

Politechnika Rzeszowska
Wydział Elektrotechniki i Informatyki
Katedra Informatyki i Autmatyki

Społeczeństwo informacyjne

PROJEKT

Aplikacja służąca ankietyzacji społeczeństwa

Kamil Malita (152167)
Mateusz Miś (152175)

Rzeszów, 2020

Spis treści

1. Założenia.....	3
1.1 Strona użytkownika	3
1.2 Strona urzędnika	3
1.3 Grupy docelowe.....	3
2. Stack technologiczny	4
2.1 Część Back-Endowa	4
2.2 Część frontendowa	5
3. Implementacja.....	6
3.1 Implementacja serwerowa	6
3.2 Implementacja frontendowa	7
4. Możliwości rozwoju aplikacji.....	12

1. Założenia

Celem projektu było wykonanie aplikacji dzięki której użytkownicy mogliby odpowiadać na wiele ankiet tworzonych przez lokalnych urzędników. Użytkownicy tacy po zalogowaniu mają możliwość wybrania dowolnej ankiety i udzielenia odpowiedzi zgodnie z ich zdaniem. Głosy oddane byłyby anonimowo, jednak samo głosowanie wymaga zalogowania do systemu.

1.1 Strona użytkownika

Użytkownicy mają do dyspozycji:

- Stronę rejestracji,
- Stronę logowania,
- Możliwość oddawania głosów na ankiety pogrupowane według zadanych kategorii,
- Obejrzenie wyników głosowania po zakończonej ankiecie.

1.2 Strona urzędnika

Urzednicy mają do dyspozycji:

- Stronę logowania,
- Możliwość dodawania nowych ankiet o określonej kategorii,
- Możliwość dodawania kategorii oraz ich usuwanie,
- Możliwość generowania wykresów wartości po zakończeniu ankiety.

1.3 Grupy docelowe

Aplikacja jest skierowana do wszystkich grup wiekowych. Niestety nie założono filtrację dostępnych ankiet dla osób dorosłych lub dzieci. Najbardziej zainteresowanymi będą osoby z przedziału wiekowego 14-45, które według danych statystycznych są najbardziej aktywnymi uczestnikami Internetu.

Oczywiście z założenia można tworzyć ankiety ukierunkowane specjalnie na konkretne grupy wiekowe, ale dalsze rozważania zostały poruszone w sekcji: „Możliwości rozwojowe aplikacji”.

2. Stack technologiczny

2.1 Część Back-Endowa

Strona serwerowa aplikacji została napisana w języku Java przy użyciu dostępnego frameworka Spring Boot. Jest to framework pozwalający tworzyć aplikacje serwerowe typu stand-alone.



Ikona Spring

Struktura plików jest typowa dla tego typu aplikacji i składa się z serwisów, repozytoriów oraz kontrolerów. Każdy z elementów odpowiada za osobną logikę aplikacji.

Repozytoria tworzą połączenia z bazą danych i odpowiadają za wymianę informacji z bazą danych połączonych z aplikacją.

Servisy odbierają informacje z repozytoriów i obrabiają dane do postaci wynikowej. Na tym etapie odbywa się najczęściej czasochłonnych i procesorowych obliczeń.

Kontrolery wystawiają nasze dane a świat zewnętrzny pod odpowiednim adresem końcowym. Odbierają dane z serwisów i przetwarzają ich formę na odpowiednią formę wynikową, np. do postaci JSON.

Z częścią Back-Endową związana jest nieodzownie **baza danych**.

W aplikacji została wykorzystana baza H2. Jest to baza typu „In Memory”, tzn. utrzymuje stan swoich danych tylko podczas trwania aplikacji. Po padnięciu lub wyłączenia serwera dane te są bezpowrotnie tracone. Model ten znacznie pomaga w testowaniu i działaniu aplikacji lokalnie bez konieczności na każdym urządzeniu testowym osobnej konfiguracji bazy danych lokalnie lub konieczności posiadania łączności z siecią wifi w celu uruchomienia aplikacji z sieciową wersją bazy danych.



Ikona bazy h2

Łączenie do bazy danych odbywa się poprzez framework tzw. hibernate, który jest najpopularniejszym narzędziem komunikacji z bazą danych dla języka Java. Poszczególne Encje z bazy są odwzorowane jako klasy w Javie, natomiast połączenie pomiędzy tymi tabelami tworzy się przy użyciu specjalnych konstrukcji udostępnianych przez Springa.

Informacje udostępniane dla części graficznej (strony internetowej) są wykonywane przy pomocy zapytań REST-owych. Część wizualna „odpytuje” część serwerową o konkretne zapytania i wyświetla je w odpowiedni dla siebie, zaprogramowany wcześniej, sposób.

2.2 Część frontendowa

Strona frontendowa przygotowana została w oparciu o framework Vue JS oraz bibliotekę elementów interfejsu graficznego Vuetify.



Logo Vue JS

W strukturze plików podobnie jak po stronie serwera można wyróżnić pewną strukturę. Główną część stanowi katalog zawierający dostępne w aplikacji widoki. Każdy z zawartych w nim plików stanowić będzie osobny widok w aplikacji do którego użytkownik będzie mógł się dostać z poziomu interfejsu użytkownika.

Istotą frameworku Vue JS jest dzielenie widoków na mniejsze pod części zwane komponentami. Zawarte one zostały w katalogu Components. Poszczególne komponenty stanowią niewielkie, niezależne elementy interfejsu graficznego służące np do wyświetlenia konkretnych danych czy zebrania danych od użytkownika. Każdy komponent posiada własny zestaw danych na których wykonuje operacje oraz własne metody. Do sprawnego działania aplikacji wykorzystuje się komunikację między komponentami realizowaną poprzez przekazywanie danych czy wywoływanie zdarzeń.

Dostęp do części danych jest niezbędny z poziomu całej aplikacji. Do takiego właśnie celu framework Vue JS oferuje narzędzie zwane Vuex. Przechowuje się w nim dane takie jak np. dane o użytkowniku w celu zapewnienia dostępu do nich z poziomu wszystkich komponentów. Zapewnia to że dane te nie muszą być wielokrotnie pobierane z serwera co usprawnia działanie aplikacji.

Warstwą komunikującą się z serwerem jest warstwa serwisów. Wykorzystuje ona narzędzie zwane Axios do komunikacji z udostępnionym przez serwer API poprzez protokół http.

Do przyspieszenia prac nad rozwojem interfejsu użytkownika wykorzystano bibliotekę elementów UI zwaną Vuetify. Jest to biblioteka przygotowana specjalnie pod framework Vue JS. Zawiera ona sporą liczbę gotowych elementów graficznych takich jak menu nawigacyjne,

przyciski, ikony itd. Korzystanie z niej znacząco przyspiesza pracę gdyż nie wymaga od programisty nakładania na elementy HTML stylu CSS ręcznie bo jest to dostępne bezpośrednio w bibliotece.

3. Implementacja

3.1 Implementacja serwerowa

Przykładowa klasa opisująca użytkownika w bazie danych:

```
@Getter
    @Setter
    @Entity
    public class Users {
        @Id
        private String username;
        private String password;
        private boolean enabled;
        private String name;
        private String surname;
        private LocalDateTime created;
        private String email;

        @OneToOne(cascade = CascadeType.REMOVE)
        @JoinColumn(name = "username")
        private Authorities authority;

        @OneToMany(cascade = CascadeType.REMOVE)
        @JoinColumn(name = "username", updatable = false, insertable = false)
        private List<Questionnaire> questionnaires;
```

Odwzorowanie takie pozwala aby automatycznie kod został wygenerowany jako tabela w bazie danych. Pola klasy są mapowane na pola w tabeli, a adnotacja `@JoinColumn` tworzy powiązanie (jako klucz obcy) pola `username` do tabeli `Authorities`.

Idąc dalej mamy repozytoria, które są przejściem pomiędzy tabelami a bazą danych w celu obsługi i odbierania danych z niej dla tabel. Zwykle zachodzi zależność, że na każdą tabelę Encji przypada jedna implementacja Repository.

Kolejnym ważnym punktem do wyszczególnienia są Servisy, który pozwalają odbierać dane z Repozytoriów i je przetwarzać (łączyć, często filtrować czy sprawdzać dane) do wyniku zadowalającego dla osoby lub automatu korzystającego z danego serwisu.

```
private final
CategoryRepository
categoryRepository;
```

```

    public List<Category> getCategories() {
        return categoryRepository.findAll();
    }

    public Category addCategory(CategoryDto category) {
        Authentication principal =
            SecurityContextHolder.getContext().getAuthentication();

        if(principal.getAuthorities().stream().noneMatch(grantedAuthority ->
            grantedAuthority.toString().equals("ROLE_ADMIN")))
            throw new UserAccessForbidden(principal.getName());
    }

```

Powyższy fragment serwisu kategorii pozwala pobrać metodą `getCategories` listę kategorii z bazy lub dodać nową za pomocą metody `addCategory`, gdzie parametrem jest nowa kategoria do dodania. W tym miejscu odbywa się sprawdzanie poprawności praw użytkownika do wykonania akcji. O ile pobrać listę kategorii może każdy o tyle dodać już mogą tylko użytkownicy posiadający prawa „admina”, czyli w tym przypadku urzędnicy.

Kolejnym bardzo ważnym elementem są kontrolery, które sprawują władzę wejścia requestom zewnętrznym do aplikacji serwerowej:

```

    private final QuestionnaireService questionnaireService;

    @GetMapping("/questionnaire/{id}")
    public Questionnaire getQuestionnaireById(@PathVariable Long id) {
        return questionnaireService.getQuestionnaireById(id);
    }

```

Powyższa część kontrolera obsługi ankiet pozwala pod adresem: *adres_dmyślny/questionnaire/{id}* dostania się do dowolnej ankiety określonym id. Chęć otrzymania takich informacji trafia do serwisu odpowiedzialnego za przetworzenie takiego zapytania, a z niego wywoływana jest metoda `getQuestionnaireById(id)` z parametrem `id`, który jest wartością całkowitą z id ankiety z bazy danych.

3.2 Implementacja frontendowa

Implementację strony frontndowej rozpoczęto od przygotowania głównego komponentu w którym osadzone będą poszczególne widoki. Zawarto w nim również panel nawigowania który widoczny jest jedynie po zalogowaniu. Struktura głównego komponentu aplikacji przedstawiona została poniżej:

```

<template>
  <v-app id="app">
    <nav-menu v-if="user" />

```

```

<app-bar v-if="user" />

<v-main>
  <router-view />
</v-main>
</v-app>
</template>

```

Kolejnym etapem było przygotowanie warstwy serwisu komunikującej się z wystawionym przez serwer API. Fragment jednego z serwisów przedstawiono poniżej:

```

import server_api from './axios_config'

const BASE_URL = '/v1'

export function getQuestionnaires () {
  return new Promise((resolve, reject) => {
    return server_api.get(`${BASE_URL}/questionnaires`)
      .then(response => {
        resolve(response.data)
      })
      .catch(error => {
        console.error(error)
        reject(error)
      })
  })
}

```

Zauważyć można że graficzny interfejs przygotowany jest tak aby odpowiednio reagował zarówno na operacje zakończone powodzeniem jak i te które się nie powiodą, obsługując w odpowiedni sposób błędy zwracane od serwera.

Przygotowano również strukturę Vuexa przechowującego dane dostępne globalnie w całej aplikacji.

```

Vue.use(Vuex)

export default new Vuex.Store({
  state: state,
  mutations: mutations,
  actions: actions,
  getters: getters
})

```

```

export default {
  user: null,

  categories: {
    loaded: false,
    data: []
  },

  questionnaires: {
    loaded: false,

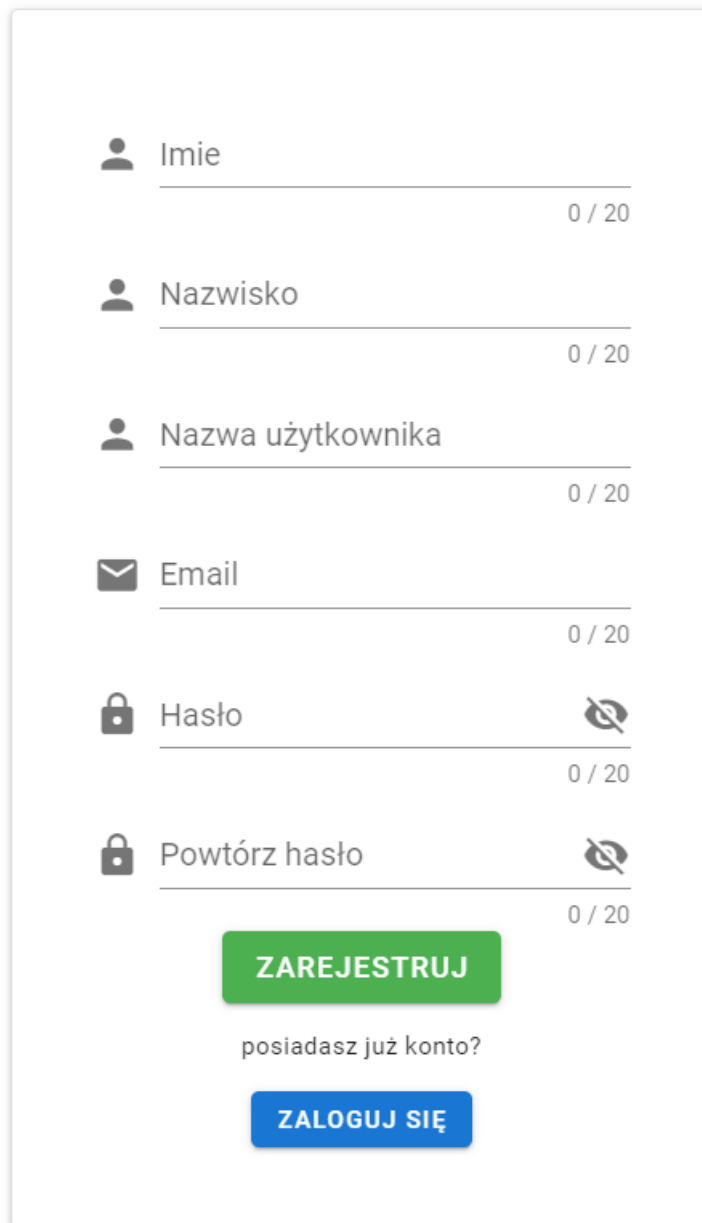
```



```
data: []  
}  
}
```

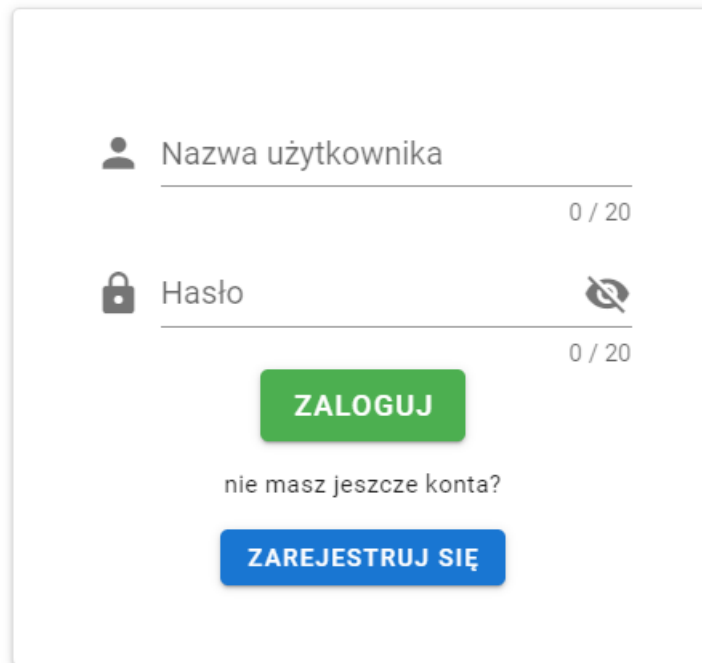
Przechowywanie danych globalnie pozwala na zmniejszenie ilości zapytań wysyłanych na serwer gdyż poszczególne komponenty mogą uzyskać dostęp do danych lokalnie.

Po przygotowaniu logiki rozpoczęto implementację poszczególnych widoków aplikacji rozpoczynając od panelu logowania oraz rejestracji nowych użytkowników.



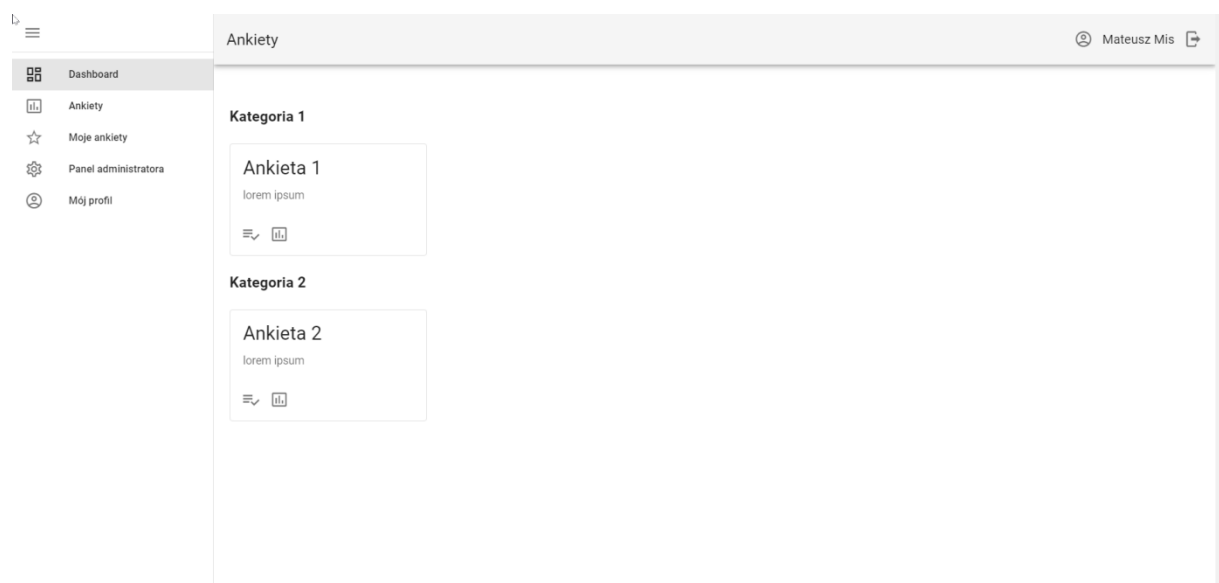
The image shows a registration form with the following fields and elements:

- Imie** (Name): Input field with a person icon, character count 0 / 20.
- Nazwisko** (Surname): Input field with a person icon, character count 0 / 20.
- Nazwa użytkownika** (Username): Input field with a person icon, character count 0 / 20.
- Email**: Input field with an envelope icon, character count 0 / 20.
- Hasło** (Password): Input field with a lock icon and a toggle icon, character count 0 / 20.
- Powtórz hasło** (Repeat password): Input field with a lock icon and a toggle icon, character count 0 / 20.
- ZAREJESTRUJ**: Green button.
- posiadasz już konto?**: Text label.
- ZALOGUJ SIĘ**: Blue button.

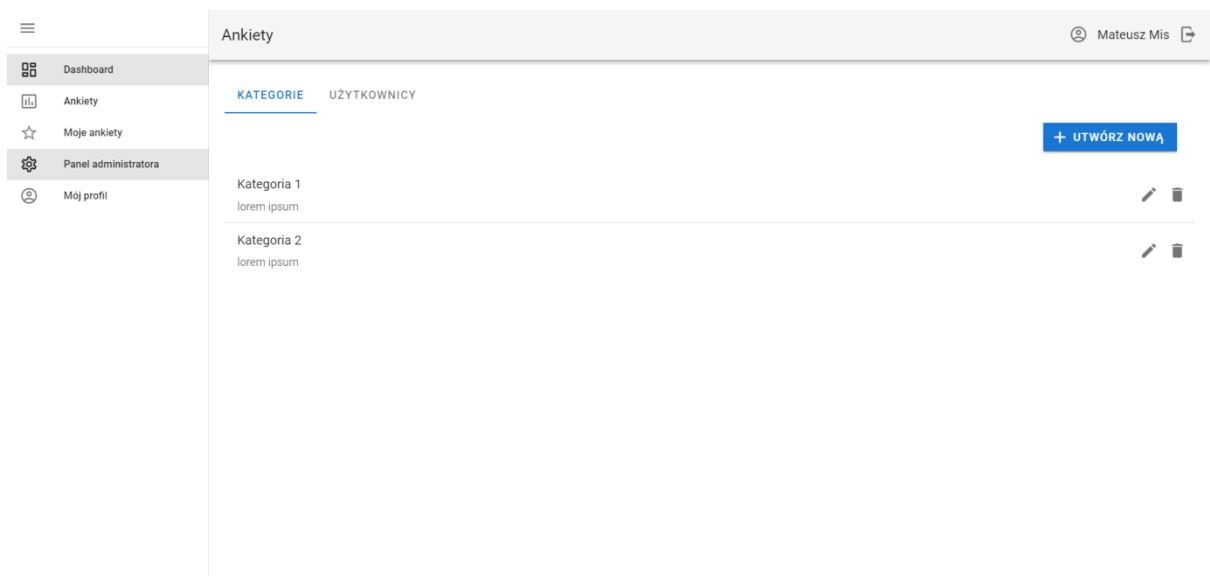


A login and registration form with a white background and a light gray border. It features two input fields: 'Nazwa użytkownika' (Username) and 'Hasło' (Password), both with a 0/20 character count. A green 'ZALOGUJ' (Login) button is centered below the fields. Below the button is the text 'nie masz jeszcze konta?' (don't you have an account yet?) and a blue 'ZAREJESTRUJ SIĘ' (Register) button.

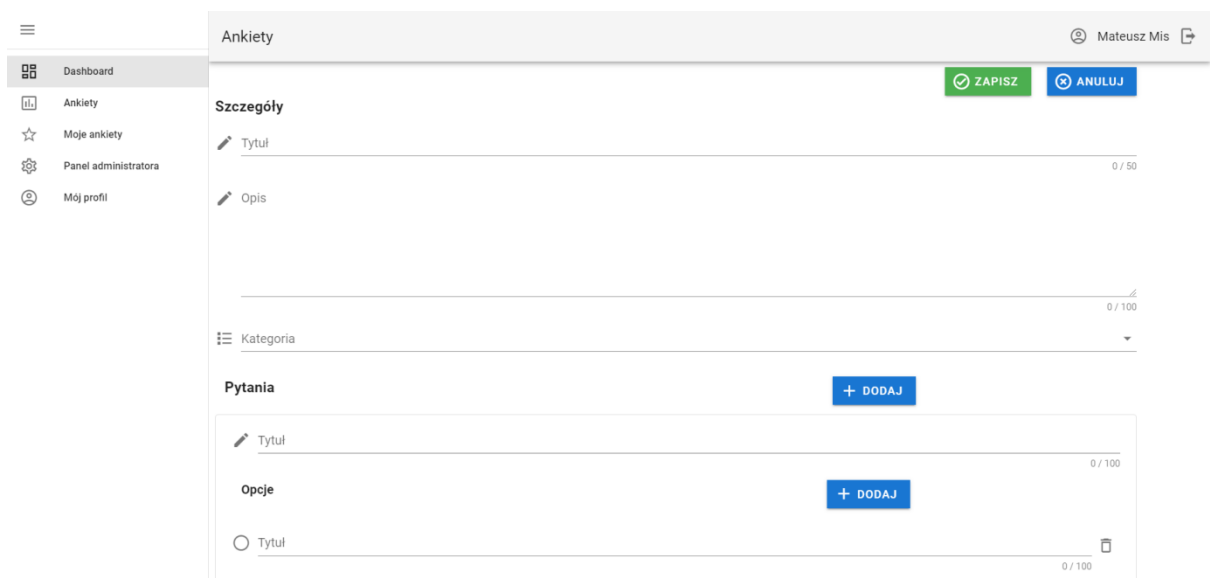
Po zalogowaniu użytkownik uzyskuje dostęp do dashboardu z poziomu którego widzi najnowsze ankiety dostępne w poszczególnych kategoriach.



Administrator posiada uprawnienia do tworzenia nowych kategorii do których dodawane mogą być ankiety.



System udostępnia prosty system tworzenia ankiet.



Umożliwia również na proste ich wypełnianie.

Dashboard

Ankiety

Moje ankiety

Panel administratora

Mój profil

Ankiety

Mateusz Mis

Szczegóły

Tytuł: Ankieta 1

Opis: lorem ipsum

Kategoria: Kategoria 1

Pytania

Treść: Pytanie 1

Opcje

☐ opcja 1

☐ opcja 2

ZAPISZ

ANULUJ

4. Możliwości rozwoju aplikacji

Aplikacja jest przygotowana pod dalszy rozwój. Podstawowym zagadnieniem byłoby przepięcie bazy danych na jakąś wersję stabilną i bezpieczną w razie wyłączenia aplikacji, do dyspozycji mamy dowolną bazę SQL-ową, jak np. PostgreSQL czy OracleSQL.

Kolejnym polem do rozwoju aplikacji mogłoby być dodanie odpowiednich grup wiekowych dla użytkowników młodszych, pełnoletnich i osoby starsze. Dzięki temu trafienie do odpowiedniej grupy wiekowej byłoby jeszcze łatwiejsze i precyzyjniejsze. Dodatkowo wchodzi tu w grę sprawa nie udostępniania młodszym osobom treści dla pełnoletnich.