내일은 플러터

더 멋진 내일(Tomorrow)을 위한 내일(My Career)

응용실전편
부록

02

내일은 플러터

고급 주제를 활용한 예제

01 나만의 지도 만들기

02 SNS 만들기

01 나만의 지도 만들기

✓핵심 키워드

네이버 API, 네이버 지도, Database, Marker

여기서는 무얼 배울까

이번에는 네이버 지도를 사용하는 방법을 배운다. 데이터 베이스를 활용하여 지도와 함께 다루는 법을 학습하여 새로운 장소를 등록하고, 기존 장소를 수정하면서 지도에서 마커를 띄워 보도록 하자.

네이버 지도

네이버 지도를 사용하기 위해서는 다음과 같은 절차를 따라야 한다.

네이버 개발자 센터 가입

네이버 개발자 센터(https://developers.naver.com)에 가입하고 로그인한다. 개발자 센터에서 애플리케이션을 등록하고 API 키를 발급받아야 한다.

애플리케이션 등록

네이버 개발자 센터에서 "내 애플리케이션"을 클릭하고 "애플리케이션 등록"을 선택한다. 애플리케이션 이름과 사용할 API를 선택하여 애플리케이션을 등록한다. 등록 후에는 API 키를 발급받을 수 있다.

네이버 지도 API 추가

애플리케이션 등록 후에는 "API 설정" 탭으로 이동하여 사용할 네이버 지도 API를 추가해야 한다. "지도/로컬" 카테고리에서 "지도" API를 선택하고 "사용 설정"을 클릭한다.

API 키 발급

API 설정이 완료되면 "API 키" 탭에서 API 키를 발급받을 수 있다. 발급된 API 키는 앱에서 네이버 지도를 사용하기 위해 필요하며, main함수에 추가될 예정이다.

의존성 추가

네이버 지도 API와 관련된 의존성을 추가하기 위해 naver_map_plugin을 사용한다. pubspec.yaml 파일에 다음과 같이 의존성을 추가한다.

```
dependencies:
  flutter:
    sdk: flutter
  naver_map_plugin: ^X.X.X
```

패키지 설치

터미널 또는 명령 프롬프트에서 프로젝트 폴더로 이동한 상태에서 다음 명령어를 실행하여 필요한 패키지를 설치한다.

터미널 명령

```
flutter pub get
```

이제 프로젝트 설정이 완료되었다. 다음 단계로 앱의 구조와 화면을 설계하고 기능을 구현해 나갈 수 있다.

메인 화면(Main Screen)

```
import 'package:flutter/material.dart';
import 'package:flutter_naver_map/flutter_naver_map.dart';
import '/widgets/map_view.dart';

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await NaverMapSdk.instance.initialize(clientId: 'API_KEY');
  runApp(MyApp());
```

```
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Places List',
      theme: ThemeData(
        primarySwatch: Colors.blue,
        visualDensity: VisualDensity.adaptivePlatformDensity,
      ),
      home: MapView(),
    );
  }
}
```

위의 코드에서는 WidgetsFlutterBinding.ensureInitialized();를 호출하여 플러터 엔진과 호스트 플랫폼 간의 연결을 초기화한다. 그리고 NaverMapSdk를 초기화하고 MyApp 위젯을 실행한다. NaverMapSdk.instance.initialize(clientId: 'API_KEY'): 이 부분에서는 네이버 지도 SDK를 초기화하며, 여기서 사용자는 자신의 API_KEY로 수정하여야 한다. MapView()는 다른 파일(/widgets/map_view.dart)에 있는 위젯을 의미한다. widgets 폴더 내에 있으며, 수정하여 사용할 수 있다.

모델 (Model)

```
class Place {
  int? id;
  String name;
  double latitude;
  double longitude;
  String? phoneNumber;
  String? imagePath;

  Place({
    this.id,
    required this.name,
    required this.latitude,
    required this.longitude,
    this.phoneNumber,
```

```
    this.imagePath,
  });

  Map<String, dynamic> toMap() {
    return {
      'id': id,
      'name': name,
      'latitude': latitude,
      'longitude': longitude,
      'phoneNumber': phoneNumber,
      'imagePath': imagePath,
    };
  }

  factory Place.fromMap(Map<String, dynamic> map) {
    return Place(
      id: map['id'],
      name: map['name'],
      latitude: map['latitude'],
      longitude: map['longitude'],
      phoneNumber: map['phoneNumber'],
      imagePath: map['imagePath'],
    );
  }
}
```

id, name, latitude, longitude, phoneNumber, imagePath는 Place 객체의 속성을 나타낸다. 각각 장소의 ID, 이름, 위도, 경도, 전화번호, 이미지 경로를 나타낸다. toMap() 메서드는 Place 객체의 속성을 Map 형태로 변환한다. 이렇게 하면 데이터베이스에 저장하거나 네트워크를 통해 전송하기 쉬워진다. fromMap() 팩토리 메서드는 Map을 받아서 해당 정보로부터 Place 객체를 생성합니다. 해당 파일은 models 폴더 내에 place.dart라는 이름으로 만들어져 있으며 수정하여 사용할 수 있다.

데이터베이스 (Database)

```
import 'dart:io';

import 'package:path/path.dart';
import 'package:path_provider/path_provider.dart';
```

```
import 'package:sqflite/sqflite.dart';
import 'models/place.dart';

class DatabaseHelper {
  static final _databaseName = "places_database.db";
  static final _databaseVersion = 1;

  static final table = "places";

  static final columnId = "id";
  static final columnName = "name";
  static final columnLatitude = "latitude";
  static final columnLongitude = "longitude";
  static final columnPhoneNumber = "phoneNumber";
  static final columnImagePath = "imagePath";

  DatabaseHelper._privateConstructor();
  static final DatabaseHelper instance = DatabaseHelper._privateConstructor();

  static Database? _database;
  Future<Database> get database async {
    if (_database != null) return _database!;
    _database = await _initDatabase();
    return _database!;
  }

  _initDatabase() async {
    Directory documentsDirectory = await getApplicationDocumentsDirectory();
    String path = join(documentsDirectory.path, _databaseName);
    return await openDatabase(path,
        version: _databaseVersion, onCreate: _onCreate);
  }

Future _onCreate(Database db, int version) async {
    await db.execute('''
          CREATE TABLE $table (
            $columnId INTEGER PRIMARY KEY,
            $columnName TEXT NOT NULL,
            $columnLatitude REAL NOT NULL,
            $columnLongitude REAL NOT NULL,
            $columnPhoneNumber TEXT,
            $columnImagePath TEXT
          )
```

```
        ''');

    await db.transaction((txn) async {
      var batch = txn.batch();
      batch.insert(table, {
        columnName: 'Initial place 1',
        columnLatitude: 37.5665,
        columnLongitude: 126.9780,
        columnPhoneNumber: '010-0000-0000',
        columnImagePath: '/image1',
      });
      batch.insert(table, {
        columnName: 'Initial place 2',
        columnLatitude: 37.5666,
        columnLongitude: 126.9781,
        columnPhoneNumber: '010-1111-1111',
        columnImagePath: '/image2',
      });
      await batch.commit();
    });
  }

  Future<int> insert(Place place) async {
    Database db = await database;
    return await db.insert(table, place.toMap());
  }

  Future<int> getMaxPlaceId() async {
    final db = await database;
    var result = await db.rawQuery("SELECT MAX(id)+1 as id FROM places");
    if (result.isEmpty) {
      return 1;
} else {
      int id = result.first["id"] == null ? 1 : result.first["id"] as int;
      return id;
    }
  }

  Future<List<Place>> getAllPlaces() async {
    Database db = await database;
    final List<Map<String, dynamic>> maps = await db.query(table);

    return List.generate(maps.length, (i) => Place.fromMap(maps[i]));
```

```
  }

  Future<List<Place>> getPlaceList() async {
    Database db = await instance.database;
    final List<Map<String, dynamic>> placeMaps = await db.query('places');

    return List.generate(placeMaps.length, (i) {
      return Place(
        id: placeMaps[i]['id'],
        name: placeMaps[i]['name'],
        latitude: placeMaps[i]['latitude'],
        longitude: placeMaps[i]['longitude'],
      );
    });
  }

  Future<int> update(Place place) async {
    Database db = await database;
    return await db.update(table, place.toMap(),
        where: '$columnId = ?', whereArgs: [place.id]);
  }

  Future<int> delete(int id) async {
    Database db = await database;
    return await db.delete(table, where: '$columnId = ?', whereArgs: [id]);
  }
}
```

Flutter 앱에서 SQLite 데이터베이스를 관리하는 데 도움을 주는 클래스인 'DatabaseHelper'를 정의하였다. 이 클래스는 Singleton 패턴을 사용하여 앱 전체에서 단 한 번만 인스턴스화되며, 모든 데이터베이스 작업은 이 인스턴스를 통해 실행된다. 해당 파일은 database 폴더 내에 database_helper.dart라는 이름으로 만들어져 있으며 수정하여 사용할 수 있다.

지도 화면 (Widget)

```
import 'package:flutter/material.dart';
import 'package:flutter_naver_map/flutter_naver_map.dart';
import '/models/place.dart';
import '/database/database_helper.dart';
```

```
import 'place_form.dart';

class MapView extends StatefulWidget {
  const MapView({Key? key}) : super(key: key);

  @override
  _MapViewState createState() => _MapViewState();
}

class _MapViewState extends State<MapView> {
  NaverMapController? _controller;

  @override
  void initState() {
    super.initState();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("Naver Map View"),
      ),
      body: Stack(
        children: [
          NaverMap(
            options: const NaverMapViewOptions(),
            forceGesture: false,
            onMapReady: (controller) {
              _controller = controller;
              _loadMarkers();
            },
          ),
          Align(
            alignment: Alignment.bottomRight,
            child: Padding(
              padding: const EdgeInsets.all(16.0),
              child: FloatingActionButton(
                onPressed: _navigateToPlaceForm,
                child: const Icon(Icons.add),
              ),
            ),
          ),
```

```
        ],
      ),
    );
  }

  void _navigateToPlaceForm() async {
    await Navigator.push(
      context,
      MaterialPageRoute(
        builder: (context) => PlaceForm(),
      ),
    );
  }

  Future<void> _loadMarkers() async {
    List<Place> places = await DatabaseHelper.instance.getAllPlaces();

    for (Place place in places) {
      NMarker marker = NMarker(
        id: place.id.toString(),
        position: NLatLng(place.latitude, place.longitude),
      );

      if (_controller != null) {
        await _controller!.addOverlay(marker);
      }
    }
  }
}
```

위 코드에서는 네이버 지도를 사용하여 장소를 표시하는 MapView 위젯을 정의하였다. NaverMapController? _controller는 NaverMap 위젯과 상호작용하기 위한 컨트롤러 객체이며, NaverMap은 네이버 지도를 표시하는 위젯이다. onMapReady 콜백에서 컨트롤러를 초기화하고, _loadMarkers 함수를 호출하여 데이터베이스에서 장소 정보를 가져와 마커로 표시한다. 해당 파일은 widgets 폴더 내에 map_view.dart라는 이름으로 만들어져 있으며 수정하여 사용할 수 있다.

마커 아이템 (Widget)

```
import 'package:flutter/material.dart';
import '/models/place.dart';

class PlaceItem extends StatelessWidget {
  final Place place;
  final VoidCallback onDelete;

  PlaceItem({required this.place, required this.onDelete});

  @override
  Widget build(BuildContext context) {
    return ListTile(
      title: Text(place.name),
      subtitle:
          Text('Latitude: ${place.latitude}, Longitude: ${place.longitude}'),
      trailing: IconButton(
        icon: Icon(Icons.delete),
        onPressed: onDelete,
      ),
    );
  }
}
```

위 코드는 장소(Place) 객체를 표시하는 위젯이며, 해당 파일은 widgets 폴더 내에 place_item.dart라는 이름으로 만들어져 있으며 수정하여 사용할 수 있다.

마커 리스트 (Widget)

```
import 'package:flutter/material.dart';
import 'package:helloworld/models/place.dart';
import 'package:helloworld/database/database_helper.dart';
import 'package:helloworld/widgets/place_item.dart';

class PlacesList extends StatefulWidget {
  @override
  _PlacesListState createState() => _PlacesListState();
}
```

```
class _PlacesListState extends State<PlacesList> {
  late Future<List<Place>> _placesList;

  @override
  void initState() {
    super.initState();
    _updatePlacesList();
  }

  void _updatePlacesList() {
    setState(() {
      _placesList = DatabaseHelper.instance.getPlaceList();
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Places List'),
      ),
      body: FutureBuilder<List<Place>>(
        future: _placesList,
        builder: (context, snapshot) {
          if (snapshot.connectionState == ConnectionState.waiting) {
            return Center(child: CircularProgressIndicator());
          } else if (snapshot.hasError) {
            return Center(child: Text('Error: ${snapshot.error}'));
          } else {
            return ListView.builder(
              itemCount: snapshot.data?.length ?? 0,
              itemBuilder: (context, index) {
                Place place = snapshot.data![index];
                return PlaceItem(
                  place: place,
                  onDelete: () {
                    if (place.id != null) {
                      DatabaseHelper.instance.delete(place.id!).then((_) {
                        _updatePlacesList();
                      });
                    }
                  },
```

```
            );
          },
        );
      }
    },
  ),
 );
 }
}
```

이 코드는 데이터베이스에서 장소 목록을 가져와서 리스트 형태로 표시하는 PlacesList 위젯이다. _updatePlacesList() 메소드는 데이터베이스에서 장소 목록을 가져와 _placesList를 업데이트하고, 화면을 다시 그리도록 setState를 호출한다. builder에서는 스냅샷(snapshot) 객체에 따라 대응되는 UI를 반환하며, 연결 상태(connectionState)에 따라 로딩 중, 에러 발생, 또는 데이터가 있는 경우 각각 다른 UI가 표시된다. 해당 파일은 widgets 폴더 내에 place_list.dart라는 이름으로 만들어져 있으며 수정하여 사용할 수 있다.

마커 입력 화면 (Widget)

```
import 'package:flutter/material.dart';
import 'models/place.dart';
import 'database/database_helper.dart';

class PlaceForm extends StatefulWidget {
  final Place? place;

  PlaceForm({this.place});

  @override
  _PlaceFormState createState() => _PlaceFormState();
}

class _PlaceFormState extends State<PlaceForm> {
  final _formKey = GlobalKey<FormState>();
  late TextEditingController _nameController;
  late TextEditingController _latitudeController;
  late TextEditingController _longitudeController;
  late TextEditingController _phoneNumberController;
```

```
@override
void initState() {
  super.initState();
  _nameController = TextEditingController(text: widget.place?.name ?? '');
  _latitudeController =
      TextEditingController(text: widget.place?.latitude.toString() ?? '');
  _longitudeController =
      TextEditingController(text: widget.place?.longitude.toString() ?? '');
  _phoneNumberController =
      TextEditingController(text: widget.place?.phoneNumber ?? '');
}

@override
void dispose() {
  _nameController.dispose();
  _latitudeController.dispose();
  _longitudeController.dispose();
  _phoneNumberController.dispose();
  super.dispose();
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text(widget.place == null ? 'Add Place' : 'Edit Place'),
      actions: [
        IconButton(
          icon: Icon(Icons.save),
          onPressed: _savePlace,
        ),
      ],
    ),
    body: Form(
      key: _formKey,
      child: SingleChildScrollView(
        padding: EdgeInsets.all(16.0),
        child: Column(
          children: [
            TextFormField(
              controller: _nameController,
              decoration: InputDecoration(labelText: 'Name'),
```

```
              validator: (value) {
                if (value == null || value.isEmpty) {
                  return 'Please enter a name';
                }
                return null;
              },
            ),
            TextFormField(
              controller: _latitudeController,
              decoration: InputDecoration(labelText: 'Latitude'),
              keyboardType: TextInputType.number,
              validator: (value) {
                if (value == null || value.isEmpty) {
                  return 'Please enter latitude';
                }
                return null;
              },
            ),
            TextFormField(
              controller: _longitudeController,
              decoration: InputDecoration(labelText: 'Longitude'),
              keyboardType: TextInputType.number,
              validator: (value) {
                if (value == null || value.isEmpty) {
                  return 'Please enter longitude';
                }
                return null;
              },
            ),
            TextFormField(
              controller: _phoneNumberController,
              decoration: InputDecoration(labelText: 'Phone Number'),
              keyboardType: TextInputType.phone,
            ),
          ],
        ),
      ),
    ),
  );
}

void _savePlace() async {
  if (!_formKey.currentState!.validate()) {
```

```
      return;
    }

    int maxId = await DatabaseHelper.instance.getMaxPlaceId();
    int newId = maxId + 1;

    Place place = Place(
      id: newId,
      name: _nameController.text,
      latitude: double.parse(_latitudeController.text),
      longitude: double.parse(_longitudeController.text),
      phoneNumber: _phoneNumberController.text,
    );

    if (widget.place == null) {
      await DatabaseHelper.instance.insert(place);
    } else {
      await DatabaseHelper.instance.update(place);
    }

    Navigator.pop(context);
  }
}
```

위 코드는 장소 정보를 입력받는 폼을 표시하고, 입력된 정보를 데이터베이스에 저장하는 PlaceForm 위젯이다. dispose() 메소드는 위젯이 제거될 때 호출되며, TextController들을 dispose하여 메모리 누수를 방지한다. Form에서는 폼을 생성하고 유효성 검사를 수행하기 위해 GlobalKey와 함께 사용되며, SingleChildScrollView을 통해 컬럼 내용이 화면 크기보다 클 경우 스크롤 가능한 영역으로 만든다. 해당 파일은 widgets 폴더 내에 place_form.dart라는 이름으로 만들어져 있으며 수정하여 사용할 수 있다.

02 회원가입 & 로그인

✓핵심 키워드

Firebase, Firestore, 이메일 인증, 네이버 로그인, 카카오 로그인

여기서는 무얼 배울까

Firebase를 이용한 회원 서비스를 만든다. 이메일 인증을 통한 회원가입 서비스를 만들고 네이버 계정으로 로그인, 카카오 계정으로 로그인하는 방법을 알아보자. 이를 활용하여 회원 서비스에서 회원가입과 네이버, 카카오 계정으로 로그인할 수 있다.

프로젝트 설정 및 환경 구성

Firebase 프로젝트 설정

- Firebase 콘솔(https://console.firebase.google.com)에 접속하여 프로젝트를 생성하고, 앱을 추가한다.
- Firebase 프로젝트에 앱을 추가한다. Flutter 앱을 추가하려면, Firebase 아이콘을 선택한다.
- Firebase CLI를 설치하고 로그인을 한다.

```
firebase login
```

프로젝트 생성 및 설정

- 터미널 또는 명령 프롬프트에서 원하는 디렉토리로 이동한다.
- 다음 명령어를 실행하여 새로운 Flutter 프로젝트를 생성한다.

```
flutter create my_app
```

- 생성한 프로젝트 폴더로 이동한다.

```
cd my_social_app
```

- 디렉토리에서 다음 명령어를 실행한다.

```
dart pub global activate flutterfire_cli
```

- 다음 명령어를 사용하여 path 설정을 해준다.

```
export PATH= $PATH $HOME/.pub-cache/bin
```

- 다음 Flutter 프로젝트 디렉토리의 루트에서 다음 명령어를 실행한다. Firebase 페이지에서 자세한 프로젝트명을 가져와서 사용한다.

```
flutterfire configure --project=[프로젝트명]
```

Firebase 초기화 및 플러그인 추가

```
import 'package:firebase_core/firebase_core.dart';
import 'firebase_options.dart';

// ...

await Firebase.initializeApp(
   options: DefaultFirebaseOptions.currentPlatform,
);
```

Android 앱 종속성 추가

- android/app/build.gradle 파일을 열어 다음과 같이 수정한다.

```
// 상단에 추가
plugins {
   id "com.android.application"
   apply plugin: 'com.google.firebase.crashlytics'
   id "kotlin-android"
```

```
    id "dev.flutter.flutter-gradle-plugin"
}

apply plugin: 'com.google.gms.google-services'

// dependencies 블록 안에 추가
implementation platform('com.google.firebase:firebase-bom:28.4.1')
implementation 'com.google.firebase:firebase-analytics'
implementation 'com.google.firebase:firebase-auth'

// 맨 하단에 추가
googleServices { jsonFileEnabled true }
```

iOS 앱 종속성 추가

- ios/Podfile 파일을 열어 다음과 같이 수정한다.

```
# 상단에 추가
platform :ios, '10.0'

target 'Runner' do

  # Flutter 플러그인들은 이미 존재하는 코드 위로 올라갑니다.
  # Firebase/Core 모듈 추가 (필수)
  pod 'Firebase/Core'

  # Firebase/Auth 모듈 추가 (Firebase 인증 사용 시)
  pod 'Firebase/Auth'

end
```

의존성 추가

```
dependencies:
  flutter:
    sdk: flutter
  firebase_core: ^X.X.X
  firebase_auth: ^X.X.X
```

- 터미널 또는 명령 프롬프트에서 프로젝트 폴더로 이동한 상태에서 다음 명령어를 실행하여 필요한 패키지를 설치한다.

```
flutter pub get
```

이제 프로젝트 설정이 완료되었다. 다음 단계로 앱의 구조와 화면을 설계하고 기능을 구현해 나갈 수 있다.

메인 화면 (main.dart)

```
import 'package:flutter/material.dart';
import 'package:firebase_core/firebase_core.dart';
import 'register_page.dart';
import 'firebase_options.dart';


void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform,
  );
  runApp(App());
}

class App extends StatelessWidget {
  final Future<FirebaseApp> _initialization = Firebase.initializeApp();

  @override
  Widget build(BuildContext context) {
    return FutureBuilder(
      future: _initialization,
      builder: (context, snapshot) {
        if (snapshot.connectionState == ConnectionState.done) {
          return MaterialApp(
            title: 'Firebase Auth Demo',
            home: RegisterPage(),
          );
        }
        if (snapshot.hasError) {
          return Text('Error initializing Firebase');
```

```
        }
        return CircularProgressIndicator();
      },
    );
  }
}
```

WidgetsFlutterBinding.ensureInitialized()은 Flutter 앱이 실행되기 전에 필요한 초기화 작업을 수행한다. Firebase.initializeApp()은 Firebase를 초기화하는 비동기 함수이며, DefaultFirebaseOptions.currentPlatform 옵션을 사용하여 플랫폼에 맞는 Firebase 설정으로 초기화한다. _initialization 변수는 Firebase 초기화 작업의 비동기 Future 객체를 저장한다.

회원가입 페이지 (register_page.dart)

```
import 'package:flutter/material.dart';
import 'package:firebase_auth/firebase_auth.dart';

class RegisterPage extends StatefulWidget {
  @override
  _RegisterPageState createState() => _RegisterPageState();
}

class _RegisterPageState extends State<RegisterPage> {
  final FirebaseAuth _auth = FirebaseAuth.instance;
  final TextEditingController emailController = TextEditingController();
  final TextEditingController passwordController = TextEditingController();
  bool isLoading = false;

  void registerWithEmailPassword(String email, String password) async {
    setState(() {
      isLoading = true;
    });

    try {
      if (email.isEmpty || password.isEmpty) {
        throw ('이메일과 비밀번호를 입력해주세요.');
      }
```

```
      UserCredential userCredential =
      await _auth.createUserWithEmailAndPassword(
        email: email,
        password: password,
      );

      showDialog(
        context: context,
        builder: (context) => AlertDialog(
          title: Text('회원가입 성공'),
          content: Text('회원가입이 성공했습니다.'),
          actions: [
            TextButton(
              onPressed: () => Navigator.pop(context),
              child: Text('확인'),
            ),
          ],
        ),
      );
    } catch (e) {
      showDialog(
        context: context,
        builder: (context) => AlertDialog(
          title: Text('회원가입 실패'),
          content: Text(e.toString()),
          actions: [
            TextButton(
              onPressed: () => Navigator.pop(context),
              child: Text('확인'),
            ),
          ],
        ),
      );
    } finally {
      setState(() {
        isLoading = false;
      });
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
```

```
        appBar: AppBar(title: Text('Register')),
        body: Stack(children: <Widget>[
          Padding(
              padding: const EdgeInsets.all(16),
              child: Column(children: <Widget>[
                TextField(
                    decoration: InputDecoration(hintText: 'Email'),
                    controller: emailController),
                TextField(
                    decoration: InputDecoration(hintText: 'Password'),
                    controller: passwordController,
                    obscureText: true),
                ElevatedButton(
                    onPressed: !isLoading
                        ? () {
                      registerWithEmailPassword(
                          emailController.text, passwordController.text);
                    }
                        : null,
                    child: Text('Register'))
              ])),
          if (isLoading) Center(child: CircularProgressIndicator())
        ]));
  }
}
```

auth 변수는 FirebaseAuth 인스턴스를 저장하는데 사용된다. Firebase 인증 기능을 사용하기 위해 필요하다. emailController, passwordController 변수는 TextField 위젯과 연결되어 입력된 이메일과 비밀번호 값을 가져올 수 있도록 한다. 이메일과 비밀번호를 매개변수로 받아 Firebase에 회원가입 요청을 보낸다. 함수 내부에서는 입력값이 유효한지 확인하고, _auth.createUserWithEmailAndPassword() 메서드를 사용하여 실제 회원가입 작업을 수행한다. 성공적으로 회원가입이 완료되면 AlertDialog로 성공 메시지 창이 표시된다. 실패할 경우 AlertDialog로 실패 메시지와 에러 정보가 표시된다.

카카오 로그인

카카오 SDK

- 카카오 개발자 사이트(https://developers.kakao.com/)에 접속하여 새로운 앱을 등록한다.
- 새로 등록된 앱에서 플랫폼 메뉴에 들어간다.

안드로이드 플랫폼 등록

- 패키지명에 안드로이드 앱의 패키지명을 적는다. 프로젝트 폴더의 andriod/app/src/main/AndroidManifest.xml 파일을 열고 파일 최상단 태그의 package에 해당하는 값을 입력하면 된다. 형식은 com.example.my_app으로 작성되어 있다.
- 마켓 URL은 공백으로 설정한다.
- 키 해시를 등록하기 위해서는 OpenSSL을 설치한다. 환경 변수 Path에 OpenSSL의 bin 폴더의 경로를 추가해 주고 다음과 같은 명령어를 통해 debug key hash를 추출할 수 있다.

```
keytool -exportcert -alias androiddebugkey -keystore {path} -storepass
android -keypass android | openssl sha1 -binary | openssl base64
```

- {path} 부분에는 debug.keystore가 있는 경로를 입력해 주면 된다.

iOS 플랫폼 등록

- 번들 ID만 입력하면 iOS 플랫폼 등록은 완료된다. 번들 ID는 ios/Runner.xcodeproj/project.pbxproj 파일에 있는 PRODUCT_BUNDLE_IDENTIFIER 속성에 작성되어 있다.

카카오톡 로그인 활성화

- 플랫폼 등록이 완료되었으면 카카오 로그인 메뉴에 들어간다.
- OFF로 설정되어 있는 활성화 설정을 ON으로 수정하여 카카오톡 로그인 API를 사용할 수 있다.
- 동의항목 메뉴에서 개인정보, 접근 권한에서 자신이 사용하고자 하는 설정을 추가하여 활용할 수 있다.

의존성 추가

```
dependencies:
  flutter:
    sdk: flutter
  kakao_flutter_sdk: ^X.X.X
```

- 터미널 또는 명령 프롬프트에서 프로젝트 폴더로 이동한 상태에서 다음 명령어를 실행하여 필요한 패키지를 설치한다.

```
flutter pub get
```

API 키 설정

- Main 함수 runApp() 메서드 호출 전에 Flutter SDK를 다음과 같이 초기화한다.

```
void main() {
  ...
  KakaoSdk.init(nativeAppKey: '${YOUR_NATIVE_APP_KEY}');
  runApp(MyApp());
  ...
}
```

- '${YOUR_NATIVE_APP_KEY} 부분은 내 어플리케이션 요약 정보에서 네이티브 앱 키 부분으로 수정하여 작성한다.
- 카카오 로그인 기능을 구현하기 위해서는 리다이렉션(Redirection)을 통해 인가 코드를 받아야 한다. 안드로이드의 경우 android/app/src/AndroidManifest.xml에 액티비티(Activity) 설정을 다음과 같이 추가한다.

```
<activity
    android:name="com.kakao.sdk.auth.AuthCodeHandlerActivity"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <category android:name="android.intent.category.BROWSABLE" />

        <data android:host="oauth"
```

```
            android:scheme="kakao${NATIVE_APP_KEY}" />
    </intent-filter>
</activity>
```

- iOS의 경우 ios/Runner/AppDelegate.swift 파일 내부에 다음 코드 스니펫을 삽입하여 Kakao SDK 초기화 코드를 작성한다.

```
import KakaoSDKCommon

func application(_ application: UIApplication, didFinishLaunchingWithOptions
launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
    ...
    KakaoSDKCommon.initSDK(appKey: "YOUR_NATIVE_APP_KEY")
    ...
}
```

로그인 구현 예제

```
// 카카오 로그인 구현 예제

// 카카오톡 실행 가능 여부 확인
// 카카오톡 실행이 가능하면 카카오톡으로 로그인, 아니면 카카오계정으로 로그인
if (await isKakaoTalkInstalled()) {
  try {
      await UserApi.instance.loginWithKakaoTalk();
      print('카카오톡으로 로그인 성공');
  } catch (error) {
    print('카카오톡으로 로그인 실패 $error');

    // 사용자가 카카오톡 설치 후 디바이스 권한 요청 화면에서 로그인을 취소한 경우,
    // 의도적인 로그인 취소로 보고 카카오계정으로 로그인 시도 없이 로그인 취소로 처리
    (예: 뒤로 가기)
    if (error is PlatformException && error.code == 'CANCELED') {
        return;
    }
    // 카카오톡에 연결된 카카오계정이 없는 경우, 카카오계정으로 로그인
    try {
        await UserApi.instance.loginWithKakaoAccount();
        print('카카오계정으로 로그인 성공');
    } catch (error) {
```

```
        print('카카오계정으로 로그인 실패 $error');
      }
    }
  } else {
    try {
      await UserApi.instance.loginWithKakaoAccount();
      print('카카오계정으로 로그인 성공');
    } catch (error) {
      print('카카오계정으로 로그인 실패 $error');
    }
  }
```

추가 항목 동의 받기

```
User user;
  try {
    user = await UserApi.instance.me();
  } catch (error) {
    print('사용자 정보 요청 실패 $error');
    return;
  }

  // 사용자의 추가 동의가 필요한 사용자 정보 동의 항목 확인
  List<String> scopes = [];

  if (user.kakaoAccount?.emailNeedsAgreement == true) {
    scopes.add('account_email');
  }
  if (user.kakaoAccount?.birthdayNeedsAgreement == true) {
    scopes.add("birthday");
  }
  if (user.kakaoAccount?.birthyearNeedsAgreement == true) {
    scopes.add("birthyear");
  }
  if (user.kakaoAccount?.ciNeedsAgreement == true) {
    scopes.add("account_ci");
  }
  if (user.kakaoAccount?.phoneNumberNeedsAgreement == true) {
    scopes.add("phone_number");
  }
  if (user.kakaoAccount?.profileNeedsAgreement == true) {
```

```
    scopes.add("profile");
  }
  if (user.kakaoAccount?.ageRangeNeedsAgreement == true) {
    scopes.add("age_range");
  }

  if (scopes.length > 0) {
    print('사용자에게 추가 동의 받아야 하는 항목이 있습니다');

    // OpenID Connect 사용 시
    // scope 목록에 "openid" 문자열을 추가하고 요청해야 함
    // 해당 문자열을 포함하지 않은 경우, ID 토큰이 재발급되지 않음
    // scopes.add("openid")

    // scope 목록을 전달하여 추가 항목 동의 받기 요청
    // 지정된 동의 항목에 대한 동의 화면을 거쳐 다시 카카오 로그인 수행
    OAuthToken token;
try {
     token = await UserApi.instance.loginWithNewScopes(scopes);
     print('현재 사용자가 동의한 동의 항목: ${token.scopes}');
   } catch (error) {
     print('추가 동의 요청 실패 $error');
     return;
   }

   // 사용자 정보 재요청
   try {
     User user = await UserApi.instance.me();
     print('사용자 정보 요청 성공'
         '\n회원번호: ${user.id}'
         '\n닉네임: ${user.kakaoAccount?.profile?.nickname}'
         '\n이메일: ${user.kakaoAccount?.email}');
   } catch (error) {
     print('사용자 정보 요청 실패 $error');
   }
  }
```

OpenID Connect를 사용하는 앱의 경우, 추가 항목 동의 받기 요청 시 동의 항목 키를 전달하는 scope 파라미터 값에 openid를 반드시 포함해야 한다. 해당 파라미터 값을 포함하지 않을 경우, ID 토큰이 재발급되지 않는다.

네이버 로그인

카카오 SDK

- 네이버 개발자 사이트(https://developers.naver.com/)에 접속하여 새로운 애플리케이션을 등록한다. 사용 API에서 네이버 로그인 API를 설정하고, 앱 패키지 이름을 작성한다. 다운로드 URL은 임의로 작성해 주고, URL Scheme은 자신이 사용할 임의의 값을 넣어 준다.
- 등록한 애플리케이션의 설정 페이지에서 "애플리케이션 정보" 섹션에서 클라이언트 ID와 클라이언트 시크릿 값을 확인한다.

의존성 추가

```
dependencies:
  flutter:
    sdk: flutter
  Flutter_naver_login: ^X.X.X
```

- 터미널 또는 명령 프롬프트에서 프로젝트 폴더로 이동한 상태에서 다음 명령어를 실행하여 필요한 패키지를 설치한다.

```
flutter pub get
```

안드로이드 설정

- android/app/src/main/res/values/strings.xml 파일에 클라이언트 ID 값을 다음과 같이 설정한다.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="client_id">[client_id]</string>
    <string name="client_secret">[client_secret]</string>
    <string name="client_name">[client_name]</string>
</resources>
```

- android/app/src/main/AndroidManifest.xml 파일에 다음 설정들을 추가한다.

```
<application
       android:name="io.flutter.app.FlutterApplication"
       android:label="flutter_naver_login_example"
       android:icon="@mipmap/ic_launcher">
       <meta-data
          android:name="com.naver.sdk.clientId"
          android:value="@string/client_id" />
       <meta-data
          android:name="com.naver.sdk.clientSecret"
          android:value="@string/client_secret" />
        <meta-data
          android:name="com.naver.sdk.clientName"
          android:value="@string/client_name" />
                        ...
```

iOS 설정

- ios/Runner/Info.plist 파일에 다음 설정들을 추가한다.

```
<key>CFBundleURLTypes</key>
<array>
   <dict>
      <key>CFBundleTypeRole</key>
      <string>Editor</string>
      <key>CFBundleURLName</key>
      <string>naver</string>
      <key>CFBundleURLSchemes</key>
      <array>
         <string>[CALLBACK_URL_SCHEME]</string>
      </array>
   </dict>
</array>

<key>NaverClientID</key>
<string>[CLIENT_ID]</string>

<key>NaverCallbackUrlScheme</key>
<string>[CALLBACK_URL_SCHEME]</string>
```

```
<key>NaverServiceAppName</key>
<string>[YOUR_APP_NAME]</string>
```

- ios/Runner/AppDelegate.swift 파일을 열고 최하위에 다음 코드를 추가한다.

```
import NaverThirdPartyLogin

extension AppDelegate {
    func handleNaverLoginURL(_ url: URL, options: [UIApplication.OpenURLOptionsKey:
    Any]) -> Bool {
        if let naverLoginConnection = NaverThirdPartyLoginConnection.
        getSharedInstance() {
            if naverLoginConnection.application(UIApplication.shared, open:
            url, options: options) {
                return true
            }
        }

        return false
    }
}

// AppDelegate 클래스 내부에서 application(_:open:options:) 메서드 오버라이드
override func application(_ app: UIApplication, open url: URL, options:
[UIApplication.OpenURLOptionsKey : Any] = [:]) -> Bool {
    if handleNaverLoginURL(url, options: options) {
        return true
    }

    return super.application(app, open: url, options: options)
}
```

로그인 구현 예제

```
import 'package:flutter/material.dart';
import 'package:flutter_naver_login/flutter_naver_login.dart';

class MyApp extends StatelessWidget {
  @override
```

```
  Widget build(BuildContext context) {
    return MaterialApp(
      home: LoginPage(),
    );
  }
}

class LoginPage extends StatelessWidget {
  Future<void> handleLogin() async {
    NaverLoginResult result = await FlutterNaverLogin.logIn();
    if (result.status == NaverLoginStatus.loggedIn) {
      String id = result.account.id;
      String email = result.account.email;

      // 네이버 로그인 성공 후 필요한 처리 수행
      // ...

      // 예시: 다음 화면으로 이동
      Navigator.push(
        context,
        MaterialPageRoute(builder: (context) => HomePage()),
      );
    } else {
      // 네이버 로그인 실패 혹은 취소된 경우
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('로그인')),
      body: Center(
        child: RaisedButton(
          onPressed: handleLogin,
          child: Text('네이버로 로그인'),
        ),
      ),
    );
  }
}

class HomePage extends StatelessWidget {
  Future<void> handleLogout() async {
```

```
    await FlutterNaverLogin.logOut();

    // 로그아웃 후 필요한 처리 수행
    // ...

    // 예시: 이전 화면으로 돌아가기
    Navigator.pop(context);
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('홈')),
      body: Center(
        child: RaisedButton(
          onPressed: handleLogout,
          child: Text('로그아웃'),
        ),
      ),
    );
  }
}
```