

Autor: Miłosz Kukulski 258990
Grupa: Poniedziałek 13:15
Prowadzący: Dr inż. Marek Piasecki
Termin oddania: 20.06.2023

Programowanie aplikacji mobilnych

Projekt android

1. Założenia projektowe

Aplikacja daje możliwość założenia konta przez użytkownika oraz zamawiania usług cateringowych online, po złożeniu zamówienia, zapisuje zamówienie do bazy danych. Użytkownik przy rejestracji przechodzi również weryfikację mailową, na podanego maila przychodzi 6 cyfrowy kod, który trzeba wpisać w ostatnim kroku rejestracji, aby konto zostało założone, a dane zapisały się do bazy danych. Administrator ma przypisane konto, tzn. login i hasło zostało ustawione w bazie danych. Przechodzi on proces logowania. Po poprawnym zalogowaniu ma dostęp do danych użytkowników oraz możliwość ich usuwania. Użytkownika składa zamówienie mając do wyboru trzy „restauracje”. Zamówienia wyświetlają się w podsumowaniu na bieżąco w listView.

2. Implementacja funkcjonalności

Aplikację mogę podzielić na 4 moduły:

- Moduł bazy danych
- Moduł procesu rejestracji
- Moduł użytkownika
- Moduł administratora

Ciężko mi określić procentowy udział w ich wykonaniu. Wszystkie moduły robiłem samodzielnie korzystając z różnych poradników znalezionych w internecie, a wykorzystane znalezione fragmenty kodu przechodziły moją modyfikację oraz dokładną analizę.

Moduł bazy danych:

Utworzyłem bazę danych SQLite przy pomocy klasy SQLiteOpenHelper zgodnie z poradnikami.

```
class Database(context: Context) : SQLiteOpenHelper(context, name: "AppBase.db", factory: null, version: 2) {  
  
    override fun onCreate(db: SQLiteDatabase?) {  
        db?.execSQL(logreg_create)  
    }  
}
```

Klasa Database dziedziczy po SQLiteOpenHelper, który jest pomocnikiem do zarządzania operacjami na bazie danych. W metodzie onCreate są wywołane metody execSQL z odpowiednimi zapytaniami SQL do stworzenia tabel.

Stworzyłem schemat dla tabeli logreg oraz dodatkowo dwie tabele pizza i admin. Schemat definiuje nazwę tabeli i nazwy kolumn, które są potem używane do tworzenia tabel w bazie danych.

```
object LoginSchema {  
    object LoginEntry : BaseColumns {  
        const val TABLE_NAME = "logreg"  
        const val COLUMN_NAME_LOGIN = "login"  
        const val COLUMN_NAME_PASSWORD = "password"  
        const val COLUMN_NAME_EMAIL = "email"  
    }  
}
```

Utworzona klasa PizzaDatabaseHelper jest pomocnikiem do zarządzania tabelą pizza. W metodzie addPizza jest dodawana nowe danie do tabeli. Metoda getData zwraca kursor do wszystkich dań w tabeli. Metoda clearPizzaTable usuwa wszystkie dania z tabeli.

```
class PizzaDatabaseHelper(context: Context) :  
    SQLiteOpenHelper(context, DATABASE_NAME, factory: null, DATABASE_VERSION) {  
    companion object {  
        private const val DATABASE_NAME = "pizza_database"  
        private const val DATABASE_VERSION = 1  
    }  
}
```


Klasa AdminDatabaseHelper jest pomocnikiem do zarządzania tabelą admin. W metodzie onCreate jest tworzona tabela i dodawany jest administrator.

```
class AdminDatabaseHelper(context: Context) :  
    SQLiteOpenHelper(context, DATABASE_NAME, factory: null, DATABASE_VERSION) {  
    companion object {  
        private const val DATABASE_NAME = "admin_database"  
        private const val DATABASE_VERSION = 2  
    }  
}
```

Struktura bazy danych okazała się raczej prostsza niż sobie to wyobrażałem, więc nie było na tym etapie specjalnych problemów.

Moduł procesu rejestracji:

Pierwszym krokiem użytkownika jest założenie konta.



Podaj nazwę konta

Marek

Podaj email

milek.kukulski@o2.pl

Podaj hasło do konta

Powtórz hasło

ZAREJESTRUJ

MASZ JUŻ KONTO? ZAŁOGUJ SIĘ!

Hasła muszą być identyczne, na podany mail przychodzi 6 cyfrowy kod w celu weryfikacji.



Wysłano kod weryfikacyjny
na twój adres email

Podaj kod weryfikacyjny

np. 678423

POTWIERDŹ

WRÓĆ DO LOGOWANIA

Po wpisaniu poprawnego kodu, konto zostaje założone i dodane do bazy, a użytkownik może przejść do logowania.



Marek

ZALOGUJ

NIE MASZ JESZCZE KONTA? ZAREJESTRUJ SIĘ!

WRÓĆ DO STRONY STARTOWEJ

Jeżeli login i hasło są poprawne, użytkownik zostaje zalogowany i może składać swoje zamówienie cateringowe. Przy braku lub błędnych danych wyświetlane są „toasty” z odpowiednim komunikatem w celu ponownej próby.

W kodzie przechodzę do konkretnych aktywności za pomocą Intentów. Pobierane są referencje do pól tekstowych, które będą wykorzystane do wprowadzenia danych przez użytkownika (login, email, hasło i powtórzenie hasła). Dodatkowo odnajdywane są referencje do przycisków "Rejestracja" oraz "Zaloguj się". Listener dla przycisku rejestracji, registerButton, sprawdza czy wszystkie pola są wypełnione oraz czy wprowadzone hasło w obu polach jest takie samo. Jeśli tak, następuje przekierowanie do aktywności VerificationActivity z przekazanymi danymi o loginie, hasle i emailu. Jeżeli dane są nieprawidłowe, wyświetlany jest komunikat z informacją o błędzie. Tworzę obiekt dbHelper, który pozwala na dostęp do bazy danych w trybie zapisu, obiekt ContentValues, który przechowuje nasze wartości w formie klucz-wartość. Generuje losowy 6-cyfrowy kod, który zostanie wysłany do użytkownika. Tworzę stałe, które będą używane do wysyłania wiadomości e-mail oraz funkcję sendEmail do wysyłania e-maila z kodem weryfikacyjnym. Konfiguruje ona połączenie SMTP i tworzy wiadomość e-mail.

```
var code = ""
for (i in 1..6) {
    code += Random.nextInt( from: 0, until: 9)
}

val senderEmail = "app.catering1@gmail.com"
val senderPassword = "vzaaapfmjfsimfwx"
val subject = "Application_Catering:Code verification"
val body = "Hello,thank you for using our app!!\nThis is your verifiaction code:"+code+"\nYour login:"+name+"\nYour password:"+password

fun sendEmail(senderEmail: String, senderPassword: String, receiverEmail: String, subject: String, body: String) {
    val properties = Properties()
    properties["mail.smtp.host"] = "smtp.gmail.com"
    properties["mail.smtp.port"] = "587"
    properties["mail.smtp.auth"] = "true"
    properties["mail.smtp.starttls.enable"] = "true"
    properties["mail.smtp.ssl.trust"] = "smtp.gmail.com"

    val session = Session.getInstance(properties, object : Authenticator() {
        override fun getPasswordAuthentication(): PasswordAuthentication {
            return PasswordAuthentication(senderEmail, senderPassword)
        }
    })
}
```

Moduł użytkownika:

Użytkownik po zalogowaniu ma do wyboru trzy „restaurację”.

Catering

Wybierz restaurację !

IGI PIZZA

RAGU - PRACOWNIA MAKARONU

BURGER LTD



Po chęci zamówienia np. pizzy z IGI PIZZA, użytkownik proszony jest o wybór rodzaju pizzy oraz takich danych jak adres i termin dostarczenia zamówienia.

Wybierz swoją pizzę! :)

Pizza

Pizza: Pepperoni - 30zł

Adres

Termin wraz z godziną dostarczenia zamówienia

ZAMÓW

Po wypełnieniu danych i złożeniu zamówienia, dostajemy potwierdzenie i podsumowanie zamówienia.

Podsumowanie

Dziękujemy, Twoje zamówienie jest w trakcie realizacji! :)

Pizza: Pepperoni
, Cena: 30 zł
, Adres: Poziomkowa 6
, Termin: 21.06.2023, 17:00

Tworzenie modułu użytkownika sprawiło mi największą przyjemność. Każda restauracja ma swoją bardzo podobną aktywność. Do wyboru dania stworzyłem autouzupełniające pole tekstowe oraz ArrayAdapter z wykorzystaniem layoutu list_item i tablicy items.

```

val autoComplete: AutoCompleteTextView = findViewById(R.id.auto_complete)

val adapter = ArrayAdapter(context: this, R.layout.list_item, items)

autoComplete.setAdapter(adapter)

autoComplete.setOnItemClickListener =
    AdapterView.OnItemClickListener { adapterView, view, i, l ->
        val itemSelected = adapterView.getItemAtPosition(i).toString()
        val namePrice = itemSelected.split(...delimiters: "-")
        val pizzaName = namePrice[0].trim()
        val pizzaPrice = namePrice[1].trim().replace(oldValue: "zł", newValue: "").trim().toInt()

        selectedPizza = pizzaName
        selectedPrice = pizzaPrice

        Toast.makeText(context: this, text: "Wybrano: $itemSelected", Toast.LENGTH_SHORT).show()
    }

```

Moduł Administratora

Administrator posiada swoją klasę i tabelę:

```

override fun onCreate(db: SQLiteDatabase) {
    val createTableStatement = """
CREATE TABLE $TABLE_ADMIN (
    _id INTEGER PRIMARY KEY AUTOINCREMENT,
    $COLUMN_LOGIN TEXT,
    $COLUMN_PASSWORD TEXT
)
""".trimIndent()

    db.execSQL(createTableStatement)

    // Dodaj administratora
    val adminLogin = "maciek123"
    val adminPassword = ("domek123") // Hasło powinno być zahaszkowane
    val insertAdminStatement = """
INSERT INTO $TABLE_ADMIN (
    $COLUMN_LOGIN,
    $COLUMN_PASSWORD
) VALUES (
    '$adminLogin',
    '$adminPassword'
)
""".trimIndent()

    db.execSQL(insertAdminStatement)
}

```

Przechodzi on również przez proces logowania, jednak jego dane są przypisane wcześniej w bazie. Po zalogowaniu ma podgląd do danych zarejestrowanych użytkowników oraz opcję usuwania konta za pomocą podania ID użytkownika:

Catering

USUŃ

ID: 4
Numer konta: 1
login: admin
hasło: admin
email: milek.kukulski@o2.pl

ID: 5
Numer konta: 2
login: Marek
hasło: marek
email: milek.kukulski@o2.pl

Kiedy deleteButton jest naciskany, kod najpierw sprawdza, czy wprowadzone ID użytkownika jest prawidłowe (czyli czy nie jest puste). Jeżeli tak, wykonuje zapytanie usuwające w bazie danych. Zapytanie usuwające ma postać: `db.delete(LoginSchema.LoginEntry.TABLE_NAME, selection, selectionArgs)`, gdzie `selection` to warunek usuwania, a `selectionArgs` to argumenty, które są wprowadzane do tego warunku.

```
val deleteEditText = findViewById<EditText>(R.id.id)
val deleteButton = findViewById<Button>(R.id.usun)

deleteButton.setOnClickListener { it: View!
    val accountIdToDelete = deleteEditText.text.toString().toIntOrNull()

    if (accountIdToDelete != null) {
        // Zapytanie usuwające
        val selection = "${BaseColumns._ID} = ?"
        val selectionArgs = arrayOf(accountIdToDelete.toString())
        db.delete(LoginSchema.LoginEntry.TABLE_NAME, selection, selectionArgs)
    }
}
```

Odświeżanie danych: Po usunięciu użytkownika, wywoływana jest funkcja `refreshData()`, która odświeża dane wyświetlane na ekranie. Ta funkcja wykonuje zapytanie do bazy danych, aby pobrać aktualną listę użytkowników, a następnie wyświetla te informacje w `displayTextView`. W przypadku błędu (na przykład, jeśli wprowadzone ID użytkownika jest puste lub nieprawidłowe), kod wyświetla powiadomienie z prośbą o wprowadzenie poprawnego numeru ID użytkownika. Po usunięciu użytkownika, wyświetlane jest powiadomienie informujące o pomyślnym usunięciu. Funkcja `refreshData()` jest wywoływana dwa razy - raz na początku, aby zainicjować dane, które są wyświetlane na ekranie, a drugi raz po usunięciu użytkownika, aby odświeżyć te dane.


```

fun refreshData() {
    val cursor = db.query(
        LoginSchema.LoginEntry.TABLE_NAME,
        columns: null,
        selection: null,
        selectionArgs: null,
        groupBy: null,
        having: null,
        orderBy: null
    )
}

```

3. Podsumowanie

Tworzenie aplikacji mobilnej sprawiło mi dużą przyjemność. Poświęciłem bardzo dużo czasu, jednak nie zdążyłem zaimplementować aplikacji do perfekcyjnego stopnia. Mimo wszystko jestem bardzo zadowolony ze swoich postępów. Ze względu na ograniczenie czasowe nie zrobiłem automatycznego tworzenia bazy z zamówieniem dla zalogowanego użytkownika, zamiast tego po ponownym zalogowaniu użytkownika zamówione wcześniej dane są usuwane z tabeli. Dlatego też admin nie ma podglądu do zamówień, a danych o użytkownikach. Nie ma to jednak wpływu na istotne aspekty projektu. Powyższy opis aplikacji jest troszkę szczegółowy jeżeli chodzi o funkcjonalności aplikacji, przyczyną przekroczenia formatu są liczne screenshoty za co przepraszam.

4. Wykorzystane materiały

<https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper>

<https://developer.android.com/training/data-storage/sqlite#kotlin>

<https://learn.microsoft.com/en-us/dotnet/api/android.database.sqlite.sqlitedatabase.rawQuery?view=xamarin-android-sdk-13>

<https://www.youtube.com/watch?v=jXSNobmB7u4&t=83s>

<https://suluksm.medium.com/how-to-create-a-login-register-app-with-kotlin-using-fragments-and-room-database-mvvm-76147970f754>

<https://www.vetbossel.in/kotlin-login-registration-android/>

<https://auth0.com/blog/get-started-android-authentication-kotlin/>