Wydział Informatyki Politechnika Białostocka Systemy Mobilne(PS)	Data: 22.01.2023
Dokumentacja Projektu Temat: Komunikator tekstowy Autorzy: Michał Wołosewicz 109958 Patryk Wójtowicz 109960 Magda Zaborowska 109962	Prowadzący: mgr inż. Patryk Milewski

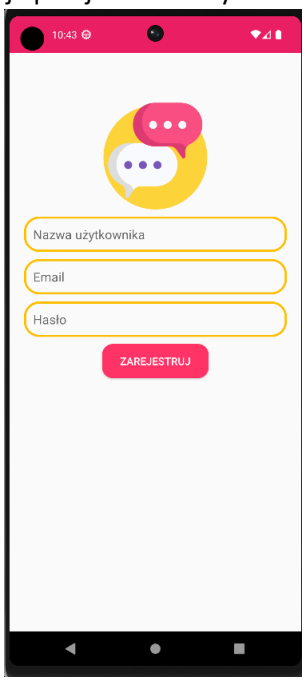
1. Opis Projektu

Celem projektu było stworzenie prostego komunikatora tekstowego na systemy Android w oparciu o technologie Kotlin i Firebase. Użytkownik w wypadku nieudanej autentykacji zobaczy wiadomość typu popup „błąd logowania”/ „błąd rejestracji”. Poprawne zalogowanie lub zarejestrowanie(user jest dodawany do bazy danych) pozwoli zobaczyć listę pozostałych użytkowników (bez osoby zalogowanej na danym urządzeniu). Po wybraniu osoby otwiera się aktywność chatu, jeżeli wcześniej nie prowadzono konwersacji wtedy są tworzone unikalne „pomieszczenia” dla nadawcy i adresata, w których zapisywane są wiadomości (message oraz id senderId).

2. Opis Funkcjonalności

- ## Rejestracja

  Po wprowadzeniu poprawnych danych użytkownik zostanie dodany do bazy danych i aplikacja przejdzie do listy kontaktów.

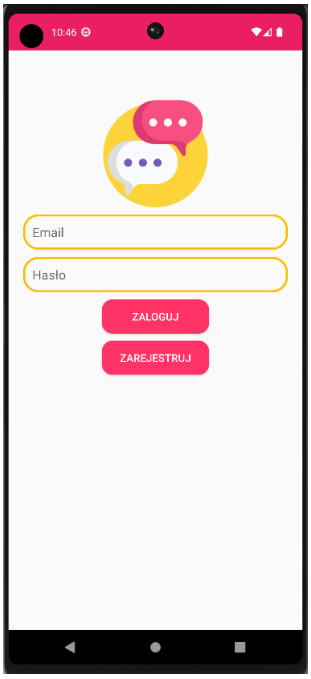


Rysunek 1 Widok activity_sign_up

```
private  fun signUp(name: String, email: String, password: String){

    mAuth.createUserWithEmailAndPassword(email, password)
        .addOnCompleteListener(this) { task ->
            if (task.isSuccessful) {
                addUserToDatabase(name, email, mAuth.currentUser?.uid!!)

                val intent = Intent(this@SignUp, MainActivity::class.java)
                finish()
                startActivity(intent)

            } else {
                Toast.makeText(this@SignUp, "Wystąpił błąd przy
rejestracji", Toast.LENGTH_SHORT).show()
            }
        }

}
```

- **Logowanie**

  Podobnie jak poprzednio w przypadku poprawnych danych logowania (w bazie danych istnieje użytkownik o podanym email i haśle) zostaniemy przeniesieni do listy kontaktów (MainActivity).

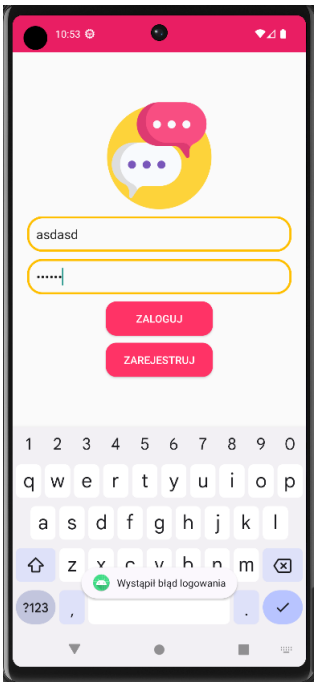


Rysunek 2 Widok activity_log_in

```
private  fun login(email: String, password: String){
    mAuth.signInWithEmailAndPassword(email, password)
        .addOnCompleteListener(this) { task ->
            if (task.isSuccessful) {

                val intent = Intent(this@Login, MainActivity::class.java)
                finish()
                startActivity(intent)

            } else {
                Toast.makeText(this@Login, "Wystąpił błąd logowania",
```

```
Toast.LENGTH_SHORT).show()
            }
        }
}
```

- **Powiadomienia popup (Toast)**
  Po wprowadzeniu złych danych przy logowaniu lub rejestracji wyskoczą odpowiednie komunikaty.



Rysunek 3 Powiadomienie o błędzie logowania

```
else {
    Toast.makeText(this@SignUp, "Wystąpił błąd przy rejestracji",
Toast.LENGTH_SHORT).show()
}
else {
    Toast.makeText(this@Login, "Wystąpił błąd logowania",
Toast.LENGTH_SHORT).show()
}
```

- **Lista kontaktów**
  To jest MainActivity, w którym użytkownik może wybrać rozwijane menu (3 pionowe kropki na toolbar) lub innego użytkownika, by otworzyć z nim chat.

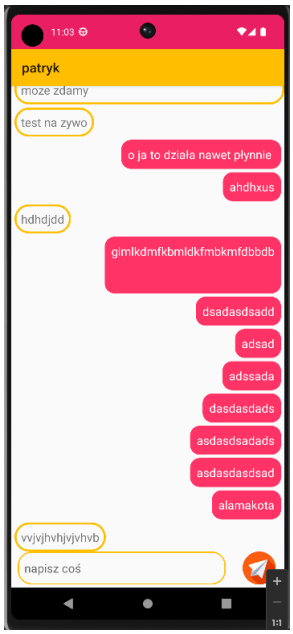
Rysunek 4 Widok activity_main

```
override fun onCreateOptionsMenu(menu: Menu?): Boolean {
    menuInflater.inflate(R.menu.menu, menu)
    return super.onCreateOptionsMenu(menu)
}
```

```
mDbRef.child("user").addValueEventListener(object : ValueEventListener{
    override fun onDataChange(snapshot: DataSnapshot) {

        userList.clear()

        for(postSnapshot in snapshot.children){
            val currentUser = postSnapshot.getValue((User::class.java))
            if(mAuth.currentUser?.uid != currentUser?.uid){
                userList.add(currentUser!!)
            }
        }
        adapter.notifyDataSetChanged()
    } … })
```

- **Wysyłanie i odbieranie wiadomości**

  Po wybraniu osoby z listy otwiera się aktywność chatu, jeśli wcześniej nie pisaliśmy żadnych wiadomości z danym użytkownikiem zostaną stworzone prywatne pokoje dla nadawcy i adresata (dane w nich są identyczne), w których będą zapisywane nasze wiadomości.

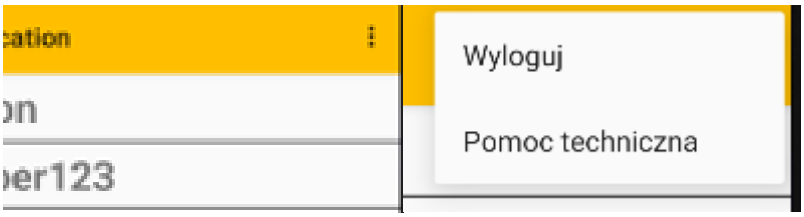


Rysunek 5 Widok activity_chat

```
// dodawanie danych do rycyclerView
mDbRef.child("chats").child(senderRoom!!).child("messages")
    .addValueEventListener(object : ValueEventListener{
        override fun onDataChange(snapshot: DataSnapshot) {

            messageList.clear()

            for(postSnapshot in snapshot.children){

                val message = postSnapshot.getValue(Message::class.java)
                messageList.add(message!!)
            }
            messageAdapter.notifyDataSetChanged()

        }…})
```

```
//dodawanie wiadomości do firebase
sendButton.setOnClickListener(){

    val message = messageBox.text.toString()
    val messageObject = Message(message,senderUid)

    mDbRef.child("chats").child(senderRoom!!).child("messages").push()
        .setValue(messageObject).addOnSuccessListener {

mDbRef.child("chats").child(receiverRoom!!).child("messages").push()
                .setValue(messageObject)
        }
    messageBox.setText("")
}
```

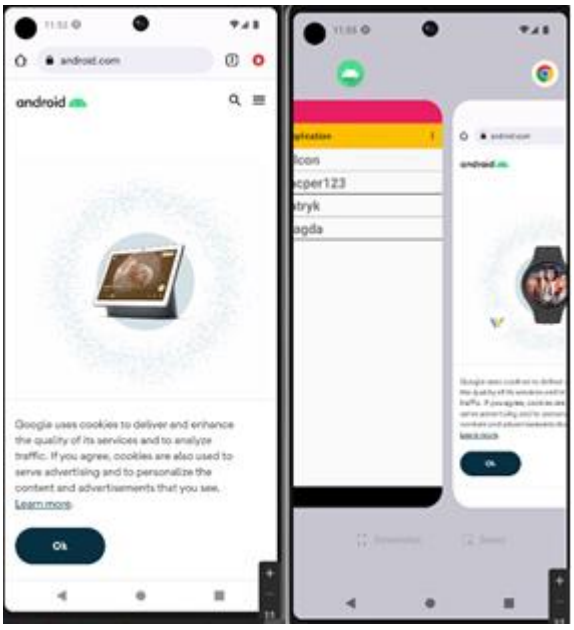
- **Menu rozwijane**
  Po rozwinięciu menu pojawią się dwie opcje wyloguj i Pomoc techniczna.



Rysunek 6 Menu rozwijane

```
override fun onOptionsItemSelected(item: MenuItem): Boolean {
    if(item.itemId == R.id.logout) {
        mAuth.signOut()
        val intent = Intent(this@MainActivity, Login::class.java)
        finish()
        startActivity(intent)
        return true;
    }
    if(item.itemId == R.id.TechSupp) {
        val webIntent: Intent = Uri.parse("https://www.android.com").let {
webpage ->
            Intent(Intent.ACTION_VIEW, webpage)
        }
        startActivity(webIntent)
        return true;
    }
    return true;
}
```

- **Wylogowanie**
  Zamyka aktywności main i wraca do logowania.
- **Pomoc Techniczna**
  Przekierowuje użytkownika do aplikacji przeglądarki na stronę www.android.com.



Rysunek 7 Przekierowanie użytkownika do innej aplikacji

3. Szczególnie interesujące zagadnienia projektowe

API Firebase znacznie ułatwiło pracę z projektem. Przechowywanie, zarządzanie użytkownikami i ich uwierzytelnianie (**firebase.auth**) nie sprawiło większych problemów dzięki dostarczonym pakietom. Baza danych przechowuje i zarządza także pokojami chat'ów i zapisanymi w nich wiadomości, dzięki czemu opóźnienia między wysłaniem i dostarczeniem wiadomości jest minimalne.

4. Instrukcja instalacji

Należy przygotować urządzenie z systemem Android (**minimum 6.0 Marshmallow**) oraz pobrać projekt i otworzyć go w Android Studio. Następnie połączyć docelowe urządzenie i wybrać opcję „Run App"*.

*Aby instalacja się powiodła należy włączyć opcje developerskie na urządzeniu i opcję instalacji z nieznanych źródeł.

5. Instrukcja użytkownika

Wierzymy, że nasza aplikacja jest intuicyjna i nie potrzebuje dodatkowej instrukcji, w razie jakichkolwiek pytań proszę się skontaktować się z nami- na pewno pomożemy.

6. Wnioski

Kotlin był odpowiednim wyborem do tego projektu, ponieważ jest nowoczesnym i wydajnym językiem programowania do programowania na Androida, a wykorzystanie Firebase znacznie ułatwiło pracę z bazą danych i dostarczyło nam kilka gotowych rozwiązań, które są dobrze opisane w dokumentacji na https:// firebase.google.com/docs.

7. Samoocena i wymagania projektowe z cez2.wi.pb.pl

Aplikacja jest atrakcyjna wizualnie oraz działa płynnie i responsywnie(została przetestowana na kilku fizycznych urządzeniach i nie stwierdzono bugów), wdrożyliśmy większość wymagań i uważamy, że zasługujemy na oceny bardzo dobre.

	Kategoria	Zakres	Punkty
1	Struktura projektu	Umiejętne dobranie komponentów budulcowych aplikacji mobilnej, zgodnie z ich przeznaczeniem	1
		Poprawna struktura projektu - podział na klasy, interfejsy, typy wyliczeniowe, rozmieszczenie ich w pakietach	1
		Konfiguracja projektu - dobranie minimalnej wersji API, pobranie niezbędnych zależności (plik build.gradle), poprawna konfiguracja budowania projektu, odpowiednie	2

		ustawienia w pliku konfiguracyjnym projektu (AndroidManifest.xml)	
		Złożoność: liczba klas, poziom skomplikowania, dobór adekwatnych do celu technik programistycznych oraz zastosowanie dobrych praktyk	3
2	Programistyczny interfejs aplikacji	Poprawność zastosowania metod cyklu życia komponentów budulcowych aplikacji (w szczególności aktywności)	1
		Zabezpieczenie przed utratą danych w przypadku zniszczenia aktywności (np. w wyniku obracania ekranu)	2
		Nawigacja w aplikacji, przejścia między widokami	2
		Komunikacja między poszczególnymi komponentami aplikacji, wymiana informacji	1
		Uruchamianie innych aplikacji z poziomu danej aktywności (np. YouTube)	1
3	Interfejs użytkownika i zasoby aplikacyjne	Dobór rodzajów układów (layout) zgodnie z przeznaczeniem. Różnorodność zastosowanych rodzajów układów.	2
		Stosowne użycie różnorodnych kontrolek zgodne z ich przeznaczeniem	1
		Zastosowanie fragmentów	2
		Zastosowanie listy wraz z powiązanymi obiektami (np. RecyclerView + ViewHolder + Adapter)	2
		Dodatkowe elementy poprawiające komfort korzystania z aplikacji i poziom wizualny (np. CardView, Bubbles, animacje)	2
		Wielokrotne zastosowanie widoków, składanie widoków (merge, include)	2
		Menu (różne rodzaje)	2
		Wyszukiwanie	2
		Powiadomienia (Notifications)	2
		Okna dialogowe, wiadomości typu pop-up (np. Snackbar)	1
		Umiędzynarodowienie	1
		Wydzielenie stałych do odpowiednich plików z zasobami	1
		Zastosowanie dobrych praktyk opisanych np. w Material Design	1
		Zapewnienie poprawnego działania aplikacji na urządzeniach z ekranami o różnych rozdzielczościach	1
4	Zasoby sprzętowe	Zastosowanie minimum dwóch czujników, zamiennie z zastosowaniem lokalizacji lub aparatu fotograficznego; uzasadnione wykorzystanie większej liczby zasobów skutkuje otrzymaniem większej liczby punktów (większa złożoność)	4
5	Baza danych	Implementacja lokalnej bazy danych SQLite (z wykorzystaniem Room)	1
		Relacje między encjami: OneToOne, OneToMany, ManyToMany	3
		Konwertery typów, dodatkowe adnotacje (np. @Ignore, @Index, @Fts4)	2
		Różnorodne zapytania do każdej z encji (min. CRUD)	2
		Złożone zapytania, migracja bazy danych	2

6	Zasoby sieciowe (API)	Połączenie z API zewnętrznym poprzez webserwisy (REST)	10
		Suma	45