**PAŃSTWOWA WYŻSZA SZKOŁA ZAWODOWA**

**W NOWYM SĄCZU**

**INSTYTUT TECHNICZNY**

**PRACA DYPLOMOWA**

**Społecznościowy agregat wiadomości z dedykowanymi filtrami artykułów i mechanizmem współdzielenia**

**Autor: Kmak Karol**

**Kierunek: Informatyka stosowana**

**Nr albumu: 28264**

**Promotor: dr hab. Witold Przygoda**

**Akceptacja promotora: ……………………………………………………**  data i podpis

**NOWY SĄCZ 2022**

Spis treści

[1 Wstęp 4](#_Toc99366564)

[2 Cel i założenia projektu 5](#_Toc99366566)

[2.1 Środowiska programowe 6](#_Toc99366567)

[3 Wykonanie 7](#_Toc99366568)

[3.1 System bazodanowy 8](#_Toc99366569)

[3.2 Strona interntowa do zarządzania agregatami 13](#_Toc99366570)

[3.3 Aplikacja mobilna i obsługa zapytań przez serwer 20](#_Toc99366571)

[4 Testy aplikacji 34](#_Toc99366572)

[4.1 Testy aplikacji mobilnej 34](#_Toc99366573)

[4.2 Testy systemu bazodanowego i narzędzi do zarządzania agregatami 38](#_Toc99366574)

[5 Podsumowanie i wnioski 42](#_Toc99366575)

[6 Bibliografia 43](#_Toc99366576)

[7 Spis rysunków 43](#_Toc99366577)

[8 Załączniki 44](#_Toc99366578)

# Wstęp

Rozwój internetu umożliwiający co raz większej liczbie osób na realizację swoich dziennikarskich pasji spowodował powstanie niezliczonej liczby portali, blogów i innych źródeł, które dla użytkownika nie chcącego poświęcać zbyt wielu godzin na śledzenie różnych portali by móc być na bieżąco z wydarzeniami ze świata stworzył rynek agregatów wiadomości. Są to usługi zbierające artykuły z wielu portali i przedstawiające wybrane artykuły według zainteresowań użytkownika usługi.

Istniejące rozwiązania działają na podstawie jednej z dwóch podstawowych metod. Pierwsza metoda to wykorzystanie algorytmów, które na podstawie zbieranych danych o ilości odwiedzin danej strony, czasie spędzonym na stronie przekazują użytkownikowi artykuły, które są w danym momencie ważne, interesujące i jak najlepiej dopasowane do użytkownika dzięki porównaniu z historycznymi danymi zainteresowań użytkownika. Rozwiązanie to potrafi najczęściej trafnie określić zbiór artykułów mogących zainteresować czytelnika, ale nie jest w stanie rozróżnić informacji nieprawdziwych, czy tzw. clickbaitów i innych artykułów nie przedstawiający pożądanych dla użytkownika wartości.

Drugim z popularnych rozwiązań są artykuły specjalnie wyselekcjonowane przez zajmującą się tym redakcję. Zapewnia to o wiele lepszą jakość artykuł dzięki weryfikacji przez osoby doświadczone, ale wadą tego rozwiązania jest fakt, że pracownicy takiej redakcji nie są w stanie przeanalizować wszelkich możliwych źródeł, oraz mają swoje poglądy, które mogą wpłynąć na proces decyzyjny wybrania, bądź odrzucenia danego artykułu.

Dlatego platforma pozwalająca na wykorzystanie najlepszych cech dominujących rozwiązań, a jednocześnie minimalizująca ich wady jest rozwiązaniem mogącym łatwo zdobyć pozycję na rynku pozwalającą na stabilne finansowanie przynosząc zyski dla właściciela platformy. Społecznościowy agregat wiadomości ma za zadanie oddać poszczególnym użytkownikom, organizacjom możliwość pełnienia funkcji redakcji filtrującej treści dla innych użytkowników. Użytkownik aplikacji będzie miał możliwość wybrania treści zweryfikowanych przez konkretnego, wybranego przez siebie innego użytkownika i/lub za pomocą systemu tagów wybrać zakres zainteresowań, na podstawie którego będą mu przekazywane subskrybowane przez niego treści. W celu zapewnienia użytkownikowi możliwości odfiltrowania treści dla niego mało ważnych, takich jak np. plotki aplikacja posiadać będzie możliwość określenia kilkustopniowego poziomu ważności.



# Cel i założenia projektu

Celem pracy jest stworzenie aplikacji mobilnej systemu Android pozwalającej użytkownikowi przeglądanie listy polecanych artykułów, która będzie pobierana i aktualizowana z serwera bazodanowego. Aplikacja po naciśnięciu przez użytkownika w artykuł otworzy okno domyślnej przeglądarki zainstalowanej na urządzeniu i załaduje wybrany przez użytkownika artykuł. Użytkownik ze specjalnej zakładki będzie mógł przeglądać dostępne agregaty i je zasubskrybować dzięki czemu artykuły z tego agregatu będą się pojawiać w głównej zakładce aplikacji. Aplikacje będzie też oferować możliwość odfiltrowania z listy wyników artykułów nieistotnych za pomocą kilkustopniowej własności ważność każdego artykułu, która będzie ustalana przez zarządzających poszczególnymi agregatami.

W przypadku gdy użytkownik aplikacji nie będzie miał czasu by zapoznać się z treścią artykułu, lub gdy z innych powodów będzie chciał zapisać artykuł do przeczytania w innym terminie otrzymaną taką funkcję, oraz specjalną zakładkę przechowującą zapisane przez użytkownika artykuły.

Drugim celem pracy jest stworzenie lokalnego systemu bazodanowego przechowującego informacje o agregatach, oraz strony internetowej pozwalającej użytkownikom na zarządzanie swoimi agregatami poprzez dodawanie nowych artykułów, zmianę ich właściwości lub usunięcie ich ze swojego agregatu, a także tworzenie nowych agregatów i dodawanie do nich nowych użytkowników czy zmianę ich uprawnień do agregatu.

Gotowa aplikacja ma pozwalać poszczególnym użytkownikom na zarządzanie dowolną liczbą agregatów za pomocą jednego konta, oraz pozwalać by jeden agregat mógł być zarządzany przez dowolną liczbę osób.

# Środowiska programowe

Android Studio – oficjalne otwarto źródłowe środowisko programistyczne dla mobilnych systemów operacyjnych Android. Środowisko oferuje obsługę języków programowania takich jak m.in. Java, C++, Kotlin. Zawiera funkcjonalności takie jak budowanie interfejsu aplikacji za pomocą narzędzia graficznego, zintegrowany emulator pozwalający przetestować działanie aplikacji z poziomu komputera, oraz narzędzia do kontroli kompatybilności aplikacji z poszczególnymi wersjami systemu Android, a także funkcję automatycznego tłumaczenia kodu języka Java na kod w języku Kotlin. Aplikacja będzie pisana w językach Kotlin oraz Java, a do tworzenia interfejsu graficznego wykorzystane będzie narzędzie do graficznego generowania interfejsu oraz język znaczników XML.

Funkcje lokalnego serwera będą tworzone za pomocą USBWebserver, programu pozwalającego na uruchomienie serwera Apache, MySQL, PHP, oraz phpMyAdmin. Pozwala on na bezproblemowe przeniesie i uruchomienie na dowolnym komputerze przez zastosowanie np. pamięci USB bez konieczności instalacji na komputerze z którego jest używany. Program ten został wykorzystany na podstawie łatwości użytkowania, oraz możliwości szybkiego udostępnienia usługi serwera z poziomu dowolnego komputera bez konieczności posiadania uprawnień administratora na tym komputerze.

Aplikacja bazodanowa zostanie stworzona przy użyciu phpMyAdmin/MySQL będącego narzędziem służącym do tworzenia i zarządzania relacyjnymi bazami danych MySQL, oferuje on zarządzanie bazą danych przy użyciu poleceń tekstowych, a także interfejsu graficznego dostępnego z poziomu przeglądarki internetowej.

Strona internetowa zawierająca narzędzia dla zarządzających agregatami zostanie wykonana z pomocą serwera Apache i przy wykorzystaniu języka znaczników HTML, oraz skryptowego języka programowania JavaScript. Obsługa zapytań aplikacji, oraz strony internetowej wykonana będzie z wykorzystaniem języka PHP.

# Wykonanie

Rozdział ten jest poświęcony wykonaniu aplikacji. Podzielony jest na trzy podrozdziały poświęcone odpowiednio systemowi bazodanowemu, stronie internetowej oferującej użytkownikom zarządzanie agregatem i obsługi zapytań aplikacji mobilnej, oraz podrozdział poświęcony wykonaniu aplikacji mobilnej.

Użytkownik aplikacji po jej pierwszym uruchomieniu na swojej stronie głównej ma wyświetlane najważniejsze wiadomości ze wszystkich agregatów, aby rozpocząć właściwe korzystanie z aplikacji powinien przejść do menu „agregaty” i tam wybrać stopień ważności artykułów wyświetlanych z danego agregatu, oraz włączyć dany agregat, lub zmienić stopień ważności, a także zrezygnować z subskrypcji agregatu.

Jeśli użytkownik zasubskrybował przynajmniej jeden agregat to w głównej zakładce aplikacji wyświetlone zostaną tylko te artykuły, które pochodzą z wybranego agregaty, oraz mają wartość ważności taką samą lub wyższą niż ta która została wybrana dla danego agregatu. Wyświetlone artykuły użytkownik może otworzyć, poprzez naciśnięcie na niego, lub zapisać na później za pomocą dedykowanego przycisku. Jeśli użytkownik zapisze artykuł na później, to pojawi się on w zakładce „do przeczytania”, pozostanie on w niej do czasu aż użytkownik odznaczy ten artykuł. Artykuł nie zostanie usunięty z zakładki „do przeczytania” jeśli użytkownik zrezygnuje z subskrypcji agregatu z którego artykuł pochodzi.

Aby zarządzać własnym agregatem użytkownik musi się zarejestrować na stronie internetowej. Na głównej stronie użytkownik wyświetlone zostają ostatnio dodane przez niego artykuły ze wszystkich agregatów którymi zarządza. W zakładce agregatów użytkownik ma wyświetloną listę zarządzanych przez niego agregatów, lub jeśli nie zarządza żadnym agregatem zostaje przekierowany do zakładki tworzenia nowego agregatu. Po wybraniu agregatu z listy wyświetlone zostają ostatnio dodane do niego artykuły, które może zmienić, lub dodać nowy, a także może przejść do osobnej zakładki zarządzania agregatem gdzie będzie mógł dodać, usunąć lub zmienić uprawnienia użytkowników zarządzających agregatem.

# System bazodanowy

Baza danych korzystać będzie z ustandaryzowanego systemu nazewnictwa. Wszelkie nazwy zapisane będą w języku angielskim, który jest standardowym sposobem zapisu w informatyce. Nazwy tabel zapisywane małymi literami w liczbie mnogiej. Nazwy każdej kolumny zaczynać się będą z dużej litery i będą zapisywane w liczbie pojedynczej, w przypadku wystąpienia więcej niż jednego słowa oddzielane one będą znakiem „\_”. Klucz główny będzie zapisany jako ID\_nazwa\_tabeli, gdzie nazwa będzie zapisaną w liczbie pojedynczej nazwą tabeli do której należy. Klucz obcy zapisywany będzie w analogicznej formie, zmienionym o kolejność zapisu słów tak, że nazwa tabeli będzie zapisana jako pierwsza i po niej będzie dodany człon „ID:. Nazwy tabel łączących będą zapisywane stosując nazwy łączonych tabel oddzielając je znakiem „-”, lub inaczej gdy tabela będzie przechowywać dane inne niż te niezbędne do jej funkcjonowania i znacząco rozbudowujące jej funkcjonalność, klucz główny tabeli będzie zapisywany stosując człon „ID\_”, oraz dwie pierwsze litery łączonych tabel oddzielonych znakiem „-”.

Dodawanie wpisów do bazy z poziomu języka PHP wykonywane będzie z wykorzystaniem polecenia: mysql\_query("SET NAMES UTF8")dzięki czemu dane wpisywane do użytkownika w dowolnym języku będą zapisywane i wyświetlane prawidłowo.

## Tabela użytkownicy

Aplikacja w bazie danych będzie przechowywać numer ID użytkownika, nadawany automatycznie w celu identyfikacji przez system, jego nazwę wymyśloną i wybraną przez użytkownika w momencie rejestracji, nazwa musi być unikalna i nie będzie możliwości jej zmiany po zarejestrowaniu konta, adres e-mail w celu zapewnienia możliwości odzyskania hasła, oraz krótki opis. Opis ten jest sposobem by przekazać przez użytkownika krótką notatkę o sobie i może być edytowany w każdej chwili. Pola nazwy użytkownika, hasła oraz adresu e-mail są obowiązkowe. Hasło przechowywane jest w bazie w formie zaszyfrowanego tekstu.

Dodatkowo w przypadku głównego zarządzającego agregatem obowiązkowym jest uzupełnienie danych osobistych, czyli imienia, nazwiska, daty urodzenia, oraz adresu zamieszkania przechowywanych w osobnej tabeli, połączonej tak jak na rysunku poniżej. W przypadku kolumny data zrezygnowano z zasady atomowości danych, w celu uproszczenia struktury bazy danych. Zakładana rzadkość wykorzystywania tych danych nie uzasadnia zwiększenia skomplikowania bazy.

*CREATE TABLE IF NOT EXISTS `users` (*

`ID\_user` **int**(12) **NOT** NULL **AUTO\_INCREMENT**,

`Username` **varchar**(100) **COLLATE** utf8\_polish\_ci **NOT** NULL,

`Password` **varchar**(50) **COLLATE** utf8\_polish\_ci **NOT** NULL,

`Email` **varchar**(100) **COLLATE** utf8\_polish\_ci **NOT** NULL,

`Description` **varchar**(250) **COLLATE** utf8\_polish\_ci **NOT** NULL,

**PRIMARY** **KEY** (`ID\_user`),

**UNIQUE** **KEY** `ID\_users` (`ID\_user`),

**UNIQUE** **KEY** `Username` (`Username`)

) **ENGINE**=InnoDB **DEFAULT** **CHARSET**=utf8 **COLLATE**=utf8\_polish\_ci;

*CREATE TABLE IF NOT EXISTS `users\_data` (*

`ID\_user\_data` **int**(12) **NOT** NULL,

`Name` **varchar**(50) **COLLATE** utf8\_polish\_ci **NOT** NULL,

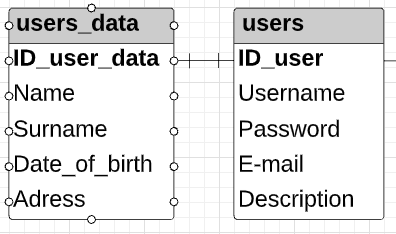
`Surname` **varchar**(50) **COLLATE** utf8\_polish\_ci **NOT** NULL,

`Date\_of\_birth` **date** **NOT** NULL,

`Adress` **varchar**(250) **COLLATE** utf8\_polish\_ci **NOT** NULL,

**PRIMARY** **KEY** (`ID\_user\_data`)

) **ENGINE**=InnoDB **DEFAULT** **CHARSET**=utf8 **COLLATE**=utf8\_polish\_ci;



Rysunek 1. Tabele przechowujące dane użytkownika

## Połączenia użytkowników z zarządzanymi agregatami

Aplikacja powinna umożliwiać zarządzanie poszczególnymi agregatami dla wielu użytkowników, aby zapewnić administratorowi możliwość łatwego i przejrzystego podglądu historii zmian dokonywanych przez każdego z użytkowników mających uprawnienia do zarządzania agregatem. A z racji faktu, że użytkownik może wyrażać chęć zarządzania więcej niż jednym agregatem np. w sytuacji gdy organizacja do której należy podjęła decyzję by tworzyć więcej niż jeden agregat aby zapewnić swoim użytkownikom jak najlepsze dopasowanie do ich preferencji. Z tego powodu koniecznym jest umieszczenie tabeli łączącej, która przechowywać będzie klucze identyfikujące z tabel przechowujących dane użytkowników i agregatów. Dzięki zastosowaniu tabeli łączącej dowolny użytkownik będzie mógł być połączony z dowolną liczbą agregatów, a jednocześnie każdy z tych agregatów będzie mógł być połączony z dowolną liczbą zarządzających nim użytkowników. Połączenie tabel wykonane jest jak na rysunku poniżej.

W tabeli łączącej zawiera się też kolumna definiująca uprawnienia użytkownika do wprowadzania zmian w agregacie. Uprawnienia są zapisywane i odczytywane w sposób binarny, a przechowywane w bazie po zamianie tej liczby na system dziesiętny. Liczba jeden oznacza posiadanie danego uprawnienia, a zero jego brak. Uprawnienia licząc od pozycji najmniejszej do największej to: główny administrator (może być tylko jeden dla danego agregatu), usuwanie administratorów agregatu, zmiana ról, dodawanie nowych administratorów, usuwanie artykułów, wprowadzanie zmian w artykułach, dodawanie artykułów.

*CREATE TABLE IF NOT EXISTS `agregats` (*

`ID\_agregat` **int**(12) **NOT** NULL **AUTO\_INCREMENT**,

`Name` **varchar**(50) **COLLATE** utf8\_polish\_ci **NOT** NULL,

`Description` **varchar**(250) **COLLATE** utf8\_polish\_ci **NOT** NULL,

**PRIMARY** **KEY** (`ID\_agregat`),

**UNIQUE** **KEY** `Name` (`Name`)

) **ENGINE**=InnoDB **DEFAULT** **CHARSET**=utf8 **COLLATE**=utf8\_polish\_ci;

*CREATE TABLE IF NOT EXISTS `users-agregats` (*

`ID\_us-ag` **int**(12) **NOT** NULL **AUTO\_INCREMENT**,

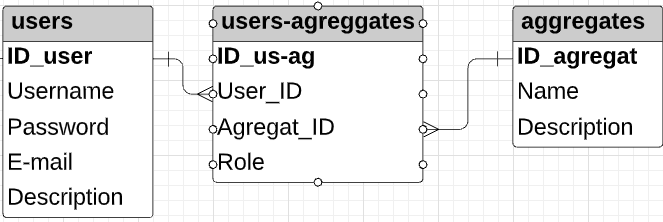
`User\_ID` **int**(12) **NOT** NULL,

`Agregat\_ID` **int**(12) **NOT** NULL,

`Role` **tinyint**(3) **NOT** NULL,

**PRIMARY** **KEY** (`ID\_us-ag`)

) **ENGINE**=InnoDB **DEFAULT** **CHARSET**=utf8 **COLLATE**=utf8\_polish\_ci;



Rysunek 2. Tabele obsługujące połączenie użytkownika z agregatem

## Tabele obsługujące działanie agregatu

Zarządzanie agregatem będzie realizowane za pomocą tabeli „agregats” przechowującej nazwę agregatu, oraz jego krótki opis, tabeli „articles” przechowującej tytuł artykułu, jego adres i pierwsze słowa artykułu mające zachęcić użytkownika do jego przeczytania. Tabele połączone będą tabelą łączącą „sugested\_articles” (rysunek poniżej) zawierającą oprócz kluczy głównych łączonych tabel kolumnę określającą ważność za pomocą wartości liczbowej w zakresie 1-6, gdzie jeden oznacza artykuły o najmniejszej ważności takie jak np. plotki, a sześć oznacza artykuł bardzo ważny. Tabela będzie też zawierać kategorię artykułu, oraz zmienną określającą zmianę ważności artykuły po przejściu do bardziej ogólnej kategorii.

*CREATE TABLE IF NOT EXISTS `articles` (*

`ID\_article` **int**(12) **NOT** NULL **AUTO\_INCREMENT**,

`Adress` **varchar**(255) **COLLATE** utf8\_polish\_ci **NOT** NULL,

`Title` **varchar**(100) **COLLATE** utf8\_polish\_ci **NOT** NULL,

`Description` **varchar**(512) **COLLATE** utf8\_polish\_ci **NOT** NULL,

**PRIMARY** **KEY** (`ID\_article`),

**UNIQUE** **KEY** `ID\_articles` (`ID\_article`),

**KEY** `ID\_articles\_2` (`ID\_article`),

**KEY** `ID\_articles\_3` (`ID\_article`)

) **ENGINE**=InnoDB **DEFAULT** **CHARSET**=utf8 **COLLATE**=utf8\_polish\_ci;

*CREATE TABLE IF NOT EXISTS `sugested\_articles` (*

`ID\_sugested\_article` **int**(12) **NOT** NULL **AUTO\_INCREMENT**,

`Agregat\_ID` **int**(12) **NOT** NULL,

`User\_ID` **int**(12) **NOT** NULL,

`Article\_ID` **int**(12) **NOT** NULL,

`Importance` **int**(1) **NOT** NULL,

`Importance\_change` **float** **NOT** NULL,

**PRIMARY** **KEY** (`ID\_sugested\_article`)

) **ENGINE**=InnoDB **DEFAULT** **CHARSET**=utf8 **COLLATE**=utf8\_polish\_ci;

Obraz zawierający stół

Opis wygenerowany automatycznie

Rysunek 3. Połączenie tabel agregatów i artykułów

## Widoki

W celu uproszczenia kodu mającego wyświetlać zawartość bazy danych użytkownikom użyte zostały widoki grupujące kilka tabel w jedną tabelę wirtualną. Widok „users\_with\_data” zawiera informację o nazwie użytkownika, jego adresie e-mail, opisie konta, oraz o danych osobistych jeśli użytkownik je uzupełnił. Wybranie danych wszystkich użytkowników i tych danych osobistych użytkowników, którzy uzupełnili swoje dane odbywa się z wykorzystaniem komendy LEFT JOIN.

**CREATE** VIEW `users\_with\_data` **AS** **SELECT**

`users`.`Username` **AS** `Username`,

`users`.`Email` **AS** `Email`,

`users`.`Description` **AS** `Description`,

`users\_data`.`Name` **AS** `Name`,

`users\_data`.`Surname` **AS** `Surname`

,`users\_data`.`Date\_of\_birth` **AS** `Date\_of\_birth`,

`users\_data`.`Adress` **AS** `Adress`

**FROM** `users` **LEFT** **JOIN** `users\_data` **ON** `users`.`ID\_user` = `users\_data`.`ID\_user\_data`

Widok „agregat\_users” zawiera nazwę agregatu, jego opis, oraz ID wraz z nazwami użytkowników przypisanych do agregatu, ich rolami, oraz ich ID. Wykonane za pomocą komendy JOIN łącząc tylko te tabele, które mają odpowiadający wpis w drugiej tabeli.

**CREATE** VIEW `agregat\_users` **AS** **SELECT**

`users`.`Username` **AS** `Username`,

`agregats`.`Name` **AS** `Name`,

`agregats`.`Description` **AS** `Description`,

`users-agregats`.`Role` **AS** `Role`,

`agregats`.`ID\_agregat` **AS** `ID\_agregat`,

`users`.`ID\_user` **AS** `ID\_user`

**FROM** ((`users` **JOIN** `users-agregats`) **JOIN** `agregats`) **WHERE** ((`users`.`ID\_user` = `users-agregats`.`User\_ID`) **AND** (`agregats`.`ID\_agregat` = `users-agregats`.`Agregat\_ID`))

Widok „agregat” zawiera tytuł, adres WWW, krótki opis artykułu, oraz informację o stopniu jego ważności i agregacie z jakiego został on dodany wraz z nazwą użytkownika który go dodał. Wykonane za pomocą komendy JOIN łącząc tylko te tabele, które mają odpowiadający wpis w drugiej tabeli.

**CREATE** VIEW `agregat` **AS** **SELECT**

`sugested\_articles`.`Importance` **AS** `Importance`,

`sugested\_articles`.`Importance\_change` **AS** `Importance\_change`,

`agregats`.`Name` **AS** `agregat\_name`,

`agregats`.`Description` **AS** `agregat\_description`,

`articles`.`Adress` **AS** `Adress`,

`articles`.`Title` **AS** `Title`,

`articles`.`Description` **AS** `article\_description`,

`users`.`Username` **AS** `Username`,

`users`.`ID\_user` **AS** `User\_ID`,

`sugested\_articles`.`ID\_sugested\_article ` **AS** `ID\_sugested\_article`,

`agregats`.`ID\_agregat` **AS** `Agregat\_ID`

**FROM** `sugested\_articles` **JOIN** `agregats` **JOIN** `articles` **JOIN** `users` **WHERE** `sugested\_articles`.`Agregat\_ID` = `agregats`.`ID\_agregat` **AND** `sugested\_articles`.`Article\_ID` = `articles`.`ID\_article` **AND** `sugested\_articles`.`User\_ID` = `users`.`ID\_user`

# Strona interntowa do zarządzania agregatami

## Logowanie

Użytkownik rozpoczyna interakcję z narzędziami do zarządzania agregatem na stronie z formularzem logowania. Wypełnienie obydwu pól logowania jest obowiązkowe, a w razie ich niewypełnienia pojawia się komunikat „Wypełnij to pole.” wskazujący na puste pole. Po wypełnieniu danych porównywane są one z istniejącymi wpisami w bazie i w przypadku podania kombinacji danych, których w bazie nie ma użytkownik otrzymuje komunikat o nieprawidłowości danych. W przypadku podania prawidłowych danych uruchomiona zostaje sesja i w danych sesji zapisywane są zmienne „loggedin” z wartością 1, oraz „id” z numerem ID użytkownika, następnie użytkownik zostaje przeniesiony na główną stronę swojego konta. Z strony głównej użytkownik może też przejść do formularza rejestracyjnego poprzez przycisk „Zarejestruj się”.

$q = "select \* from users where Username like '$user' and Password like '$pass'";

$result = mysqli\_query($conn, $q);

**if**(mysqli\_num\_rows($result) > 0)

{

session\_start();

$\_SESSION["loggedin"] = 1;

$row = $result->fetch\_assoc();

$\_SESSION["id"] = $row["ID\_user"];

header("Location: http://localhost/index.php");

}

**else**

{

**echo** "<script>alert('Błędne hasło lub nazwa użytkownika');

window.location.href = '/index.html';</script>";

}

## Rejestracja

Na stronie rejestracyjnej użytkownik wpisuje swój unikalny login, hasło oraz adres e-mail, wypełnienie wszystkich pól jest obowiązkowe, a w razie pominięcia jest wyświetlany komunikat o konieczności uzupełnienia pola, wraz ze wskazaniem pustego pola. Jeśli podany przez użytkownika login jest już używany, to zostaje on o tym poinformowany i musi wybrać nowy. Jeśli nazwa użytkownika się nie powtarza z żadnym innym będącym w bazie to zostaje on zarejestrowany, a następnie tworzona zostaje sesja przechowująca informację o zalogowaniu i ID użytkownika, a następnie zostaje automatycznie przekierowany na główną stronę swojego konta.

$czy\_jest = "select \* from users where Username like '$user'";

$result = mysqli\_query($conn, $czy\_jest);

**if**(mysqli\_num\_rows($result) > 0)

{

**echo** "<script>alert('Nazwa użytkownika jest już zajęta');

window.location.href = '/rejestracja.html'</script>";

}

**else**

{

$q = "INSERT INTO agregat.users (Username,Password,Email) VALUES ('$user','$pass','$email')";

**if**($conn->query($q) === **TRUE**)

{

$q2 = "select \* from users where Username like '$user' and Password like '$pass'";

$result = mysqli\_query($conn, $q2);

session\_start();

$\_SESSION["loggedin"] = 1;

$\_SESSION["id"] = $row["ID\_users"];

header("Location: http://localhost/index.php");

}

}

## Strona główna

Po zalogowaniu użytkownika za pomocą parametru ID zapisanego w nowej sesji jest wysyłane zapytanie do bazy by wyświetlić ostatnie dwadzieścia artykułów dodanych z tego konta użytkownika. Po najechaniu na tytuł artykułu zostaje on podświetlony, a po naciśnięciu użytkownik zostaje przekierowany na adres wybranego artykułu. W przypadku gdy użytkownik nie dodał jeszcze żadnego artykułu pojawia się komunikat zachęty.

W górnej części strony pojawia się menu nawigacyjne pozwalające użytkownikowi przejść do zakładek zarządzania agregatem, zmiany podstawowych danych konta, oraz zakładka zmiany danych osobistych. Elementy menu zostają podświetlone po najechaniu nie.

$q = "select Title,article\_description,Adress from agregat where User\_ID like '".$\_SESSION["id"]."' ORDER BY ID\_sugested\_article DESC LIMIT 20";

$result = mysqli\_query($conn, $q);

**if** ($result->num\_rows > 0)

{

**while**($row = $result->fetch\_assoc())

{

**echo** "<div class = 'article'>";

**echo** "<h3 onclick = 'link(\"".$row['Adress']."\")'>".$row["Title"]."</h3>";

**echo** $row["article\_description"];

}

}

**Else**

{

**echo** "Brak artykułów. Może coś dodaj?";

}

## Strona „Moje agregaty”

Po przejściu do zakładki „Moje agregaty” zostaje wyświetlona lista agregatów z którymi jest powiązany użytkownik, oraz opcja stworzenia nowego agregatu, w przypadku braku żadnego agregatu, użytkownik jest automatycznie przekierowywany do formularza tworzenia nowego agregatu.

$q = "select Name, ID\_agregat from agregat\_users where ID\_user = '".$\_SESSION["id"]."'";

$result = mysqli\_query($conn, $q);

**if** ($result->num\_rows > 0)

{

**while**($row = $result->fetch\_assoc())

{

**echo** "<div class = 'agregat' id = '".$row["ID\_agregat"]."' onclick = 'agregat(".$row["ID\_agregat"].",0)''>".$row["Name"]."</div>";

}

}

**else**

{

header("Location:/nowy\_agregat.html");

}

Po naciśnięciu na agregat uruchomiona zostaje funkcja, która za pomocą ukrytego formularza wyświetla listę pięćdziesięciu artykułów ostatnio dodanych do wybranego agregatu pozwalająca na podejrzenie zawartości polecanego artykułu, oraz jego zmianę, a także podświetla nazwę tego agregatu dając użytkownikowi wizualną informację który agregat jest aktualnie wybrany. Udostępniona zostaje też opcja dodania nowego artykułu, oraz zarządzania agregatem. Po wybraniu innego agregatu zostaje podświetlona nazwa nowego agregatu i wczytana lista artykułów powiązanych z nowo wybranym agregatem.

**echo** "<div class=\"articles\">";

$q2 = "select Title,article\_description,Adress,ID\_sugested\_article from agregat where Agregat\_ID like '".$id."' ORDER BY ID\_sugested\_article DESC LIMIT 50";

$result2 = mysqli\_query($conn, $q2);

**if** ($result2->num\_rows > 0)

{

**while**($row = $result2->fetch\_assoc())

{

**echo** "<div class = 'article'>";

**echo** "<h3 onclick = 'link(\"".$row['Adress']."\")'>".$row["Title"]."</h3>";

**echo** $row["article\_description"];

**echo** "</br><button type=\"button\" onclick = \"zmien(".$row["ID\_sugested\_article"].")\">Zmień</button>";

}

}

**echo** "</div>";

## Dodawania i zmiana artykułu

Po naciśnięciu przycisku „Dodaj nowy artykuł”, lub „Zmień” pod wybranym artykułem za pomocą ukrytego formularza przesyłany jest numer ID agregatu i numer ID artykułu jeśli wybrana została opcja zmiany już istniejącego artykułu, numery to zostają przesłane do pól formularza dodawania artykułu.

**if**($\_POST["id\_art"] != "")

{

$id\_article = $\_POST["id\_art"];

$id\_agregat = $\_POST["id\_agr"];

**echo** "<script>zmien('".$id\_article."','".$id\_agregat."')</script>";

}

**else**

{

$id\_agregat = $\_POST["id\_agr"];

**echo** "<script>nowy('".$id\_agregat."')</script>";

}

Formularz dodawania i zmiany artykułu zawiera pola tytułu, adresu, krótkiego opisu artykułu, jego ważności, oraz zmiany ważności artykułu. Jeśli pobrana wartość pola ID artykułu nie będzie pusta, to wykonany zostanie kod odpowiedzialny za edycję istniejącego wpisu.

$q1 = "UPDATE articles SET Adress = '$adres', Title = '$tytul', Description = '$opis' WHERE ID\_article = '$id\_art'";

$q2 = "UPDATE sugested\_articles SET Importance = '$impor', Importance\_change = '$impor\_chan' WHERE Article\_ID = '$id\_art' AND Agregat\_ID = '$id\_agr'";

**if** ($conn->query($q1) && $conn->query($q2) == **TRUE**)

**echo** "Record updated successfully";

**else**

**echo** "Error updating record: " . $conn->error;

Jeśli wartość pola ID artykułu była pusta to do bazy wpisywana jest zawartość tabeli „articles”, z której następnie pobierany jest numer ID właśnie dodanego artykułu i za jego pomocą dodawany jest wpis do tabeli „sugested\_articles”.

$q1 = "INSERT INTO agregat.articles (Adress,Title,Description) VALUES ('$adres','$tytul','$opis')";

$result1 = mysqli\_query($conn, $q1);

$q2 = "select \* from articles where Adress like '$adres' AND Title like '$tytul' AND Description like '$opis'";

$result2 = mysqli\_query($conn, $q2);

**if** ($result2->num\_rows > 0)

{

**while**($row = $result2->fetch\_assoc())

{

$ID\_article = $row["ID\_article"];

*//pobranie ID dodanego artykułu*

}

**if**(ID\_article != "")

{

$q3 = "INSERT INTO agregat.sugested\_articles (Agregat\_ID,User\_ID,Article\_ID,Importance,Importance\_change) VALUES ('$id\_agr','".$\_SESSION["id"]."','$ID\_article','$impor','$impor\_chan')";

$result3 = mysqli\_query($conn, $q3);

header("Location:/index.php");

}

**else**

**echo** "<script>alert('Nie udało się dodać artykułu')</script>";

}

**else**

**echo** "błąd";

## Zarządzanie agregatem

Po wejściu do zakładki zarządzania agregatem użytkownikowi zostaje wyświetlona lista użytkowników agregatu. Właściciel agregatu ma pełne uprawnienia i nie można zmienić jego uprawnień. Uprawnienia pozostałych użytkowników wyświetlane są w postaci „chek-boxu”, czyli listy, której poszczególne elementy można zaznać bądź odznaczać. Po załadowaniu tej strony zaznaczone są te uprawnienia użytkownika, które posiada a uprawnienia nieposiadane są odznaczone. By zmienić uprawnienia należy zaznaczyć te pola uprawnień jakie mają użytkownikowi być przyznane i nacisnąć przycisk „ustaw uprawnienia”.

$q2 = "select \* from agregat\_users where ID\_agregat like '$agregat\_ID'";

$result = mysqli\_query($conn, $q2);

**while**($row = $result->fetch\_assoc())

{

**echo** "<p>".$row["Username"];

*//Wyświetlanie statusu właściciela agregatu*

**if**($row["Role"]>=100)

**echo** " - Właściciel";

**else**

{

*//Wyświetlanie statusu administratora agregatu*

**if**(($row["Role"]%100)>=10)

**echo** "</br><input type='checkbox' name='admin' value='TRUE' id='".$row["ID\_user"]."\_ad' checked>";

**else**

**echo** "</br><input type='checkbox' name='admin' v alue='TRUE' id='".$row["ID\_user"]."\_ad'>";

**echo** "<label for='admin'>Admnistrator</label>";

*//Wyświetlanie statusu redaktora agregatu*

**if**(($row["Role"]%10)>=1)

**echo** "</br><input type='checkbox' name='redaktor' value='TRUE' id='".$row["ID\_user"]."\_red' checked>";

**else**

**echo** "</br><input type='checkbox' name='redaktor' value='TRUE' id='".$row["ID\_user"]."\_red'>";

**echo** "<label for='redaktor'>Redaktor</label></br>";

**echo** " <button onclick= zmien(".$row["ID\_user"].",1)>ustaw uprawnienia</button>";

**echo** " <button onclick= zmien(".$row["ID\_user"].",2)>usuń użytkownika</button>";

**echo** "</p>";

}

}

Z poziomu tej strony można też usunąć użytkownika, a za pomocą przycisku „Dodaj użytkownika” otwiera się formularz zawierający pole do wpisania nazwy użytkownika, oraz opcja zaznaczenia uprawnień przyznawanych użytkownikowi.

$rola = 0;

$admin = $\_POST["admin"];

**if** ($\_POST['admin'] == 'TRUE')

$rola = 10;

$redaktor = $\_POST["redaktor"];

**if** ($\_POST['redaktor'] == 'TRUE')

$rola += 1;

$czy\_jest = "select \* from users where Username like '$user'";

$result = mysqli\_query($conn, $czy\_jest);

**if**(mysqli\_num\_rows($result) > 0)

{

**while**($row = $result->fetch\_assoc())

{

$user = $row["ID\_user"];

}

$q = "INSERT INTO agregat.`users-agregats` (User\_ID,Agregat\_ID,Role) VALUES ('$user','".$\_SESSION["agregat"]."','$rola')";

$result2 = mysqli\_query($conn, $q);

}

## Zmiana podstawowych danych konta użytkownika

W podstronie „Zmiana podstawowych danych” użytkownik ma możliwość zmiany swojego adresu e-mail, swojego hasła, oraz dodać nowy opis swojego konta. Zmiany są zapisywane tylko do tych pól, które zostają uzupełniane pozostawiając bez zmian wszelkie dane, których użytkownik zmieniać nie chce.

**if**($pass != "")

{

$q1 = "UPDATE users SET Password = '$pass' WHERE users.ID\_user = ".$\_SESSION["id"];

**if** ($conn->query($q1) === **TRUE**)

{

**echo** "Record updated successfully";

}

}

**if**($mail != "")

{

$q1 = "UPDATE users SET Email = '$mail' WHERE users.ID\_user = ".$\_SESSION["id"];

**if** ($conn->query($q1) === **TRUE**)

{

**echo** "Record updated successfully";

}

}

**if**($desc != "")

{

$q1 = "UPDATE users SET Description = '$desc' WHERE users.ID\_user = ".$\_SESSION["id"];

**if** ($conn->query($q1) === **TRUE**)

{

**echo** "Record updated successfully";

}

}

## Dodanie danych osobistych użytkownika

W zakładce ustawień osobistych użytkownik może dodać swoje imię, nazwisko, datę urodzenia, oraz miejsce zamieszkania. Strona ta nie oferuje możliwości późniejszej zmiany wprowadzonych przez użytkownika danych.

$czy\_istnieje = "select \* from users\_data where ID\_user\_data like '$\_SESSION[id]'";

$result = mysqli\_query($conn, $czy\_istnieje);

**if**(mysqli\_num\_rows($result) > 0)

{

**echo** "<script>window.location.href = \"/index.php\";</script>";

}

**else**

{

$q = "INSERT INTO agregat.users\_data (ID\_user\_data,Name,Surname,Date\_of\_birth,Adress) VALUES ('$\_SESSION[id]','$name','$surn','$date','$adres')";

**if**($conn->query($q) === **TRUE**)

{

**echo** "string";

}

**else**

{

**echo** "błąd";

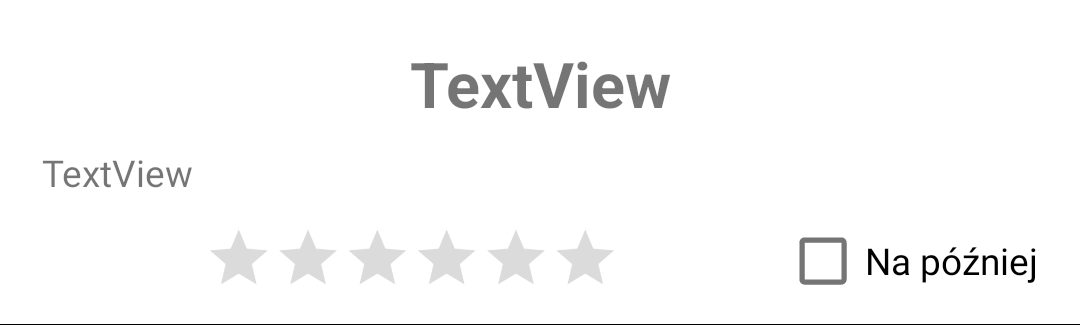
}

}

# Aplikacja mobilna i obsługa zapytań przez serwer

## Strona główna

Użytkownik rozpoczyna używanie aplikacji z poziomu aktywności „MainActivity”, zawiera ona menu nawigacyjne utworzone za pomocą kontenera „BottomNavigationView”. Główną część aktywności zajmuje „RecyclerView” pozwalający na tworzenie i wyświetlanie dynamicznej listy elementów. „RecyclerView” obsługujący główną stronę wczytuje layout „article\_row.xml”, zawiera on pola tekstowe przechowujące tytuł, oraz krótki wstęp do artykułu. Pod nimi znajduje się pasek oceny ważności artykułu pokazujący użytkownikowi to jak ważny powinien dany artykuł, a obok niego znajduje się przycisk służący do zapisania danego artykułu na później. Na samym dole znajduje się wąska linia mająca na celu oddzielać od siebie poszczególne artykuły.



Rysunek 4. Layout definiujący sposób prezentowania artykułów

Do „RecyclerView” przypisywany jest Linear Layout Manager odpowiadający za wyświetlanie zawartości w formie listy elementów „article\_row”. Następnie wywoływane są funkcje zmien(), oraz fetchJson().

Funkcja fetchJson ma za zadanie pobrać treść strony /app/index.php z adresu serwera. Jeśli istnieje to pobierana jest zawartość przechowywanego lokalnie na urządzeniu w którym zainstalowana pliku SharedPreferences, plik ten oferuje przechowywanie lokalnie prostych typów danych takich jak Int, Float, String itd. Pobierana z niego jest wartość zmiennej „table\_size” przechowującej liczbę aktualnie subskrybowanych agregatów, liczba ta następnie jest zapisywana w zmiennej „i”, jeśli zmienna „table\_size” nie istnieje, to do zmiennej „i” przypisywana jest wartość 0.

**private** **fun** fetchJson()

{

**val** table: SharedPreferences = **this**.getSharedPreferences("agregat\_table", MODE\_PRIVATE)

**var** agregaty = ""

**var** i = table.getInt("table\_size", 0)

Jeśli zmienna „i” jest większa niż zero to do zmiennej tekstowej „agregaty” wpisywana jest część zapytania SQL przechowująca część warunku filtrującego wyświetlane artykuły, dzięki czemu aplikacja nie wyświetla artykułów z agregatów, które nie są subskrybowane przez użytkownika, lub te artykuły których stopień ważności jest niższy niż wybrany przez użytkownika. Przy użyciu pętli WHILE do treści warunku zapisywane są wszystkie agregaty zasubskrybowane przez użytkownika.

**while** (i > 0){

agregaty += "(`agregat\_name` LIKE \"" + (table.getString("agregat\_$i", "").toString()) + "\" AND `Importance` >= " + table.getInt("importance\_$i", 0) + ")"

println("Ważność: " + table.getInt("importance\_$i", 0))

i--

**if** (table.getString("agregat\_"+(i), "") != ""){agregaty+= " OR "}

**else**{**break**}

}

println("Warunek: "+agregaty)

Następnie tworzona jest treść żądania HTML, zawierająca zakodowany warunek SQL, za pomocą biblioteki OkHttp wysyłane jest utworzone żądanie na adres serwera. Następnie za pomocą biblioteki GSon tworzony jest obiekt przechowujący pobrane z serwera artykuły, oraz utworzony zostaje obiekt klasy „MainAdatper”, która ma na celu prawidłowe wyświetlanie pobranych z serwera danych. W razie wystąpienia błędów tworzony jest komunikat zawierający treść błędu.

**val** formBody:RequestBody = FormBody.Builder()

.add("agregat", agregaty).build()

**val** url = "http://192.168.1.10/app/index.php"

**val** request = Request.Builder().url(url).post(formBody).build()

**val** client = OkHttpClient()

client.newCall(request).enqueue(object: Callback{

**override** **fun** onResponse(call: Call?, response: Response?) {

**val** body = response?.body()?.string()

**val** gson = GsonBuilder().create()

**val** homeFeed = gson.fromJson(body,Array<HomeFeed>::class.java)

**val** recyclerViewMain: RecyclerView =

findViewById(R.id.recyclerView\_main)

runOnUiThread {

recyclerViewMain.adapter = MainAdapter(homeFeed, **this**@MainActivity)

}

}

**override** **fun** onFailure(call: Call, e: IOException) {

runOnUiThread {

Toast.makeText(applicationContext, e.message, Toast.LENGTH\_LONG).show()

}}})}

## Wyświetlanie artykułów z bazy

Jeśli użytkownik nie zasubskrybował żadnego agregatu to do strony wysyłającej dane do aplikacji wysyłana zostaje pusta treść warunku i wykonywany jest kod wyświetlający z bazy tylko te artykuły, których stopień ważności jest wyższy lub równy 5 sprawiając tym samym, że użytkownik, który nie zasubskrybował jeszcze żadnego agregatu otrzyma tylko najważniejsze artykuły ze wszystkich agregatów. Jeśli użytkownik zasubskrybował jeden lub więcej agregat to wysłany zostaje warunek do odfiltrowania tylko tych treści, które użytkownik chce otrzymać. Otrzymane dane z bazy są następnie zapisywane do tabeli, a to tabela za pomocą funkcji „json\_encode()” zostaje przekazana aplikacji.

$conn = **new** mysqli("localhost","root","usbw","agregat");

**if** (mysqli\_connect\_errno())

{

**echo** "Failed to connect to MySQL: " . mysqli\_connect\_error();

**die**();

}

error\_reporting(**E\_ERROR** | **E\_WARNING** | **E\_PARSE**);

$test = $\_POST["agregat"];

**if**($test == ""){

$stmt = $conn->prepare("SELECT Adress,Title,article\_description, Importance FROM agregat WHERE Importance > 4;");

}

**else**{

$stmt = $conn->prepare("SELECT Adress,Title,article\_description, Importance FROM agregat WHERE $test;");

}

$stmt->execute();

$stmt->bind\_result($Adress, $Title, $article\_description, $Importance);

$articles = **array**();

**while**($stmt->fetch())

{

$temp = **array**();

$temp['Adress'] = $Adress;

$temp['Title'] = $Title;

$temp['article\_description'] = $article\_description;

$temp['Importance'] = $Importance;

array\_push($articles, $temp);

}

**echo** json\_encode($articles);

## Klasa MainAdpater

Klasa „MainAdapter” w celu prawidłowego działania dziedziczy po klasie „RecyclerView”. Zawiera w sobie metodę zwracającą liczbę elementów wyświetlanych za pomocą „RecyclerView”.

**class** MainAdapter(**private** **val** homeFeed: Array<HomeFeed>, **private** **val** context: Context): RecyclerView.Adapter<CustomViewHolder>() {

**override** **fun** getItemCount(): Int{

**return** homeFeed.count()

}

Metoda „onCreateViewHolder” ma na celu utworzyć layout dla poszczególnych elementów listy, layout pobierany jest z pliku „article\_row”. Do tego celu wykorzystywany „LayoutInflater”, do którego przesyłany „article\_row”.

**override** **fun** onCreateViewHolder(parent: ViewGroup, viewType: Int): CustomViewHolder {

**val** layoutInflater = LayoutInflater.from(parent.context)

**val** cellForRow = layoutInflater.inflate(R.layout.article\_row, parent, **false**)

**return** CustomViewHolder(cellForRow, **null**, context)

}

Metoda „onBindViewHolder” zmienia zawartość layoutu „article\_row” na kolejne pobrane z bazy artykuły przypisując wartości przechowywane w „homeFeed” do odpowiedniego obiektu „holder”, wskazującego na poszczególne elementy widoku.

**override** **fun** onBindViewHolder(holder: CustomViewHolder, position: Int) {

**val** article = homeFeed[position]

holder.itemView.findViewById<TextView>(R.id.textView\_article\_title).text = article.Title

holder.itemView.findViewById<TextView> (R.id.textView\_article\_description).text = article.article\_description

holder.itemView.findViewById<RatingBar>(R.id.ratingBar).rating = article.Importance.toFloat()

holder.itemView.findViewById<RatingBar>(R.id.ratingBar).isEnabled = **false**

holder.url = article.Adress

Następnie sprawdza czy dany artykuł został zapisany pobierając plik SharedPrefernces i sprawdzając czy jest w nim zapisany artykuł o tym tytule, jeśli jest zapisany to zaznacza odpowiednie pole pokazując użytkownikowi, że dany artykuł został przez niego zapisany na później.

**val** pref: SharedPreferences = context.getSharedPreferences("articles", Context.MODE\_PRIVATE)

**if** (pref.getString("title\_${article.Title}", **null**) == article.Title){

holder.itemView.findViewById<CheckBox>(R.id.checkBox).isChecked = **true**

}

Do przycisku „na później” dołączone zostaje nasłuchiwanie czy przycisk został naciśnięty, a następnie w zależności od tego jaka została przypisana wartość do tego przycisku wywołana zostaje funkcja zapisywania, lub usuwania artykułu z listy na później.

holder.itemView.findViewById<CheckBox>(R.id.checkBox).setOnClickListener{

**if** (holder.itemView.findViewById<CheckBox>(R.id.checkBox).isChecked){

holder.save(article.Titleholder.itemView.findViewById<CheckBox> (R.id.checkBox).isChecked, article.article\_description, article.Adress, article.Importance)

}

**else**{

holder.save(article.Title, holder.itemView.findViewById<CheckBox>(R.id.checkBox).isChecked)

}

Klasa „CustomViewHolder” przechowuje poszczególne widoki będące częścią „RecyclerView”. W momencie inicjalizacji tworzy nasłuchiwanie naciśnięcia artykułu i jeśli zostanie on naciśnięty to tworzy nową aktywność i przesyła do niej adres internetowy wybranego artykułu.

**class** CustomViewHolder(**private** **val** view: View, **var** url:String? = **null**, **private** **val** context: Context): RecyclerView.ViewHolder(view) {

companion object {

const **val** link = "article\_link"

}

init {

view.setOnClickListener {

**val** intent = Intent(view.context, ArticlePage::class.java)

intent.putExtra(link, url)

view.context.startActivity(intent)

}

}

Zapisywanie danych odbywa się za pomocą dwóch plików SharedPreferences. Plik „articles” przechowuje tylko tytuł artykułu, służy on do szybkiego odnalezienia czy dany artykuł został zapisany do przeczytania później. Plik „articles\_table” przechowuje tytuł, opis, adres strony, stopień jej ważności, oraz ilość zapisanych artykułów.

Metoda „save()” sprawdza jaka została przekazana jej wartość przycisku „na później”, jeśli przekazana została wartość „true”, to w pliku SharedPreferences o nazwie „articles” zapisywany jest tytuł artykułu i wywoływana jest funkcja „table()” z odpowiednimi parametrami. Jeśli przycisk na później przekazał wartość „false”, to przeszukiwany jest plik SharedPreferences o nazwie „articles\_table” by uzyskać pozycję w tabeli odpowiadającego artykułu, usunięta zostaje zmienna przechowująca tytuł i wywołana zostaje funkcja „table()”

**fun** save(title: String, switch: Boolean?, desc: String? = "",

url: String? = "", imp: Int = 0) {

**val** pref: SharedPreferences = context.getSharedPreferences("articles", Context.MODE\_PRIVATE)

**val** editor = pref.edit()

**if** (switch != **null** && switch == **true**) {

editor.apply {

putString("title\_$title", title)

}.apply()

table(**true**, title, pref.all.size, desc, url, imp)

Toast.makeText(context, "Zapisano $title", Toast.LENGTH\_SHORT).show()

}

**else** {

**val** table: SharedPreferences = context.getSharedPreferences("articles\_table", Context.MODE\_PRIVATE)

**var** i = 1

**while** (table.getString("title\_$i", "") != title){

i++

}

table(**false**, title, i)

editor.remove("title\_$title").apply()

Toast.makeText(context, "Usunięto $title", Toast.LENGTH\_SHORT).show()

}

}

Funkcja „table()” jest główną funkcją obsługującą zapisywanie artykułów, sprawdza czy przekazano jej wartość „true”, jeśli tak, to zapisuje tytuł, opis, adres, oraz ważność artykułu do zmiennych zapisywanych według wzoru nazwa zmiennej + pozycja w tabeli, a następnie aktualizowana jest zmienna przechowująca rozmiar tabeli.

**private** **fun** table(boolean: Boolean, title: String, rozmiar: Int, desc: String? = "error-MainAdapter", url: String? = "error-MainAdapter", imp: Int = 0){

**val** table: SharedPreferences = context.getSharedPreferences("articles\_table",

Context.MODE\_PRIVATE)

**val** editor = table.edit()

**if** (boolean){

println("pozycja: $rozmiar")

editor.apply {

putString("title\_$rozmiar", title)

putString("desc\_$rozmiar", desc)

putString("url\_$rozmiar", url)

putInt("imp\_$rozmiar", imp)

putInt("table\_size", rozmiar)

}.apply()

println("rozmiar tabeli: "+table.all.size)

}

Jeśli do funkcji „table()” przekazana została wartość „false” to funkcja sprawdza czy dany artykuł był zapisany na ostatniej pozycji w tabeli, jeśli tak to zostaje on usunięty i rozmiar tabeli zostaje zmniejszony o 1. Jeśli artykuł był zapisany w środku tabeli, to w jego miejsce zostaje zapisany artykuł z ostatniej pozycji tabeli, a następnie ostatnia pozycja zostaje usunięta i rozmiar tabeli jest zmniejszany o 1.

**else**{

**if**(rozmiar == table.getInt("table\_size", 0)) {

println("usunięto element numer: $rozmiar")

editor.remove("title\_$rozmiar").apply()

editor.remove("desc\_$rozmiar").apply()

editor.remove("url\_$rozmiar").apply()

editor.remove("imp\_$rozmiar").apply()

editor.apply {

putInt("table\_size", table.getInt("table\_size", 0) - 1)

}.apply()

}**else**{

**val** ostatni = table.getInt("table\_size", 0)

**val** tekst = table.getString("title\_$ostatni", "błąd")

**val** opis = table.getString("desc\_$ostatni", "błąd")

**val** adres = table.getString("url\_$ostatni", "błąd")

**val** import = table.getInt("imp\_$ostatni", 0)

println("Usunięto w środku: $ostatni")

editor.apply {

putString("title\_$rozmiar", tekst)

putString("desc\_$rozmiar", opis)

putString("url\_$rozmiar", adres)

putInt("imp\_$rozmiar", import)

putInt("table\_size", ostatni-1)

}.apply()

editor.remove("title\_$ostatni").apply()

editor.remove("desc\_$ostatni").apply()

editor.remove("url\_$ostatni").apply()

editor.remove("imp\_$ostatni").apply()

println("rozmiar tabeli po usunięciu: "+table.getInt("table\_size", 0))

}

}

## Menu nawigacyjne

Menu nawigacyjne jest obsługiwane za pomocą funkcji „zmien()”, nasłuchuje ona naciśnięcie przycisków i rozpoczyna aktywność odpowiedniej zakładki.

**private** **fun** zmien(){

**val** navigationView: BottomNavigationView = findViewById(R.id.navigationView)

navigationView.setOnItemSelectedListener {

**when** (it.itemId){

R.id.home -> {

**val** i = Intent(**this**@Agregat, MainActivity::class.java)

startActivity(i)}

R.id.agregat -> {

**val** i = Intent(**this**@Agregat, Agregat::class.java)

startActivity(i)

}

R.id.zapisane -> {

**val** i = Intent(**this**@Agregat, Saved::class.java)

startActivity(i)

}

}

**true**

}

}

## Widok dostępnych agregatów

W zakładce agregaty użytkownikowi wyświetlane są wszystkie aktualnie dostępne agregaty. „RecyclerView” obsługujący widok agregatów wczytuje layout „agregat\_row.xml”, zawiera on pola tekstowe przechowujące nazwę agregatu, jego krótki opis, oraz przycisk pozwalający na jego subskrypcję, lub rezygnację z niej. Pod nimi znajduje się pasek oceny ważności artykułów, które mają być wyświetlane dla danego agregatu, można ją zmienić poprzez wybranie nowej wartości, a następnie wyłączenie i ponowne włączenie subskrypcji agregatu. Na samym dole znajduje się wąska linia mająca na celu oddzielać od siebie poszczególne agregaty.



Rysunek 5. Layout definiujący sposób prezentowania dostępnych agregatów

Widok agregatów działa na analogicznej zasadzie do głównego widoku aplikacji, wywoływana jest funkcja „fetchJson()”, która pobiera z adresu serwera przekazywane jej dane, a następnie te pobrane dane zapisuje do klasy „AgregatFeed”, w razie braku połączenia tworzony jest komunikat z treścią błędu.

**private** **fun** fetchJson(){

**val** url = "http://192.168.1.10/app/agregat.php"

**val** request = Request.Builder().url(url).build()

**val** client = OkHttpClient()

client.newCall(request).enqueue(object: Callback{

**override** **fun** onResponse(call: Call?, response: Response?) {

**val** body = response?.body()?.string()

**val** gson = GsonBuilder().create()

**val** agregatFeed = gson.fromJson(body, Array<AgregatFeed>::class.java)

**val** recyclerViewAgregat: RecyclerView = findViewById(R.id.recyclerView\_agregat)

runOnUiThread {

recyclerViewAgregat.adapter = AgregatAdapter(agregatFeed, **this**@Agregat)

}

}

**override** **fun** onFailure(call: Call, e: IOException) {

runOnUiThread {

Toast.makeText(applicationContext, e.message, Toast.LENGTH\_LONG).show()

}

}

})

}

## Wyświetlanie agregatów z bazy

Jeśli aplikacja chce uzyskać listę dostępnych agregatów, to wysyła żądanie do serwera, który pobiera z bazy dane o wszystkich zapisanych agregatach, zapisuje pozyskane agregaty do tabeli, którą następnie za pomocą funkcji „json\_encode()” przekazuje do aplikacji w formie pliku Json.

$conn = **new** mysqli("localhost","root","usbw","agregat");

**if** (mysqli\_connect\_errno())

{

**echo** "Failed to connect to MySQL: " . mysqli\_connect\_error();

**die**();

}

$stmt = $conn->prepare("SELECT Name, Description FROM agregats;");

$stmt->execute();

$stmt->bind\_result($Name, $Description);

$agregats = **array**();

**while**($stmt->fetch())

{

$temp = **array**();

$temp['Name'] = $Name;

$temp['Description'] = $Description;

array\_push($agregats, $temp);

}

**echo** json\_encode($agregats);

## Klasa AgregatAdapter

Klasa AgregatAdpater analogicznie do MainAdapter zawiera metodę przechowującą rozmiar listy agregatów, funkcję wywoływaną podczas tworzenia nowego elementu listy, która tworzy wygląd elementu, oraz funkcję przypisującą poszczególne wartości do kolejnych elementów listy. Sprawdza czy agregat jest subskrybowany i zaznacza odpowiednie wartości jeśli zostały one zapisane przez użytkownika.

**class** AgregatAdapter(**private** **val** agregatFeed: Array<AgregatFeed>, **private** **val** context: Context): RecyclerView.Adapter<AgregatViewHolder>() {

**override** **fun** getItemCount(): Int{

**return** agregatFeed.count()

}

**override** **fun** onCreateViewHolder(parent: ViewGroup, viewType: Int): AgregatViewHolder {

**val** layoutInflater = LayoutInflater.from(parent.context)

**val** cellForRow = layoutInflater.inflate(R.layout.agregat\_row, parent, **false**)

**return** AgregatViewHolder(cellForRow, context)

}

**override** **fun** onBindViewHolder(holder: AgregatViewHolder, position: Int) {

**val** agregat = agregatFeed[position]

holder.itemView.findViewById<TextView> (R.id.textView\_article\_title).text = agregat.Name

holder.itemView.findViewById<TextView> (R.id.textView\_article\_description).text = agregat.Description

**val** pref: SharedPreferences = context.getSharedPreferences("agregat", MODE\_PRIVATE)

holder.itemView.findViewById<RatingBar>(R.id.ratingBar).rating = pref.getInt("importance\_"+agregat.Name, 0).toFloat()

holder.itemView.findViewById<SwitchCompat>

(R.id.switchAgregat).isChecked = pref.getBoolean("agregat\_"+agregat.Name, **false**)

holder.itemView.findViewById<SwitchCompat> (R.id.switchAgregat).setOnClickListener{

holder.save(agregat.Name, holder.itemView.findViewById<SwitchCompat>(R.id.switchAgregat).isChecked, holder.itemView.findViewById<RatingBar>(R.id.ratingBar).rating.toInt())

}

}

}

Klasa „AgregatViewHolder” analogicznie do odpowiadającej jej klasy głównej strony aplikacji przechowuje widok poszczególnych agregatów. Zawiera ona metodę „save()”, która zapisuje do pliku SharedPreferences o nazwie „agregat” nazwę agregatu i wybrany dla niego stopień ważności artykułów, lub usuwa wybrany wpis z pliku. Metoda ta jest zastosowana dla szybkiego odnajdywania i pokazywania zapisanych przez użytkownika agregatów. Metoda „save()” wywołuje też metodę „table()”, która tworzy tabelę zapisywanych agregatów, która jest dostępna z głównej strony aplikacji i za jej pomocą tworzony jest warunek filtrowania artykułów należących do odpowiednich agregatów i mających odpowiedni stopień ważności.

**fun** save(text: String, switch: Boolean?, imp: Int){

**val** pref: SharedPreferences = context.getSharedPreferences("agregat", MODE\_PRIVATE)

**val** editor = pref.edit()

**if**(switch != **null** && switch ==**true**) {

editor.apply {

putBoolean("agregat\_$text", switch)

putInt("importance\_$text", imp)

}.apply()

table(**true**, text, pref.all.size/2, imp)

Toast.makeText(context, "Zapisano$text$switch", Toast.LENGTH\_SHORT).show()

}

**else**{

**val** table: SharedPreferences = context.getSharedPreferences("agregat\_table", MODE\_PRIVATE)

**var** i = 1

**while** (table.getString("agregat\_$i", "") != text){

i++

}

editor.remove("agregat\_$text").apply()

editor.remove("importance\_$text").apply()

table(**false**, text, i, 0)

Toast.makeText(context, "Usunięto", Toast.LENGTH\_SHORT).show()

}

}

Metoda „table()” w zależności od wysłanych jej atrybutów analogicznie do podobnej funkcji w klasie „MainAdaptre” dodaje do tabeli nowy wpis, lub go usuwa, gdy usuwany jest wpis znajdujący się w środku tabeli to w jego miejsce zapisywany jest wpis z ostatniej pozycji tabeli, a ostatnia pozycja zostaje następnie usunięta.

**private** **fun** table(boolean: Boolean, name: String, rozmiar: Int, imp: Int){

**val** table: SharedPreferences = context.getSharedPreferences("agregat\_table", MODE\_PRIVATE)

**val** editor = table.edit()

**if** (boolean){

println("pozycja: $rozmiar")

editor.apply {

putString("agregat\_$rozmiar", name)

putInt("importance\_$rozmiar", imp)

putInt("table\_size", rozmiar)

}.apply()

println("rozmiar tabeli: "+table.all.size)

}

**else**

{

**if**(rozmiar == table.getInt("table\_size", 1)) {

println("usunięto element numer: $rozmiar")

editor.remove("agregat\_$rozmiar").apply()

editor.remove("importance\_$rozmiar").apply()

editor.apply {

putInt("table\_size", table.getInt("table\_size", 0) - 1)

}.apply()

}

**else**

{

**val** ostatni = table.getInt("table\_size", 0)

**val** tekst = table.getString("agregat\_$ostatni", "")

**val** impor = table.getInt("importance\_$ostatni", 0)

println("Usunięto w środku: $ostatni")

editor.apply {

putString("agregat\_$rozmiar", tekst)

putInt("importance\_$rozmiar", impor)

putInt("table\_size", ostatni-1)

}.apply()

editor.remove("agregat\_$ostatni").apply()

editor.remove("importance\_$ostatni").apply()

println("rozmiar tabeli po usunięciu: " + table.getInt("table\_size", 0))

}

}}

## Widok artykułów zapisanych na później

Widok artykułów zapisanych na później korzysta z tego samego Layoutu, który jest używany w głównej stronie aplikacji. W odróżnieniu od pozostałych stron nie zawiera funkcji „fetchJson()”, zamiast tego jej adapter jest wywoływany bezpośrednio.

## Klasa „SavedAdapter”

Klasa „SavedAdapter” analogicznie do pozostałych klas tworzy listę elementów, rozmiar listy jest pobierany z pliku SharedPrefences o nazwie „articles\_table”. Za pomocą funkcji „onBindViewHolder()” do odpowiednich pól elementów listy zapisanych artykułów są przypisywane odpowiednie, zapisane w pamięci urządzenia dane artykułu.

**class** SavedAdapter(**private** **val** context: Context): RecyclerView.Adapter<SavedViewHolder>(){

**override** **fun** getItemCount(): Int{

**val** pref: SharedPreferences = context.getSharedPreferences("articles\_table", Context.MODE\_PRIVATE)

**return** pref.getInt("table\_size", 0)

}

**override** **fun** onCreateViewHolder(parent: ViewGroup, viewType: Int): SavedViewHolder {

**val** layoutInflater = LayoutInflater.from(parent.context)

**val** cellForRow = layoutInflater.inflate(R.layout.article\_row, parent, **false**)

**return** SavedViewHolder(cellForRow, context)

}

**override** **fun** onBindViewHolder(holder: SavedViewHolder, position: Int) {

**val** pref: SharedPreferences = context.getSharedPreferences("articles\_table", Context.MODE\_PRIVATE)

holder.itemView.findViewById<TextView> (R.id.textView\_article\_title).text = pref.getString("title\_"+(position+1), "error-SavedAdapter")

holder.itemView.findViewById<TextView> (R.id.textView\_article\_description).text = pref.getString("desc\_"+(position+1), "error-SavedAdapter")

holder.url = pref.getString("url\_"+(position+1), "")

holder.itemView.findViewById<RatingBar>(R.id.ratingBar).rating = pref.getInt("imp\_"+(position+1), 3).toFloat()

holder.itemView.findViewById<RatingBar> (R.id.ratingBar).isEnabled=**false**

holder.itemView.findViewById<CheckBox> (R.id.checkBox).isChecked=**true**

holder.itemView.findViewById<CheckBox> (R.id.checkBox).setOnClickListener{

holder.save(

holder.itemView.findViewById<TextView> (R.id.textView\_article\_title).text.toString()

)

}

}}

Klasa „SavedViewHolder” analogicznie do głównej strony w momencie inicjalizacji rozpoczyna nasłuchiwanie naciśnięcia na artykuł i w jego momencie otwiera stronę z wybranym artykułem.

**class** SavedViewHolder(**private** **val** view: View, **private** **val** context: Context, **var** url:String? = **null**): RecyclerView.ViewHolder(view) {

init {

view.setOnClickListener {

**val** intent = Intent(view.context, ArticlePage::class.java)

intent.putExtra(CustomViewHolder.link, url)

view.context.startActivity(intent)

}

}

Funkcje „save()” i „table()” działają identycznie jak w głównej części strony. Zapisują lub usuwają z listy zapisanych artykułów w momencie zaznaczenia lub odznaczenia danego artykułu. Po odznaczeniu, artykuł pozostaje wyświetlany w zakładce zapisanych artykułów do czasu opuszczenia tej zakładki, więc w przypadku przypadkowego odznaczenia artykułu pozostaje możliwość jego ponownego zaznaczenia.

**fun** save(title: String = "") {

**val** pref: SharedPreferences = context.getSharedPreferences("articles", Context.MODE\_PRIVATE)

**val** table: SharedPreferences = context.getSharedPreferences("articles\_table", Context.MODE\_PRIVATE)

**var** i = 1

**while** (table.getString("title\_$i", "") != title){

i++

}

**val** editor = pref.edit()

editor.remove("title\_$title").apply()

table(i)

Toast.makeText(context, "Usunięto", Toast.LENGTH\_SHORT).show()

}

**private** **fun** table(rozmiar: Int){

**val** table: SharedPreferences = context.getSharedPreferences("articles\_table", Context.MODE\_PRIVATE)

**val** editor = table.edit()

**if**(rozmiar == table.getInt("table\_size", 0)){

println("usunięto element numer: $rozmiar")

editor.remove("title\_$rozmiar").apply()

editor.remove("desc\_$rozmiar").apply()

editor.remove("url\_$rozmiar").apply()

editor.apply {

putInt("table\_size", table.getInt("table\_size", 0) - 1)

}.apply()

}

**else**

{

**val** ostatni = table.getInt("table\_size", 0)

**val** tekst = table.getString("title\_$ostatni", "błąd")

**val** opis = table.getString("desc\_$ostatni", "błąd")

**val** adres = table.getString("url\_$ostatni", "błąd")

println("Usunięto w środku: $ostatni")

editor.apply {

putString("title\_$rozmiar", tekst)

putString("desc\_$rozmiar", opis)

putString("url\_$rozmiar", adres)

putInt("table\_size", ostatni-1)

}.apply()

editor.remove("title\_$ostatni").apply()

editor.remove("desc\_$ostatni").apply()

editor.remove("url\_$ostatni").apply()

println("rozmiar tabeli po usunięciu: " + table.getInt("table\_size", 0))

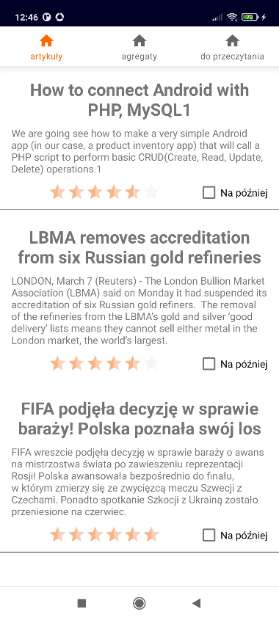
}

}

# Testy aplikacji

# Testy aplikacji mobilnej

Testy działania aplikacji mobilnej przeprowadzono z telefonu Xiaomi POCO M3 PRO zalogowanego do tej samej sieci Wi-Fi do której zalogowany jest komputer z uruchomionym programem USBWebSerwer. Test ma na celu sprawdzenie poprawności działania podstawowych funkcji aplikacji. Test jest rozpoczęty na świeżo zainstalowanej aplikacji. Po jej uruchomieniu, jako że nie jest subskrybowany jeszcze żaden agregat ukazuje się strona zawierająca najważniejsze artykuły ze wszystkich agregatów.



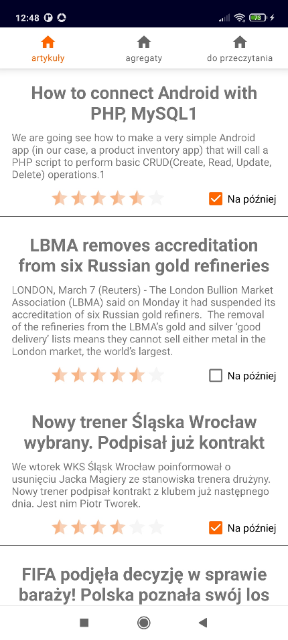
Rysunek 6. Aplikacja po jej pierwszym uruchomieniu

W zakładce agregaty wybrane zostały dwa przykładowe agregaty, by sprawdzić czy wyświetlane są tylko te artykuły, które pochodzą z agregatów zostały zasubskrybowane i czy w sposób prawidłowy odfiltrowane zostają artykuły o zbyt małej ważności.



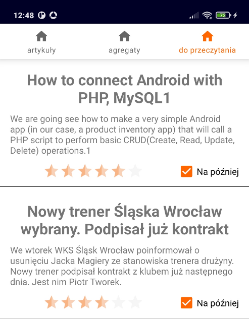
Rysunek 7. Zasubskrybowane agregaty

Po zasubskrybowaniu odpowiednich agregatów wybrane prawidłowo zostały wyświetlone artykuły według wybranych opcji, a następnie wybrano dwa artykuły do zapisania na później.



Rysunek 8. Zapisanie artykułów do przeczytania na później

Artykuły zostają poprawnie wyświetlone w zakładce „na później”.



Rysunek 9. Wyświetlenie artykułów zapisanych na później

Po naciśnięciu na artykuł zostaje on wyświetlony w aplikacji i użytkownik może go przeczytać bez wychodzenia z aplikacji nie tracąc żadnej części doświadczenia użytkowania danej strony.



Rysunek 10. Otworzenie przykładowej strony

# Testy systemu bazodanowego i narzędzi do zarządzania agregatami

Testy działania systemu bazodanowego i narzędzi do zarządzania agregatami przeprowadzono z komputera pełniącego jednocześnie funkcję serwera, oraz z połączonego w sieci lokalnej telefonu typu smartphone. Test ma na celu sprawdzenie czy wszystkie zaimplementowane funkcje działają poprawnie, dlatego rozpoczęty zostanie on wejścia na stronę i założenia nowego konta użytkownika.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Rysunek 11. Tworzenie konta

Zakładamy przykładowe konto użytkownika z danymi jak na rysunku powyżej. Po naciśnięciu przycisku „Zarejestruj” użytkownik został przeniesiony na stronę główną. Użytkownik nie dodał jeszcze żadnego artykułu, więc pojawia się komunikat zachęcający.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Rysunek 12. Strona główna

Po przejściu do zakładki „Moje agregaty” użytkownik nie przynależący do żadnego agregatu zostaje przekierowany do formularza tworzenia nowego agregatu. W celu przetestowania funkcji utworzony zostanie przykładowy agregat.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Rysunek 13. Tworzenie nowego agregatu

Po stworzeniu agregatu należy dodać do niego kilka artykułów.



Rysunek 14. Dodawanie artykułu

Po dodaniu kilku przykładowych artykułów pojawiają się one w zakładce „Moje agregaty” w kolejności od najnowszego.

Obraz zawierający tekst

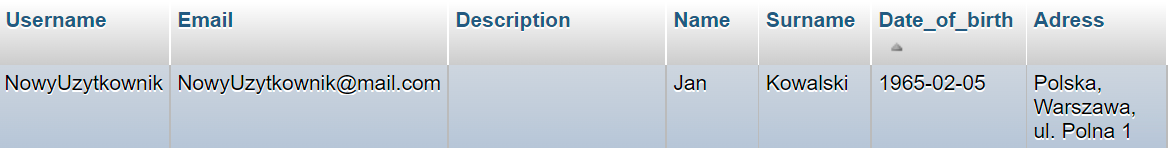
Opis wygenerowany automatycznie

Rysunek 15. Widok agregatu po dodaniu przykładowych artykułów

Uzupełnianie przykładowych danych osobistych

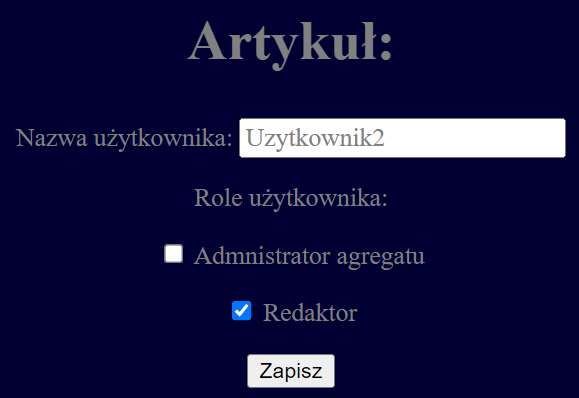


Rysunek 16. Dodawanie danych osobistych



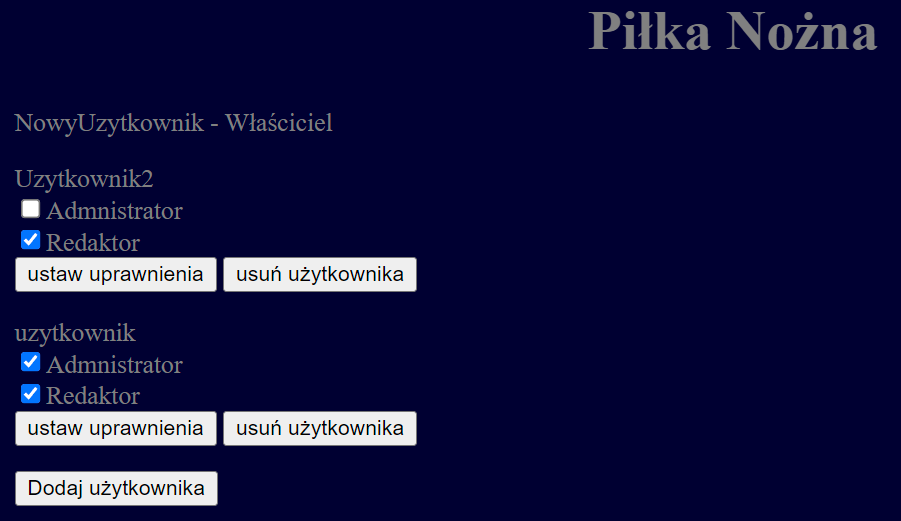
Rysunek 17. Potwierdzenie dodania danych

Dodawanie nowego użytkownika do agregatu.



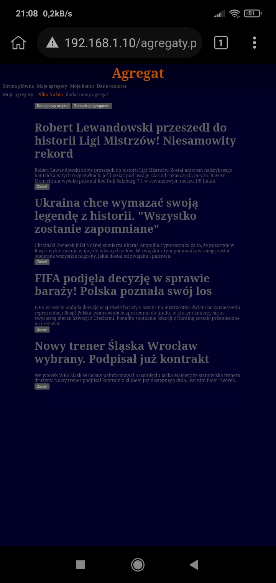
Rysunek 18. Dodawanie użytkownika do agregatu

Widok zarządzania agregatem po dodaniu nowych użytkowników do agregatu



Rysunek 19. Widok zarządzania agregatami

Sprawdzenie działania strony przy użyciu telefonu.



Rysunek 20. Działanie strony na innym urządzeniu

# Podsumowanie i wnioski

Użytkownik pragnący zarządzać własnym agregatem może zarejestrować się na przeznaczonej do tego stronie internetowej, z niej może stworzyć dowolną liczbę agregatów. Użytkownik agregatu może dodać dowolną liczbę użytkowników do wspólnego zarządzania agregatem, zmienić przypisane im uprawnienia, lub też usunąć ich z listy zarządzających agregatem. Użytkownik może dodać, zmienić, lub usunąć artykuł z listy polecanych.

Aplikacja mobilna przedstawia użytkownikowi polecane dla niego artykułu na podstawie zasubskrybowanych agregatów i odfiltrowuje mniej interesujące artykuły na podstawie wybranego przez użytkownika stopnia ważności. Użytkownik z poziomu aplikacji może otworzyć wybrany artykuł i go przeczytać, lub zapisać go na później i uzyskać do niego dostęp przy użyciu przeznaczonej do tego celu zakładki. Aplikacja zapisuje wszelkie dane lokalnie, dzięki czemu użytkownik nie musi się martwić o wyciek danych, a także może uzyskać dostęp do zapisanych do przeczytania później artykułów nawet wtedy gdy serwer bazodanowy jest tymczasowo nieosiągalny.

Projekt by osiągnąć pełną funkcjonalność i być udostępnionym użytkownikom powinien zostać ulepszony poprzez: dodanie użytkownikom opcji zarządzania agregatami z poziomu aplikacji mobilnej, rozszerzając tym samym funkcjonalność aplikacji, oraz usunięcie konieczności dodatkowego uruchamiania przeglądarki internetowej w celu dodawania nowego artykułu do polecanych. Stworzenie rozszerzenia do przeglądarek internetowych pozwalającego na dodanie artykułu bez konieczności przełączenia się na stronę pełniącą funkcję centrum zarządzania, umożliwiłoby to błyskawiczne dodawanie nowych artykułów. Stworzenie strony pozwalającej na śledzenie zmian wprowadzanych przez poszczególnych użytkowników. Dzięki temu większe organizacje miałyby narzędzie pozwalające na ocenę jakości pracy poszczególnych użytkowników. Dodanie systemu tagów, który by pozwalał użytkownikowi śledzić informację o interesującym go temacie bez konieczności subskrypcji poszczególnych agregatów. Stworzenie funkcji wyświetlającej sugestie opisu, oraz ważności artykułu na bazie decyzji podjętych już przez innych użytkowników, którzy do swojego agregatu dodali ten sam artykuł. Ulepszenie opcji wyszukiwania użytkowników wraz z możliwością podejrzenia ich przynależności. Ulepszenie funkcjonalności edycji artykułów, danych użytkownika poprzez dodanie opcji podglądu zmienianych pól. Implementację oprogramowania serwerowego na ogólnodostępnym serwerze.

# Bibliografia

Alifio Tutorials(11.02.2018). *How to connect android application to MySQL Database in XAMP Server*. Pobrane z https://youtu.be/4e8be8xseqE, data pobrania [12.02.2022]

Anupam Chugh(16.06.2020). *Android Alert Dialog using Kotlin*. Pobrane z https://www.journaldev.com/, data pobrania [12.02.2022]

Dawn Griffiths, David Griffiths(2020). *Kotlin. Rusz głową!* O'Reilly/Helion

Jolanta Pokorska(2013). Kwalifikacja E.14. Część 1. Tworzenie stron internetowych, Helion

Jolanta Pokorska(2013). Kwalifikacja E14. Część 2. Tworzenie baz danych i administrowanie bazami. Podręcznik do nauki zawodu technik informatyk, Helion

Jolanta Pokorska(2013). Kwalifikacja E.14. Część 3. Tworzenie aplikacji internetowych. Podręcznik do nauki zawodu technik informatyk, Helion

# Spis **rysunków**

Rysunek 1. Tabele przechowujące dane użytkownika 9

Rysunek 2. Tabele obsługujące połączenie użytkownika z agregatem 10

Rysunek 3. Połączenie tabel agregatów i artykułów 11

Rysunek 4. Layout definiujący sposób prezentowania artykułów 20

Rysunek 5. Layout definiujący sposób prezentowania dostępnych agregatów 28

Rysunek 6. Aplikacja po jej pierwszym uruchomieniu 35

Rysunek 7. Zasubskrybowane agregaty 36

Rysunek 8. Zapisanie artykułów do przeczytania na później 37

Rysunek 9. Wyświetlenie artykułów zapisanych na później 37

Rysunek 10. Otworzenie przykładowej strony 38

Rysunek 11. Tworzenie konta 39

Rysunek 12. Strona główna 39

Rysunek 13. Tworzenie nowego agregatu 39

Rysunek 14. Dodawanie artykułu 40

Rysunek 15. Widok agregatu po dodaniu przykładowych artykułów 40

Rysunek 16. Dodawanie danych osobistych 40

Rysunek 17. Potwierdzenie dodania danych 40

Rysunek 18. Dodawanie użytkownika do agregatu 41

Rysunek 19. Widok zarządzania agregatami 41

Rysunek 20. Działanie strony na innym urządzeniu 41

# Załączniki