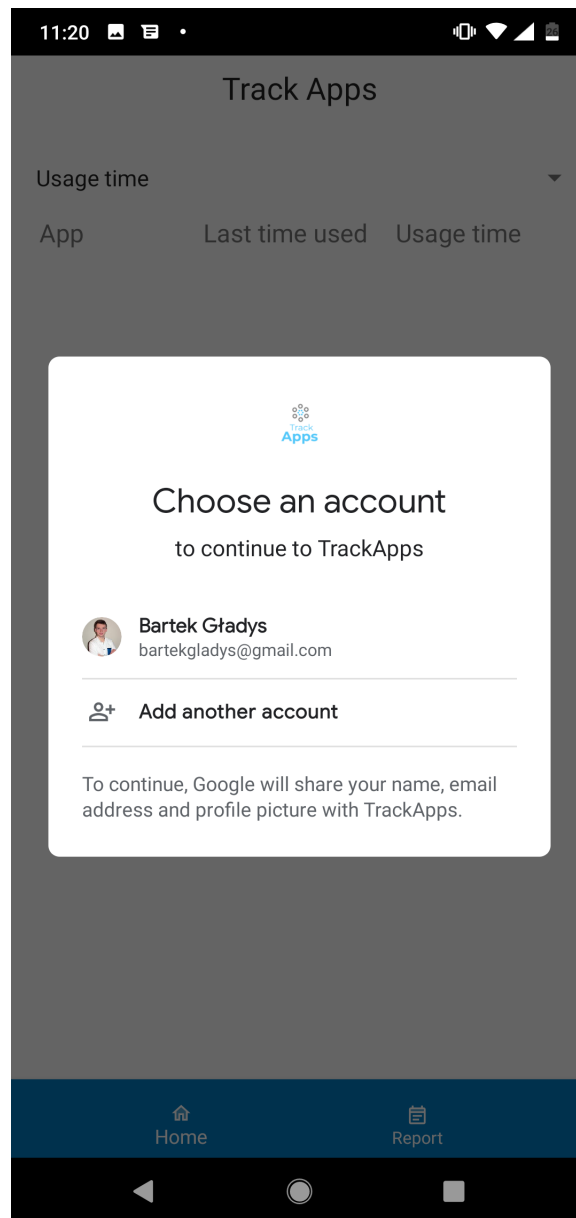
Rysunek 4.1: Szczegóły aplikacji wykonanej w ramach pracy inżynierskiej w Google Play.

lista z ostatnio odwiedzonymi 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



Rysunek 4.2: Ustawianie zezwoleń do śledzenia historii aktywności dla aplikacji wykonanej w ramach pracy inżynierskiej.

cechą autoryzacji w ten sposób jest to, że nawet w przypadku gdy użytkownik zmieni urządzenie mobilne to jego wszystkie wpisy historyczne i oceny będą zapisane wewnątrz systemu.

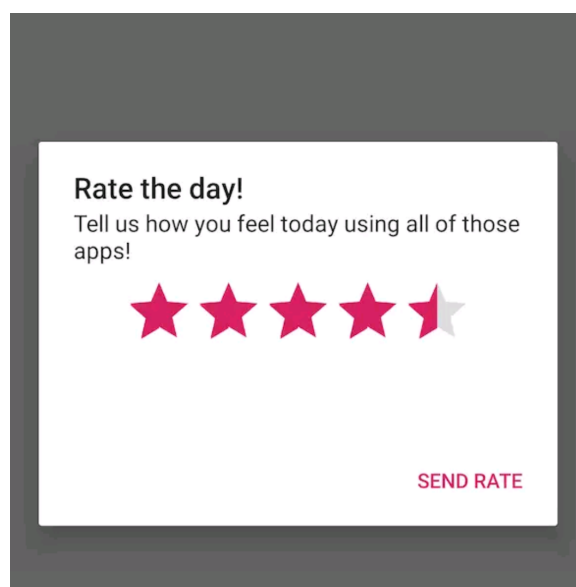


Rysunek 4.3: Autoryzacja poprzez korzystanie z usługi Google Play Services wewnątrz aplikacji wykonanej w ramach pracy inżynierskiej.

4.2 Scenariusze korzystania z systemu

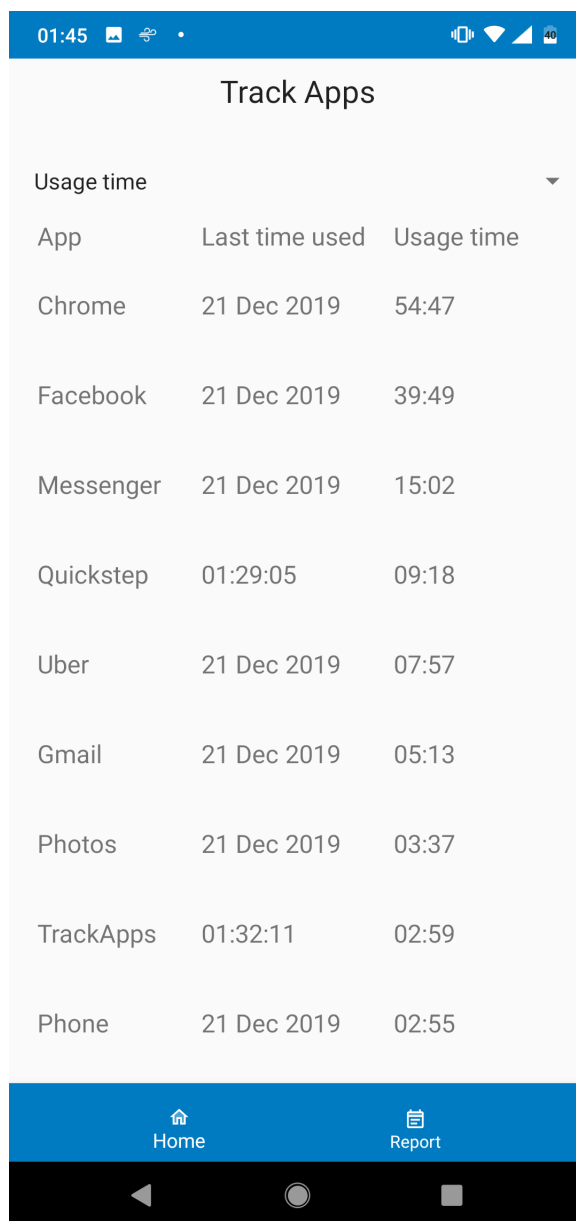
Jedyną aktywnością wymaganą do poprawnego zbierania i klasyfikacji danych jest wpisywanie przez użytkownika oceny samopoczucia w zakresie od 0 do 5.

Proces ten jest widoczny na przedstawionym rysunku 4.4.



Rysunek 4.4: Widok oceny dnia w aplikacji wykonanej w ramach pracy inżynierskiej.

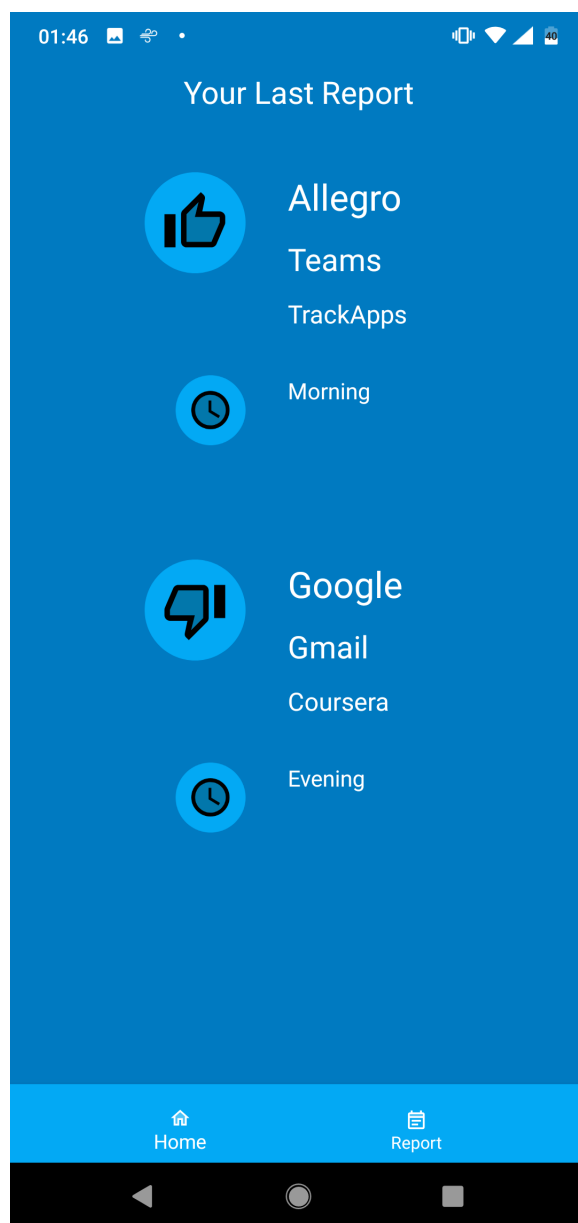
Widok ten jest wyświetlony cyklicznie – raz każdego dnia i nie można go „wywołać” ręcznie poprzez nawigację w programie. Aplikacja posiada dwa standardowe widoki po jakich użytkownik może się poruszać. Pierwszym z nich jest widoczna na zrzucie ekranu 4.5 lista, która może być sortowana po trzech atrybutach: nazwie programu, spędzonym na aplikacji czasie oraz 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 najmniejsze. Dodatkowo, widnieje tam informacja o tym, jaka pora dnia jest najodpowiedniejsza do korzystania z urządzenia mobilnego oraz przeciwnie o jakich godzinach używanie telefonu wpływa na użytkownika negatywnie.



The screenshot shows a mobile application titled 'Track Apps'. At the top, there is a status bar with the time 01:45 and various icons. Below the title, there is a dropdown menu currently set to 'Usage time'. The main content is a table with three columns: 'App', 'Last time used', and 'Usage time'. The table lists several applications with their last usage date and time, and the total usage time. At the bottom, there is a navigation bar with two buttons: 'Home' and 'Report'.

Track Apps		
Usage time ▼		
App	Last time used	Usage time
Chrome	21 Dec 2019	54:47
Facebook	21 Dec 2019	39:49
Messenger	21 Dec 2019	15:02
Quickstep	01:29:05	09:18
Uber	21 Dec 2019	07:57
Gmail	21 Dec 2019	05:13
Photos	21 Dec 2019	03:37
TrackApps	01:32:11	02:59
Phone	21 Dec 2019	02:55

Rysunek 4.5: Widok aktywności odbywających się w ostatnich dwudziestu czterech godzinach w aplikacji wykonanej w ramach pracy inżynierskiej. Znajdująca się tu lista pokazuje nazwę aplikacji, ostatni punkt w czasie kiedy użytkownik z niej korzystał, oraz sumę czasu, jaki użytkownik poświęcił danej aplikacji.



Rysunek 4.6: Widok raportu w aplikacji wykonanej w ramach pracy inżynierskiej. Na początku przedstawione są aplikacje, które wpływają pozytywnie na samopoczucie użytkownika oraz pora dnia w której posługiwanie się urządzeniem mobilnym wpływa korzystnie. Analogicznie, poniżej przedstawione są aplikacje które wpływają negatywnie na samopoczucie i pora dnia w ciągu której posługiwanie się urządzeniem mobilnym wpływa niekorzystnie.

Rozdział 5

Specyfikacja wewnętrzna

W tym rozdziale zostaną przedstawione wszystkie elementy które nie są widoczne dla użytkownika, a spełniają fundamentalną rolę w kontekście poprawnego działania całego systemu zbudowanego w ramach pracy inżynierskiej. W pierwszym etapie opisana będzie transformacja historycznych danych aktywności z postaci natywnej dla systemu Android do postaci umożliwiającej klasyfikację. Następnie, w sekcji 5.2 opisane zostaną algorytmy które były brane pod uwagę podczas wyboru modelu do klasyfikacji. W sekcji 5.3 znajdą się informacje na temat struktury projektu i tego jak dane serwisy i aplikacja mobilna są ze sobą połączone. W sekcji 5.4 zostaną opisane szczegóły implementacji aplikacji mobilnej. W sekcji 5.5 będzie przedstawiona struktura bazy danych. Następnie, w sekcji 5.6 pojawią się informacje na temat serwisu odbierającego dane z urządzeń mobilnych. Finalnie, w sekcji 5.7 zostaną przedstawione szczegóły dotyczące serwisu zajmującego się analizą danych i tworzeniem raportów.

5.1 Przygotowanie danych

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 ak-

Tablica 5.1: Przykładowe dane historyczne pochodzące z Usage Stats API w systemie Android

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

tywność z godziny 12:00 do klasy „Południe”. Dzięki takiej zamianie można było w pewien sposób przyspieszyć algorytm oraz uzyskać wyniki lepsze w kontekście interpretacji. Następną kwestią było zagadnienie dotyczące klasyfikacji wszystkich aktywności z jednego dnia do jednego rekordu posiadającego ocenę w tymże dniu. Z tego względu oryginalne dane widoczne w tabeli 5.1 zostały przekształcone do wektora danych, który opisuje sumę czasu w sekundach dla każdej aplikacji oraz każdej pory dnia i jest widoczny w tabeli 5.2. Dodatkowo widać, że w procesie transformacji danych użyto kodowanie „1 z n” [11] zarówno dla nominalnych danych odnośnie pory dnia jak i nazw aplikacji. W ten właśnie sposób było możliwe przygotowanie danych historycznych w jednym wierszu i przypisanie 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. Przykładowo, użytkownik wystawił oceny 1, 2, 3, 4, 5, średnia tych ocen wynosi 3, więc w momencie gdy użytkownik przy kolejnej próbie wystawienia oceny wybierze 4, to model przypisze do zbioru aktywności z tego dnia klasę pozytywną. Gdyby jednak użytkownik wybrał ocenę 2, to model przypisał by do danego zbioru klasę negatywną. Podczas początkowej fazy projektu próbowano ten sam problem rozwiązać z użyciem regresji, a więc każdy unikalny zbiór aktywności posiadał by przypisaną ocenę. Niestety, błąd treningowy był na tyle duży, że postanowiono zrezygnować z regresji i jako model wybrano klasyfikator dokonujący wyboru między dwoma etykietami. Więcej informacji na temat działania klasyfikatora regresji logistycznej i innych algorytmów, które były wzięte pod uwagę, znajdują się w następnej sekcji.

Tablica 5.2: Przykładowe dane historyczne po transformacji z postaci oryginalnej dla Usage Stats API w systemie Android do postaci umożliwiającej klasyfikację

time_in_sec	youtube	facebook	day_part_night	day_part_afternoon
28252.161	19391.801	8860.36	19320.018	8932.143

5.2 Opis algorytmów klasyfikujących

W tej sekcji zostaną opisane wszystkie z czterech algorytmów, które zostały przetestowane pod kątem jakości uzyskanego modelu. Tymi algorytmami są: regresja logistyczna, k -NN (z ang. *k nearest neighbours* – k najbliższych sąsiadów), drzewo decyzyjne oraz MLP (z ang. *Multilayer Perceptron* – najpopularniejszy typ sztucznej sieci neuronowej). Więcej informacji na temat samego procesu testowania i walidacji znajduje się w rozdziale 6.

Pierwszy z nich, 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 $b_0 + b_1 \cdot x_1 + b_2 \cdot x_2 + \dots + b_n \cdot x_n$, natomiast b opisuje wagi dla poszczególnych wartości x , gdzie x_i to czas wyrażony w sekundach dla pewnej aplikacji i . 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 \mathbf{x} , rezultatem będzie właśnie ta składowa.

Kolejny algorytm – k -NN [15] 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 liczbę 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-

perparametry natomiast są ustawiane na początku algorytmu i nie zmieniają się dynamicznie podczas całego procesu. Z uwagi na fakt, że k -NN jest metodą nieparametryczną wynika więc, że model ten nie uczy się podczas etapu prognozy, a jest jedynie korygowany za każdym razem gdy dostarczone są nowe dane.

Drzewo decyzyjne tak samo jak k -NN jest modelem nieparametrycznym. Kłasyfikator 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 wybrany został współczynnik Giniego [14] (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 Giniego jest mniejszy, tym rozgałęzienie jest lepsze. Prawdopodobnie powodem tego, że AUC było tak niskie jest mała ilość danych treningowych.

MLP [16], czyli 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*). 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 liczba iteracji została ustawiona na 1000. Jako funkcje aktywacyjną wybrano $\max(0, x)$, często opisaną jako relu (z ang. *rectified linear unit*). Rozmiar sieci neuronowej to jedna warstwa, w której zawiera

się 25 neuronów.

Na początku projektu, planowano również zastosować naiwny klasyfikator bayesowski [20], ale jako że algorytm ten w podstawowej wersji wspiera tylko dane o postaci „Prawda/Fałsz”, to odstąpiono od tej decyzji. Wybór algorytmów poddanych eksperymentowi opierał się o popularność w kontekście algorytmów klasyfikacyjnych. Ze względu na to, że wszystkie dane były tworzone i zbierane podczas pracy inżynierskiej, to nie było ich na tyle dużo, aby czas trwania danych algorytmów klasyfikacyjnych był brany pod uwagę podczas wybierania konkretnego modelu.

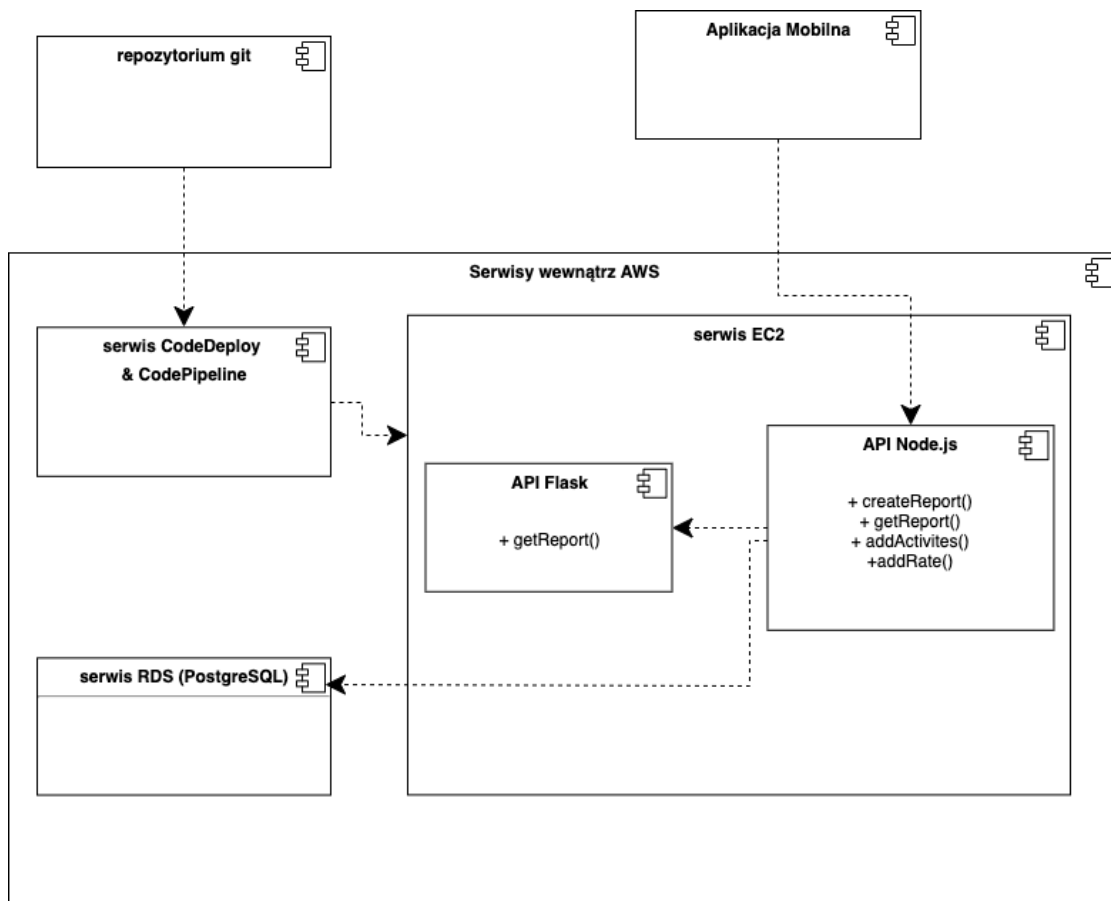
5.3 Struktura projektu

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, oraz aplikacja analizująca dane i generująca raporty napisana w języku Python. Na diagramie 5.1 przedstawiono komunikację pomiędzy poszczególnymi elementami systemu. Natomiast, w tabeli 5.3 znajdują się wszystkie biblioteki wykorzystane bezpośrednio podczas implementacji.

Wszystkie komponenty poza aplikacją mobilną znajdują się wewnątrz AWS. Aplikacje bezpośrednio komunikujące się z bazą danych są uruchomione wewnątrz klastra EC2 (z ang. *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 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 ściąganie zależności, zainstalowanie pa-

kietów 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.

5.4 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 pomocnicze istniejące wewnątrz klasy *UsageStatsComparator* rozszerzające funkcjonalność generycznej klasy *Comparator*. Klasa ta również implementuje metodę *getView*, która pobiera 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* – flaga mówiąca o tym, że aplikacja pojawiła się na głównym planie urządzenia mobilnego, *MoveToBackground* – flaga mówiąca o tym, że dana aplikacja została zminimalizowana. 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 uruchamia aplikację po raz pierwszy to pojawi się modalne okno dialogowe 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 uwierzytel-

nienia został napisany w klasie *ApiHandler* 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 również informacja o tym jaka pora dnia powinna być bardziej sprzyjająca w kontekście użytkowania 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 dodania dodatkowych funkcjonalności do każdego żądania. Wykorzystując ten wzorec można w łatwy sposób „udekorować” pewną metodę w dodatkową funkcjonalność, która zostanie wykonana przed lub po wywołaniu. 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 samopoczucia. Do komunikacji z serwisem zewnętrznym w celu przesłania wybranej przez użytkownika oceny służy po raz kolejny klasa *ApiHandler* oraz metoda *rate*. Codzienne wywoływanie 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

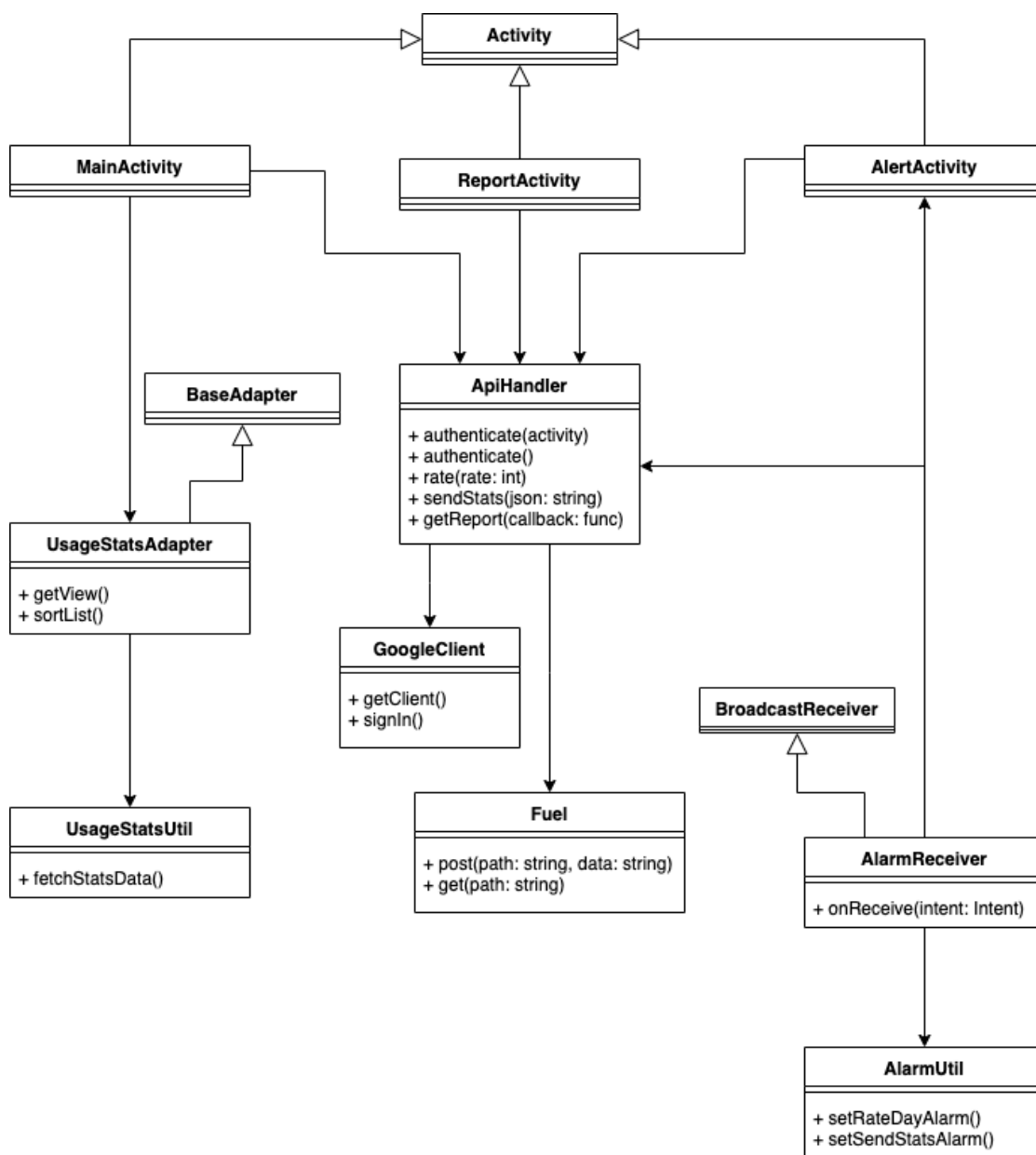
o wybranej godzinie uruchomić wybraną funkcję. Implementacja tego rozwiązania opiera się o nadpisanie bazowej metody *onReceive* dodając funkcjonalność, która nasłuchuje na dany typ intencji. Jeśli takowa intencja się pojawi, zostanie stworzona nowa, do której przypisana będzie flaga informująca o tym, 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 użytkowania 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*. Metody te są wykorzystywane w celu autoryzacji z serwisem zewnętrznym gdy aplikacja działa w tle oraz w celu przesłania historycznych aktywności. 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.

5.5 Baza Danych

Silnikiem bazodanowym, który został wybrany jest już wcześniej napomniany PostgreSQL. Głównym atutem tego narzędzia jest ogromna liczba dostępnych typów danych i funkcji pomagających w pewien sposób te dane konwertować. Na diagramie 5.3 widać podział bazy danych 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.



Rysunek 5.2: Diagram przedstawiający zależności między klasami w aplikacji mobilnej.

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 [19] (JavaScript Object Notation), czyli

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 całego 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. Niestety, relacja posiadająca atrybut w tej formie nie spełnia pierwszej postaci normalnej. Z uwagi na fakt, że atrybut report jest jednostką zagnieżdżoną, encja user posiada nieelementarne wartości.

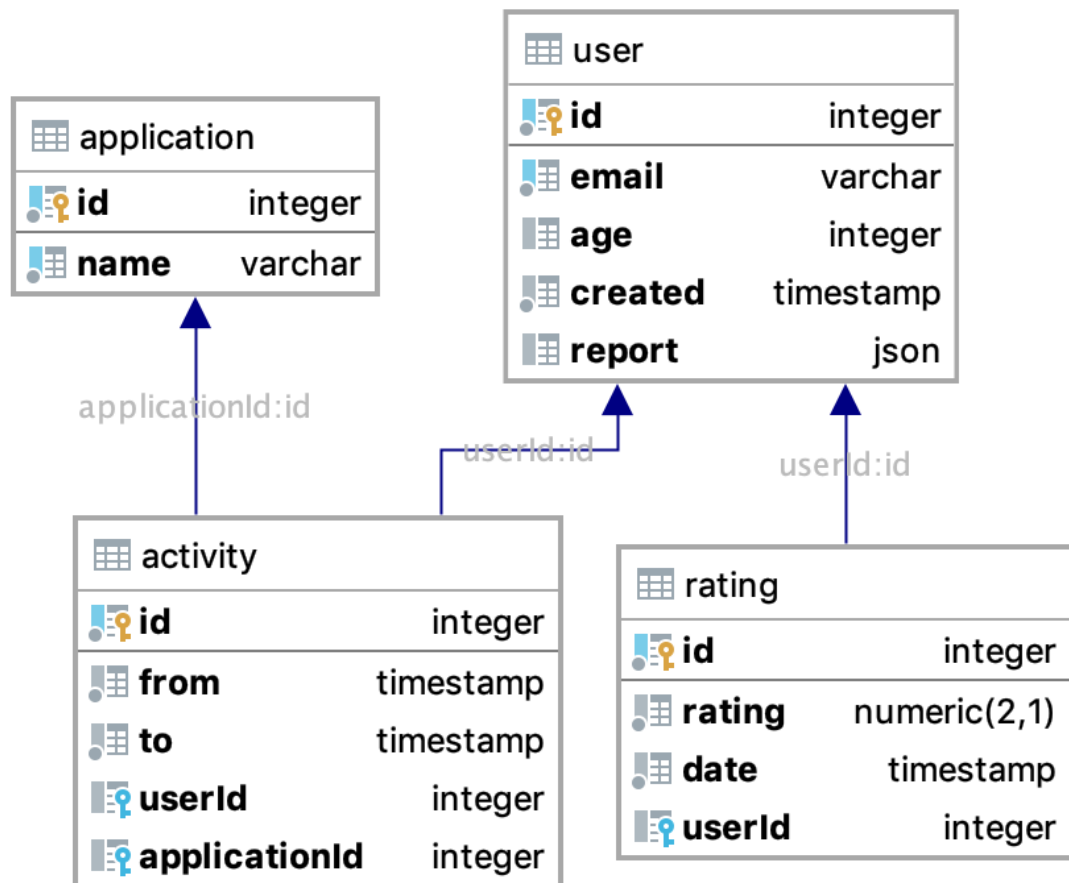
Tabela rating posiada numeryczne pole rating przyjmujące wartości między 0, a 5. Posiada również pole date w celu zapisania daty kiedy dana ocena została zapisana. Znajduje się tu również klucz obcy tabeli user, dzięki któremu można przypisać ocenę do danego użytkownika.

Pomiędzy tabelami user i application, na podstawie tabeli activity, jest stworzona relacja „wiele do wielu“. Poza kluczami obcymi wymaganych relacji, występują tu również pola from i to, które bezpośrednio informują o tym, w jakich godzinach dana aktywność występowała.

5.6 Serwis 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 implementacje zgodnie domeną problemu. Jedyną wadą wspomaganą 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 [12] i jak nazwa wskazuje zapewnia ona mapowanie



Rysunek 5.3: Diagram przedstawiający schemat bazy danych.

obiektoowo-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 inny 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 granice między warstwą przechowującą dane, a

logiką domenową, dlatego też podczas implementacji użyto wzorca Repozytorium. Jest on bardzo dobrze wspierany przez typeorm i umożliwia separację między wymienionymi powyżej warstwami, poprzez tworzenie tzw. repozytorium, osobno dla każdego obiektu domenowego w projekcie.

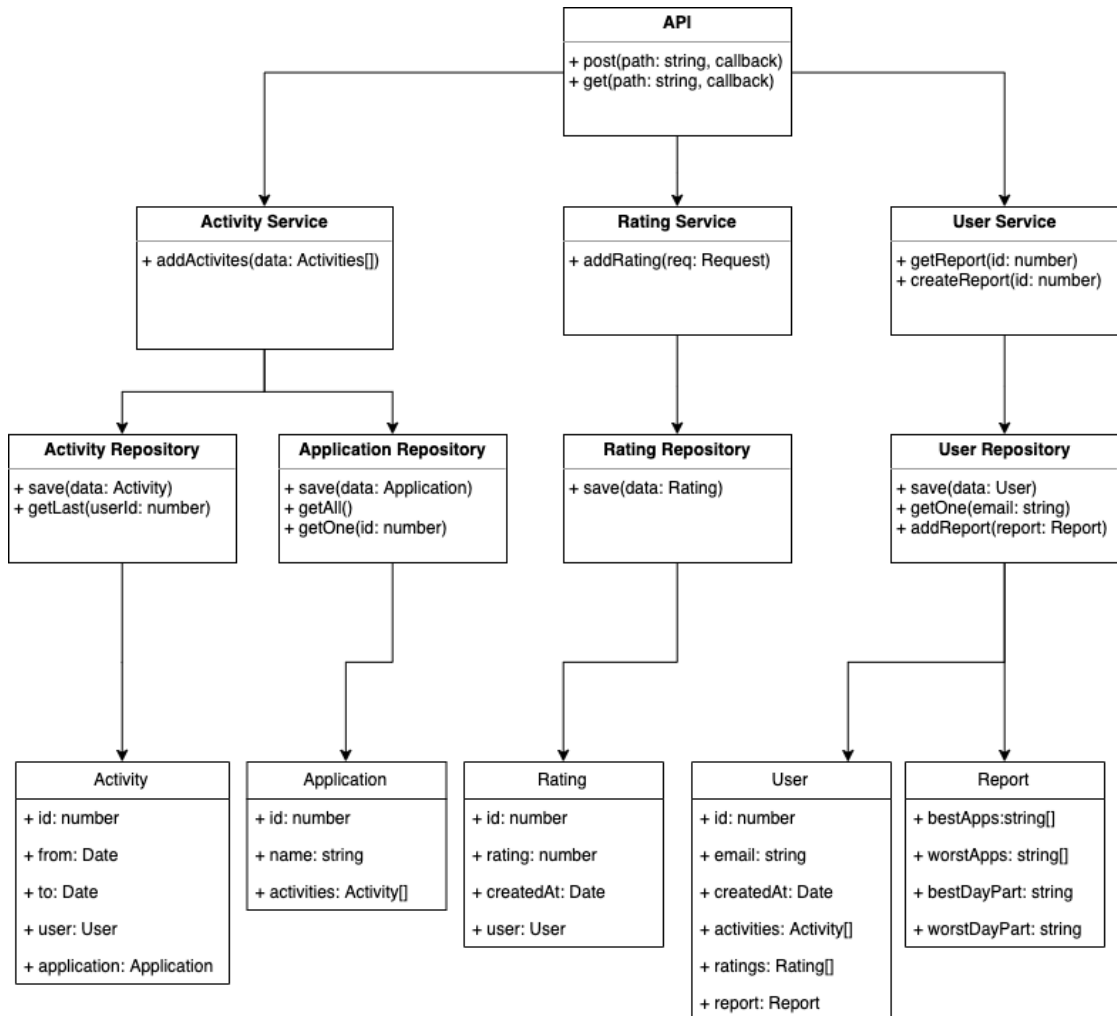
Inną kluczową biblioteką jest google-auth-library, która konwertuje i weryfikuje nagłówki przychodzących żądań z aplikacji mobilnych dzięki czemu możliwe jest przypisanie użytkownika do danych historycznych. Na samym końcu, przy samej obsłudze wymiany informacji z klientami mobilnymi oraz innymi serwisami 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.

Kolejna 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 axios w celu utworzenia żądania HTTP i wysłania go do systemu zajmującego się przetworzeniem

danych. Jako rezultat operacji zwrócony zostanie model, będący w stanie przedstawić aplikacje wpływające na użytkownika pozytywnie i negatywnie.



Rysunek 5.4: Diagram przedstawiający zależności między elementami w aplikacji Node.js.

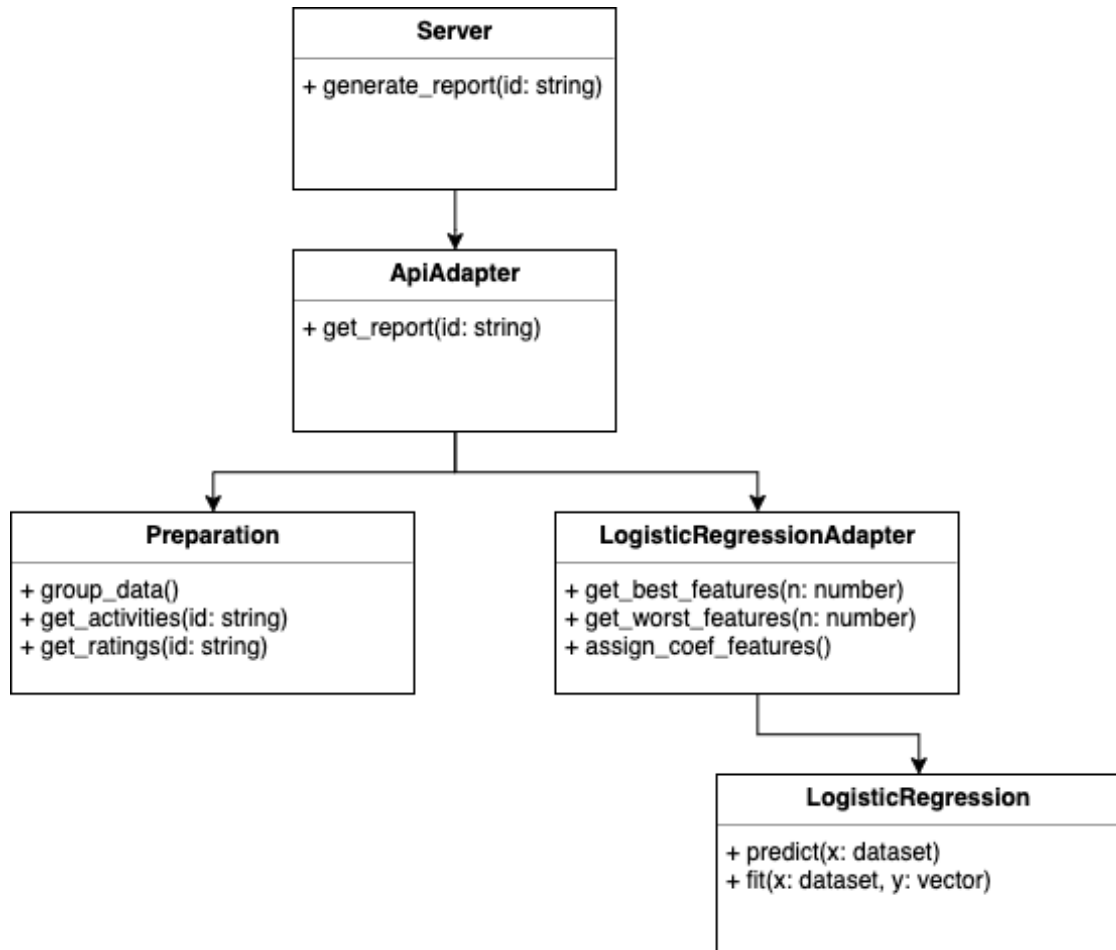
5.7 Serwis odpowiedzialny za analizę danych i generowanie raportów

Aplikacja została napisana w języku Python ze względu na dużą liczbę gotowych do użycia narzędzi wspomagających transformacje 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 systemu klasyfikacyjnego, który będzie mógł przetworzyć statystyki na predykcję i odpowiednio je zinterpretować, wykorzystana została biblioteka *sklearn*. Aby umożliwić połączonemu z aplikacjami mobilnymi serwisowi łatwy dostęp do tworzenia raportów, użyto również bibliotekę *Flask* w celu udostępnienia interfejsu będącego w stanie przyjąć 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* [10]. W trakcie wywołania metody *predict* i *fit*, algorytm przyporządkuje do każdego atrybutu (w tym kontekście atrybut oznacza pewną aplikację, która występuje

w historii aktywności użytkownika) specjalne wagi, które są mnożone przez ilość sekund poświęconych na daną aplikację. Wagi te zmieniają się dynamicznie w momencie gdy do systemu dodana jest nowa ocena i określają czy dana aplikacja działa pozytywnie, czy też negatywnie na samopoczucie docelowego użytkownika.



Rysunek 5.5: Diagram przedstawiający zależności między elementami w aplikacji napisanej w języku Python.

Tablica 5.3: Tabela przedstawiająca wszystkie biblioteki które zostały bezpośrednio użyte podczas implementacji.

Nazwa Biblioteki	Opis Biblioteki
Aplikacja Mobilna (Android)	
fuel	Biblioteka ułatwiająca tworzenie żądań HTTP.
google-play-services	Biblioteka połączona z serwisem Google Auth umożliwiającą uwierzytelnienie poprzez użycie konta Google.
gson	Biblioteka ułatwiająca transformacje danych do postaci JSON i z postaci JSON do natywnych obiektów.
Serwis zarządzający bazą danych i obsługą żądań z aplikacji mobilnych (Node.js)	
typeorm	Biblioteka umożliwiająca stosowanie mapowanie obiektowo-relacyjnego w bazach danych.
express	Biblioteka ułatwiająca obsługę żądań HTTP przychodzących do serwisu.
axios	Biblioteka ułatwiająca tworzenie żądań HTTP.
google-auth	Biblioteka umożliwiająca walidację nagłówków żądań HTTP.
Serwis odpowiedzialny za analizę danych i generowanie raportów (Python)	
numpy	Biblioteka dostarczająca wiele funkcji pomocniczych, które ułatwiają operacje na wektorach i macierzach.
pandas	Biblioteka pozwalająca w łatwy sposób operować na zbiorach danych.
flask	Biblioteka ułatwiająca obsługę żądań HTTP przychodzących do serwisu.
sklearn	Biblioteka dostarczająca gotowe do użycia algorytmy z zakresu uczenia maszynowego.
psycopg2	Biblioteka pozwalająca połączyć się z bazą danych o silniku PostgreSQL.

Rozdział 6

Weryfikacja i walidacja

Rozdział ten opisuje sposoby weryfikacji i walidacji stworzonego projektu. W pierwszej sekcji opisany został proces testowania i praktyki umożliwiające łatwiejsze budowanie serwisów oraz połączeń między nimi. W kolejnej sekcji przedstawiona została analiza pozwalająca wybrać najbardziej wydajny algorytm klasyfikujący.

6.1 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 RDS (z ang. *Amazon Relational Database Service*), 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 CI/CD [17] (ang. *Continuous Integration / 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.

6.2 Porównanie i wybór algorytmu użytego do klasyfikacji

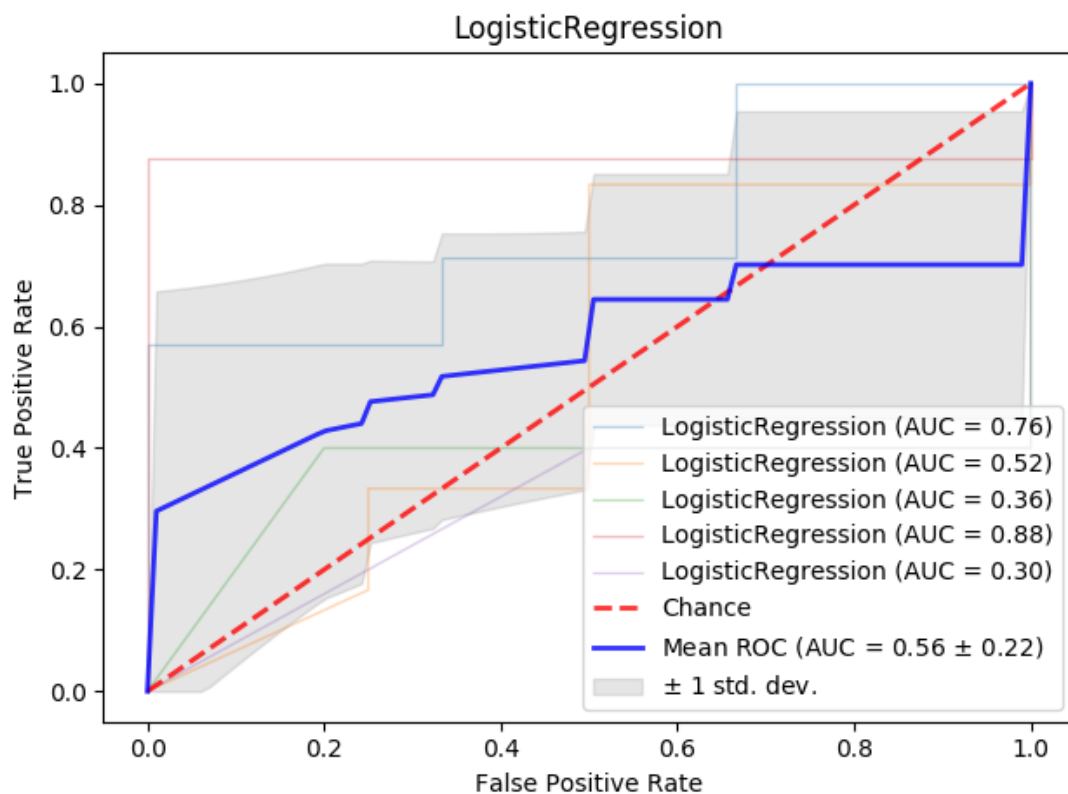
Biorąc pod uwagę ilość danych w innych eksperymentach z wykorzystaniem uczenia maszynowego można stwierdzić, że w tym przypadku danych było stosunkowo mało (w ciągu jednego miesiąca dla jednego użytkownika można było uzyskać maksymalnie 30 ocen, czyli 30 wierszy w bazie danych).

W celu wytrenowania modelu oraz sprawdzenia jaki klasyfikator sprawdza się najlepiej użyto k -krotnej walidacji krzyżowej dla $k = 5$. Podczas tego procesu zbiór danych jest podzielony na k równych co do wielkości podzbiorów, gdzie każdy z nich raz pełni rolę podzbioru testowego i $k - 1$ razy rolę podzbioru treningowego. Ze względu na to, że nie należy się spodziewać zbalansowanych danych (czasem może być więcej przykładów pozytywnych, oznaczających wysokie samopoczucie, a czasem więcej negatywnych, oznaczających niskie samopoczucie), nie został użyty taki wskaźnik jak trafność (z ang. *accuracy*). Miary tej nie stosuje się w przypadku zbiorów niezbalansowanych w problemach klasyfikacyjnych, ponieważ klasyfikacja wszystkich przykładów do jednej klasy da i tak wysoką wartość tej miary. 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* – k najbliższych sąsiadów), drzewo decyzyjne oraz MLP (z ang. *Multilayer Perceptron* – najpopularniejszy typ sztucznej sieci neuronowej). Szczegółowe informacje o każdym z algorytmów oraz parametry jakie zostały dla nich zastosowane przedstawiono w rozdziale 5.

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. Jeżeli wynik modelu rozstrzygającego problem dwuklasowy znajduje się w pobliżu tej linii, oznacza to, że skuteczność danego klasyfikatora jest bardzo zła i równie dobrze

zachował by się klasyfikator działający na podstawie losowania między dwoma klasami. 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 wysoką wartość średniej AUC względem innych algorytmów oraz łatwość interpretacji modelu.

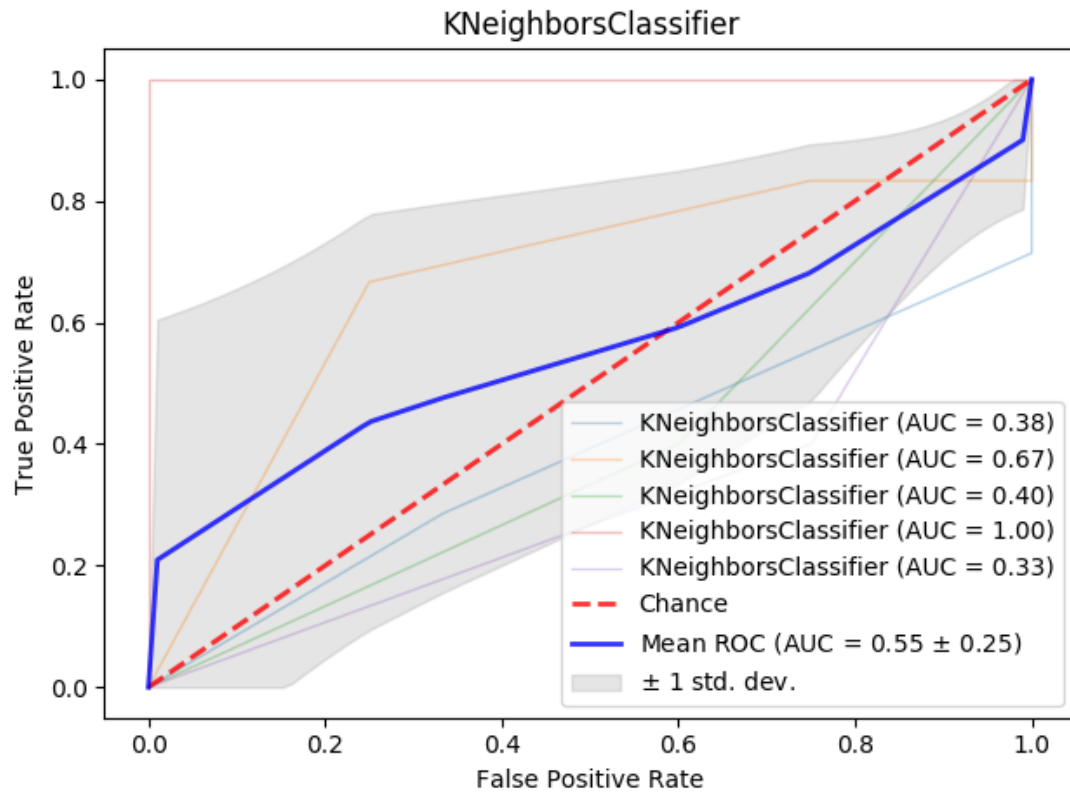
Przez interpretowalność modelu [18] rozumie się to, w jaki sposób dany algorytm dotarł do pewnych rezultatów. Jako, że regresja logistyczna jest modelowana jako liniowa funkcja, to z łatwością można uzyskać znaczenie danego atrybutu w całym procesie i dowiedzieć się, czy wysoka wartość tego atrybutu wpłynie na pozytywną lub negatywną prognozę (atrybuty posiadające wagi ujemne będą sugerowały negatywną prognozę, a wagi dodatnie odwrotnie).



Rysunek 6.1: Wykres przedstawiający AUC dla regresji logistycznej.

Podobny wynik uzyskano używając modelu k -NN widocznego na wykresie 6.2,

jednak interpretowalność, będąca kluczową jednostką w tym projekcie była zdecydowanie lepsza dla regresji.

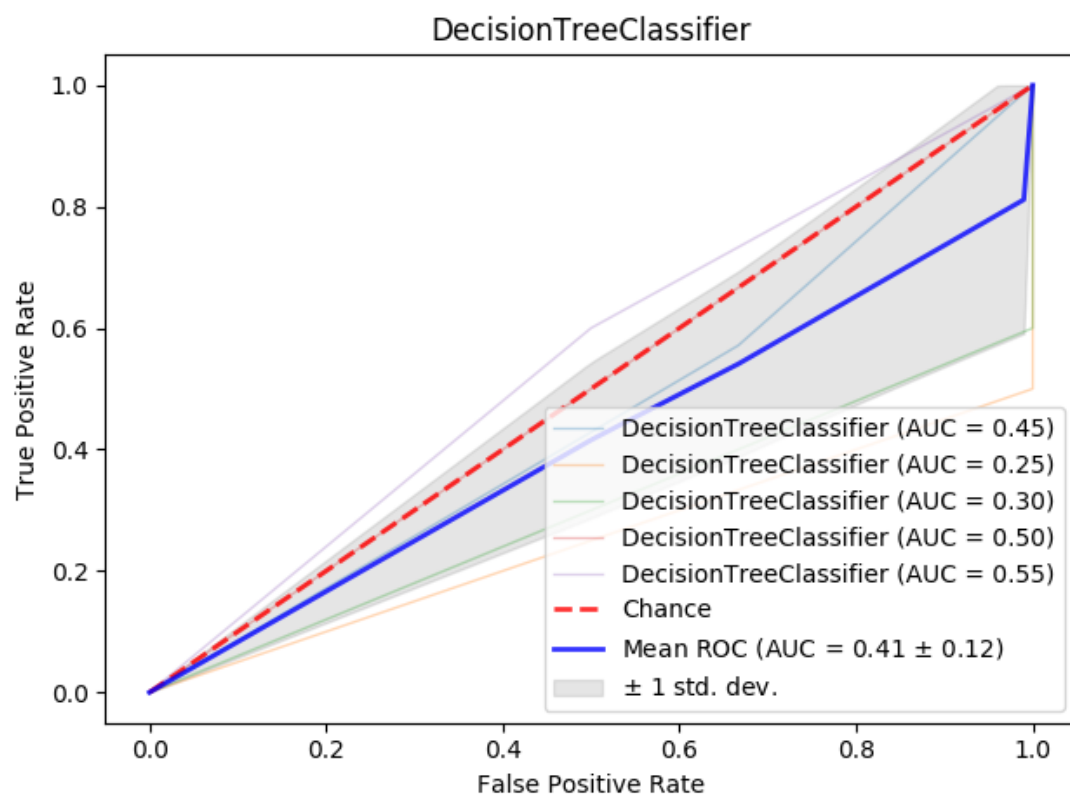


Rysunek 6.2: Wykres przedstawiający AUC dla KNN.

Kolejnym algorytmem posiadającym bardzo dobrą interpretowalność jest drzewo decyzyjne którego wyniki pokazane są na rysunku 6.3. AUC tego klasyfikatora rozwiązującego problem dwuklasowy wynosi mniej niż 0.5 co oznacza, że model częściej wskazywał klasę przeciwną.

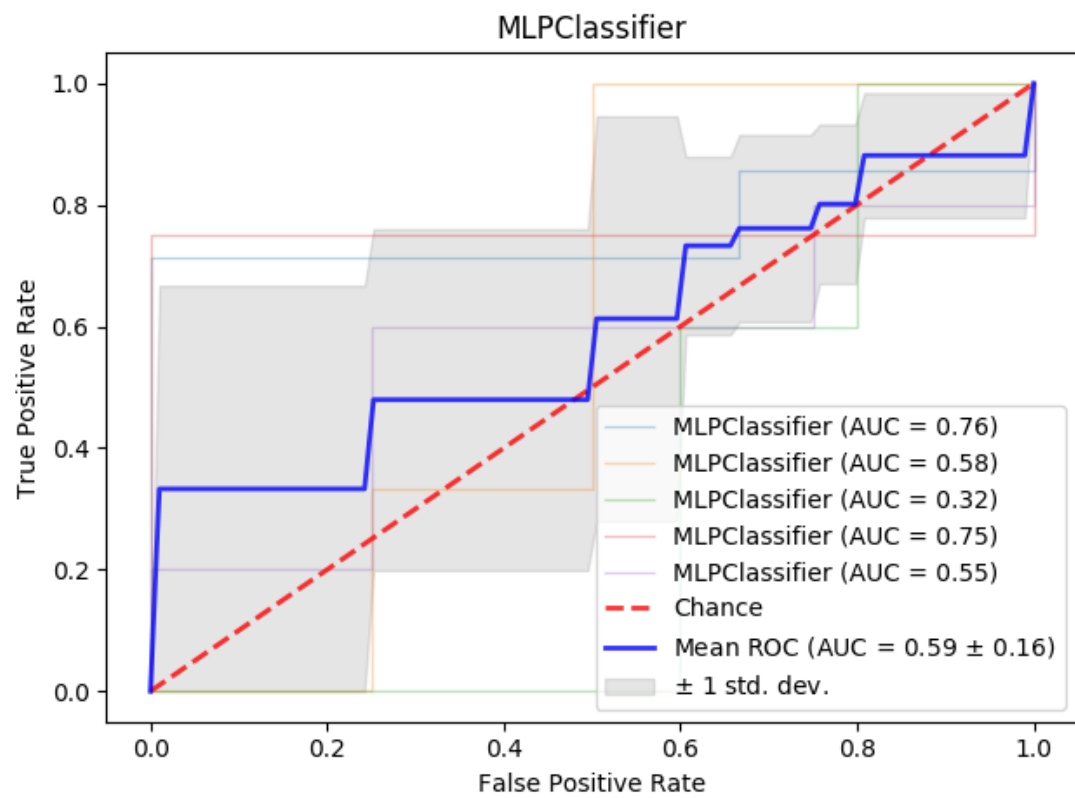
Ostatnim modelem poddanym eksperymentowi jest MLP, dla którego wyniki są widoczne na rysunku 6.4. Model ten uzyskał największą średnią AUC, ale jego interpretacja jest dużo trudniejsza niż w przypadku regresji logistycznej.

Eksperymenty można uznać za udane, ponieważ na początku projektu pojawiły się wątpliwości, czy w jakikolwiek sposób można przewidzieć samopoczucie użytkownika biorąc pod uwagę jedynie to z jakich aplikacji on korzystał i w jakiej porze dnia. Maksymalne AUC (dla algorytmu MLP) wynoszące 0.59 nie jest liczbą



Rysunek 6.3: Wykres przedstawiający AUC dla drzewa decyzyjnego.

wysoką, ale na pewno spisuje się lepiej niż klasyfikacja polegająca na losowaniu między dwoma etykietami.



Rysunek 6.4: Wykres przedstawiający AUC dla MLP.

Rozdział 7

Wnioski i napotkane problemy

Zadanie w ramach pracy inżynierskiej niewątpliwie pozwoliło na zgłębienie wielu technologii i rozwiązań codziennie stosowanych w wielu profesjonalnych projektach. Udało się osiągnąć cel pracy, którym było stworzenie narzędzia umożliwiającego zapisanie i analizę historycznych aktywności użytkownika wewnątrz urządzenia mobilnego. Udało się również zebrać te dane, co wymagało zainstalowania aplikacji na kilku urządzeniach i przekonania innych osób do czynnego oceniania swojego samopoczucia, gdy tylko pojawiała się notyfikacja. Kwestią sporną może być powodzenie klasyfikacji danych, ponieważ AUC będące głównym miernikiem jakości modelu, opisane w rozdziale 6, wynosiło jedynie 0.59. Trzeba jednak pamiętać, że na samopoczucie użytkownika wpływa wiele innych czynników, a same aplikacje mobilne nie są jedynym czynnikiem technologicznym. Użytkownik może również używać aplikacji desktopowych, czy też korzystać większość czasu z przeglądarki internetowej lub oglądać telewizję. Klasyfikator użyty w projekcie nie ma pojęcia o innych aspektach życia użytkownika. Wszystkie dane pochodzą z historii aktywności i ocen wystawianych przez właściciela urządzenia mobilnego, a więc można uznać, że uzyskana wartość jest wynikiem zadowalającym.

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 po-

bierania 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 wyszedł z aplikacji”). W taki też sposób zostało zaimplementowane pobieranie aktywności. Zdarzenia były pobierane, filtrowane i następnie chronologicznie segregowane aby uzyskać ciągły przebieg wszystkich aktywności odbywających się na urządzeniu mobilnym dla konkretnego użytkownika.

Innym 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.

Kolejnym 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.

Wracając 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 musiałby udostępnić również informacje na temat tego ile czasu przeznaczają na korzystanie z aplikacji komputerowych oraz stron internetowych. W kolejnym etapie można byłoby 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 badałyby one czy aktualny czas jaki użytkownik przeznaczają na daną aplikację może na niego działać z niepożądanym efektem. Analiza i przekształcenie danych poprzez dyskretyzację również mogłaby 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

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

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

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

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

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

CI/CD (ang. *Continuous Integration / Continuous Delivery*) Proces bezpiecznego i często dostarczania sprawdzonych zmian w kodzie.

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

Intencja (ang. *Intent*) Podstawowy komponent systemu Android, dzięki któremu można m.in. otwierać nowe aktywności lub aplikacje..

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

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

- MLP (z ang. *Multilayer Perceptron*) Najpopularniejszy typ sztucznej sieci neuro-
nowej.
- PostgreSQL Jeden z trzech najpopularniejszych otwartych systemów zarządzania relacyj-
nymi bazami danych.
- RDS (ang. *Amazon Relational Database Service*) Serwis umożliwiający stworzenie
i konfigurację relacyjnej bazy danych.
- relu (z ang. *rectified linear unit*) Funkcja aktywacyjna o postaci $\max(0, x)$.
- ROC (z ang. *Receiver Operating Characteristic*) Krzywa opisująca funkcję odcię-
cia 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).
- k -NN (ang. *k nearest neighbours*) Algorytm k najbliższych sąsiadów.

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 QualityTime*
Mobidays, Inc.
<https://www.qualitytimeapp.com/>
- [9] *Opis TimeDoctor*
TimeDoctor, 30 listopad 2019
<https://www.timedoctor.com/features.html>
- [10] *Building A Logistic Regression in Python, Step by Step*
Susan Li, 29 wrzesień 2017
<https://towardsdatascience.com/building-a-logistic-regression-in-python-step-by-step-becd4d56c9c8>
- [11] *One Hot Encoding in Python*
Susan Li, 10 wrzesień 2018
<https://blog.cambridgespark.com/robust-one-hot-encoding-in-python-3e29bfcec77e>
- [12] *TypeORM - TypeScript guide*
David Herron, 18 lipiec 2019
<https://levelup.gitconnected.com/complete-guide-to-using-typeorm-and-typescript-for-data-persistence-in-node-js-module-bfce169959d9>
- [13] *Naive Bayes Classifier*
Rohith Gandhi, 5 maj 2018
<https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c>
- [14] *Introduction to Machine Learning and Data Mining*
Tue Herlau, Mikkel N. Schmidt and Morten Mørup, 31 styczeń 2019, pp. 156 - 157
Technical University of Denmark
- [15] *A Study on Influence of Normalization Methods on Music Genre Classification Results Employing knn Algorithm*
Rosner Aldona, Michalak Marcin and Kostek B.
2013. STUDIA INFORMATICA, vol. 34, no. 2A, pp. 411-423

-
- [16] *When to Use MLP, CNN, and RNN Neural Networks*
Jason Brownlee, 23 lipiec 2018
<https://machinelearningmastery.com/when-to-use-mlp-cnn-and-rnn-neural-networks/>
- [17] *What is CI/CD? Continuous integration and continuous delivery explained*
Isaac Sacolick, 10 maj 2018
<https://www.infoworld.com/article/3271126/what-is-cicd-continuous-integration-and-continuous-delivery-explained.html>
- [18] *An introduction to variable and feature selection.*
I. Guyon, A. Elisseeff
Journal of Machine Learning Research, 3(7-8):1157-1182, 2003.
- [19] *JSON Type*
The PostgreSQL Global Development Group, styczeń 2020
<https://www.postgresql.org/docs/9.2/datatype-json.html>
- [20] *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 \LaTeX owe i końcowa wersja w pdf),
- źródła programu,
- dane testowe.

Spis rysunków

3.1	Diagram przedstawiający codzienny proces „oceny dnia” przez użytkownika.	6
3.2	Diagram przedstawiający proces odczytywania raportu w aplikacji mobilnej.	6
3.3	Diagram przedstawiający cykliczny proces przesyłania danych historyczny do serwisu zewnętrznego.	7
4.1	Szczegóły aplikacji wykonanej w ramach pracy inżynierskiej w Google Play.	10
4.2	Ustawianie zezwoleń do śledzenia historii aktywności dla aplikacji wykonanej w ramach pracy inżynierskiej.	11
4.3	Autoryzacja poprzez korzystanie z usługi Google Play Services wewnątrz aplikacji wykonanej w ramach pracy inżynierskiej.	12
4.4	Widok oceny dnia w aplikacji wykonanej w ramach pracy inżynierskiej.	13
4.5	Widok aktywności odbywających się w ostatnich dwudziestu czterech godzinach w aplikacji wykonanej w ramach pracy inżynierskiej. Znajdująca się tu lista pokazuje nazwę aplikacji, ostatni punkt w czasie kiedy użytkownik z niej korzystał, oraz sumę czasu, jaki użytkownik poświęcił danej aplikacji.	14

4.6	Widok raportu w aplikacji wykonanej w ramach pracy inżynierskiej. Na początku przedstawione są aplikacje, które wpływają pozytywnie na samopoczucie użytkownika oraz pora dnia w której posługiwanie się urządzeniem mobilnym wpływa korzystnie. Analogicznie, poniżej przedstawione są aplikacje które wpływają negatywnie na samopoczucie i pora dnia w ciągu której posługiwanie się urządzeniem mobilnym wpływa niekorzystnie.	15
5.1	Infrastruktura projektu.	22
5.2	Diagram przedstawiający zależności między klasami w aplikacji mobilnej.	26
5.3	Diagram przedstawiający schemat bazy danych.	28
5.4	Diagram przedstawiający zależności między elementami w aplikacji Node.js.	30
5.5	Diagram przedstawiający zależności między elementami w aplikacji napisanej w języku Python.	32
6.1	Wykres przedstawiający AUC dla regresji logistycznej.	37
6.2	Wykres przedstawiający AUC dla KNN.	38
6.3	Wykres przedstawiający AUC dla drzewa decyzyjnego.	39
6.4	Wykres przedstawiający AUC dla MLP.	40

Spis tablic

5.1	Przykładowe dane historyczne pochodzące z Usage Stats API w systemie Android	18
5.2	Przykładowe dane historyczne po transformacji z postaci oryginalnej dla Usage Stats API w systemie Android do postaci umożliwiającej klasyfikację	19
5.3	Tabela przedstawiająca wszystkie biblioteki które zostały bezpośrednio użyte podczas implementacji.	33