Wrocław University
of Science and Technology

Projekt zespołowy

Aplikacja DAT

Kucza Miłosz 262760

Kukulski Miłosz 258990

Matysiak Marcin 256850

Kacper Wiącek 259378

Prowadzący: Dr Inż. Konrad Kluwak

1. Cel ćwiczenia

- Aplikacja została stworzona w celu umożliwienia użytkownikom kontroli nad określonymi urządzeniami. Składa się ona z menu głównego, layoutów logowania i rejestracji, kodu weryfikacyjnego oraz prostej bazy danych opartej na SQLiteHelper. W celu zarejestrowania się w aplikacji, użytkownik otrzymuje kod weryfikacyjny wysłany na jego adres e-mail, który jest wykorzystywany do aktywacji funkcji rejestracji.
- Aby zapewnić odpowiednią skalowalność, layouty aplikacji zostały dostosowane do różnych rozmiarów telefonów, dzięki czemu użytkownicy mogą korzystać z aplikacji na urządzeniach o różnych rozdzielczościach ekranu. Po zalogowaniu się, na ekranie głównym aplikacji wyświetlana jest lista dostępnych urządzeń. Po wybraniu konkretnego urządzenia, użytkownik może uzyskać dostęp do menu konfiguracyjnego, gdzie może dokonywać odpowiednich ustawień.
- Dodatkowo, została stworzona symulacja w języku Python, wykorzystując framework Flask. Symulacja ta może służyć do testowania różnych funkcjonalności aplikacji przed ich wdrożeniem.

2. Wstęp teoretyczny

- W dzisiejszych czasach aplikacje mobilne odgrywają kluczową rolę w naszym codziennym życiu, umożliwiając nam wygodny dostęp do różnorodnych usług i funkcjonalności za pomocą naszych smartfonów. Tworzenie aplikacji mobilnych wymaga zrozumienia i wykorzystania różnych elementów technologicznych, które umożliwiają efektywny rozwój i wdrażanie tych aplikacji.

- W niniejszym projekcie skupiamy się na stworzeniu aplikacji mobilnej, która wykorzystuje szereg narzędzi i technologii w celu zapewnienia optymalnej funkcjonalności i użyteczności. Projekt obejmuje wykorzystanie SQLiteHelper, który umożliwia zarządzanie prostą bazą danych opartą na SQLite. Baza danych przechowuje informacje o urządzeniach i innych istotnych danych dotyczących funkcjonalności aplikacji. SQLiteHelper zapewnia prosty i wygodny sposób tworzenia, modyfikowania i odczytywania danych w bazie SQLite.

- Kolejnym istotnym aspektem projektu są layouty, czyli układy ekranów, które pozwalają użytkownikom interakcję z aplikacją. W projekcie zaprojektowano layouty związane z logowaniem, rejestracją i kodem weryfikacyjnym, które zapewniają przejrzysty układ i odpowiednie pola tekstowe, przyciski i elementy wizualne. Dostosowanie layoutów do różnych rozmiarów telefonów jest kluczowe, aby zapewnić użytkownikom odpowiednią skalowalność i komfort korzystania z aplikacji niezależnie od rozmiaru ekranu urządzenia.

- Funkcja rejestracji za pomocą kodu weryfikacyjnego jest również istotnym elementem projektu. Użytkownik otrzymuje kod weryfikacyjny na adres e-mail, który wprowadza podczas procesu rejestracji. Wprowadzenie poprawnego kodu weryfikacyjnego potwierdza tożsamość użytkownika, zapewniając bezpieczną rejestrację i ochronę danych.

- Menu główne aplikacji jest kluczowym elementem interfejsu użytkownika, umożliwiającym dostęp do różnych opcji i funkcji. Projekt zakłada również rozwinięcie menu głównego, aby zapewnić użytkownikom większą elastyczność i dostęp do dodatkowych funkcji aplikacji.

- Oprócz aplikacji mobilnej, projekt obejmuje również symulację w języku Python przy użyciu frameworka Flask. Symulacja ta zapewnia dodatkową funkcjonalność i komunikację z użytkownikiem. Flask, będący lekkim i elastycznym frameworkiem, umożliwia tworzenie aplikacji internetowych. W projekcie aplikacji mobilnej symulacja w Flask może być wykorzystana do przetwarzania danych, zarządzania logiką aplikacji i obsługi żądań HTTP.

- Podsumowując, projekt aplikacji mobilnej oparty na narzędziach takich jak SQLiteHelper, layouty, kod weryfikacyjny, menu główne i symulacja w języku Python przy użyciu frameworka Flask pozwala na stworzenie kompleksowej aplikacji, która umożliwia użytkownikom zarządzanie urządzeniami i konfigurację funkcjonalności. Wykorzystanie tych różnorodnych elementów technologicznych zapewnia efektywne i elastyczne tworzenie aplikacji mobilnych, dostosowanych do różnych urządzeń i potrzeb użytkowników.

3. Implementacja funkcjonalności

Aplikację dzielimy na 3 moduły:

- Moduł bazy danych
- Moduł procesu rejestracji i logowania
- Moduł menu głównego

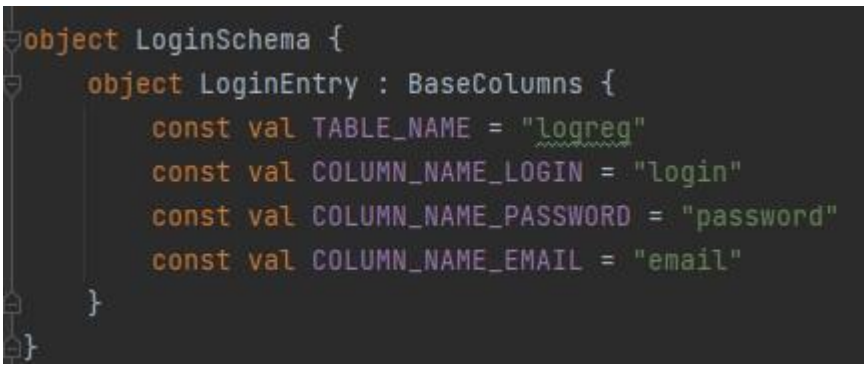
3.1. Moduł bazy danych

Moduł bazy danych:
Ten moduł odpowiada za zarządzanie bazą danych w aplikacji. Wykorzystuje narzędzia takie jak SQLiteHelper do tworzenia, modyfikowania i odczytywania danych w bazie SQLite. W tym module przechowywane są informacje o urządzeniach i inne istotne dane dotyczące funkcjonalności aplikacji. Zapewnia on interfejs do komunikacji z bazą danych i wykonuje operacje związane z przechowywaniem i pobieraniem danych.
Utworzyliśmy 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)
    }
```

Rys. 1. Kod będący konstruktorem klasy Database, która dziedziczy po klasie SQLiteOpenHelper i służy do inicjalizacji obiektu bazy danych SQLite z odpowiednimi parametrami.

Klasa Database dziedziczy po SQLiteOpenHelper, który jest pomocnikiem do zarządzania operacjami na bazie danych. W metodzie onCreate, wywołując metody execSQL z odpowiednimi zapytaniami SQL, tworzone są tabele dla czterech różnych schematów: logowania, menu, statystyk i dane o urządzeniach. Każdy z tych schematów zawiera nazwy tabeli oraz nazwy kolumn, które są używane do odpowiedniego utworzenia struktury tabel w bazie danych. Te schematy przechowują różne rodzaje danych, zapewniając odpowiednią funkcjonalność aplikacji.



Rys. 2. Obiekt LoginSchema definiujący strukturę tabeli "logreg" dla modułu logowania, zawierającą kolumny dla loginu, hasła i adresu e-mail.

Rysunek 3: Definiowanie tabele dla ekranu menu głównego, statystyk i urządzeń do przechowywania lokalnie danych aplikacji

```
private const val logreg_create = "CREATE TABLE ${LoginSchema.LoginEntry.TABLE_NAME} (" +
        "${BaseColumns._ID} INTEGER PRIMARY KEY AUTOINCREMENT," +
        "${LoginSchema.LoginEntry.COLUMN_NAME_LOGIN} TEXT," +
        "${LoginSchema.LoginEntry.COLUMN_NAME_PASSWORD} TEXT," +
        "${LoginSchema.LoginEntry.COLUMN_NAME_EMAIL} TEXT)"

private const val menu_create = "CREATE TABLE ${MenuSchema.MenuEntry.TABLE_NAME} (" +
        "${BaseColumns._ID} INTEGER PRIMARY KEY AUTOINCREMENT," +
        "${MenuSchema.MenuEntry.COLUMN_NAME_NAME} TEXT," +
        "${MenuSchema.MenuEntry.COLUMN_NAME_IP} TEXT," +
        "${MenuSchema.MenuEntry.COLUMN_NAME_STATE} TEXT," +
        "${MenuSchema.MenuEntry.COLUMN_NAME_CONNECTIME} TEXT," +
        "${MenuSchema.MenuEntry.COLUMN_NAME_USEDATA} TEXT)"

private const val device_create = "CREATE TABLE ${DeviceSchema.DeviceEntry.TABLE_NAME} (" +
        "${BaseColumns._ID} INTEGER PRIMARY KEY AUTOINCREMENT," +
        "${DeviceSchema.DeviceEntry.COLUMN_NAME_CONNECTED} TEXT," +
        "${DeviceSchema.DeviceEntry.COLUMN_NAME_LIMIT} TEXT," +
        "${DeviceSchema.DeviceEntry.COLUMN_NAME_DISCONNECTIONTIME} TEXT," +
        "${DeviceSchema.DeviceEntry.COLUMN_NAME_CONNECTIONTIME} TEXT," +
        "${DeviceSchema.DeviceEntry.COLUMN_NAME_LASTUSED} TEXT," +
        "${DeviceSchema.DeviceEntry.COLUMN_NAME_LASTACTIVITY} TEXT)"

private const val static_create = "CREATE TABLE ${StatisticSchema.StatisticEntry.TABLE_NAME} (" +
        "${BaseColumns._ID} INTEGER PRIMARY KEY AUTOINCREMENT," +
        "${StatisticSchema.StatisticEntry.COLUMN_NAME_DAILY} TEXT," +
        "${StatisticSchema.StatisticEntry.COLUMN_NAME_WEEKLY} TEXT," +
        "${StatisticSchema.StatisticEntry.COLUMN_NAME_MONTHLY} TEXT,"+
        "${StatisticSchema.StatisticEntry.COLUMN_NAME_PREVMONTHLY} TEXT)"
```

Rysunek 4: Utworzenia powyższych tabel za pomocą polecenia CREATE TABLE w języku SQL

```
private const val logreg_delete = "DROP TABLE IF EXISTS ${LoginSchema.LoginEntry.TABLE_NAME}"
private const val menu_delete = "DROP TABLE IF EXISTS ${MenuSchema.MenuEntry.TABLE_NAME}"
private const val device_delete = "DROP TABLE IF EXISTS ${DeviceSchema.DeviceEntry.TABLE_NAME}"
private const val static_delete = "DROP TABLE IF EXISTS ${StatisticSchema.StatisticEntry.TABLE_NAME}"
```

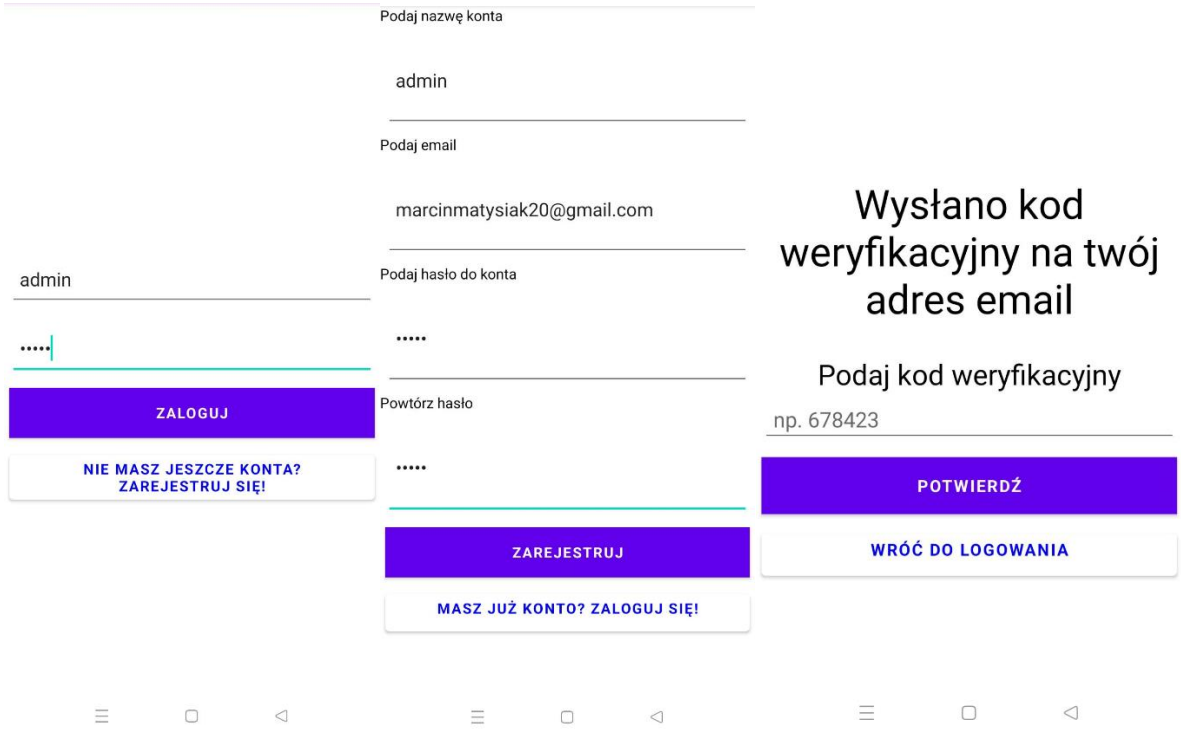
Rysunek 5: Usuwanie utworzonych tabel za pomocą polecenia DROP TABLE w języku SQL

```
override fun onCreate(db: SQLiteDatabase?) {
    db?.execSQL(logreg_create)
    db?.execSQL(menu_create)
    db?.execSQL(device_create)
    db?.execSQL(static_create)
}

override fun onUpgrade(db: SQLiteDatabase?, oldVersion: Int, newVersion: Int) {
    db?.execSQL(logreg_delete)
    db?.execSQL(menu_delete)
    db?.execSQL(device_delete)
    db?.execSQL(static_delete)
    onCreate(db)
}

override fun onDowngrade(db: SQLiteDatabase?, oldVersion: Int, newVersion: Int) {
    onUpgrade(db, oldVersion, newVersion)
}
```

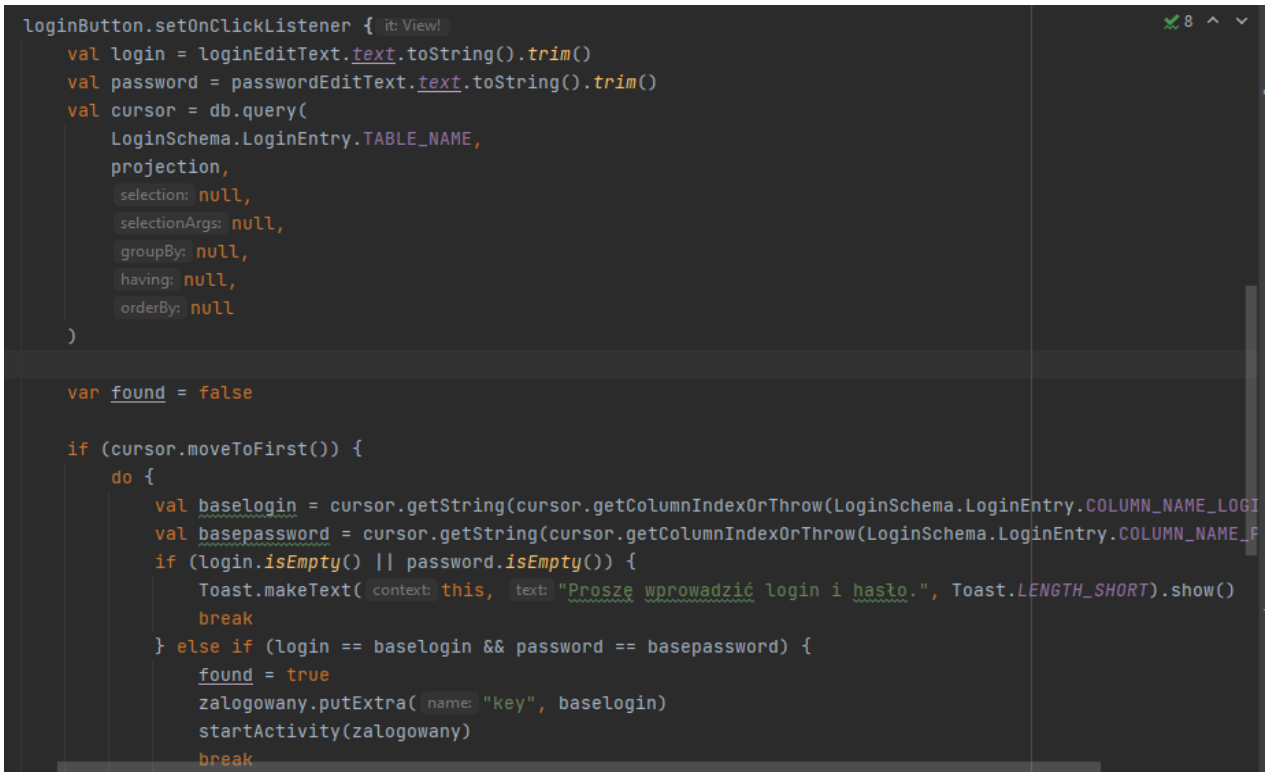
Rysunek 6: Metody tworzenia i modyfikowania bazy danych naszej aplikacji

3.2. Moduł procesu rejestracji i logowania

Ten moduł składa się z trzech aktywności: logowania, rejestracji i weryfikacji. Po uruchomieniu aplikacji pierwsze co widzimy to aktywność logowania, w której (jeśli posiadamy konto) możemy podać swoje dane logowania, po naciśnięciu przycisku zaloguj tworzymy kursor, który przeszukuje bazę danych w poszukiwaniu danej nazwy oraz hasła. Jeśli takowe znajdzie następuje przejście do głównej aktywności, w przeciwnym przypadku zostanie wyświetlony komunikat o błędnych danych logowania. Jeśli takiego konta nie posiadamy możemy przejść za pomocą przycisku do aktywności rejestracji gdzie musimy wprowadzić takie dane jak nazwa, mail oraz hasło. W każdym momencie możemy cofnąć się do logowania lub przejść dalej przenosząc nas do aktywności weryfikacji. W tej aktywności wysyłany jest kod weryfikacyjny za pomocą Java Mail API wraz z nazwą oraz hasłem użytkownika. Otrzymany kod należy przepisać do konkretnego pola. Jeśli kod weryfikacyjny jest poprawny dane w rejestracji zostają zapisane do bazy danych, a użytkownik może wrócić do logowania i zalogować się za pomocą zarejestrowanych danych.

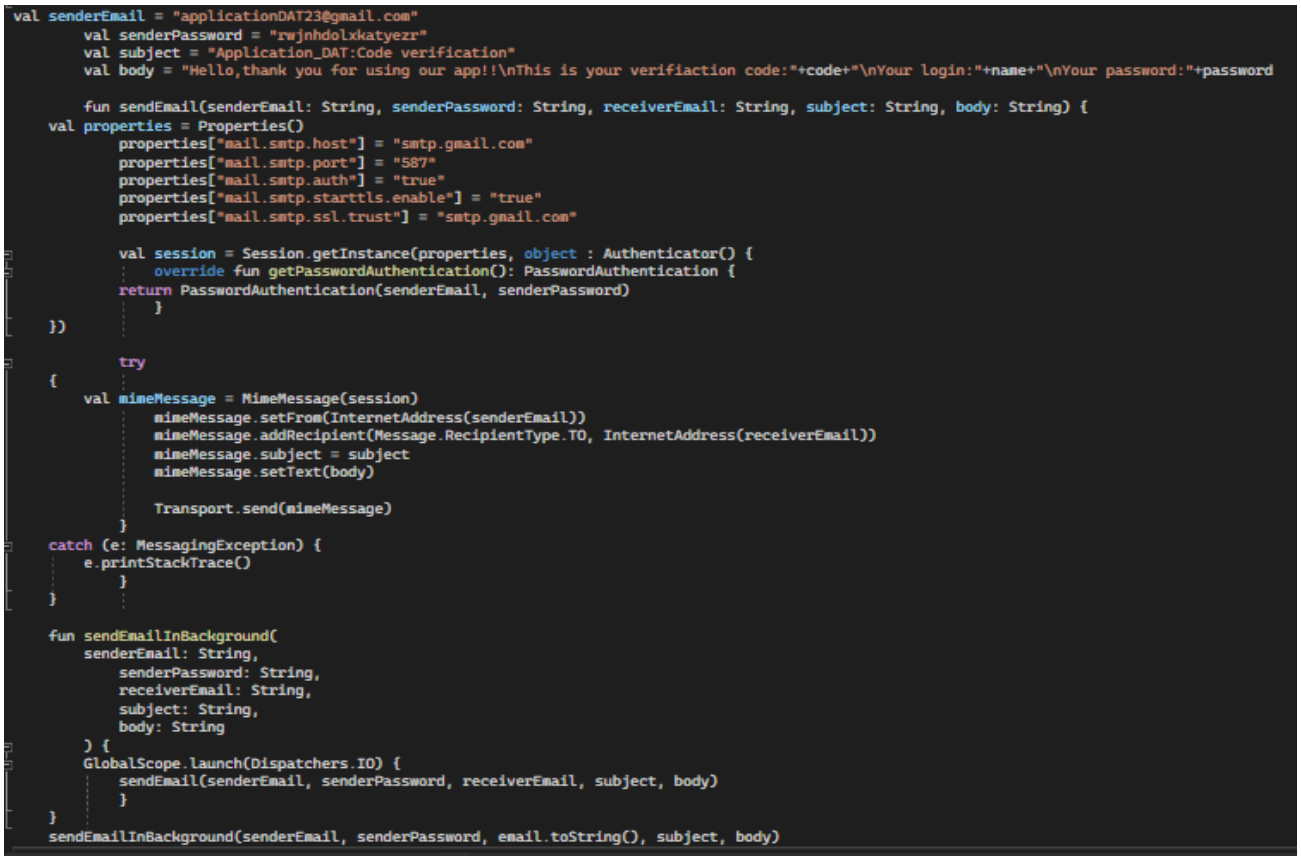


Rys. 7. Wygląd aktywności logowania rejestracji i weryfikacji.



```
loginButton.setOnClickListener { it: View!
    val login = loginEditText.text.toString().trim()
    val password = passwordEditText.text.toString().trim()
    val cursor = db.query(
        LoginSchema.LoginEntry.TABLE_NAME,
        projection,
        selection: null,
        selectionArgs: null,
        groupBy: null,
        having: null,
        orderBy: null
    )

    var found = false

    if (cursor.moveToFirst()) {
        do {
            val baselogin = cursor.getString(cursor.getColumnIndexOrThrow(LoginSchema.LoginEntry.COLUMN_NAME_LOGI
            val basepassword = cursor.getString(cursor.getColumnIndexOrThrow(LoginSchema.LoginEntry.COLUMN_NAME_P
            if (login.isEmpty() || password.isEmpty()) {
                Toast.makeText( context: this,  text: "Proszę wprowadzić login i hasło.", Toast.LENGTH_SHORT).show()
                break
            } else if (login == baselogin && password == basepassword) {
                found = true
                zalogowany.putExtra( name: "key", baselogin)
                startActivity(zalogowany)
                break
```

Rysunek 8 Kod przeszukujący tabele logowania w celu znalezienia odpowiednich danych podanych przez użytkownika w celu zalogowania się



```
val senderEmail = "applicationDAT23@gmail.com"
        val senderPassword = "rwjnhdolxkatyezr"
        val subject = "Application_DAT: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)
                }
    })

            try
    {
        val mimeMessage = MimeMessage(session)
                mimeMessage.setFrom(InternetAddress(senderEmail))
                mimeMessage.addRecipient(Message.RecipientType.TO, InternetAddress(receiverEmail))
                mimeMessage.subject = subject
                mimeMessage.setText(body)

                Transport.send(mimeMessage)
            }
    catch (e: MessagingException) {
        e.printStackTrace()
            }
    }

    fun sendEmailInBackground(
        senderEmail: String,
            senderPassword: String,
            receiverEmail: String,
            subject: String,
            body: String
        ) {
        GlobalScope.launch(Dispatchers.IO) {
            sendEmail(senderEmail, senderPassword, receiverEmail, subject, body)
            }
    }
    sendEmailInBackground(senderEmail, senderPassword, email.toString(), subject, body)
```

Rysunek 9 Kod tworzący wiadomość mailową wysyłającą kod do weryfikacji oraz dane logowania

3.3 Moduł statystyk

Ekran statystyk w raporcie służy do prezentowania różnych informacji i danych dotyczących aktywności lub wyników w formie statystycznej. Podsumowuje on zgromadzone dane i przedstawia je w sposób uporządkowany i czytelny dla użytkownika.

Na szczycie ekranu statystyk wyświetlana jest nazwę urządzenia. Poniżej tego elementu znajdują się informację o czasie przed ekranem. Ten element ma wyśrodkowane wyświetlanie tekstu. Następnie kolejno wyświetlane są różne statystyki:

- „Dzisiejsza aktywność" - informacje o czasu dzisiejszej aktywności danego urządzenia w sieci.
- „Średnia dzienna z tygodnia" – informacje o średnim czasie czasu aktywności z tygodnia.
- „Średnia dzienna z obecnego miesiąca" = informacje o średnim czasie aktywności z obecnego miesiąca.
- „Średnia dzienna z poprzedniego miesiąca" – informacje o średnim czasie aktywności z poprzedniego miesiąca.

W skrócie, ekran statystyk przedstawia różne statystyki związane z aktywnością urządzenia. Wyświetla się nazwa urządzenia oraz informacje o czasie przed ekranem. Dodatkowo, prezentowane są statystyki dotyczące dzisiejszej aktywności, średniej dziennie z tygodnia, średniej dziennie z obecnego miesiąca oraz średniej dziennie z poprzedniego miesiąca.

3.4 Moduł menu głównego

Moduł menu głównego służy do prezentowania głównych opcji i funkcji dostępnych w aplikacji. Moduł wyświetla nazwę routera oraz informację o stanie włączenia routera. Menu główne umożliwia przewijanie zawartości oraz wyświetla informacje o podłączonych urządzeniach. Dla każdego urządzenia zawarte są informacje o urządzeniu takie jak nazwa, status, adres IP, czas połączenia i zużycie danych. Pod tymi informacjami znajdują się przyciski do statystyk i zarządzania danym urządzeniem. Na samym końcu modułu menu głównego znajdują się przyciski do ustawień aplikacji i wyjścia. Ogólnie rzecz biorąc, moduł menu głównego przedstawia listę podłączonych urządzeń wraz z informacjami o nich. Użytkownik może przewijać tę listę, a dla każdego urządzenia dostępne są przyciski do wyświetlania statystyk i zarządzania. Na dole ekranu znajdują się przyciski do ustawień aplikacji i wyjścia z aplikacji.

Menu główne jest kluczowym elementem interfejsu użytkownika, który zapewnia użytkownikowi łatwy dostęp do podstawowych funkcji i nawigacji. Pozwala to na intuicyjne poruszanie się i korzystanie z pełnej gamy dostępnych opcji i funkcji.

```
class MenuActivity : AppCompatActivity(){

    private lateinit var myStatistics: Intent
    private lateinit var myDevices: Intent

    // Define ApiService interface outside onCreate
    interface ApiService {
        @GET("/data")
        fun getData(): Call<List<DataModel>>
    }
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_menu)
        val myLogin = Intent(this, LoginActivity::class.java)
        myStatistics = Intent(this, StatisticsActivity::class.java)
        myDevices = Intent(this, DevicesActivity::class.java)
// Create Retrofit instance
        val retrofit = Retrofit.Builder()
            .baseUrl("http://192.168.137.1:5000/sprawdz_urzadzenie/")
            .addConverterFactory(GsonConverterFactory.create())
            .build()
```

```
// Create ApiService instance
        val apiService = retrofit.create(ApiService::class.java)

        val call = apiService.getData()
        call.enqueue(object : Callback<List<DataModel>> {
            override fun onResponse(
                call: Call<List<DataModel>>,
                response: Response<List<DataModel>>
            ) {
                if (response.isSuccessful) {
                    val data = response.body()
                    if (data != null && data.isNotEmpty()) {
                        val firstData = data[0] // Pobierz pierwszy element z listy danych
                        val nazwaTextView = findViewById<TextView>(R.id.name)
                        val statusTextView = findViewById<TextView>(R.id.status)
                        val zuzyteDaneTextView = findViewById<TextView>(R.id.dane)

                        // Przypisz wartości do TextView
                        nazwaTextView.text = "sadaadasda"
                        statusTextView.text = firstData.status
                        zuzyteDaneTextView.text = firstData.zuzyteDane
                    }
                } else {
                    val nazwaTextView = findViewById<TextView>(R.id.name)
                    nazwaTextView.text = "sadaadasda"
                    Log.d( tag: "MenuActivity",  msg: "Error response: ${response.code()}")
                }
            }

            override fun onFailure(call: Call<List<DataModel>>, t: Throwable) {
                // Handle network failure
                // Print error message or perform error handling
            }
        })
        val back = findViewById<Button>(R.id.back)
        back.setOnClickListener { it: View!
            startActivity(myLogin)
        }
    }
}
```

Rysunek 10 : Implementacja logiki menu głównego

3.5 Moduł ustawień podłączonych urządzeń

Moduł ustawień urządzeń służy do konfiguracji różnych opcji i preferencji właśnie dotyczących urządzeń podłączonych do routera.

Ekran ustawień urządzeń ma następującą strukturę:

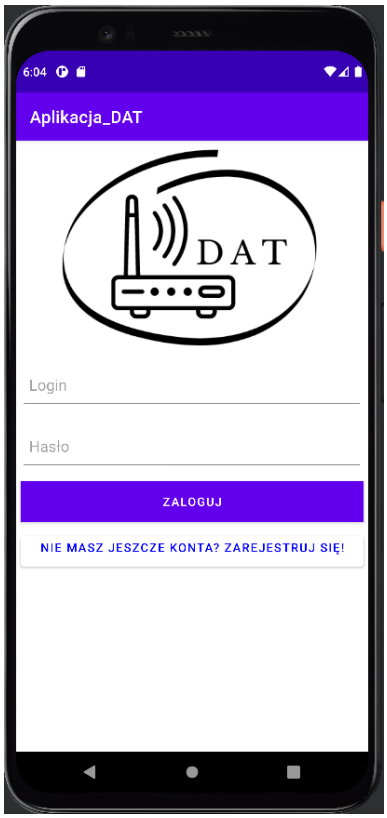
- Nagłówek - wyświetla nazwę urządzenia.
- Sekcja "Settings" - funkcja podłączenia do WiFi.
- Sekcja "Ustaw limit czasu": umożliwia wybór limitu czasu z rozwijanej listy.
- Sekcja "Ustaw godzinę rozłączenia": umożliwia wprowadzenie godziny rozłączenia w formacie czasu.
- Sekcja "Ustaw godzinę podłączenia": umożliwia wprowadzenie godziny podłączenia w formacie czasu.
- Informacje o ostatnim użyciu: wyświetla informację o ostatnio używanej aplikacji oraz informację o ostatniej aktywności.
- Przyciski: "Wróć", który służy do powrotu, przycisk "Zapisz", który służy do zapisania wprowadzonych zmian.

Ekran ustawień umożliwia dostosowanie różnych aspektów działania urządzenia, takich jak połączenie z siecią, limity czasowe i harmonogramy podłączania. Użytkownik może kontrolować te ustawienia w zależności od swoich preferencji i potrzeb.

```
class DevicesActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_devices)

        val items = listOf("1min","5min","10min","15min","20min","30min","40min","50min","1h",

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

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

        autoComplete.setAdapter(adapter)

        autoComplete.onItemClickListener = AdapterView.OnItemClickListener {
                adapterView, view, i, l ->

            val itemSelected = adapterView.getItemAtPosition(i)
            Toast.makeText( context: this, text: "item: $itemSelected",Toast.LENGTH_SHORT).show()

        }
        val back = findViewById<Button>(R.id.button3)
        val myIntent = Intent( packageContext: this, MenuActivity::class.java)
        back.setOnClickListener { it: View!
            startActivity(myIntent)
        }
    }

}
```

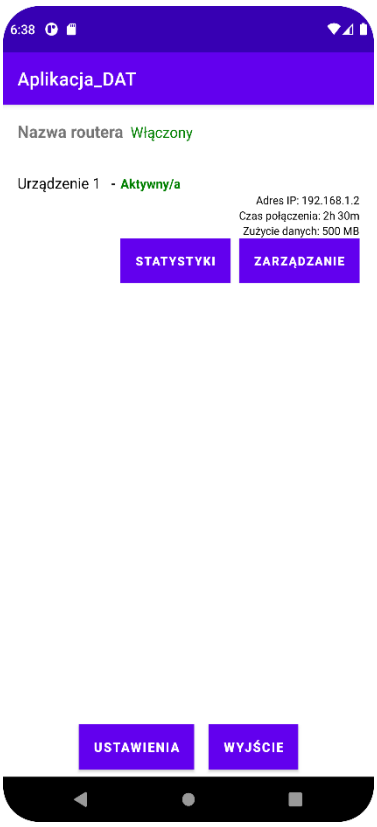
Rysunek 11: Prosta Implementacja logiki ekranu statystyk

3.6 Interfejs graficzny aplikacji

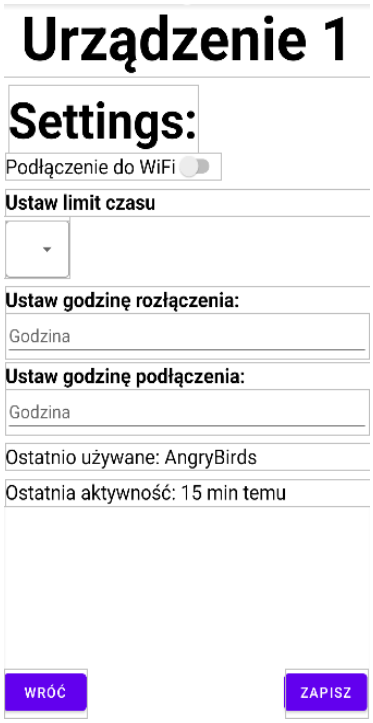


Rysunek 12: Ekran logowania

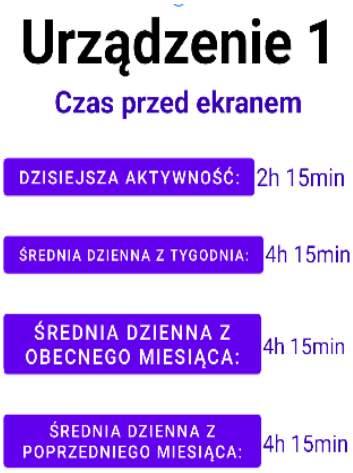
Rysunek 13: Ekran rejestracji



Rysunek 14: Menu główne



Rysunek 14: Ekran ustawień urządzeń



Rysunek 15: Ekran statystyk

4. Flask

- Aplikacja Flask obsługuje serwer HTTP i ma kilka endpointów, które umożliwiają interakcję z urządzeniami. Oto podsumowanie funkcjonalności aplikacji:
- Endpoint /sprawdz_urzadzenie obsługuje żądania typu GET. Odpowiedź na tym endpointcie zwraca JSON z polem "aktywne" ustawionym na True.
- Dodano możliwość zwracania statusu wszystkich urządzeń poprzez endpoint /status_urzadzen. Ten endpoint nie wymaga żadnych parametrów i zwraca wszystkie urządzenia wraz z ich statusem.
- Możliwe jest wybranie tylko jednego urządzenia poprzez parametr "nazwa_urzadzenia" w endpointcie /wybrane_urzadzenie. Ta funkcjonalność umożliwia otrzymanie statusu i informacji tylko dla wybranego urządzenia. Nazwę urządzenia można przekazać jako parametr w przeglądarce lub w zapytaniu GET.
- Dodano możliwość filtrowania urządzeń na podstawie statusu "aktywne" poprzez parametr zapytania GET w endpointcie /filtruj_urzadzenia. Jeśli przekazano wartość "aktywne=true" lub "aktywne=false", zostaną zwrócone tylko urządzenia o odpowiednim statusie.
- Dodano opcję wyboru dowolnej liczby urządzeń poprzez przekazanie listy nazw urządzeń jako parametru zapytania GET w endpointcie /wybierz_urzadzenia. Można również zastosować filtrowanie i ograniczanie liczby zwracanych urządzeń na podstawie listy nazw urządzeń oraz parametrów "aktywne" i "liczba_urzadzen" w zapytaniu GET. Dzięki temu można bardziej precyzyjnie określić, jakie urządzenia mają zostać zwrócone.

  Dzięki tym funkcjonalnościom aplikacja umożliwia elastyczne zarządzanie urządzeniami i precyzyjne wybieranie tych, na których użytkownik skupia swoją uwagę.

5. Wnioski i podsumowanie

- Domowy asystent technologiczny dla rodziców jest wszechstronnym narzędziem, które może pomóc w zarządzaniu technologią w domu i zapewnieniu bezpieczeństwa dzieciom. Zapewnia on kontrolę nad czasem spędzonym na technologii, monitoruje treści online, wspiera rozwój umiejętności społecznych i dostarcza zasobów edukacyjnych dla rodziców. Korzystanie z takiej aplikacji może przyczynić się do zdrowego i odpowiedzialnego korzystania z technologii przez dzieci, podnosząc świadomość rodziców na temat cyfrowego świata i zachowując bezpieczeństwo online dla najmłodszych użytkowników.

- Tworzenie aplikacji mobilnych wymaga zrozumienia i wykorzystania różnorodnych narzędzi i technologii, takich jak bazy danych, layouty, kod weryfikacyjny i frameworki. W przypadku opisywanego projektu, wykorzystano SQLiteHelper do zarządzania bazą danych, odpowiednio dostosowane layouty dla różnych rozmiarów ekranów oraz kod weryfikacyjny do potwierdzenia tożsamości użytkowników. To pokazuje, że zrozumienie i umiejętne wykorzystanie tych narzędzi są kluczowe dla efektywnego tworzenia aplikacji mobilnych.

- Bezpieczeństwo i ochrona danych są istotnymi aspektami projektowania aplikacji. W przypadku opisanej aplikacji, proces rejestracji i logowania obejmuje wysyłanie kodu weryfikacyjnego na adres e-mail użytkownika. To zapewnia dodatkową warstwę bezpieczeństwa i ochronę przed nieautoryzowanym dostępem. Wniosek z tego jest taki, że projektowanie aplikacji z uwzględnieniem mechanizmów bezpieczeństwa jest niezbędne dla ochrony danych użytkowników.

- Wdrażanie symulacji, takiej jak opisana symulacja w języku Python przy użyciu frameworka Flask, może być wartościowym narzędziem testowym przed pełnym wdrożeniem aplikacji mobilnej. Symulacja pozwala na przetestowanie różnych funkcjonalności i logiki aplikacji, co może przyczynić się do poprawy jakości i użyteczności ostatecznej wersji aplikacji. Wniosek z tego jest taki, że symulacje mogą być przydatnym narzędziem w procesie rozwoju aplikacji mobilnych.

- Projekt zespołowy opisujący aplikację mobilną do zarządzania urządzeniami i konfiguracji funkcjonalności demonstruje zastosowanie różnorodnych narzędzi i technologii, jak również uwzględnienie aspektów bezpieczeństwa i testowania symulacyjnego. Projekt ten pokazuje, jak ważne jest zrozumienie i umiejętne wykorzystanie tych elementów w celu stworzenia efektywnej i użytecznej aplikacji mobilnej.

- Moduł menu głównego zapewnia użytkownikowi łatwy dostęp do głównych opcji i funkcji aplikacji. Dzięki temu użytkownik może szybko nawigować po aplikacji i zarządzać podłączonymi urządzeniami.

- Ekran statystyk dostarcza użytkownikowi uporządkowanych informacji i danych dotyczących aktywności urządzenia. Pozwala to na monitorowanie czasu aktywności oraz porównywanie statystyk z różnych okresów, takich jak dzisiaj, średnia dzienna z tygodnia, obecnego miesiąca i poprzedniego miesiąca. Dzięki temu użytkownik może śledzić i analizować wykorzystanie urządzeń w sieci.

- Moduł ustawień podłączonych urządzeń umożliwia użytkownikowi dostosowanie różnych parametrów i preferencji związanych z podłączonymi urządzeniami. Obejmuje to funkcje takie jak połączenie z siecią Wi-Fi, ustawianie limitów czasowych, harmonogramy rozłączania i podłączania urządzeń. Użytkownik ma pełną kontrolę nad tymi ustawieniami, co pozwala mu dostosować działanie urządzeń do swoich potrzeb.