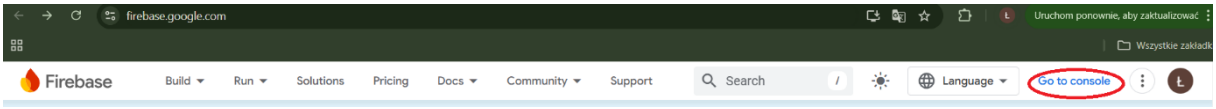
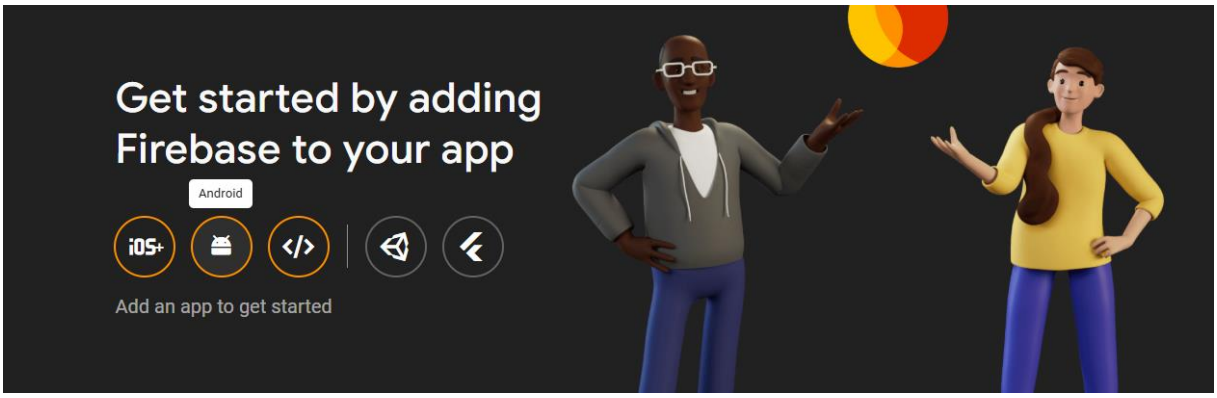
Retrofit i Firebase 1

Firebase to platforma do tworzenia aplikacji mobilnych i internetowych.
Tworzenie aplikacji wykorzystującej Firebase zaczyna się zazwyczaj od bazy danych NoSQL w usłudze Cloud Firestore.

firebase.google.com
Przechodzimy do konsoli I tworzymy nowy projekt w Firebase (równocześnie tworząc nowy projekt androidowy w android studio):



Postępujemy zgodnie z instrukcją dla aplikacji Android:



Rejestrujemy aplikację (musi mieć takie samo id jak w projekcie androidowym, w build.gradle np.:
applicationId = "com.example.weatherapp"),
pobieramy json do katalogu app,
oraz dodajemy plugin w build gradle na poziomie projektu:

```
id("com.google.gms.google-services") version "4.4.2" apply false
```

Dodajemy plugin na poziomie aplikacji:

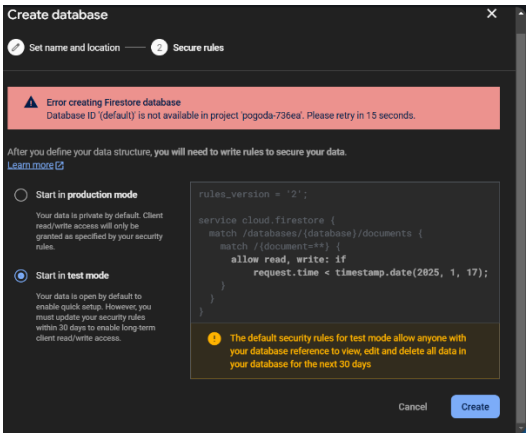
```
id("com.google.gms.google-services")
```

Na koniec dodajemy zależności:

```
implementation(platform("com.google.firebase:firebase-bom:33.7.0"))
implementation("com.google.firebase:firebase-firestore-ktx")
```

I synchronizujemy project.

Firebase BOM (Bill of Materials) - zarządza wersjami wszystkich bibliotek Firebase w jednym miejscu.
firebase-firestore-ktx - biblioteka Firebase dla Cloud Firestore



Tworzymy bazę (testową bez reguł autoryzacji)

Aby móc pobrać pogodę z api OpenWeatherMap należy założyć konto i skopiować klucz aplikacji.
https://openweathermap.org/
np:

```
object Constants {
            const val OPEN_WEATHER_API_KEY = "1234"
        }
```

```
implementation("com.squareup.retrofit2:retrofit:2.9.0")
implementation("com.squareup.retrofit2:converter-gson:2.9.0")
```

Żądanie może wyglądać następująco:

https://api.openweathermap.org/data/2.5/weather?q=Warsaw&units=metric&appid=12345678

Odpowiedź:

```
{
    "coord": {
        "lon": 21.0118,
        "lat": 52.2298
    },
    "weather": [
        {
            "id": 804,
            "main": "Clouds",
            "description": "overcast clouds",
            "icon": "04n"
        }
    ],
    "base": "stations",
    "main": {
        "temp": 6.9,
        "feels_like": 3.24,
```

```
        "temp_min": 6.47,
        "temp_max": 8.13,
        "pressure": 1020,
        "humidity": 94,
        "sea_level": 1020,
        "grnd_level": 1008
    },
    "visibility": 10000,
    "wind": {
        "speed": 6.17,
        "deg": 260
    },
    "clouds": {
        "all": 100
    },
    "dt": 1734501556,
    "sys": {
        "type": 2,
        "id": 2032856,
        "country": "PL",
        "sunrise": 1734504086,
        "sunset": 1734531838
    },
    "timezone": 3600,
    "id": 756135,
    "name": "Warsaw",
    "cod": 200
}
```


Wybieramy kilka danych z odpowiedzi (nazwa miasta - name, opis-description z listy weather. Poszczególnym wartościom z json odpowiadają obiekty kotlinowe. @SerializedName wskazuje na nazwę z json.

```
import com.google.gson.annotations.SerializedName

data class WeatherResponse(
    @SerializedName("name") val name: String,
    @SerializedName("weather") val weather: List<WeatherDescription>,
    @SerializedName("main") val main: MainWeather
)

data class WeatherDescription(
    @SerializedName("description") val description: String
)

data class MainWeather(
    @SerializedName("temp") val temp: Double
)
```

Retrofit upraszcza wysyłanie żądań HTTP, oraz mapowanie odpowiedzi na obiekty Kotlin

https://www.geeksforgeeks.org/retrofit-with-kotlin-coroutine-in-android/
https://square.github.io/retrofit/

Przygotowanie żądania w retrofit pozyskującego dane z api pogodowego:

@GET("data/2.5/weather") - Wskazuje, że ta funkcja wywołuje żądanie HTTP typu GET. Ścieżka "data/2.5/weather" jest dołączana do base URL zdefiniowanego wcześniej w konfiguracji Retrofit.

```
import com.example.weatherapp.model.WeatherResponse
import retrofit2.http.GET
import retrofit2.http.Query

interface WeatherApiService {
    @GET("data/2.5/weather")
    suspend fun getCurrentWeather(
        @Query("q") cityName: String, // Format: "Warsaw"
        @Query("appid") apiKey: String, // Api key
        @Query("units") units: String = "metric" // "metric" dla Celsiusza
    ): WeatherResponse
}
```

Retrofit wykona funkcję getCurrentWeather asynchronicznie.

```
Do wykonania żądania posłużymy się instancją Retrofit:
import okhttp3.OkHttpClient
import retrofit2.Retrofit
import retrofit2.converter.gson.GsonConverterFactory

object RetrofitInstance {
    private const val BASE_URL = "https://api.openweathermap.org/"

    val api: WeatherApiService by lazy {
        Retrofit.Builder()
            .baseUrl(BASE_URL)
            .addConverterFactory(GsonConverterFactory.create())
            .build()
            .create(WeatherApiService::class.java)
    }

}
```

Retrofit Instance to singleton, który przechowuje skonfigurowaną instancję Retrofit. Używamy OkHttmyClient do wysyłania żądań HTTP i GsonConverterFactory do konwertowania JSON z odpowiedzi na obiekty Kotlin typu WeatherApiService.

Pobrane dane w postaci odpowiedzi, moglibyśmy przekształcić na wpis do bazy Firestore.
Na podstawie danych jakie zwraca API OpenWeatherMap tworzymy model danych:

```
data class Weather(
    val city: String = "",
    val temperature: Double = 0.0,
    val description: String = "",
```

```
    val id: String = ""
)
```

Do dodawania, pobierania I usuwania wpisów z Firestore trzeba stworzyć klasę repozytorium, każdy wpis będzie znajdował się w „weather” :

```
import com.example.weatherapp.model.Weather
import com.google.firebase.firestore.FirebaseFirestore
import kotlinx.coroutines.tasks.await

class WeatherRepository {
    private val db = FirebaseFirestore.getInstance()
    private val collection = db.collection("weather")

    // Dodanie nowego wpisu:
    suspend fun addWeather(weather: Weather) {
        val document = collection.document()
        val weatherWithId = weather.copy(id = document.id)
        document.set(weatherWithId).await()
    }

    // Pobranie wszystkich wpisów:
    suspend fun getWeathers(): List<Weather> {
        val snapshot = collection.get().await()
        return snapshot.documents.mapNotNull {
it.toObject(Weather::class.java) }
    }

    // Usunięcie wpisu:
    suspend fun deleteWeather(id: String) {
        collection.document(id).delete().await()
    }
}
```

FirebaseFirestore – służy do komunikacji z bazą Firebase.
collection – wynik zapytania do Firestore (QuerySnapshot), zawiera listę dokumentów.

Przy pobieraniu wszystkich wpisów:

collection.get() zwraca obiekt Task<QuerySnapshot>
QuerySnapshot: Zawiera wyniki zapytania, czyli listę dokumentów z kolekcji.
Firebase używa obiektu Task do obsługi asynchroniczności.
await() przekształca funkcję z Firebase na zawieszoną, która może być użyta w korutynie lub w innej funkcji zawieszonej.

snapshot zawiera dokumenty w postaci listy DocumentSnapshot z zawartością z jsona.
Wszystkie DocumentSnapshot są mapowane na klasę Weather.

document.set(weatherWithId).await() – dodaje wpis.
collection.document(id).delete().await() – usuwa wpis.

Wpisy z Firestore możemy przechować w liście, a tą wyświetlać za pomocą jetpack compose.
Najlepszym narzędziem do edytowania listy i jej widoku będzie ViewModel.
ViewModel użyje repozytorium do usunięcia lub dodania wpisu, a następnie pobierze wszystkie wpisy aktualizując listę.

```
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.example.weatherapp.repository.WeatherRepository
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.launch

class WeatherViewModel : ViewModel() {
    private val repository = WeatherRepository()

    private val _weatherList = MutableStateFlow<List<Weather>>(emptyList())
    val weatherList: StateFlow<List<Weather>> = _weatherList

    // Pobranie danych:
    fun getWeather() {
        viewModelScope.launch {
            _weatherList.value = repository.getWeathers()
        }
    }

    // Dodanie nowego wpisu, odświeżenie listy:
    fun addWeather(weather: Weather) {
        viewModelScope.launch {
            repository.addWeather(weather)
            getWeather()
        }
    }

    // Usunięcie wpisu, odświeżenie listy:
    fun deleteWeather(id: String) {
        viewModelScope.launch {
            repository.deleteWeather(id)
            getWeather()
        }
    }
}
```

Funkcje zawieszone wywołane są w viewModelScope. viewModelScope to specyficzny zakres korutyn stworzony w ViewModel. Umożliwia on uruchamianie korutyn, które są automatycznie anulowane, gdy ViewModel jest niszczony.

Jeżeli chcemy po pobraniu danych z api jednocześnie zapisać te dane do bazy można w klasie WeatherViewModel użyć instancję Retrofit i repozytorium do zapisu i odświeżenia listy:

```
fun addWeatherFromApi(cityName: String, apiKey: String) {
    viewModelScope.launch {
        try {
            val response = RetrofitInstance.api.getCurrentWeather(cityName,
apiKey)
            val weather = Weather(
                city = response.name,
                temperature = response.main.temp,
                description = response.weather.firstOrNull()?.description
?: "N/A"
            )
            repository.addWeather(weather)
            fetchWeather()
        } catch (e: Exception) {
```

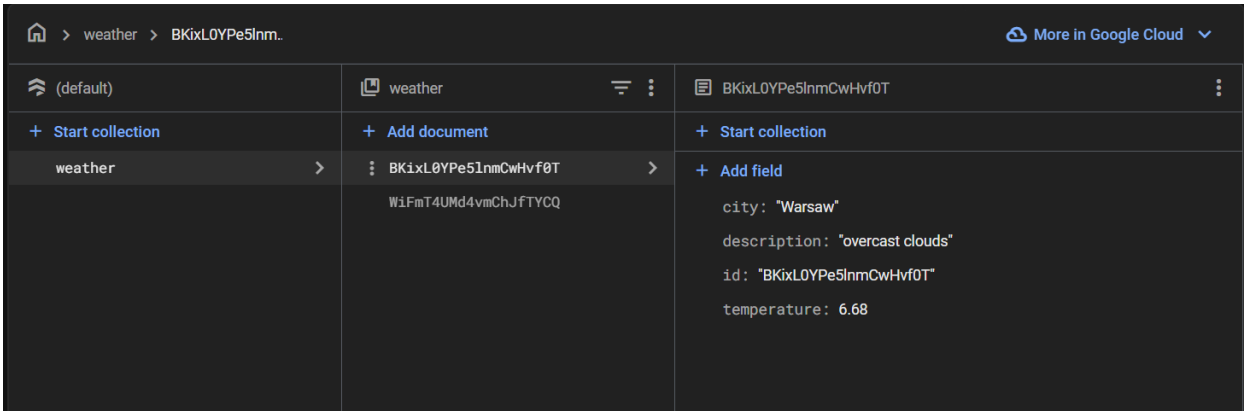
```
            e.printStackTrace()
        }
    }
}
```

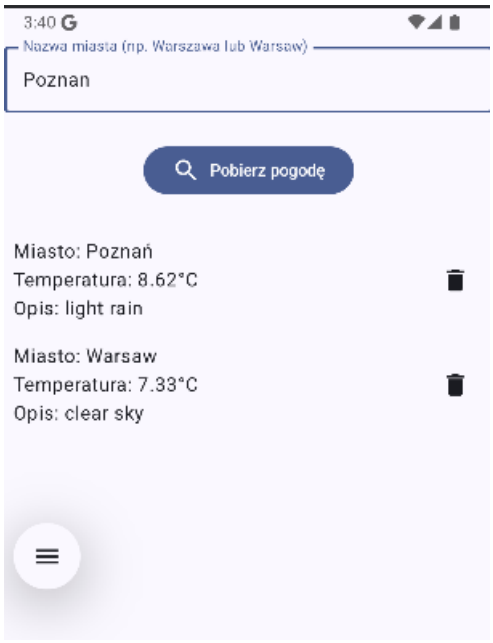
Użytkownik będzie za pomocą formularza podając odpowiednie dane i klikając przycisk pobierz pogodę, pobierze dane i dokona wpisu do bazy.

```
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.width
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Delete
import androidx.compose.material.icons.filled.Search
import androidx.compose.material3.Button
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.OutlinedTextField
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.collectAsState
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifie
import androidx.compose.ui.unit.dp
import com.example.weatherapp.model.WeatherViewModel
import com.example.weatherapp.utils.Constants
import androidx.lifecycle.viewmodel.compose.viewModel
import com.example.weatherapp.model.Weather



@Composable
fun WeatherScreen(viewModel: WeatherViewModel = viewModel(),modifier:
Modifier = Modifier) {
    var cityName by remember { mutableStateOf("")
    //Listę wypełniamy danymi z Firebase dopiero po załadowaniu widoku w
korutynie LaunchedEffect.
    val weatherList by viewModel.weatherList.collectAsState(initial =
emptyList())

LaunchedEffect(Unit) {
    viewModel.getWeathers()
}

        Box(
            modifier = modifier
                .fillMaxSize()
        ) {
```

```
            Column(
                verticalArrangement = Arrangement.Top,
                horizontalAlignment = Alignment.CenterHorizontally
            ) {
                OutlinedTextField(
                    value = cityName,
                    onValueChange = { cityName = it },
                    label = { Text("City(np Warszawa lub Warsaw)") },
                    modifier = Modifier.fillMaxWidth()
                )
//Przycisk - gdy klikniemy, pobierzemy dane z api I dodamy do bazy:
                Button(
                    onClick = {
                        viewModel.addWeatherFromApi(
                            cityName = cityName,
                            apiKey = Constants.OPEN_WEATHER_API_KEY
                        )

                    },
                    modifier = modifier,
                    enabled = cityName.isNotBlank()
                ) {
                    Icon(imageVector = Icons.Default.Search,
contentDescription = "seaerch")
                    Spacer(modifier = Modifier.width(8.dp))
                    Text("Get Weather")
                }
//LazyColumn z listą danych pogodowych:
                LazyColumn {
                    items(weatherList) { weather ->

                        WeatherItem(weather = weather,
 //funkcja do usuniecia danych pogodowych:
 onDelete = { viewModel.deleteWeather(weather.id) })
                    }
                }

            }
        }
    }
//Element kolumny z danymi pogodowymi + ikona z funkcją onDelete na
kliknięcie
Composable
fun WeatherItem(weather: Weather, onDelete: () -> Unit) {
    Row(
        modifier = Modifier
            .fillMaxWidth()
            .padding(8.dp),
        verticalAlignment = Alignment.CenterVertically
    ) {
        Column(modifier = Modifier.weight(1f)) {
            Text(text = "Miasto: ${weather.city}")
            Text(text = "Temperatura: ${weather.temperature}°C")
            Text(text = "Opis: ${weather.description}")
        }
        IconButton(onClick = onDelete) {
            Icon(imageVector = Icons.Default.Delete, contentDescription =
"Delete")
        }
    }
}
```

Wpisy w Firebase store:





Zadanie:

Pobrać za pomocą retrofit dane dotyczące książek z dowolnego api. Jeżeli api zwraca kilka książek – wybrać 1 z listy.
Zapisać te dane do bazy w Firebase.
Pozwolić użytkownikowi na dodawanie, przeglądanie oraz usuwanie tych danych.
Pozwolić użytkownikowi na modyfikowanie tych danych.