Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Instytut Automatyki i Informatyki Stosowanej

# Praca dyplomowa inżynierska

na kierunku Informatyka
w specjalności Systemy Informacyjno Decyzyjne

Platforma Informacyjnej Współpracy Ogólnowydziałowej
Aplikacja Mobilna

Tomasz Słuszniak
Numer albumu 283765

promotor
dr inż. Mariusz Kamola

WARSZAWA 2020

# Platforma Informacyjnej Współpracy Ogólnowydziałowej Aplikacja Mobilna

**Streszczenie.** Celem pracy było stworzenie aplikacji mobilnej dla urządzeń z systemem Android pozwalającej na korzystanie z głównych funkcjonalności Platformy Informacyjnej Współpracy Ogólnowydziałowej. Platforma jest serwisem centralizującym komunikaty i aktualności z różnych źródeł online należących do organów i organizacji funkcjonujących na Politechnice Warszawskiej. Aplikacja pozwala przeglądać komunikaty pochodzące z wielu stron internetowych w jednym, wygodnym miejscu. Ponadto umożliwia użytkownikowi ustalenie preferencji co do wyświetlanych treści. Głównym walorem oprogramowania jest możliwość automatycznych powiadomień o nowych treściach, co pozwala użytkownikowi pozostać na bieżąco ze wszystkimi interesującymi go kanałami informacyjnymi bez potrzeby pamiętania o codziennym przeglądaniu wszystkich ręcznie za pomocą przeglądarki internetowej.

**Słowa kluczowe:** aplikacja mobilna, Android, aktualności Politechniki, PIWO, Platforma Informacyjnej Współpracy Ogólnowydziałowej

# Faculty Cooperation Information Platform Mobile Application

**Abstract.** The aim of the project was to develop a mobile application for Android powered devices that would allow to use main features of Faculty Cooperation Information Platform (P.I.W.O.). Platform is a service that centralises news and communicates from various online sources of offices and organisations functioning at the Warsaw University of Technology. Application lets the user browse the news from various origins in one, easy to use place. Moreover, it allows the user to set his own preferences as to content shown. Main advantage of the application is the system of automatic notifications about new content, what lets the user to be up to date with all the news without the struggle to remember about daily check of all the websites manually.

**Keywords:** mobile app, Android, university news, PIWO, Platforma Informacyjnej Współpracy Ogólnowydziałowej, Faculty Cooperation Information Platform

załącznik do zarządzenia nr 28/2016 r.
Rektora PW

**Politechnika Warszawska**

„załącznik nr 3 do zarządzenia nr 24/2016 Rektora PW

.....
miejscowość i data

.....
imię i nazwisko studenta

.....
numer albumu

.....
kierunek studiów

## OŚWIADCZENIE

Świadomy/-a odpowiedzialności karnej za składanie fałszywych zeznań oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie, pod opieką kierującego pracą dyplomową.

Jednocześnie oświadczam, że:

- niniejsza praca dyplomowa nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.) oraz dóbr osobistych chronionych prawem cywilnym,
- niniejsza praca dyplomowa nie zawiera danych i informacji, które uzyskałem/-am w sposób niedozwolony,
- niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadawaniem dyplomów lub tytułów zawodowych,
- wszystkie informacje umieszczone w niniejszej pracy, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami,
- znam regulacje prawne Politechniki Warszawskiej w sprawie zarządzania prawami autorskimi i prawami pokrewnymi, prawami własności przemysłowej oraz zasadami komercjalizacji.

Oświadczam, że treść pracy dyplomowej w wersji drukowanej, treść pracy dyplomowej zawartej na nośniku elektronicznym (płycie kompaktowej) oraz treść pracy dyplomowej w module APD systemu USOS są identyczne.

.....
czytelny podpis studenta"

# Spis treści

# 1. Wstęp

## 1.1. Motywacja i cel pracy

Na Politechnice Warszawskiej, jako na jednej z największych polskich uczelni, funkcjonuje wiele organów oraz organizacji mających na celu zarówno sprawowanie kontroli nad przebiegiem studiów jak i wzbogacenie życia akademickiego uczelni. Poza samym tokiem studiów organizowanych jest wiele wydarzeń, szkoleń i warsztatów pomagających studentom w podwyższaniu ich kwalifikacji jak i stanowiących odskocznię od nauki i rozrywkę. Szeroko postrzegalny jest jednak fakt, iż w natłoku informacji wiele z komunikatów publikowanych przez różne podmioty nie dociera do odpowiednich odbiorców, czyli potencjalnie zainteresowanych studentów lub też dociera zbyt późno. Wskutek tego zjawiska zmniejszona jest chociażby frekwencja potencjalnie zainteresowanych osób na różnorakich wydarzeniach organizowanych przez podmioty takie jak przykładowo Biuro Karier Politechniki Warszawskiej. Innym problemem jest także niedoinformowanie studentów o obowiązujących terminach rejestracji na przedmioty, płatności za zaległości w nauce czy akcji kwaterunkowej. Problemem jest różnorodność, niejednolitość i nieoczywista lokalizacja niektórych miejsc zawierających istotne aktualności na stronach internetowych należących zarówno do podmiotów Uczelni jak i do organizacji nań działających. Student chcący być na bieżąco ze wszystkim, co dzieje się w jego akademickim otoczeniu, musiałby codziennie sprawdzać wiele z tych internetowych lokalizacji, co jest bardzo mało prawdopodobne z oczywistych względów. Założeniem Platformy Informacyjnej Współpracy Ogólnowydziałowej jest centralizacja źródeł aktualności i komunikatów, takich jak na przykład:

— Komunikaty dziekanatów
— Wydarzenia wydziałowe i ogólnouczelniane
— Warsztaty i szkolenia Biura Karier
— Wydarzenia kół naukowych i organizacji studenckich
— Akcje kwaterunkowe domów studenckich

Platforma jest aktualnie dostępna pod adresem www.piwopw.pl.

W skład serwisu, z punktu widzenia użytkownika, wchodzi strona internetowa oraz będąca przedmiotem niniejszej pracy dyplomowej aplikacja mobilna dla urządzeń pod kontrolą systemu operacyjnego Android. To właśnie aplikacja mobilna oferuje największą wartość, gdyż umożliwia automatyczne powiadamianie użytkownika o nowych treściach pochodzących z wyżej wymienionych źródeł i zwiększa wygodę przeglądania treści niektórych stron na urządzeniach mobilnych.

### 1.2. Dostępne źródła informacji na PW

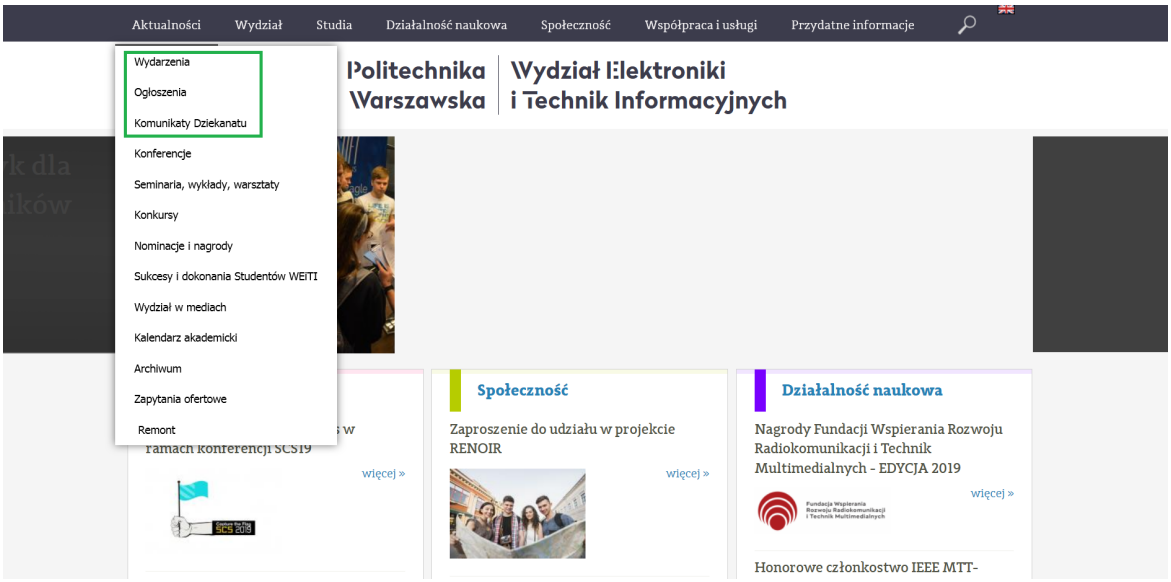
Jak zostało wspomniane, Politechnika Warszawska, jako jedna z największych uczelni w kraju, charakteryzuje się mnogością organów i instytucji nań funkcjonujących. Aktualnie w jej skład wchodzi 19 wydziałów, z których każdy posiada swoją własną, odrębną stronę internetową. Warto zaznaczyć, iż na wielu z nich treści takie jak komunikaty dziekanatu, wydarzenia oraz aktualności publikowane są w odrębnych zakładkach, co pokazane jest poniżej na Rysunkach 1 i 2. Jest to zabieg logiczny oraz zwiększający czytelność przy przeglądaniu strony za pomocą przeglądarki www, natomiast ma znaczenie w kontekście automatycznego pobierania treści przez zewnętrzne oprogramowanie, co zostanie omówione dalej. Poza wydziałami, w strukturze Politechniki funkcjonuje wiele innych organów także świadczących działania na rzecz życia akademickiego. Można podzielić je na dwie grupy:

— Organy, które publikują informacje niezbędne studentom w celu kontynuowania nauki. Są to na przykład informacje o terminach rejestracji na przedmioty prowadzone międzywydziałowo. (Między innymi Studium Wychowania Fizycznego i Sportu, Studium Języków Obcych)
— Organy, które publikują informacje niekonieczne, aczkolwiek potencjalnie interesujące dla studentów. Są to na przykład komunikaty dotyczące organizowanych szkoleń, warsztatów czy spotkań tematycznych. (Między innymi Biuro Karier, Centrum Zarządzania Innowacjami i Transferem Technologii)

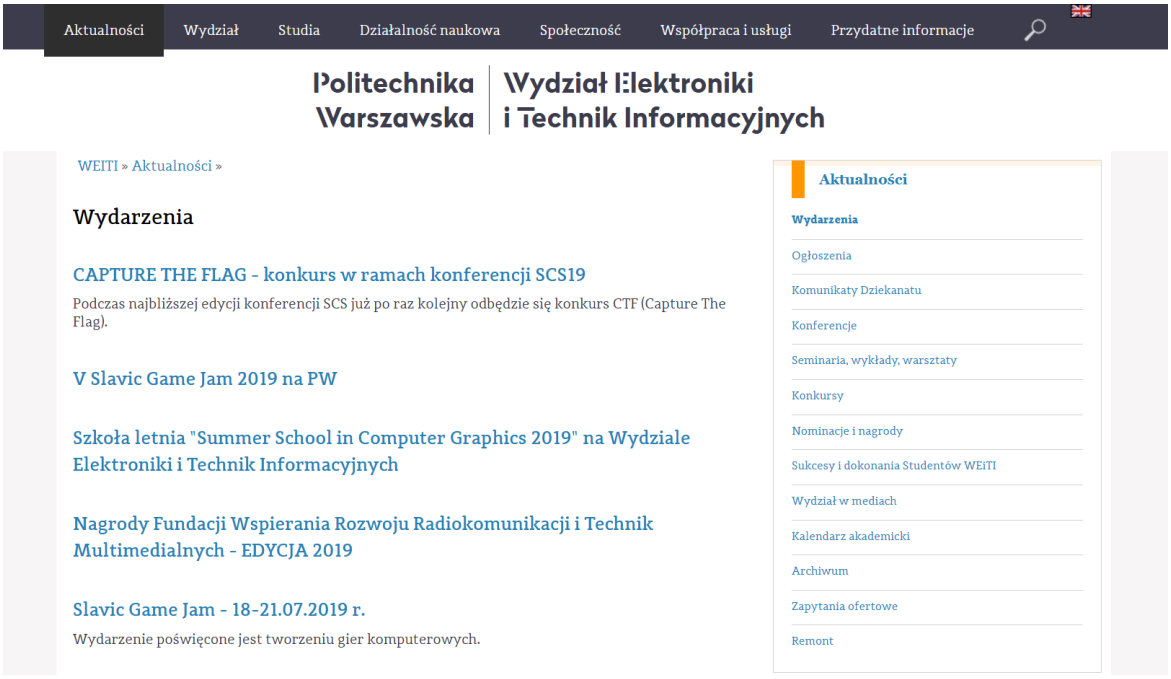
Ponadto na uczelni działalność swą prowadzi wiele kół naukowych oraz organizacji znacznie wzbogacających życie akademickie studentów. One także posiadają własne witryny www, na których umieszczają treści lub też reklamują swoje aktywności poprzez portale społecznościowe.

Pomimo, iż Politechnika w większości korzysta aktualnie ze spójnego wzornictwa stron www opartego na jednym systemie CMS, niektóre z jej organów posiadają strony niewpisujące się w ten schemat. Także organizacje i koła studenckie nie czerpią z tego standardu, prowadząc witryny znacznie różniące się między sobą wyglądem i strukturą.

Jednym ze skutków takiej sytuacji jest brak przejrzystości tej struktury informacyjnej, jako całości, dla studenta lub innej osoby mającej styczność z serwisami Politechniki. Drugim, być może znacznie poważniejszym skutkiem, jest niemożność codziennego ręcznego przeglądania wszystkich tych źródeł przez przeciętną osobę zainteresowaną, ponieważ zajmowałoby to zbyt wiele czasu. Niestety, nie wszystkie wyżej wymienione podmioty prowadzą kanały RSS pozwalające na łatwe subskrybowanie i powiadamianie zainteresowanych osób o nowych treściach.




**Rysunek 1.** Strona Wydziału Elektroniki i Technik Informacyjnych z odrębnymi zakładkami na wydarzenia, ogłoszenia oraz komunikaty dziekanatu




**Rysunek 2.** Strona wydarzeń Wydziału Elektroniki i Technik Informacyjnych

Poniżej znajduje się wycinek kodu HTML pokazanej wyżej strony Wydziału Elektroniki i Technik Informacyjnych zawierający jeden z nagłówków „wydarzeń” pokazanych na Rys. 2.

**Fragmenty kodu 1.** Fragment kodu HTML strony Wydziału Elektroniki i Technik Informacyjnych odpowiedzialny za wyświetlenie pojedynczego nagłówka w sekcji „wydarzeń".

```
<div class=''content-view-lin''>
<div class=''class-blog-post float-break''>
<div class=''attribute-header''>
<h3><a href=''/Aktualnosci/Wydarzenia/
CAPTURE-THE-FLAG-konkurs-w-ramach-konferencji-SCS19"
title="CAPTURE THE FLAG - konkurs w ramach konferencji SCS19''>
CAPTURE THE FLAG - konkurs w ramach konferencji SCS19
</a>
</h3>
</div>
<div class=''attribute-body float-break''>
<p>Podczas najblizszej edycji konferencji SCS juz po raz kolejny
odbedzie sie konkurs CTF (Capture The Flag);
</p>
</div>
</div>
</div>
```

Poniżej natomiast zamieszczono analogiczny fragment ze strony Wydziału Fizyki dotyczący ogłoszeń dziekanatu.

**Fragmenty kodu 2.** Fragment kodu HTML strony Wydziału Fizyki odpowiedzialny za wyświetlenie pojedynczego nagłówka w sekcji "ogłoszeń dziekanatu".

```
<a href=''/index.php/pl/aktualnosci/ogloszenia-dziekanatu/
item/1005-dodatkowe-szkolenie-przysposobienie-biblioteczne''>
Dodatkowe szkolenie &amp;quot;Przysposobienie biblioteczne&amp;quot;
</a>
</h2>
```

Wycinki te nie pokazują pozostałej części kodu strony, są jednak dowodem wystarczającym na pokazanie, iż strony te różnią się ze sobą strukturą na tyle, iż nie możliwe jest zaaplikowanie tego samego algorytmu parsującego do każdej z nich.

### 1.3. Charakterystyka serwisu PIWO

Jednym z podstawowych założeń Platformy jest nieinwazyjność stosowanych rozwiązań względem serwisów i stron Politechniki oraz innych wcześniej wspomnianych podmiotów mogących być uznanymi za „źródła treści". Polega to na takim zaprojek-

towaniu systemu, by nie była wymagana żadna ingerencja ze strony administratorów poszczególnych stron internetowych, będących dalej zwanymi „Źródłami". Pobranie odpowiednich treści musi dokonywać się z aktualnej postaci danego Źródła, bez implementacji specjalnych interfejsów czy innych sposobów dostępu do treści specjalnie dla Platformy.

Powodem tego jest chęć ułatwienia uczestnikom akademickiego życia uczelni procesu przekazywania informacji, do tej grupy zaliczyć można także administratorów stron i osoby publikujące na nich. Stąd pełna automatyzacja, ponieważ stworzenie kolejnego medium wymagającego ręcznej publikacji treści tylko przysporzyłoby pracy publikującym je osobom. W takiej sytuacji rozwiązanie spotkałoby się prawdopodobnie z niechęcią, a dodatkowo wystąpiłyby niespójności w czasie publikacji – treści na stronach www Źródeł mogłyby pojawiać się wcześniej niż na Platformie. Tak samo stworzenie rozwiązania wymagającego implementacji odpowiednich interfejsów po stronie Źródeł mogłoby spotkać się z problemami technicznymi i potrzebą dużego nakładu pracy ze strony osób odpowiedzialnych za nie od strony technicznej.

Odpowiednim rozwiązaniem wydaje się więc pobieranie standardowej postaci konkretnej podstrony i parsowanie jej kodu HTML odpowiednio zaprojektowanym algorytmem, w celu wyłuskania żądanej zawartości z odpowiednich znaczników HTML. Zawartość oznacza w tym wypadku listę komunikatów danego podmiotu, ich tytuły, daty publikacji, czy treści, np. komunikatów dziekanatu. Następnie zawartości te, jako poszczególne oddzielne komunikaty, zapisywane są do bazy danych Platformy.

Z pośród wspomnianych podmiotów wybrano kilka, których obsługę dodano w pierwszej kolejności:

— Biuro Karier Politechniki Warszawskie
— Wydział EiTI (Komunikaty Dziekanatu, Wydarzenia, Aktualności)
— Wydarzenia Politechniki Warszawskiej
— Niezależne Zrzeszenie Studentów
— Organizacja Studencka Enactus
— Akademicki Klub Turystyczny "Maluch"

Ponieważ algorytm „jest świadomy" z którego źródła dane komunikaty pochodzą, jest możliwe ich odpowiednie sortowanie i przeszukiwanie pod tym kątem. Treści z różnych źródeł są przypisywane do poszczególnych „Kanałów", odpowiadających nazwie danego podmiotu je publikującego. Każdy użytkownik Platformy ma więc możliwość określenia które z Kanałów go interesują, a które chce pominąć podczas przeglądania treści. Podczas rejestracji użytkownik wybiera także z którego jest wydziału, co pozwala na dobranie dla niego pewnej domyślnej listy subskrybowanych kanałów.

Komunikaty mogą zostać pobrane przez aplikacje klienckie przy użyciu specjalnie zaprojektowanego dla tego celu interfejsu po stronie serwera. Także synchronizacja pozostałych ustawień przypisanych do danego konta użytkownika dokonuje się tą drogą.

Niniejsza praca dyplomowa skupia się przede wszystkim na aspekcie aplikacji mobilnej, stąd zagadnienia związane z implementacją backendu czy frontendu serwera są omówione w stopniu minimalnym.

# 2. Realizacja pracy

## 2.1. Wstępne założenia i wymagania od aplikacji mobilnej

Pierwszym etapem realizacji projektu jest dokładne określenie wymagań, jakie stawia się przed tworzoną aplikacją. Pozwala to na łatwiejsze zaplanowanie prac oraz architektury rozwiązania. Znając priorytet poszczególnych części można także uszeregować ich wykonanie w kolejności minimalizującej ryzyko porażki projektu z powodu chociażby nagłych ograniczeń czasowych.

### 2.1.1. Wymagania konieczne

Wymagania te dotyczą podstaw działania aplikacji i muszą być spełnione by zapewnić wygodne jej użytkowanie.

— Aplikacja musi pozwalać na wyświetlenie komunikatów dostępnych także na stronie internetowej Platformy w przyjazny sposób zgodny z wytycznymi projektowania interfejsów użytkownika na systemie Android.
— Aplikacja musi automatycznie powiadamiać użytkownika o nowych komunikatach.
— Aplikacja musi zapewniać bezpieczną metodę logowania się w usłudze Platformy.
— Aplikacja musi zapewniać możliwość rejestracji nowego użytkownika. Możliwym innym podejściem byłoby przekierowanie użytkownika na stronę Platformy w celu dokonania rejestracji, jednak wydłużyłoby to ścieżkę, jaką musi przebyć potencjalny nowy użytkownik w celu rozpoczęcia korzystania z serwisu. Zwiększyłoby to tak zwany współczynnik odrzuceń. Współczynnik ten określa w tym wypadku ile osób zrezygnowało z dalszej rejestracji jeszcze przed rozpoczęciem korzystania z serwisu (np. z powodu poziomu skomplikowania lub długiego czasu oczekiwania).
— Aplikacja musi zapewniać możliwość edycji listy subskrybowanych kanałów.
— Aplikacja musi posiadać widok informacyjny zawierający informację o sposobie kontaktu z twórcami Platformy oraz odnośnik do regulaminu Platformy. Jest to naturalny wymóg związany między innymi z ułatwieniem ewentualnych zgłoszeń o problemach z aplikacją lub serwisem jako całością.

### 2.1.2. Wymagania niekonieczne

Wymagania te nie są krytyczne dla funkcjonowania aplikacji, aczkolwiek ich spełnienie znacznie podniosłoby komfort korzystania z niej lub ułatwiłoby zarządzanie ze strony administratora Platformy.

— Aplikacja powinna umożliwić także na wyświetlenie pełnej treści danego komunikatu także pozostając bez dostępu do Internetu.
— Aplikacja powinna umożliwiać edycję niektórych dodatkowych ustawień konta użytkownika.

— Aplikacja powinna posiadać mechanizm pozwalający na jej zdalne zresetowanie (wyczyszczenie lokalnej bazy danych, jeśli taka będzie istnieć) podczas synchronizacji z serwerem. W wypadku nieprawidłowej zawartości któregoś z komunikatów, która to powodowałaby błąd, możliwe byłoby jej usunięcie nie tylko z bazy danych serwera, ale także po jej propagacji z poszczególnych instalacji aplikacji. Dodatkowo, gdyby zmienił się sposób przetwarzania treści lub też ich format mogłoby okazać się, że warto zastąpić poprzednie wersje komunikatów ich nowszymi, na nowo pobranymi odpowiednikami.

### 2.1.3. Wymagania dodatkowe

Wymagania te nie są krytyczne dla funkcjonowania aplikacji oraz nie dotyczą rozwoju jej głównej funkcjonalności. Ich spełnienie jednak wniosłoby dodatkową wartość dla użytkownika.

— Aplikacja może posiadać możliwość zapisania ulubionych adresów stron www umożliwiając ich szybkie wyświetlenie z poziomu swojego interfejsu. Jest to pomocne w wypadku na przykład stron przedmiotowych i zostało zasugerowane przez użytkowników aplikacji na pewnym etapie jej rozwoju.

## 2.2. Analiza grupy docelowej, związane z tym ograniczenia i wybór technologii

Grupą docelową aplikacji są studenci Politechniki Warszawskiej. Można założyć, iż praktycznie wszyscy z nich posiadają w czasach obecnych telefony komórkowe z dostępem do Internetu mobilnego. Ceny transferu u operatorów komórkowych są obecnie dość niskie, a sama aplikacja Platformy nie będzie przesyłać dużych ilości danych na serwer ani pobierać dużego rozmiaru multimediów. Kwestia finansowa nie powinna więc być przeszkodą w korzystaniu z Platformy poza domem.

W Polsce, jako kraju będącym poza czołówką zamożności w Unii Europejskiej, na rynku mobilnym przoduje system operacyjny Android. Jest to między innymi podyktowane o wiele niższą ceną urządzeń pod jego kontrolą oraz ogólnie niższą ceną usług i akcesoriów dla tego typu telefonów. To ten właśnie system wydaje więc się odpowiedni do przygotowania aplikacji mobilnej Platformy w pierwszej kolejności, ponieważ produkt trafić wtedy może do większej liczby osób.

Co prawda na rynku dostępnych jest bardzo wiele telefonów z systemem operacyjnym od Google, jednak opracowywana aplikacja nie wykorzystuje żadnych charakterystycznych dla danych modeli funkcji czy fizycznych komponentów (jak wieloobiektywowe aparaty, technologia NFC, wykrywanie gestów dłoni czy ruchu gałek ocznych). Marka czy model urządzenia nie będą więc miały znaczenia, tak długo jak wersja systemu Android będzie odpowiednia.

Wspomniawszy o wersji systemu operacyjnego, według oficjalnych danych Google wersje Pie, Oreo, Nougat, Marshmallow, Lollipop, KitKat i JellyBean (wymienione w

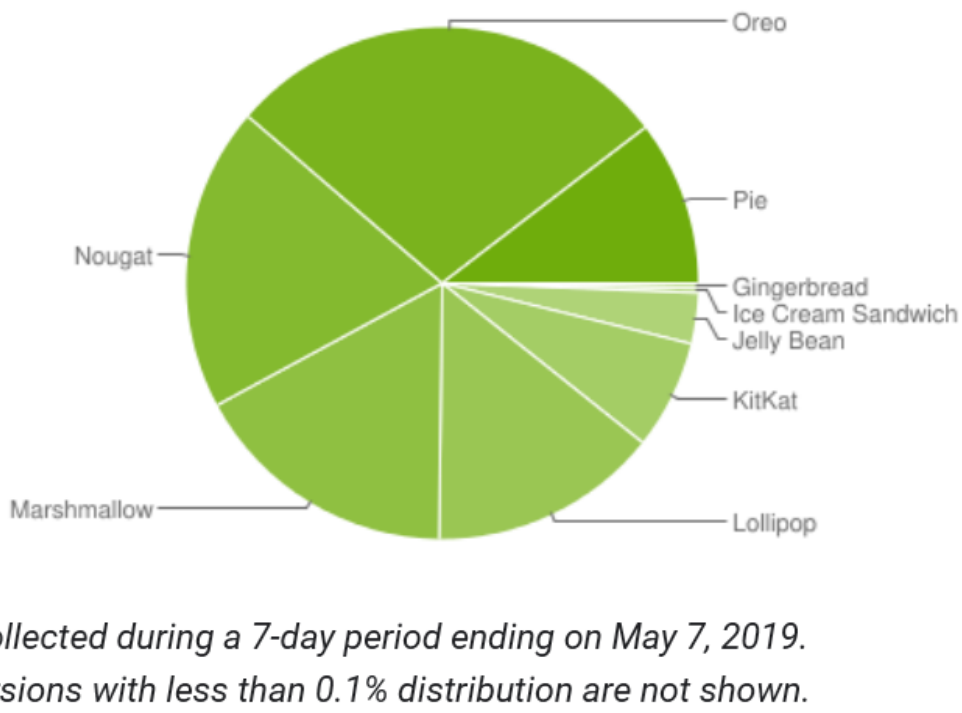
kolejności od najnowszej) stanowią obecnie około 99% działających na urządzeniach użytkowników instalacji. Fakt ten zaprezentowano na Rysunkach 3 i 4.

| Version | Codename | API | Distribution |
|---|---|---|---|
| 2.3.3 - 2.3.7 | Gingerbread | 10 | 0.3% |
| 4.0.3 - 4.0.4 | Ice Cream Sandwich | 15 | 0.3% |
| 4.1.x | Jelly Bean | 16 | 1.2% |
| 4.2.x | | 17 | 1.5% |
| 4.3 | | 18 | 0.5% |
| 4.4 | KitKat | 19 | 6.9% |
| 5.0 | Lollipop | 21 | 3.0% |
| 5.1 | | 22 | 11.5% |
| 6.0 | Marshmallow | 23 | 16.9% |
| 7.0 | Nougat | 24 | 11.4% |
| 7.1 | | 25 | 7.8% |
| 8.0 | Oreo | 26 | 12.9% |
| 8.1 | | 27 | 15.4% |
| 9 | Pie | 28 | 10.4% |

**Rysunek 3.** Tabela pokazująca aktualne pokrycie rynku różnymi wersjami systemu Android. Źródło: [1]

Grupa docelowa nie charakteryzuje się posiadaniem najnowszego na rynku sprzętu. Trzeba więc założyć, że dość znaczna część potencjalnych użytkowników posiada telefony ze starszymi wersjami systemu. Biorąc pod uwagę powyższe oraz fakt, że nie ma ku temu silnych technologicznych przeciwwskazań, jako minimalna wersja systemu Android, na których będzie mogła zostać uruchomiana aplikacja zostaje wybrana 4.2 Jelly Bean.

Projektowanie rozwiązań dla systemu Android jest znacznie uzależnione od tego wyboru. Dla nowszych wersji systemu dostępnych jest więcej bibliotek i rozwiązań znacznie ułatwiających pracę. Wersja 4.2 przyniosła ze sobą głównie zmiany dotyczące wydajności pracy systemu i oprogramowania na nim uruchomionego. Z punktu widzenia interfejsu użytkownika, wprowadzono między innymi możliwość zagnieżdżania Fragmentów, czyli klas odpowiadających za zarządzanie niektórymi elementami graficznymi.




**Rysunek 4.** Wykres kołowy ilustrujący pokrycie rynku różnymi wersjami systemu Android. Źródło: [1]

## 2.3. Wybór technologii

### 2.3.1. Języki programowania

W ramach przygotowań do wykonania aplikacji mobilnej wykonano porównanie podstawowych języków programowania dla platformy Android. Wzięto pod uwagę jedynie języki wspierane natywnie, jako że dają one największą swobodę projektowania przy równoczesnym zapewnieniu niezawodności i względnej optymalności kodu. Rozwiązania takie jak Xamarin, pozwalający pisać rozbudowane aplikacje pod wiele platform, charakteryzują się dużym narzutem na wydajność oraz objętość kodu wprowadzając równocześnie utrudnienia związane z architekturą i projektowaniem z racji na konieczność częściowego napisania kodu oddzielnie dla każdej z docelowych platform (np. obsługa graficznego interfejsu użytkownika).

Kotlin, obecnie będący językiem natywnym dla platformy Android, jest językiem stosunkowo młodym, aczkolwiek aktywnie rozwijanym i wspieranym. Posiada wiele usprawnień ułatwiających programowanie oraz zwiększających znacznie czytelność kodu pozwalając uniknąć tzw. „boilerplate code". Są to fragmenty kodu, które w niezmiennej formie wykorzystywane są w wielu miejscach programu lub też są wymagane lecz nie wnoszą charakterystycznych dla danej aplikacji zachowań, są powszechne. Boilerplate code stanowi pewnego rodzaju narzut na ilość kodu danego rozwiązania, wydłużając czas pracy programisty oraz utrudniając testy. Kotlin zapewnia także

między innymi mechanizm tzw. „null-safety", czyli bezpiecznego obchodzenia się z wartościami null. Polega on w skrócie na tym, że wymuszone jest określenie w kodzie dla każdej zmiennej i funkcji czy może ona przyjąć lub zwrócić wartość null. Zależnie od tego wyboru Kotlin zapewnia różne mechanizmy obsługi takich wartości - np. wykonywanie fragmentów kodu tylko gdy zadana zmienna nie zawiera aktualnie wartości null. Obecna jest także idea obiektów zaprzyjaźnionych (zamiast funkcji i zmiennych statycznych), co zwiększa wygodę korzystania z nieinstancyjnych funkcji i danych (Źródła: [2], [3]).

Java natomiast jest językiem starszym, przez wiele lat to ona pełniła rolę języka natywnego aplikacji na Androida (Źródło: [4]). Jest dobrze udokumentowana, na temat programowania w Javie ukazały się także niezliczone tytuły zarówno w formie elektronicznej jak i papierowej. Jako język bardzo rozpowszechniony, działający na wielu platformach i urządzeniach, posiada wręcz ogromne wsparcie społeczności, wielką liczbę materiałów w Internecie pokrywających tematy metodologii programowania, podchodzenia do poszczególnych zadań projektowych czy też rozwiązywania specyficznych problemów. Java, aktywnie rozwijana od wielu lat, oferuje także wiele bibliotek adresujących zarówno ogólne, jak i dość specyficzne potrzeby. Jest to język sprawdzony w wielu zastosowaniach.

Warto zauważyć, że Kotlin oferuje pełną kompatybilność i interoperacyjność z Javą. Oznacza to, że w obrębie jednej aplikacji można wykorzystać oba te języki bez dodatkowego wysiłku. Biorąc pod uwagę powyższy fakt oraz zalety zarówno Kotlina jak i Javy, zdecydowano się na równoczesne skorzystanie z oby tych języków w finalnej wersji aplikacji. Krok ten stwarza także okazję do nauki sposobów łącznia obu tych języków oraz bezpośredniego porównania niektórych ich funkcjonalności.

### 2.3.2. Narzędzia

Android Studio jest oficjalnym środowiskiem od Google służącym do rozwijania natywnego oprogramowania przeznaczonego na urządzenia pozostające pod kontrolą systemu operacyjnego Android. Posiada bardzo rozbudowane funkcjonalności oraz narzędzia znacznie ułatwiające pisanie kodu, jak automatyczne uzupełnianie czy sugestie rozwiązań zaistniałych problemów. Pozwala także na uruchamianie pisanych programów na emulatorach w celach testowania oraz na uruchamianie kodu z małymi zmianami bez konieczności całego procesu budowania aplikacji (tak zwana funkcja „Instant Run" dostępna w najnowszych wersjach narzędzi Android Studio), co znacznie przyspiesza testowanie drobnych poprawek. Wybór tego środowiska jest więc naturalny i oczywisty.

GitHub to jeden z najpopularniejszych na świecie systemów kontroli wersji. Pozwala na bezpieczne przechowywanie kodu oraz plików projektu na zdalnych serwerach, co umożliwia także na wygodny dostęp z każdego komputera i ułatwia pracę nad projektem. Dodatkowo, umożliwia na równoczesne rozwijanie równoległych wersji

projektu czy cofanie zmian. GitHub jest bardzo rozbudowanym systemem, w tym miejscu nie ma więc potrzeby opisywania tutaj wszystkich jego zalet. To właśnie on jednak wybrany został dla celów realizacji tej pracy.

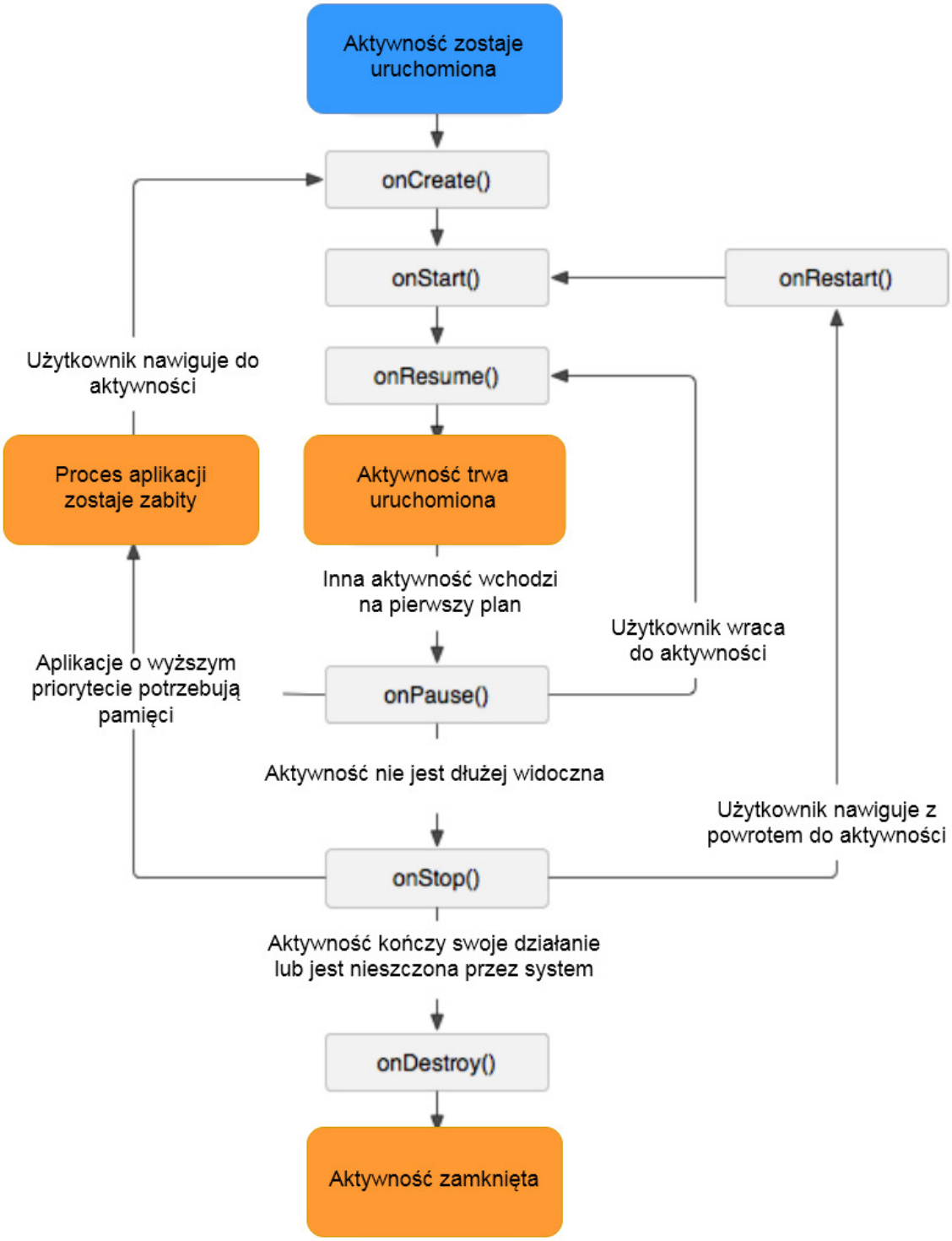
### 2.4. Cykl życia aplikacji mobilnej Android

Każda aplikacja mobilna dla platformy Android podlega z góry narzuconemu cyklowi funkcjonowania. Określa on stany, w jakich znajduje się aplikacja podczas swojej aktywności oraz gdy jest nieaktywna. Wyróżnić można kilka rodzajów procesów:

— Proces pierwszoplanowy
— Proces widoczny
— Proces serwisu
— Proces w cache'u

Procesem, z którym użytkownik ma najczęściej kontakt, jest proces pierwszoplanowy. Jest to, w uproszczeniu, każda uruchomiona aplikacja, z którą aktualnie użytkownik wchodzi w interakcję lub jest wyświetlana „na wierzchu". Aplikacje, a konkretnie ich interfejsy użytkownika, opierają się o Aktywności. Stąd też każda aktywność musi odpowiednio obsługiwać poszczególne stany cyklu życia aplikacji. Mowa jest więc w tym przypadku o cyklu życia aktywności, który to prezentuje Rysunek 5. Przedstawiono na nim kluczowe momenty/działania wpływające na aktywność oraz funkcje wywoływane w celu obsługi tych zdarzeń.

Obsługa poszczególnych zdarzeń jest ważna w aplikacjach wysoce interaktywnych, zwłaszcza w takich, w których użytkownik wprowadza duże ilości danych a ich ewentualna utrata byłaby uciążliwa i niepożądana. Dane takie powinny być tymczasowo zapisywane i wczytywane przy ponownym wyświetlaniu aktywności, jeśli w międzyczasie zaszłaby potrzeba na usunięcie aktywności z pamięci lub przerysowanie od początku jej interfejsu graficznego. W aplikacji będącej przedmiotem opisywanego projektu nie występuje natomiast taka konieczność. Brak jest sytuacji, w których użytkownik wprowadza wiele danych podczas trwającego dłuższy czas procesu.




**Rysunek 5.** Cykl życia aktywności aplikacji Android. Źródło: [5]

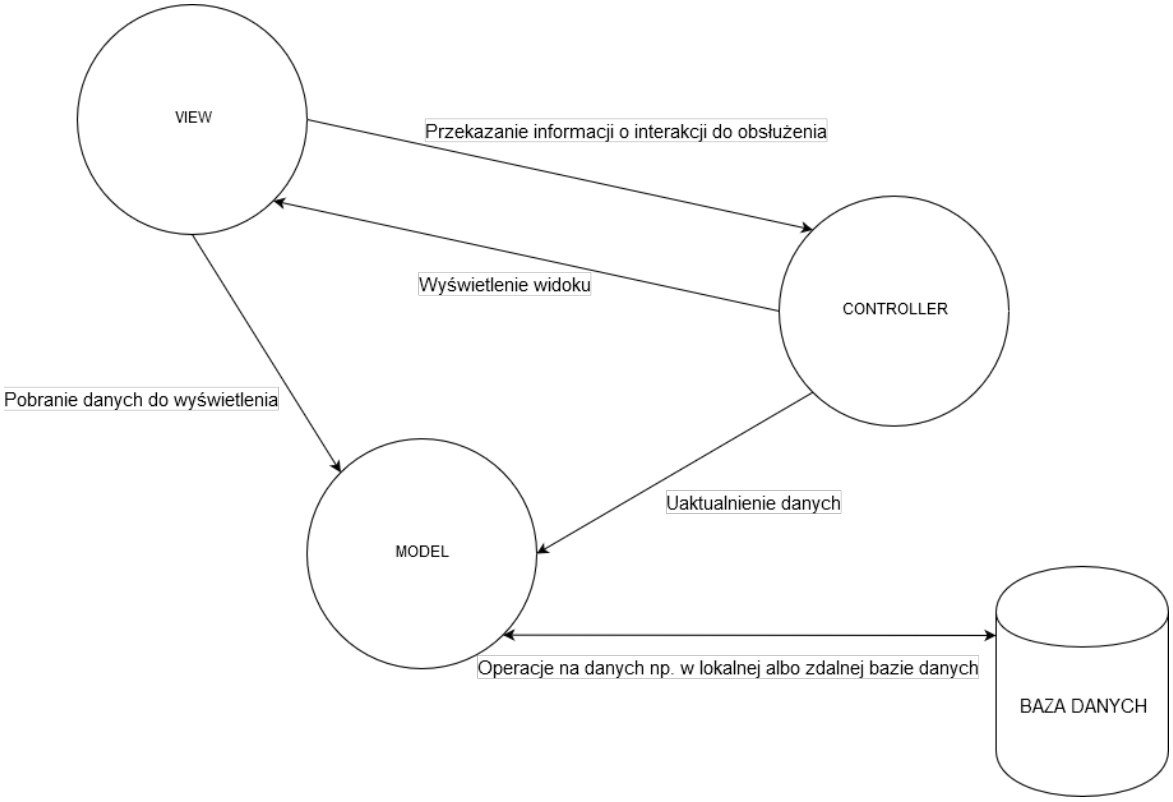
## 2.5. Architektura rozwiązania – aplikacja mobilna

### 2.5.1. Ogólny podział na moduły

Aplikacja jest klasycznym przykładem programu klienckiego pobierającego dane do wyświetlenia ze zdalnego źródła. Wyodrębnić będzie więc można moduły odpowiedzialne za dostęp do danych zdalnych (serwera), przechowywanie danych lokalnych, prezentację treści i poszczególnych ustawień, kontrolę działania usługi w tle.

Aplikację zrealizowano we wzorcu Model View Controller, który wydaje się właściwym wyborem jak dla tego typu aplikacji. Poniżej przedstawiono pokrótce założenia koncepcyjne tego wzorca.

Jak pokazano na Rysunku 6, według MVC, aplikacja dzieli się na trzy implementacyjnie odrębne części, każda z nich może składać się z wielu klas czy plików. Ważnym jest, by żadna klasa nie wchodziła w skład więcej niż jednej wspomnianej części. Poszczególne części powinny komunikować się ze sobą w jasno określony sposób, co pozwala na wprowadzanie niezależnie zmian w obrębie każdej części pozostawiając pozostałe bez konieczności ingerencji w nie. Jedynie w wypadku modyfikacji sposobu komunikowania się między sobą poszczególnych części następuje konieczność wprowadzenia zmian do wielu z nich.




**Rysunek 6.** Schemat wzorca Model View Controller

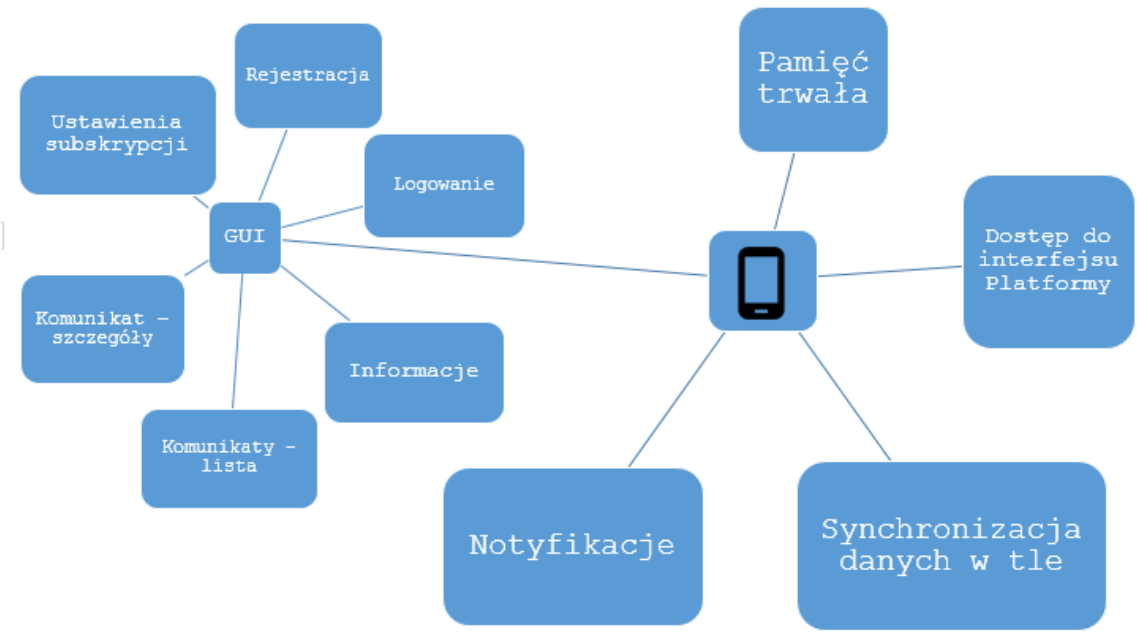
**Model** – część odpowiadająca za przechowywanie i operacje na danych. W rzeczywistości może to być klasa osłonowa pewnej lokalnej bazy danych. W tę część

wpasowuje się także tak zwany Content Provider, sugerowany przez Google sposób dostarczania danych zarówno do widoków danej aplikacji czy jej widżetów działających poza głównym kontekstem, jak i do współdzielenia danych z innymi aplikacjami. Na potrzeby aplikacji będącej przedmiotem tej pracy zastosowano lokalną bazę SQLLite w celu przechowywania części danych. Jej obsługa wykonywana jest za pomocą specjalnie napisanej w tym celu klasy osłonowej. Udostępnia ona „na zewnątrz" gotowe metody pozwalające na wykonywanie odpowiednich operacji na danych. Podejście takie pozwala zaoszczędzić na złożoności kodu ponieważ pozwala na implementację obsługi tylko niezbędnych operacji dokładnie w taki sposób, jak jest to wymagane scenariuszami użytkowania i architekturą aplikacji.

**View** – część odpowiadająca za warstwę prezentacyjną, czyli graficzny interfejs użytkownika. Obsługuje odpowiednie widoki w zależności od aktualnego stanu aplikacji oraz odbiera interakcje użytkownika, obsługując je lub w wypadku bardziej złożonych kwestii przekazując informacje o interakcji do części Controller'a.

**Controller** – część odpowiadająca za logikę biznesową aplikacji. Zarządza zarówno warstwą prezentacyjną, jak i danymi. To w niej znajdują się komponenty odpowiedzialne za kluczowe funkcje aplikacji, takie jak synchronizacja z serwerem. Poza tym w tej właśnie części zapadają decyzje co do wyświetlania notyfikacji, resetowania, rozstrzygania obecnego stanu aplikacji.

W ramach powyższego wzorca aplikację podzielono na moduły pokazane na Rysunku 7. Każdy z nich enkapsuluje pewną funkcjonalność, co pozwala na ich łatwy rozwój niezależnie od pozostałych.




**Rysunek 7.** Podział aplikacji na moduły

### 2.5.2. Moduł dostępu do interfejsu Platformy

Aplikacja korzysta ze specjalnie wystawionego w tym celu interfejsu na serwerze Platformy. Jest to interfejs REST, do którego ma się dostęp poprzez odpowiednie żądania http wysłane pod zadany adres. Cechami tego typu interfejsu widocznymi w tym rozwiązaniu są bez stanowość oraz podział systemu na klientów i serwer. Taka metoda komunikacji z serwerem jest tutaj idealnym rozwiązaniem, ponieważ jest wystarczająca pod względem dostarczanych funkcjonalności a zarazem pozwala na skorzystanie z wielu istniejących już bibliotek i opracowanych rozwiązań dotyczących tej tematyki.

Poniżej, w celu zobrazowania struktury poszczególnych żądań, przedstawiono realizujące je fragmenty kodu aplikacji korzystającego z biblioteki Retrofit (Źródło: [6]).

**Fragmenty kodu 3.** Fragment kodu służący pobraniu wiadomości nowszych od zadanej daty

```
@POST("api/news/getAfterDate")
@FormUrlEncoded
Call<DownloadMessagesResponse>
        getMessagesList(@Field("uname") String username,
                        @Field("pass") String password,
                        @Field("faculty") int faculty,
                        @Field("latestDate") String latestDate);
```

**Fragmenty kodu 4.** Fragment kodu służący pobraniu listy obecnie dostępnych kanałów

```
@POST("api/sources/getList")
@FormUrlEncoded
Call<GetChannelsResponse>
        getChannelsList(@Field("uname") String username,
                        @Field("pass") String password);
```

**Fragmenty kodu 5.** Fragment kodu służący zmianie stanu subskrypcji danego kanału

```
@POST("api/sources/toggleSubscriptionState")
@FormUrlEncoded
Call<UniversalResponse>
         setChannelSubscription(@Field("uname") String username,
                                @Field("pass") String password,
                                @Field("sourceID") int sourceId,
                                @Field("subscribed")
                                boolean subscribed);
```

**Fragmenty kodu 6.** Fragment kodu służący sprawdzeniu poprawności danych logowania podanych przez użytkownika oraz pobraniu oznaczeń wydziału i grupy studenckiej

```
@POST("logInAndro.php")
@FormUrlEncoded
Call<LogInResponse> logIn(@Field("uname") String username,
                          @Field("pass") String password);
```

**Fragmenty kodu 7.** Fragment kodu służący rejestracji nowego użytkownika Platformy z poziomu aplikacji mobilnej

```
@POST("registerAndro.php")
@FormUrlEncoded
Call<String> register(@Field("login") String login,
                      @Field("email") String email,
                      @Field("password") String password,
                      @Field("faculty") String faculty,
                      @Field("token") String token);
```

**Fragmenty kodu 8.** Fragment kodu służący pobraniu listy ulubionych adresów z serwera przypisanych do danego konta użytkownika. Wykorzystywane przy logowaniu w aplikacji mobilnej.

```
@POST("getAddressListAndro.php")
@FormUrlEncoded
Call<List<AddressResponse>>
            getFavouritesList(@Field("uname") String username,
                              @Field("pass") String password);
```

**Fragmenty kodu 9.** Fragment kodu służący sprawdzeniu aktualnej wartości numeru resetowania. Jest to numer umożliwiający zdalne wymuszenie resetu aplikacji

```
@POST("resetQueryAndro.php")
Call<String> checkResetNumber();
```

Jak można zauważyć, wszystkie wykonywane żądania są typu POST. Cechą tego typu jest fakt, iż parametry przekazywane są jako część <body>, a nie jako część URL, jak to ma miejsce w przypadku typu GET. Połączenie wykonywane jest z wykorzystaniem szyfrowania protokołu Secure Socket Layer, co zapewnia odpowiedni poziom bezpieczeństwa. W celu uwierzytelnienia i autoryzacji danego żądania wykonywanego

przez aplikację, w jego treści zawierane są także login i hasło użytkownika. Znane są obecnie inne sposoby na osiągnięcie wyżej wymienionych kwestii, jak chociażby posługiwanie się przez aplikację wygenerowanym dla niej specjalnie przy pierwszym połączeniu tokenem, jednak na potrzeby realizacji tego projektu poziom zapewnionych zabezpieczeń został uznany za wystarczający. Dodatkowo unika się sesyjności rozwiązania, jak miałoby to miejsce w wypadku użycia tokenu.

Żądanie dotyczące „numeru resetu" nie wymaga danych umożliwiających rozpoznanie użytkownika, ponieważ informacja zwrotna od serwera którą dostarcza powinna być dostępna także instalacjom aplikacji, w których nie dokonano jeszcze logowania.

Do realizacji żądań http wykorzystano bibliotekę Retrofit, ponieważ jest to bardzo popularne rozwiązanie, wykorzystywane a zarazem sprawdzone w setkach projektów zarówno prywatnych jak i komercyjnych. Posiada także aktualne wsparcie twórcy oraz społeczności. Retrofit udostępniany jest pod licencją Apache w wersji 2.0, nadaje się więc doskonale do wykorzystania w opisywanym projekcie .

Biblioteka ta wykorzystuje annotacje by automatycznie mapować odpowiednie zmienne przekazywane jako parametry danej funkcji na parametry wysyłanych żądań. Z drugiej strony, zdefiniowawszy zwracany przez daną funkcję typ (klasę), Retrofit automatycznie wykona parsowanie odpowiedzi na żądanie do właśnie obiektu tego typu (tej klasy). Klasy te są prostymi klasami złożonymi ze zmiennych odpowiednich typów (jak String czy int) także oznaczonych odpowiednimi annotacjami, które odpowiadają (także nazewnictwem) tym wchodzącym w skład odpowiedzi. Dodatkowo klasy te zawierają gettery, settery oraz metodę toString. W celu utworzenia tych klas skorzystano z narzędzia jsonschema2pojo (http://www.jsonschema2pojo.org/), które po podaniu pełnej treści odpowiedzi zwracanej przez API serwera automatycznie generuje kod odpowiednich klas, które to mogą być od razu zastosowane m.in. właśnie z biblioteką Retrofit. Fragment kodu jednej z nich zamieszczono poniżej:

**Fragmenty kodu 10.** Fragment kodu klasy odpowiadającej za strukturę wiadomości w odpowiedziach API serwera. Dla uproszczenia zawarto tu tylko dwie zmienne

```
public class OneMessage {
    @SerializedName("title")
    @Expose
    private String title;

    @SerializedName("content")
    @Expose
    private String content;
```

```
public String getTitle() {
    return title;
}

public void setTitle(String title) {
    this.title = title;
}

public String getContent() {
    return content;
}

public void setContent(String content) {
    this.content = content;
}

@Override
public String toString() {
    return "OneMessage{" +
            "title='" + title + '\'' +
            ", content='" + content + '\'' +
            ", link='" + link + '\'' +
            ", source='" + source + '\'' +
            ", date='" + date + '\'' +
            ", id=" + id +
            '}';
}
```

Annotacja @SerializedName służy określeniu nazwy zmiennej wykorzystywanej przy serializacji i deserializacji obiektu. Przy mapowaniu obiektu JSON na obiekt Javy mogą wystąpić różnice w nazewnictwie zmiennych, jako parametr tej annotacji można zadać nazwę odpowiadającego danej zmiennej Javy pola w obiekcie JSON:

```
@SerialisedName„("innaNazwa)
```

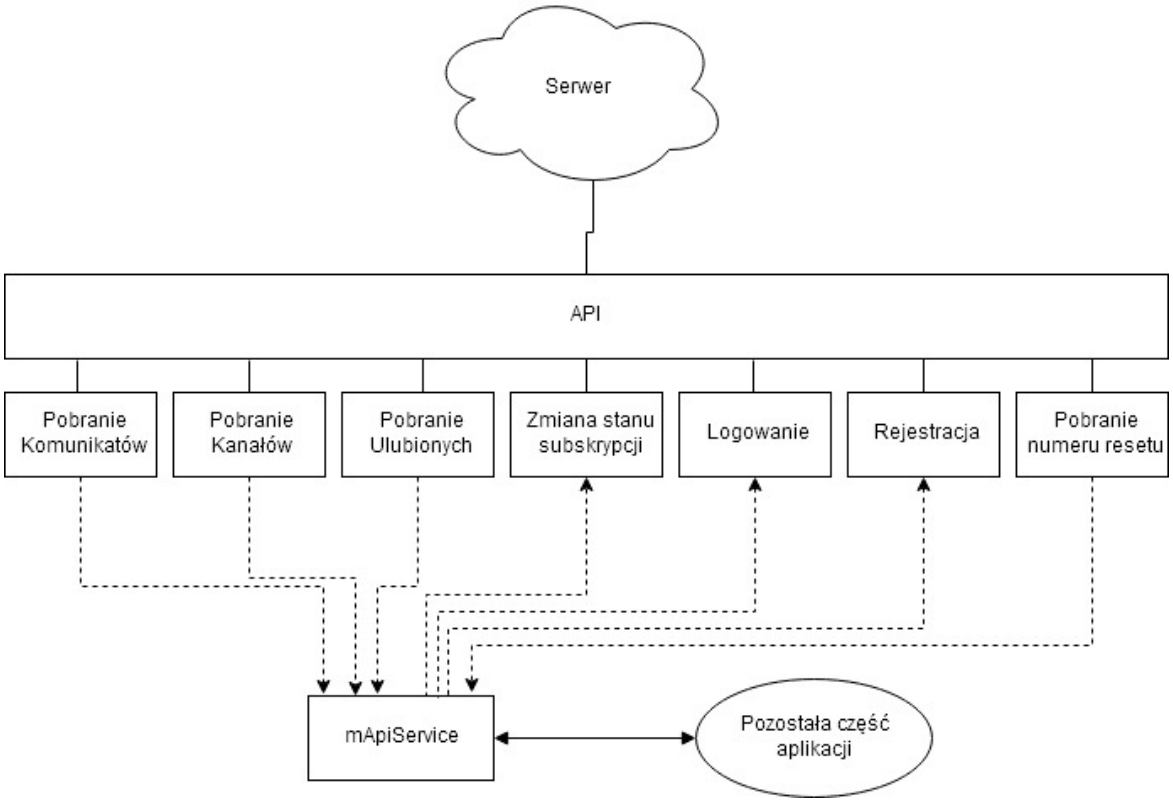
Annotacja @Expose natomiast pozwala zezwolić bądź też nie na serializację i deserializację danej zmiennej.

```
@Expose(serialize = false, deserialize = false)
```

Użyta bez parametrów, domyślnie przyjmuje obie wartości równe 'true'.

Należy pamiętać, że każde połączenie z serwerem ma nieokreślony czas wykonania - zależny od obecnego stanu sieci, serwera oraz urządzenia, na którym uruchomiona jest aplikacja. Wykorzystano więc możliwość Retrofit'a asynchronicznego wykonywania żądań do API serwera na nowym wątku. Zapobiega to blokowaniu głównego wątku graficznego interfejsu użytkownika, co jest jedną z ważniejszych zasad programowania dla platform mobilnych. Po uzyskaniu odpowiedzi, jej przetwarzanie odbywa się z powrotem na wątku głównym, co z kolei pozwala na łatwe skorzystanie z kontekstu i obecnego stanu aplikacji.

Poniżej przedstawiono uproszczony schemat działania obrazujący enkapsulację problemu połączeń z serwerem tak, by jedna klasa mogła być pośrednikiem pomiędzy serwerem a resztą aplikacji umożliwiając tym samym niezależny rozwój sposobu połączenia jak i pozostałych komponentów rozwiązania mobilnego.




**Rysunek 8.** Schemat enkapsulacji połączeń z serwerem w obrębie jednej klasy

### 2.5.3. Moduł pamięci trwałej

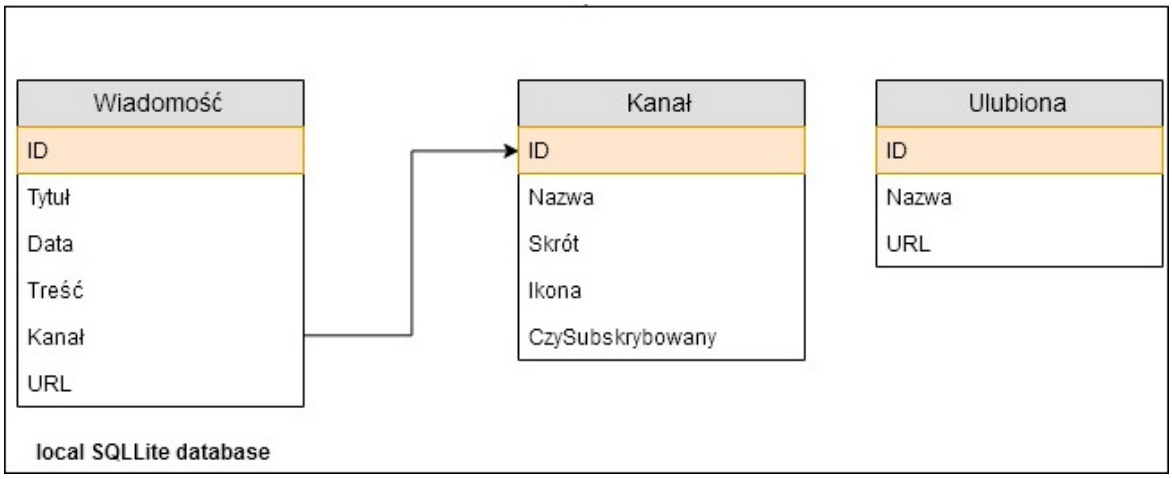
Dane do przechowania trwałego należy podzielić na dwa zestawy. **Pierwszy** z nich to dane występujące w sporej ilości, względnie często aktualizowane lub zmieniane. Są to dane dotyczące komunikatów pobranych z serwera, kanałów, z których pochodzą oraz zapisanych ulubionych adresów www. **Drugim** zestawem są dane niezbędne do funkcjonowania aplikacji, jednak nie często zmieniane i występujące w pojedynczych rekordach, mianowicie nazwa użytkownika (login), hasło użytkownika, oznaczenia wy-

działu i grupy studenckiej przypisanej do użytkownika, jak i pewne zmienne dotyczące resetowania aplikacji czy powiadomień.

Z racji na powyższy podział, wykorzystano dwa sposoby przechowywania danych w pamięci trwałej.

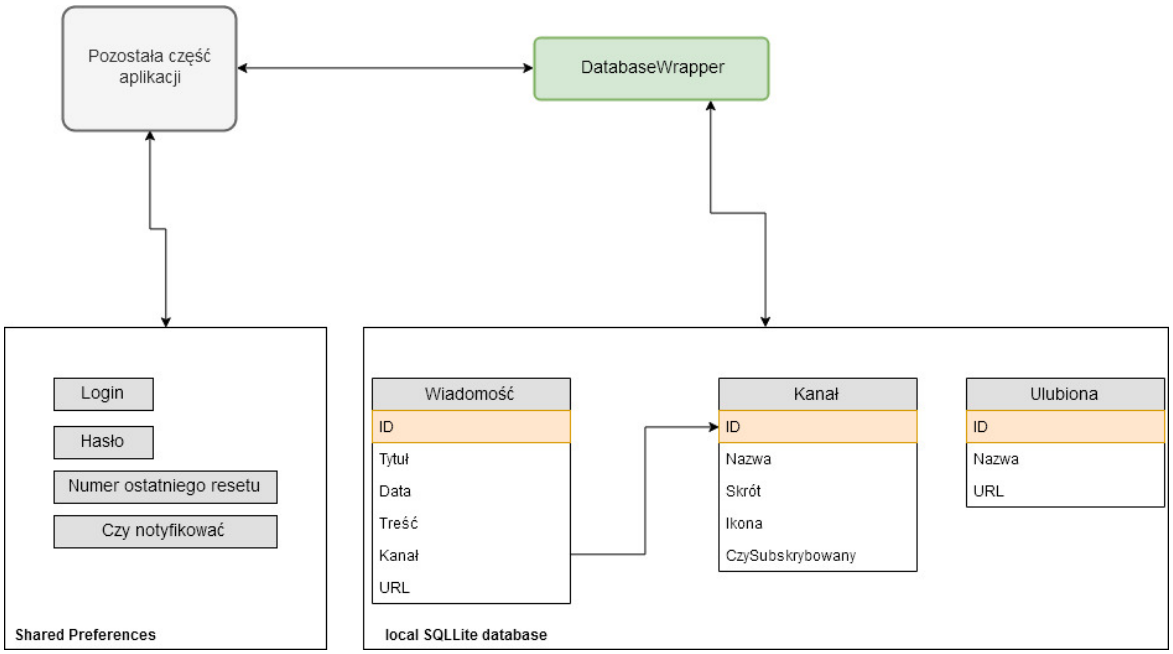
**Zestaw pierwszy** przechowywany jest w lokalnej bazie danych SQLLite.

**Zestaw drugi** przechowywany jest przy użyciu SharedPreferences, będącym prostym natywnym mechanizmem pozwalającym na przechowywanie ustawień poszczególnych aplikacji. Pozwala on na zapis poszczególnych wartości w trybie MODE_PRIVATE, który to ogranicza dostęp do nich tylko wybranej aplikacji dokonującej zapisu. Dane będą więc bezpieczne, a sama obsługa ich odczytu i zapisu będzie względnie łatwa z dowolnego miejsca aplikacji. Operacje na SharedPreferences są także dozwolone do wykonania z głównego wątku aplikacji.




**Rysunek 9.** Schemat encji bazy danych zastosowanej do przechowania **zestawu pierwszego**. Na pomarańczowo zaznaczono klucze główne.

Zdecydowano się na wykorzystanie jednej klasy pośredniczącej w dostępie do danych zestawu pierwszego, będącej swego rodzaju Content Providerem (Źródło: [7]). W związku z wyżej opisanymi wyborami, całość dostępu do danych trwałych wygląda w sposób następujący:




**Rysunek 10.** Schemat dostępu do pamięci trwałej aplikacji.

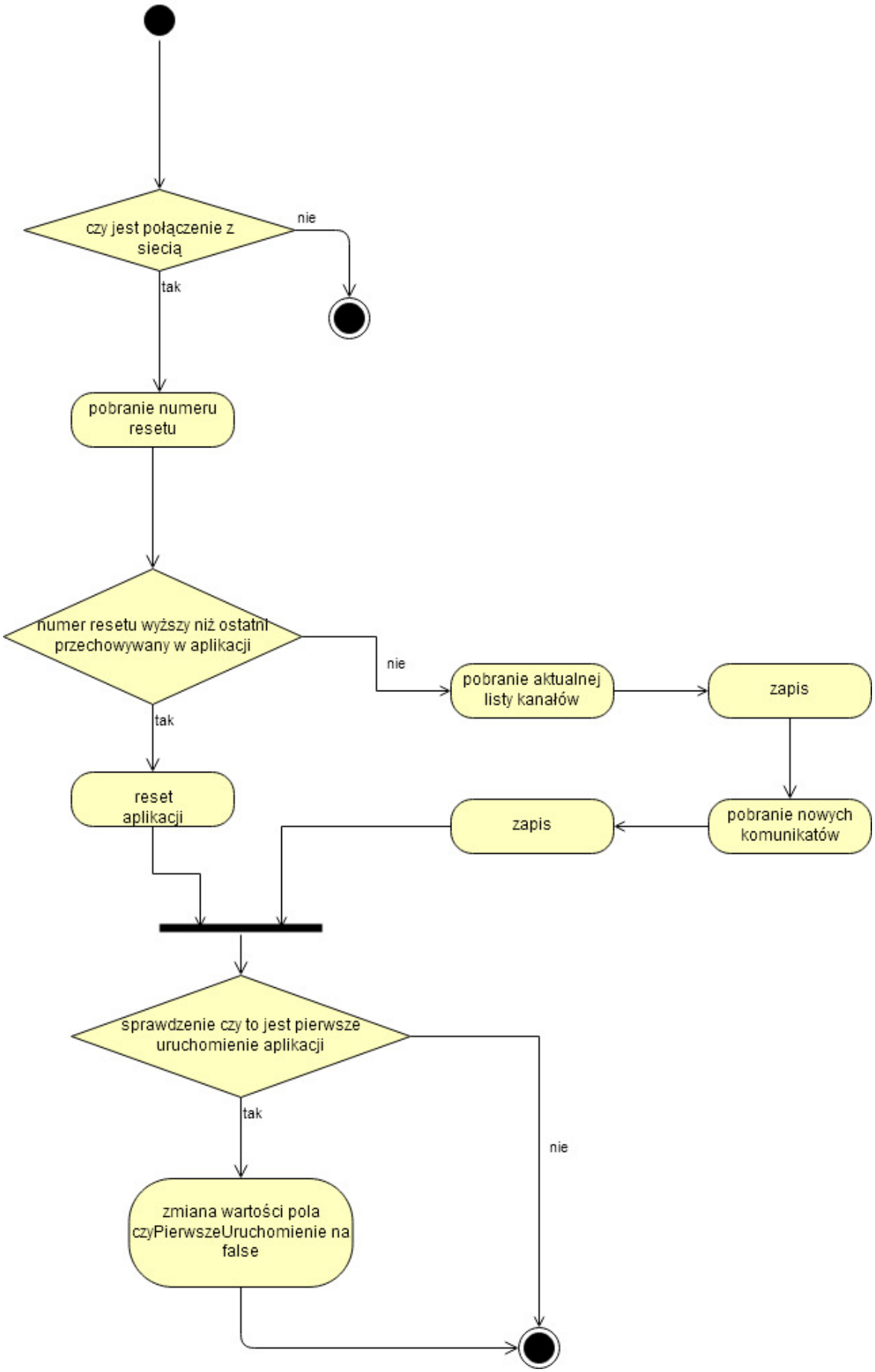
Klasa DatabaseWrapper jest tutaj wspomnianym pośrednikiem w dostępie do bazy danych. Zawiera zestaw metod pozwalających na wykonanie wszystkich niezbędnych do działania aplikacji operacji na bazie danych, to jest:

— Pobranie wszystkich wiadomości
— Pobranie wszystkich zapisanych adresów
— Pobranie wszystkich kanałów
— Pobranie liczby zapisanych adresów
— Pobranie wybranej wiadomości
— Pobranie daty najnowszej wiadomości
— Pobranie danych wybranego kanału
— Pobranie liczby kanałów
— Usunięcie wybranej wiadomości
— Usunięcie wybranego zapisanego adresu
— Usunięcie wybranego kanału
— Usunięcie wszystkich wiadomości
— Usunięcie wszystkich kanałów
— Usunięcie wszystkich zapisanych adresów
— Aktualizacja zapisanego adresu
— Zmiana wartości czySubskrybowany kanału
— Dodanie zapisanego adresu
— Dodanie wiadomości
— Dodanie kanału

Powyższe metody mogą być wywołane z innych klas aplikacji. Rozwiązanie takie pozwala na zapewnienie poziomu abstrakcji pozwalającego na rozwój pozostałej części aplikacji niezależnie od sposobu obsługi bazy danych, czy też ogólnie rzecz biorąc, od sposobu ich przechowywania, tak długo, jak jest spełniony kontrakt co do udostępnianych przez DatabaseWrapper metod operacji na danych. W celu wykonania wyżej wymienionych operacji wykorzystywane są wywołania odpowiednich zapytań języka SQL - niektóre są zdefiniowane ręcznie w kodzie aplikacji, niektóre korzystają ze wzorców dostępnych w metodach klasy SQLiteDatabase (SQLiteDatabase.query() oraz SQLiteDatabase.rawQuery()).

### 2.5.4. Moduł synchronizacji danych w tle

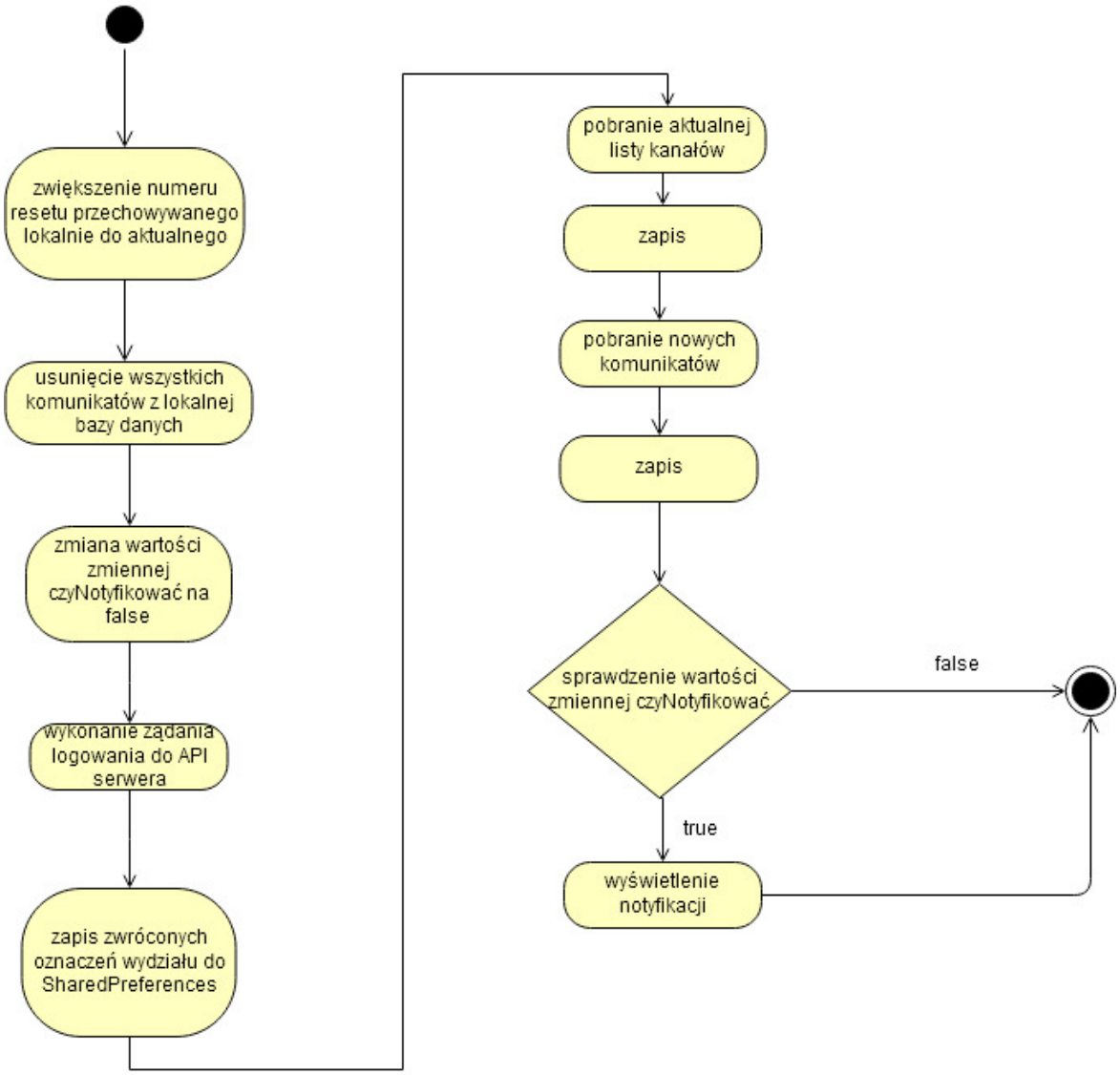
Ważną częścią funkcjonalności aplikacji jest samoczynna synchronizacja w tle z serwerem, to jest z punktu widzenia użytkownika sprawdzanie wystąpienia nowych komunikatów na subskrybowanych kanałach. Aplikacja posiada zaimplementowaną klasę będącą rozszerzeniem klasy Service, która to jest odpowiedzialna za uruchamianie synchronizacji z serwerem parę razy na dobę. Jest to wykonywane przy użyciu klas Timer oraz TimerTask. Zadanie Timera korzysta z klasy SettingsSync w celu wywołania odpowiednich interakcji z serwerem. Zadanie przypisane do Timer'a wykonywane jest w określonym interwale. Działania, w odpowiedniej kolejności, prezentują się w przybliżeniu tak, jak pokazuje Rysunek 11.




**Rysunek 11.** Schemat przebiegu synchronizacji w tle

Powyżej wspomniano wymieniony także w wymaganiach co do funkcjonalności aplikacji mechanizm zdalnego resetowania. Jest on przewidziany na wypadek, gdyby któraś z treści pobranych z serwera powodowała błąd lub też nie była obsługiwana przez starsze wersje aplikacji. Nawet jeśli dana treść zostanie poprawiona w bazie danych na serwerze, zmiana ta nie rozpropaguje się na urządzenia klienckie. Zdalne wymuszenie resetu, tj. wyczyszczenia lokalnej bazy danych, pozwoli uporać się z tym problemem. Mechanizm ten także znajdzie zastosowanie w wypadku obejścia zabezpieczeń serwera lub któregokolwiek ze źródeł podmiotów trzecich i zamieszczeniem w jednym z nich treści nieprawdziwej, niebezpiecznej lub obraźliwej. Jest to jednak czarny scenariusz.

Sam mechanizm działa w prosty sposób. Jest to zamieszczony na serwerze numer, inkrementowany za każdym razem kiedy wystąpi „incydent" i wymagany jest reset aplikacji klienckich. Dana instalacja aplikacji porównuje numer z serwera z jego kopią przechowywaną lokalnie. Jeśli zwiększył się, oznacza to, że musi wykonać reset. Sposób ten pozwala w łatwy sposób rozstrzygnąć czy dana instalacja podlegać ma resetowaniu oraz uniknąć wielokrotnego resetowania, w wypadku gdy lokalny numer dawno nie synchronizowanej instalacji aplikacji różnić się będzie o kilka jednostek.




**Rysunek 12.** Schemat przebiegu zdalnego resetu aplikacji

### 2.5.5. Moduł powiadomień

Jednym z podstawowych założeń aplikacji Platformy jest zdjęcie z użytkownika ciężaru sprawdzania czy pojawiły się nowe komunikaty. Aplikacja korzysta zatem ze standardowych notyfikacji systemu Android w celu powiadomienia użytkownika o pobraniu nowych treści.

Podczas wykonywania synchronizacji danych w tle, opisanego w poprzednim podrozdziale, pobrane zostają tylko komunikaty z datą późniejszą niż najnowszy zapisany obecnie w urządzeniu. Takie podejście gwarantuje, że jeśli pobrane zostały jakiekolwiek treści, są one „nowe", należy więc powiadomić użytkownika.

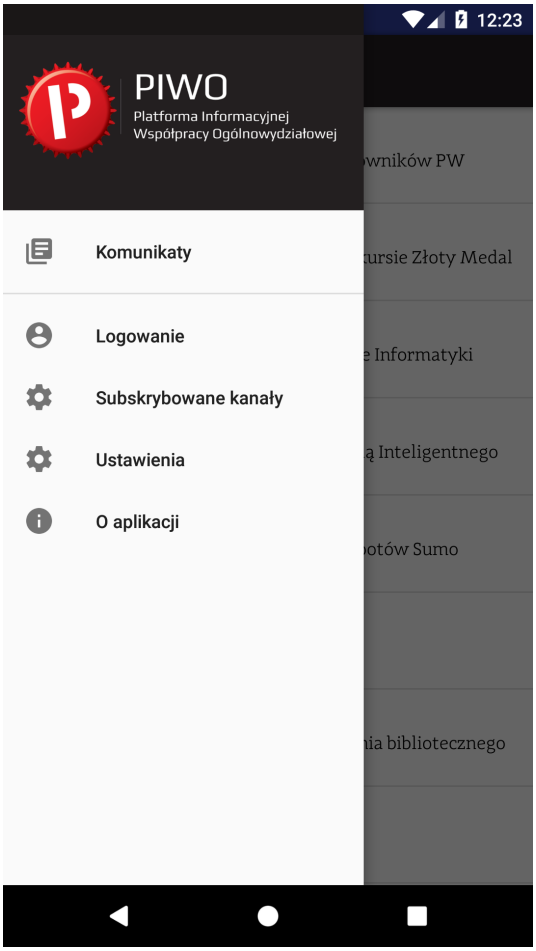
Notyfikacja tworzona jest bezpośrednio po zakończeniu zapisu do lokalnej bazy danych. W tym celu wykorzystywany jest NotificationManager. Znając liczbę pobranych komunikatów, ustawiana jest odpowiednia treść powiadomienia. Wykorzystane są tu flagi:

— onlyAlertOnce – wibracja i dźwięk nie zostaną ponowione, jeśli powiadomienie już się wyświetla. Ma to zastosowanie w sytuacji, gdy od poprzedniej synchronizacji użytkownik jeszcze nie zamknął powiadomienia o nowych treściach, a już pobrane zostały nowe.
— autoCancel – gdy użytkownik dotknie powiadomienia (otwierając tym samym aplikację), zostanie ono automatycznie odwołane (zamknięte).

### 2.5.6. Moduły GUI

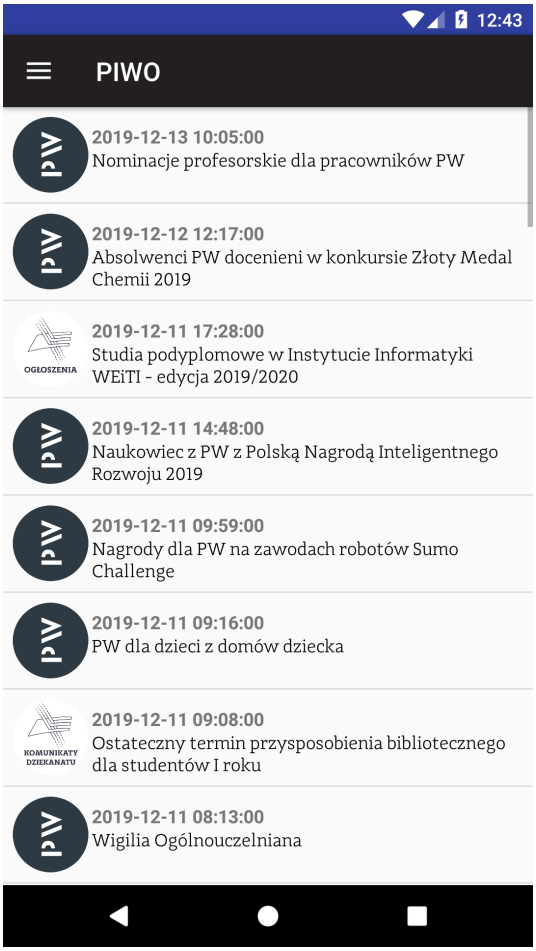
Interfejs graficzny użytkownika został zrealizowany przy użyciu Aktywności oraz Fragmentów. Prawie wszystkie ekrany aplikacji sterowane są przed jedną nadrzędną Aktywność – MainActivity. Ekran rejestracji, z racji na wykonywaną funkcję, jest wyjątkiem, który wyświetlany jest przez własną aktywność. Wykorzystanie Fragmentów pozwala na łatwiejsze zarządzanie nimi, niż w wypadku wykorzystania do obsługi każdego ekranu osobnej aktywności.

W aplikacji zdecydowano się na zastosowanie tzw. „Drawer Menu" (menu wysuwanego z boku aplikacji). Jest to rozwiązanie popularne wśród aplikacji mobilnych, jest więc intuicyjne dla wielu użytkowników (Źródło: [8]). Dodatkowo nie zabiera zbędnego miejsca na ekranie podczas gdy użytkownik go nie potrzebuje. Wybór ten pozwala także na dodawanie w przyszłości kolejnych pozycji w menu, gdy zajdzie taka potrzeba, ponieważ ilość pozycji nie zależy od limitowanego dostępnego miejsca (jak np. w wypadku Navigation Bar). Z tej także racji możliwe jest wyświetlanie zapisanych przez użytkownika zakładek jako oddzielnych, łatwo dostępnych pozycji w menu.




**Rysunek 13.** Boczne menu w aplikacji

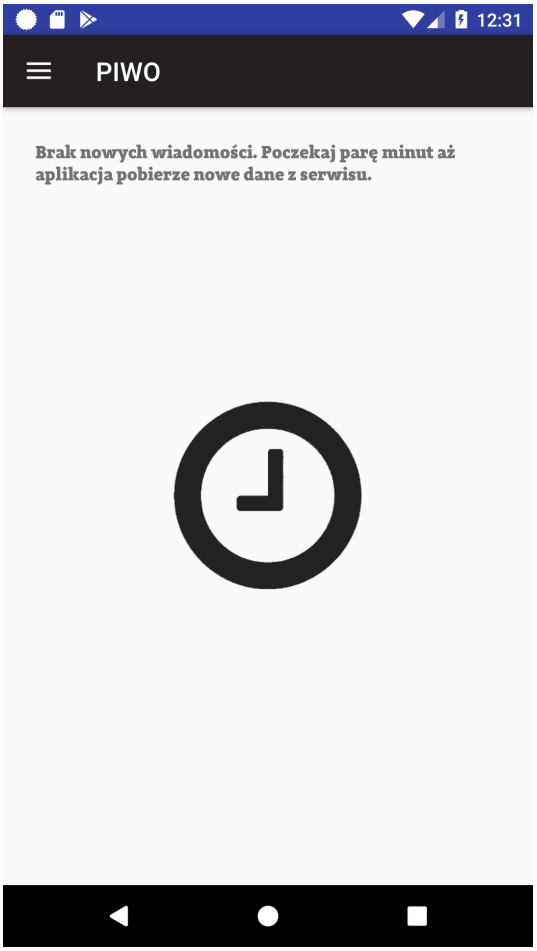
Lista komunikatów zrealizowana jest przy pomocy RecyclerView z ręcznie napisanymi widokami komórek. Jest ona bardziej zaawansowaną wariacją na temat ListView, pozwalającą ominąć potrzebę każdorazowego ponownego rysowania każdej z komórek. Kiedy dana komórka wychodzi poza widoczny na ekranie obszar, jest tymczasowo zachowywana w pamięci. Jeśli stanie się ona ponownie widoczna, zostanie natychmiastowo przywrócona bez konieczności jej tworzenia od zera. Każda komórka zawiera ikonę kanału, z którego pochodzi komunikat, jego tytuł oraz datę publikacji.




**Rysunek 14.** Widok listy komunikatów w aplikacji

Przed pierwszą synchronizacją z serwerem (lub po wykonaniu resetu aplikacji / wyczyszczenia danych aplikacji przez użytkownika z poziomu ustawień systemu) ekran listy wszystkich komunikatów wyświetli odpowiedni komunikat użytkownikowi. Lista komunikatów, obecna w normalnych warunkach, zastąpiona zostanie treścią oraz grafiką informującą o konieczności odczekania do najbliższej synchronizacji. Podobnie stanie się w momencie, gdy użytkownik jeszcze nie zalogował się w aplikacji.

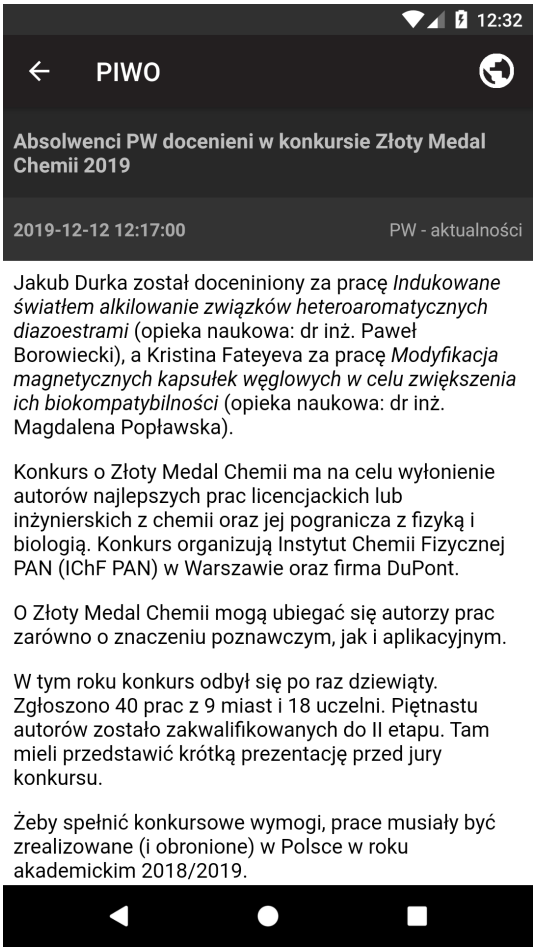

**Rysunek 15.** Widok listy komunikatów przed pierwszą synchronizacją




**Rysunek 16.** Widok listy komunikatów przed logowaniem

Ekran zawierający pełną treść danego komunikatu dzieli się na parę części prezentujących poszczególne informacje, są to:

— data publikacji komunikatu
— tytuł komunikatu
— pełna treść komunikatu
— kanał, do którego przynależy komunikat

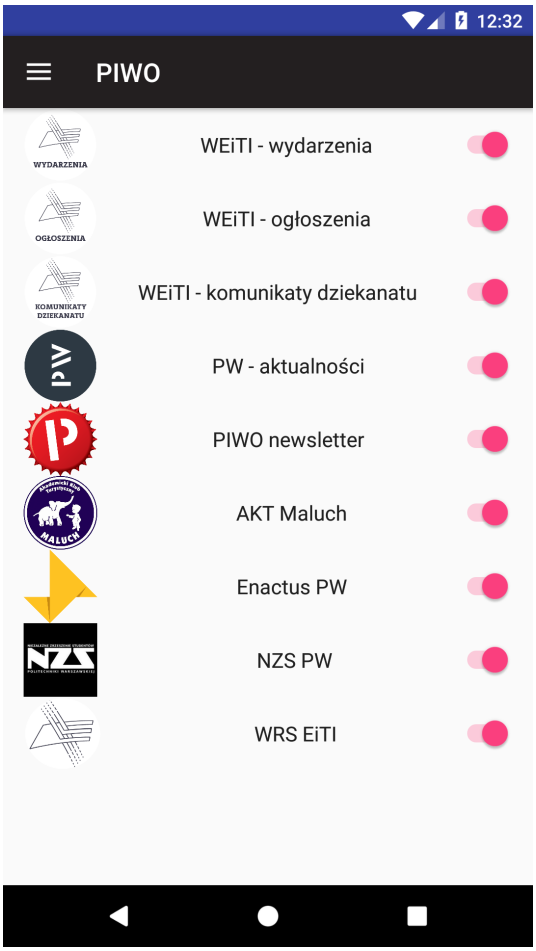



**Rysunek 17.** Widok pojedynczego komunikatu w aplikacji

Dodatkowo widoczny jest przycisk pozwalający na otwarcie strony internetowej, na której oryginalnie znajdował się dany komunikat, w domyślnej przeglądarce internetowej. Treści komunikatów prezentowane są w oryginalnej formie bez zmian ich treści czy formatowania. Także one wyświetlane są za pomocą kontrolki WebView. Warto zauważyć, że w momencie parsowania komunikatów adresy URL obrazków zamieniane są ze względnych na absolutne, pełne ścieżki, by możliwe było poprawne wyświetlanie tych grafik na stronie www oraz w aplikacji Platformy. Dodatkowo usuwane są wszelkie fragmenty JavaScript ze względów bezpieczeństwa oraz z uwagi na ew. problemy z ich wyświetlaniem.

Ekran ustawień pozwalający na subskrypcję (lub jej anulowanie) wybranych kanałów wykorzystuje kontrolki Switch w celu prezentacji użytkownikowi aktualnego stanu subskrypcji. Każdorazowo, gdy użytkownik chce zmienić stan subskrypcji, do serwera wysyłane jest odpowiednie żądanie z wykorzystaniem modułu dostępu do interfejsu Platformy. Jeśli żądanie to nie powiedzie się lub serwer zwróci negatywną odpowiedź, kontrolka zostanie cofnięta do poprzedniego stanu. Jest to ważne z punktu widzenia utrzymania spójnej wersji stanów subskrypcji na urządzeniach końcowych oraz na serwerze. W wypadku braku połączenia z Internetem, wyświetlony zostanie odpowiednie

powiadomienie typu Toast z informacją o tym fakcie. Do wyświetlania listy dostępnych kanałów wykorzystywana jest kontrolka RecyclerView. Jej zalety zostały opisane już wcześniej. Zastosowanie kontrolki w formie przewijanej pionowo listy wynika z możliwości dynamicznego dodawania nowych kanałów do Platformy. Ich ilość w niedługim czasie może wzrosnąć, ich wyświetlenie jednoczesne na jednym ekranie stałoby się niemożliwe.



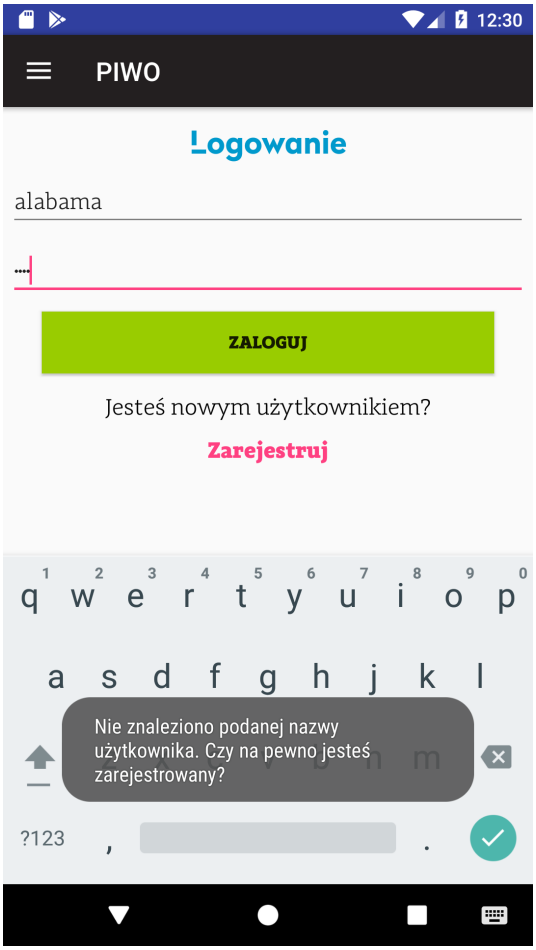

**Rysunek 18.** Widok subskrypcji kanałów

Gdy podczas logowania wystąpi błąd, użytkownikowi wyświetlone zostaną odpowiednie komunikaty za pomocą Toast'a. Poniżej podano niektóre z nich, odpowiadające wybranym sytuacjom problemowym:

— "Musisz podać dane logowania."
— "Niestety, wygląda na to, że nie masz połączenia z Internetem."
— "Nie znaleziono podanej nazwy użytkownika. Czy na pewno jesteś zarejestrowany?"
— "Niepoprawne hasło."
— "Login zawiera niepoprawne znaki."

— ”Konto nie zostało zweryfikowane. Na Twojej skrzynce mailowej powinien znajdować się mail aktywacyjny.”
— ”Niestety, wystąpił błąd po stronie serwera. Spróbuj ponownie później.”
— ”Wystąpił nieokreślony błąd.”

Zdecydowano się na skorzystanie z tej formy wyświetlania komunikatów, ponieważ nie wymaga interakcji w celu zamknięcia, nie zasłania większej części wyświetlanego interfejsu graficznego oraz jest stosunkowo łatwa w użyciu.



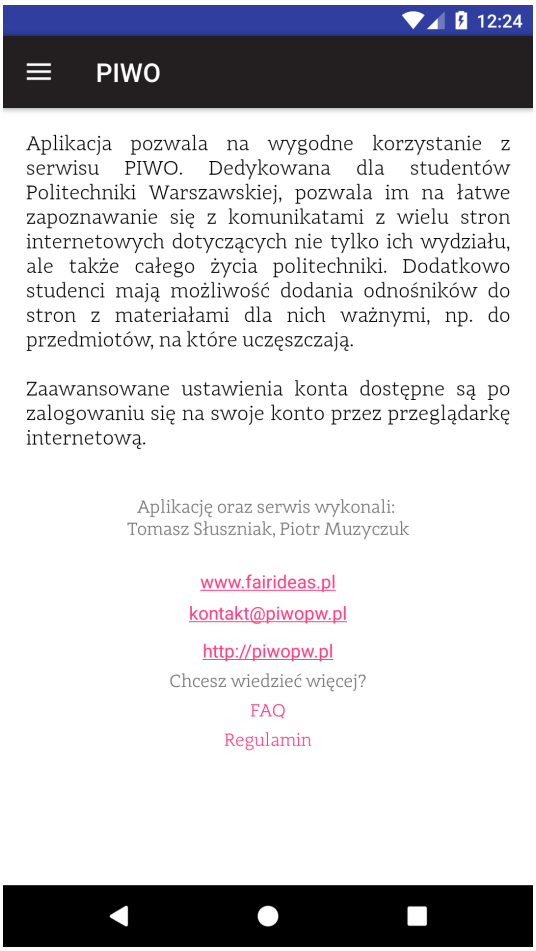

**Rysunek 19.** Błąd podczas logowania

Podobnie wyglądać będzie zachowanie aplikacji przy rejestracji nowego użytkownika. Komunikaty o błędzie lub niepowodzeniu wyświetlone zostaną w analogicznych sytuacjach, biorąc pod uwagę dodatkowo próbę duplikacji już istniejącego loginu / maila w bazie użytkowników.

Wyświetlanie zapisanych przez użytkownika adresów URL wykonywane jest za pomocą standardowej kontrolki WebView osadzonej wewnątrz Fragmentu. Pozwala to na łatwe wyświetlenie wybranej treści strony internetowej wewnątrz aplikacji. Jeśli z jakichś przyczyn nie jest możliwe wyświetlenie danej witryny wewnątrz wspomnia-

nej kontrolki, obsługa adresu URL przekazywana jest do aplikacji będącej domyślną przeglądarką internetową na danym urządzeniu.

Treści tekstowe wyświetlane w elementach GUI są eksternalizowanymi ciągami znaków, zawartymi w pliku strings.xml. Podejście takie pozwala na ich łatwiejszą podmianę, a co więcej na ewentualne tłumaczenie aplikacji na inny język – wystarczy przygotować dodatkowe pliki .xml odpowiadające innym językom i odpowiednio obsłużyć je w aplikacji. Jedyną dłuższą treścią w aplikacji, która nie jest treścią komunikatu, jest opis aplikacji znajdujący się na ekranie dostępnym pod zakładką „O Aplikacji". Tekst ten przechowywany jest w odrębnym pliku HTML, co także pozwala na jego odpowiednią stylizację. Wyświetlanie odbywa się przy użyciu standardowego WebView z ukrytymi kontrolkami.




**Rysunek 20.** Widok informacji o aplikacji

## 2.6. Architektura rozwiązania – interfejs serwera

Serwer udostępnia interfejs umożliwiający aplikacjom klienckim na synchronizację danych dotyczących zalogowanego użytkownika.

Realizacja interfejsu w ten sposób pokazany na Rysunku 8 pozwala na skorzystanie z niego z wszelkiego rodzaju aplikacji klienckich niezależnie od Platformy. Planowany jest rozwój także aplikacji dla systemu iOS, oraz ewentualnie aplikacji dla systemu Linux z myślą o urządzeniach będących częścią ekosystemu Internetu Rzeczy – np. wyświetlacze elektroniczne obecne na Wydziale Elektroniki i Technik Informacyjnych.

Dostęp do opisywanego interfejsu możliwy jest poprzez wysłanie żądań POST pod odpowiedni, publicznie dostępny, URL. Interfejs, jak i całe oprogramowanie serwera, zamieszczone są na zewnętrznym hostingu Digital Ocean. Stawianie własnego serwera wiązałoby się z wieloma niedogodnościami, na przykład globalnej dostępności – mieć na względzie trzeba chociażby ograniczenia niektórych operatorów telekomunikacyjnych oferujących osobom prywatnym dostęp do Internetu. Nie udałoby także się w rozsądnych kosztach i przy rozsądnym wysiłku osiągnąć poziomów niezawodności zbliżonych do tych oferowanych przez dostawców zewnętrznych.

# 3. Scenariusze użytkowe

Końcowa aplikacja spełnia podstawowe postawione przed nią na początku projektu założenia. Udało się pokryć następujące scenariusze użytkowe:

— Nowy użytkownik po raz pierwszy uruchamia aplikację. Nie posiada jeszcze konta w Platformie. W aplikacji korzysta więc z opcji rejestracji nowego konta. Podaje swój adres email, wymyślony login i hasło. Po udanym połączeniu aplikacji z serwerem, na podany adres email zostaje wysłany email z linkiem weryfikacyjnym.
— Użytkownik posiadający już konto w Platformie uruchamia aplikację. Na ekranie logowania podaje swoje dane, login i hasło. Jeśli są one poprawne, aplikacja pobiera ustawienia jego konta z serwera, tj.:
  — Informacje o subskrypcji poszczególnych kanałów przez danego użytkownika
  — Zapisane adresy (zakładki)

  Następnie aplikacja regularnie pobiera nowe komunikaty z serwera.
— Użytkownik będący już zalogowany w aplikacji uruchamia ją w celu przejrzenia listy nowych komunikatów.
— Użytkownik będący zalogowany w aplikacji dostaje powiadomienie o nowych komunikatach z subskrybowanych przez niego kanałów.
— Użytkownik będący zalogowany w aplikacji dodaje nowy adres do zakładek.
— Użytkownik będący zalogowany w aplikacji otwiera jedną z zapisanych zakładek

# 4. Testy

Testowanie rozwijanej aplikacji odbywało się w kilku fazach. Po pierwsze, aplikację testowano na maszynach wirtualnych uruchamianych przez Android Studio. Pozwalało to na testowanie zachowania aplikacji na różnych wersjach systemu Android. Po drugie, równolegle testy odbywały się także na urządzeniu fizycznym, telefonie Huawei P10 Lite. Podczas każdego testu wykonywano serię interakcji mających na celu przetestowanie każdego z modułów.

Powyższe testy przeprowadzane były przez autora aplikacji, mającego szeroką wiedzę na temat implementowanego rozwiązania. Dzięki temu możliwe było odwzorowanie wszystkich sytuacji mogących powodować błędy. Takie podejście jednak nie pozwala przeprowadzić testów pełnej gamy możliwych zachowań użytkownika. Powszechnie znana jest opinia, że osoby pracujące nad danych rozwiązaniem nie powinny być także jego testerami - będą odruchowo wykonywać czynności poprawnie znając ograniczenia danej aplikacji. Dlatego każde oprogramowanie trzeba także testować z pomocą niezależnych testerów lub podgrupy użytkowników końcowych.

Na etapie na którym aplikacja spełniała już podstawowe założenia funkcjonalne oraz pozwalała na to stabilność działania, zdecydowano się na opublikowanie jej w sklepie Google Play. Przed pełnym wdrożeniem wersji produkcyjnej, na każdym kolejnym etapie rozwoju publikowano wersję Beta dostępną w ramach zamkniętego programu testów, a zatem wyłącznie dla wybranych osób.

Osoby posiadające na swoich telefonach testową wersję aplikacji były po pewnym czasie pytane o wszelkie spostrzeżenia na temat aplikacji, które miały one w trakcie jej użytkowania. Spostrzeżenia te można podzielić na trzy kategorie:

— usterki, awarie, błędy mające miejsce podczas użytkowania
— niedogodności związane z użytkowaniem, zarówno pod kątem interfejsu użytkownika jak i ogólnych zasad korzystania z aplikacji
— uwagi, sugestie dotyczące zmian w wersji obecnej lub nowych funkcjonalności mogących zostać wprowadzonymi w przyszłości

Po zebraniu powyższych spostrzeżeń, zadawano każdej z osób serię pytań odnośnie tego, czy udało im się wykonać pewne operacje, które podejrzewane były o ewentualne powodowanie błędów. Jeśli osoby te nie wykonały takowych operacji, proszono je o ich wykonanie i podzielenie się nowymi spostrzeżeniami. Taka kolejność miała na celu zapewnienie, że pierwsze opinie i odczucia użytkowników zebrane w ankietach / wywiadach nie będą zniekształcone przez sugestywne pytania zbierającego odpowiedzi.

Dodatkowo Google Play zbiera logi i informacje o zaistniałych błędach dotyczących testowej instalacji aplikacji z urządzeń biorących udział w testach Beta. Informacje te także były analizowane i niejednokrotnie pozwoliły szybko zidentyfikować przyczynę błędu, który nie został wykryty na wcześniejszym etapie testów.

Po wyeliminowaniu usterek zidentyfikowanych w etapie testów, zdecydowano się na pełne wdrożenie aplikacji. Ponieważ Platforma była mocno promowana w pewnych okresach czasu, udało się zdobyć dość sporą grupę użytkowników – biorąc pod uwagę cały okres funkcjonowania Platformy od publikacji pierwszej wersji aplikacji.

Taka ilość osób niezwiązanych w żaden sposób z autorem aplikacji sama w sobie stanowiła doskonałą grupę testową. Zarówno w aplikacji jak i na jej stronie w Google Play oraz na stronie www Platformy podany był adres email umożliwiający zgłoszenie ewentualnych usterek lub sugestii dotyczących Platformy. Parokrotnie zdarzyło się, iż użytkownicy faktycznie zgłaszali błędy wcześniej nie wykryte, co pozwalało na ich identyfikację i docelowe naprawienie. Spostrzeżenia niekrytyczne uzyskane tą drogą także zostały wzięte pod uwagę przy dalszym rozwoju.

Podsumowując, aplikacja funkcjonowała od dłuższego czasu w warunkach produkcyjnych na urządzeniach różnego typu, należących do różnych użytkowników. Od pewnego czasu odbywa się to bezawaryjnie, co pozwala przypuszczać poprawność jej projektu oraz implementacji.

# 5. Podsumowanie

W ramach niniejszej pracy powstała aplikacja mobilna dedykowana Platformie Informacyjnej Współpracy Ogólnowydziałowej. Prezentuje ona aktualności i wydarzenia pochodzące od różnych podmiotów operujących na uczelni w jednym, przejrzystym i wygodnym w użytkowaniu miejscu. Program ten przeznaczony jest dla urządzeń pod kontrolą systemu operacyjnego Android.

Aplikacja została opublikowana w sklepie Google Play oraz rozpowszechniona wśród pewnej liczby studentów Wydziału Elektroniki i Technik Informacyjnych. Prowadzono kampanię promocyjną zarówno online (grupy na Facebooku, strona internetowa Wydziału - Rysunek 21) jak i standardową za pomocą m.in. plakatów obecnych na Wydziale (Rysunki 22 i 23). Opinie zebrane od użytkowników aplikacji po pewnym czasie użytkowania pozwalają wnioskować, że pomysł przyświecający rozwojowi Platformy jest jak najbardziej trafiony. Dodatkowo warto zauważyć, że użytkowanie w warunkach produkcyjnych póki co potwierdza poprawność dobranej architektury systemu. Na przestrzeni miesięcy wystąpiło kilka usterek, które udało się szybko wyeliminować, natomiast nie były to na szczęście błędy krytyczne.


**Rysunek 21.** Promocja Platformy na stronie internetowej WEiTI

W czasie rozwoju kontaktowano się zarówno z Biurem ds. Promocji i Informacji Politechniki Warszawskiej jak i z dziekanatem Wydziału Elektroniki i Technik Informacyjnych. Platforma spotkała się ogólnie z pozytywnym odbiorem ze strony organów Uczelni.


**Rysunek 22.** Jeden z plakatów promujących Platformę


**Rysunek 23.** Projekt koszulki promującej Platformę

Platforma Informacyjnej Współpracy Ogólnowydziałowej jest rozwiązaniem zaprojektowanym z myślą o skalowalności, co pozwala na jej duży rozwój w przyszłości.

Dotyczy to zarówno funkcji jak i zasięgu systemu. Jeśli chodzi o aplikację zebrano już od obecnych użytkowników oraz testerów zestaw sugestii co do jej wzbogacenia. Biorąc pod uwagę powyższe oraz obecne zaangażowanie autorów, słuszne jest przypuszczać, iż Platforma będzie dynamicznie rozwijana w niedalekiej przyszłości.

# Bibliografia


[1] *Dokumentacja Android Developers*, Dostęp zdalny z dnia 08-08-2019, `https://developer.android.com/about/dashboards`.

[2] *Oficjalna Strona Języka Kotlin*, Dostęp zdalny z dnia 21-05-2019, `https://kotlinlang.org/docs/reference/comparison-to-java.html`.

[3] *Strona Hackernoon.com*, Dostęp zdalny z dnia 20-05-2019: `https://hackernoon.com/\java-vs-kotlin-its-time-to-expandandroid-development-f08e3d6a72b6`, 2019.

[4] J. Friesen, *Java. Przygotowanie do programowania na platformę Android*. 2012.

[5] *Dokumentacja Android Developers*, Dostęp zdalny z dnia 02-09-2019, `https://developer.android.com/guide/components/activities/activity-lifecycle`.

[6] *Dokumentacja biblioteki Retrofit*, Dostęp zdalny z dnia 11-10-2019, `https://square.github.io/retrofit/`.

[7] I. F. Darwin, *Android Receptury*. 2013.

[8] *Android Developer Guide*, Dostęp zdalny z okresu od 08-08-2019 do 29-12-2019, `https://developer.android.com/guide/`.

# Wykaz symboli i skrótów

**WEiTI** – Wydział Elektroniki i Technik Informacyjnych
**PW** – Politechnika Warszawska
**Platforma** – Platforma Informacyjnej Współpracy Ogólnowydziałowej
**PIWO** – Platforma Informacyjnej Współpracy Ogólnowydziałowej

# Spis rysunków

# Fragmenty kodu