Mobile Development :

10 : Integration of External Services : Part 1

Working with a Case Study

Professor Imed Bouchrika

National School of Artificial Intelligence
imed.bouchrika@ensia.edu.dz

Outline :

- **Revision :**
  - ***Service Location Pattern : GetIt***
  - ***Database Repository : Step by Step***
- **Architecture & Technologies**
  - ***Databases & Storage Technologies***
    - ***Firestore for NoSQL data***
    - ***Supabase***
    - ***Firebase Cloud Store***
  - ***Backends Vs Direct Access : Rest API, GraphQL***
    - ***Supabase***
  - ***Background Jobs***
- **Case study : MedBox Digital App.**
  - ***Design & Architecture***
  - ***Getting the list of Doctors from : Supabase, Backends, Firestore..***

Section 1

Revision

Service Locator Pattern :

Design Pattern : Service Locator

- **Service Locator Pattern**
  - is a design pattern used to encapsulate the process of obtaining services. It provides a centralized registry or locator where consumers (clients) can query and retrieve the services they need, often by type or name.

main.dart

```dart
Future<bool> init_my_app() async {
 if (Platform.isLinux || Platform.isWindows) {
   sqfliteFfiInit();
   databaseFactory = databaseFactoryFfi;
 }

 final sl = GetIt.instance;
 sl.registerLazySingleton<ProfilesRepo>(() => ProfilesRepo());
 WidgetsFlutterBinding.ensureInitialized();
 return true;
}

void main() async {
 await init_my_app();
 runApp(const MainApp());
}

class MainApp extends StatelessWidget {
 const MainApp({super.key});

 @override
 Widget build(BuildContext context) {
   return MultiBlocProvider(
     providers: [BlocProvider(create: (_) => ProfilesCubit())],
     child: const MaterialApp(home: ProfileScreen()),
   );
 }
}
```

Initialize the Service Locator to store objects and later can be accessed by type in a singleton fashion

profile_cubit.dart

```dart
class ProfilesCubit extends Cubit<Map<String, dynamic>> {
 late final ProfilesRepo profileRepo;
 ProfilesCubit() : super({'data': [], 'state': 'loading', 'message': ''}) {
    profileRepo = GetIt.I<ProfilesRepo>();
   load();
 }
 Future<bool> load() async {
   emit({...state, 'state': 'loading', 'message': '', 'data': []});
   try {
     final records = await profileRepo.getData();
     print('Load Data..... ${records.toString()}');
     emit({...state, 'data': records, 'state': 'done', 'message': ''});
   } catch (e) {emit({...state, 'state': 'error', 'message': e.toString(), 'data': []});}
   return true;
 }
 Future<ReturnResult> insertItem(Map<String, dynamic> record) async {
   var result = ReturnResult(state: false, message: 'There is an error');
   try {
     result = await profileRepo.insertItem(record);
     await load();
   } catch (e) {emit({...state, 'error': e.toString(), 'data':
   return result;
 }

 Future<ReturnResult> updateItem(Map<String, dynamic> record)
   var result = ReturnResult(state: false, message: 'There is a
   try {
     result = await profileRepo.updateItem(record);
     await load();
   } catch (e) {emit({...state, 'error': e.toString(), 'data': []});}
   return result;
 }
 Future<ReturnResult> deleteItem(int id) async {
   var result = ReturnResult(state: false, message: 'There is an error');
   try {
```

Will get an instance of the ProfileRepo from the service location

Database Integration /Repo

```dart
import 'dart:async';
import 'package:flutter/material.dart';
import 'package:path/path.dart';
import 'package:sqflite/sqflite.dart';

class DBHelper {
 static const  database_name = "ENSIA_MY_DB.db";
 static const  database_version = 4;
 static var database;

 static Future getDatabase() async {
   if (database != null) {
     return database;
   }
   database = openDatabase(
     join(await getDatabasesPath(), _database_name),
     onCreate: (database, version) {
       database.execute('''
         CREATE TABLE  todo (
             id INTEGER PRIMARY KEY AUTOINCREMENT,
             title TEXT,
             done INTEGER,
             duedate TEXT,
             create_date TEXT)
       ''');
     },
     version:  database_version,
     onUpgrade: (db, oldVersion, newVersion) {      },
   );
   return database;
 }
}
```

01

```
class _HomeScreenState extends State<HomeScreen> {
 late Future<List<Map>> data;

 String _tx_title_value = '';
 final _tx_title_controller = TextEditingController();

 Future<List> getAllToDos() async{
   var database = await DBHelper.getDatabase();
   return database.rawQuery('''SELECT
          todo.id ,
          todo.title,
          todo.done
        FROM todo
        ''');
 }

 @override
 Widget build(BuildContext context) {
   data = getAllToDos();
   return Scaffold(
```

02

Homescreen Dart for the UI...

```
class _HomeScreenState extends State<HomeScreen> {
 late Future<List<Map>> data;

 String _tx_title_value = '';
 final _tx_title_controller = TextEditingController();

 Future<List> getAllToDos() async{
   var database = await DBHelper.getDatabase();
   return database.rawQuery('''SELECT
          todo.id ,
          todo.title,
          todo.done
        FROM todo
        ''');
 }

 @override
 Widget build(BuildContext context) {
   data = getAllToDos();
   return Scaffold(
```

02

Mixing :
UI + SQL + Specific Technology + Business Logic + Specific Data Structure inside a file....

All other files ? must be the same

```
class _HomeScreenState extends State<HomeScreen> {
 late Future<List<Map>> data;

 String _tx_title_value = '';
 final _tx_title_controller = TextEditingController();

 Future<List<Map>> getAllToDos() async{
   var database = await DBHelper.getDatabase();
   return database.rawQuery('''SELECT
          todo.id ,
          todo.title,
          todo.done
        FROM todo
        ''');
 }

 @override
 Widget build(BuildContext context) {
   data = getAllToDos();
   return Scaffold(
```

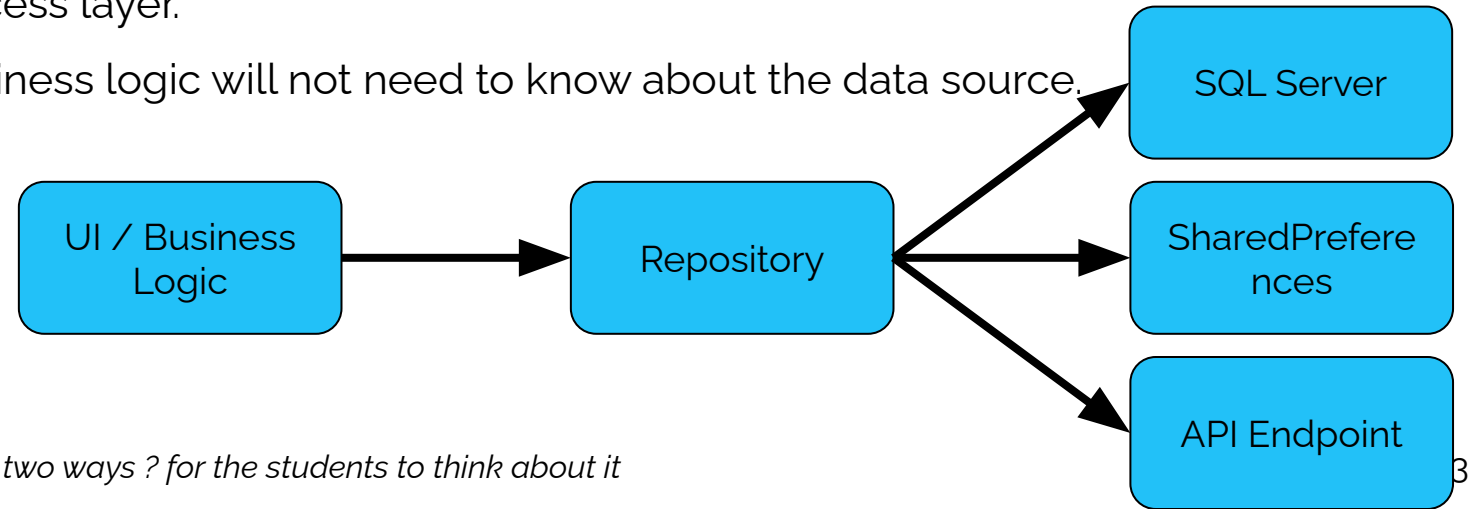
02

Changing the database table ?
All files ...

Changing to an External database ?...

Revision : Repository Design Pattern

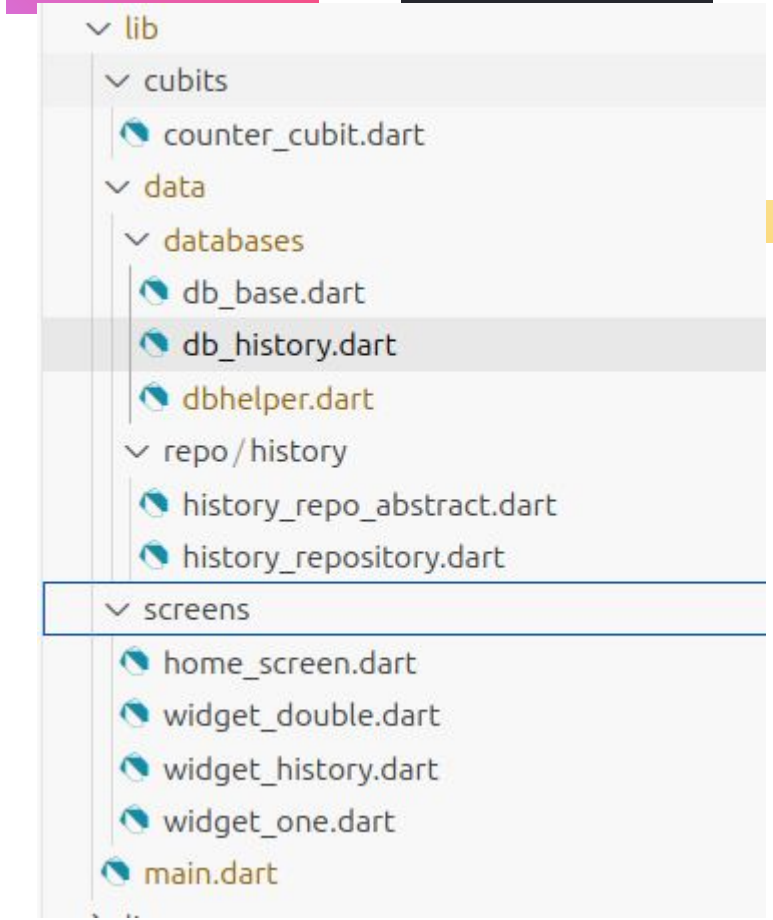
- **Repository Pattern**
  - A technique that acts as an abstraction layer between the business logic and the data access layer.
  - The business logic will not need to know about the data source.



One way communication or two ways ? for the students to think about it

main.dart

```dart
Future<bool> init my app() async {
 final historyRepo = HistoryRepositoryBase.getInstance();
 return true;
}

void main() async {
 if (Platform.isLinux || Platform.isWindows) {
   sqfliteFfiInit();
   databaseFactory = databaseFactoryFfi;
 }

 WidgetsFlutterBinding.ensureInitialized();

 await init_my_app();

 runApp(MainApp());
}

class MainApp extends StatelessWidget {
 MainApp({super.key});

 @override
 Widget build(BuildContext context) {
   return MultiBlocProvider(
     providers: [BlocProvider(create: (_) => CounterCubit())],
     child: MaterialApp(home: HomeScreen()),
   );
 }
}
```



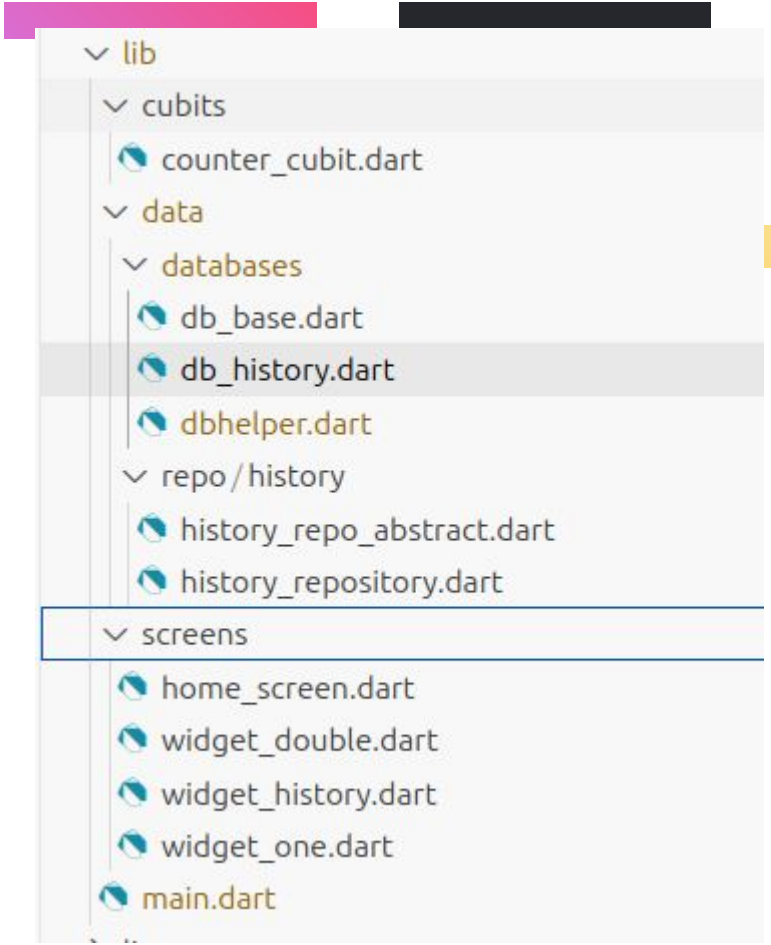
main.dart

```dart
Future<bool> init my app() async {
 final historyRepo = HistoryRepositoryBase.getInstance();

 final sl = GetIt.instance;
 sl.registerLazySingleton<HistoryRepositoryBase>(() => historyRepo  );
 return true;
}

void main() async {
 if (Platform.isLinux || Platform.isWindows) {
   sqfliteFfiInit();
   databaseFactory = databaseFactoryFfi;
 }

 WidgetsFlutterBinding.ensureInitialized();

 await init_my_app();

 runApp(MainApp());
}

class MainApp extends StatelessWidget {
 MainApp({super.key});

 @override
 Widget build(BuildContext context) {
   return MultiBlocProvider(
     providers: [BlocProvider(create: (_) => CounterCubit())],
     child: MaterialApp(home: HomeScreen()),
```

Or Just use the Service Locator Pattern

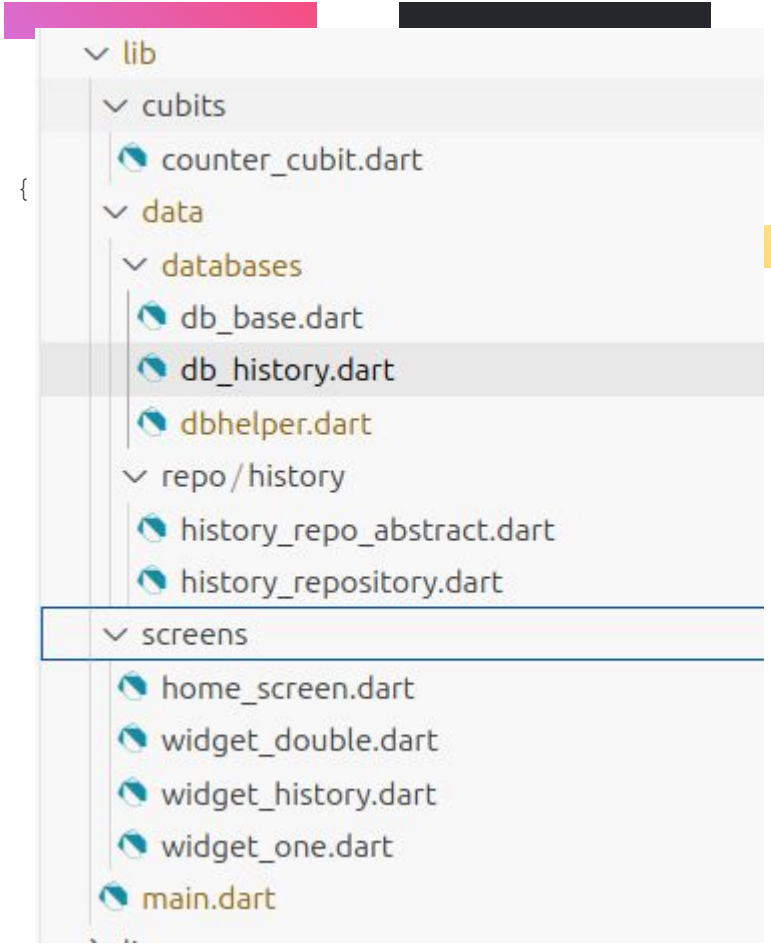
dbhelper.dart

```dart
class DBHelper {
 static const _database_name = "IncrementHistoryV1.db";
 static const  database_version = 1;
 static var database;

 static List<String> sql_codes = [DBHistoryTable.sql_code];
 static Future<Database> getDatabase() async {

   if (database != null) {
     return database;
   }

   database = openDatabase(
     join(await getDatabasesPath(), _database_name),
     onCreate: (database, version) {
       sql_codes.forEach((item) {
         database.execute(item);
       });
     },
     version:  database version,
     onUpgrade: (db, oldVersion, newVersion) {
       //do nothing...
     },
   );
   return database;
 }
}
```



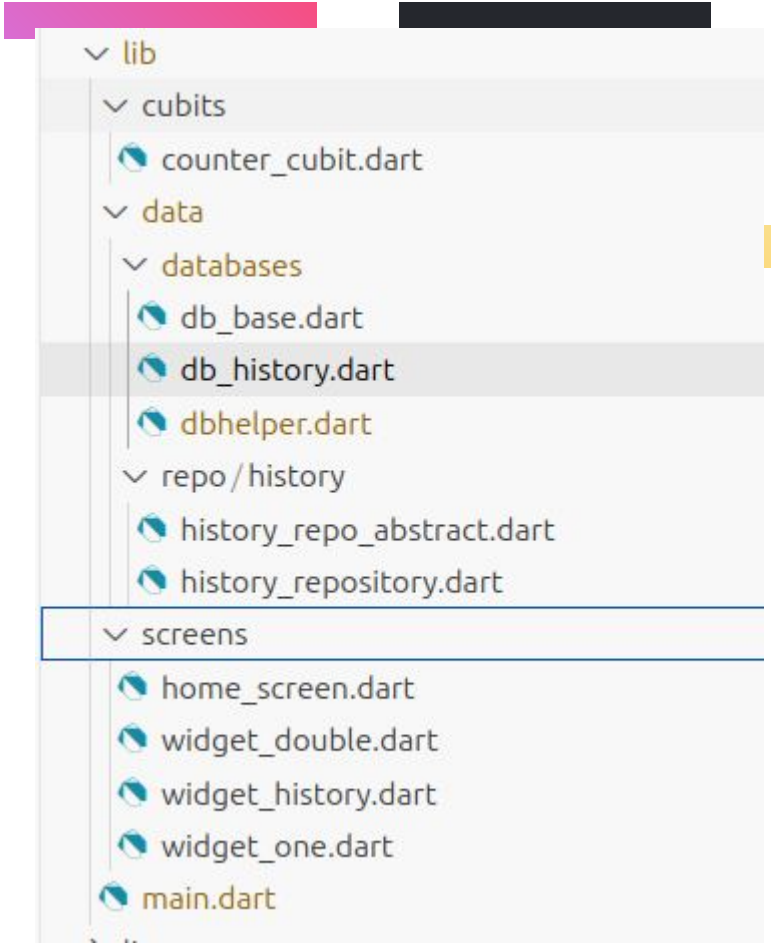
db_base.dart

```dart
class DBBaseTable {
 var db_table = 'TABLE_NAME_MUST_OVERRIDE';

 Future<bool> insertRecord(Map<String, dynamic> data) async {
   try {
     final database = await DBHelper.getDatabase();
     database.insert(
       db_table,
       data,
       conflictAlgorithm: ConflictAlgorithm.replace,
     );
     return true;
   } catch (e, stacktrace) { print('$e --> $stacktrace');}
   return false;
 }
 Future<List<Map>> getRecords() async {
   try {
     final database = await DBHelper.getDatabase();
     var data = await database.rawQuery(
       "select * from $db_table order by id DESC",
     );
     return data;
   } catch (e, stacktrace) {print('$e --> $stacktrace');}
   return [];
 }
}
```



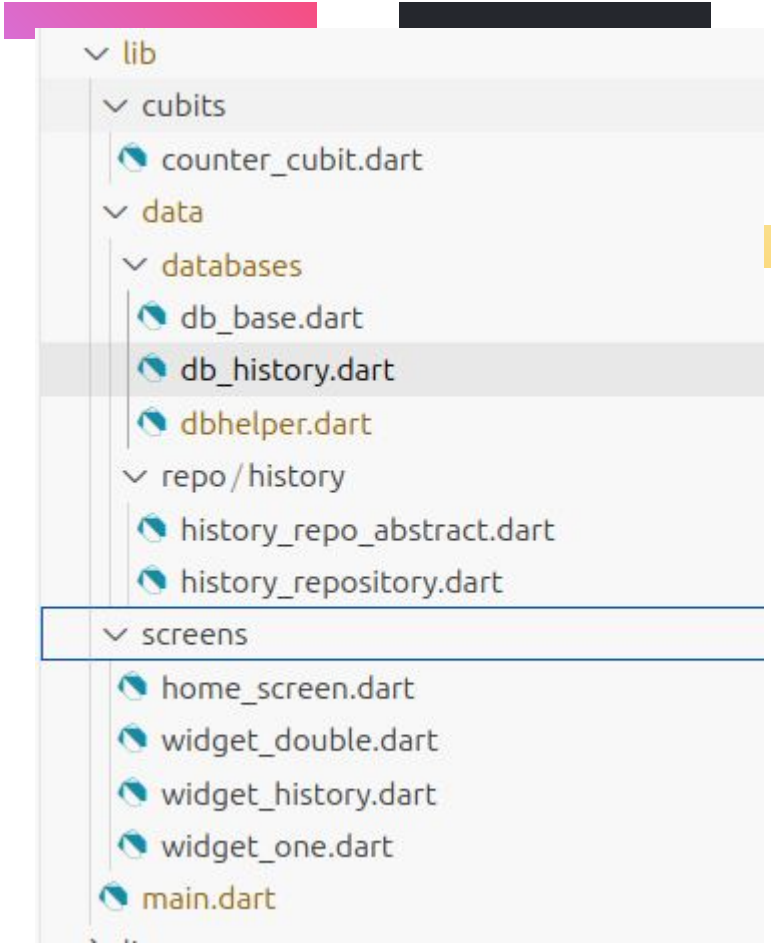
db_history.dart

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

import 'db_base.dart';

class DBHistoryTable extends DBBaseTable {
 var db_table = 'history';
 static String sql_code = '''
         CREATE TABLE  history (
             id INTEGER PRIMARY KEY AUTOINCREMENT,
             message TEXT,
             create_date TEXT
           )
       ''';
}
```



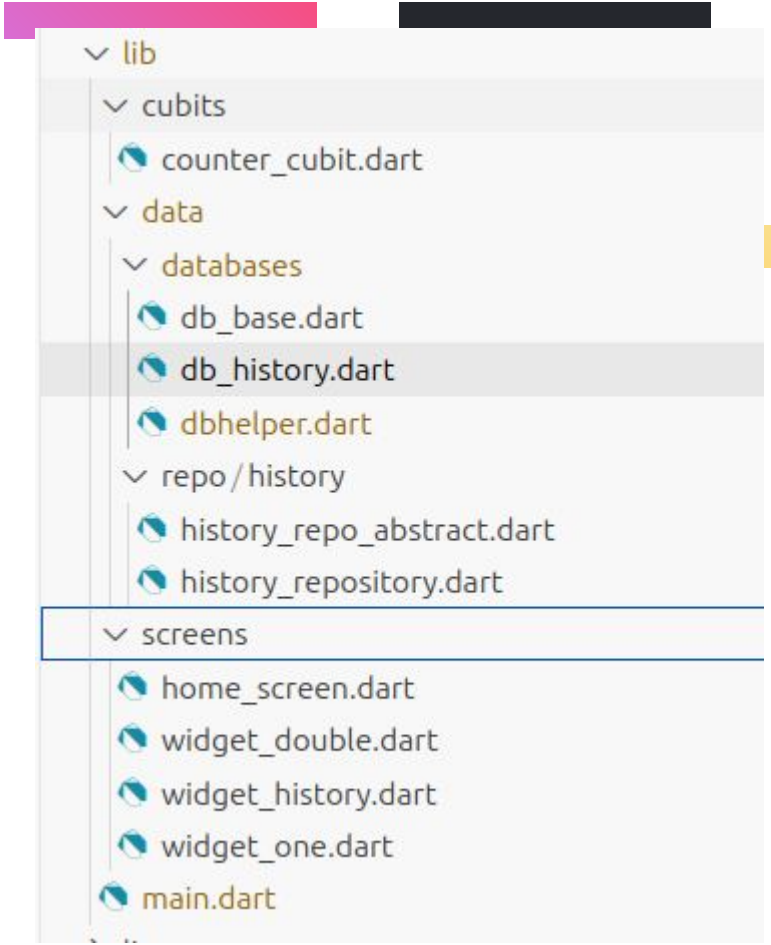
history_repo_abstract.dart

```dart
import 'history_repository.dart';

abstract class HistoryRepositoryBase {
 Future<List<String>> getData();
 Future<bool> insertData(String value);

 static HistoryRepositoryBase? _historyInstance;

 static HistoryRepositoryBase getInstance() {
   _historyInstance ??= HistoryRepository();
   return _historyInstance!; // For backend data
 }
}
```



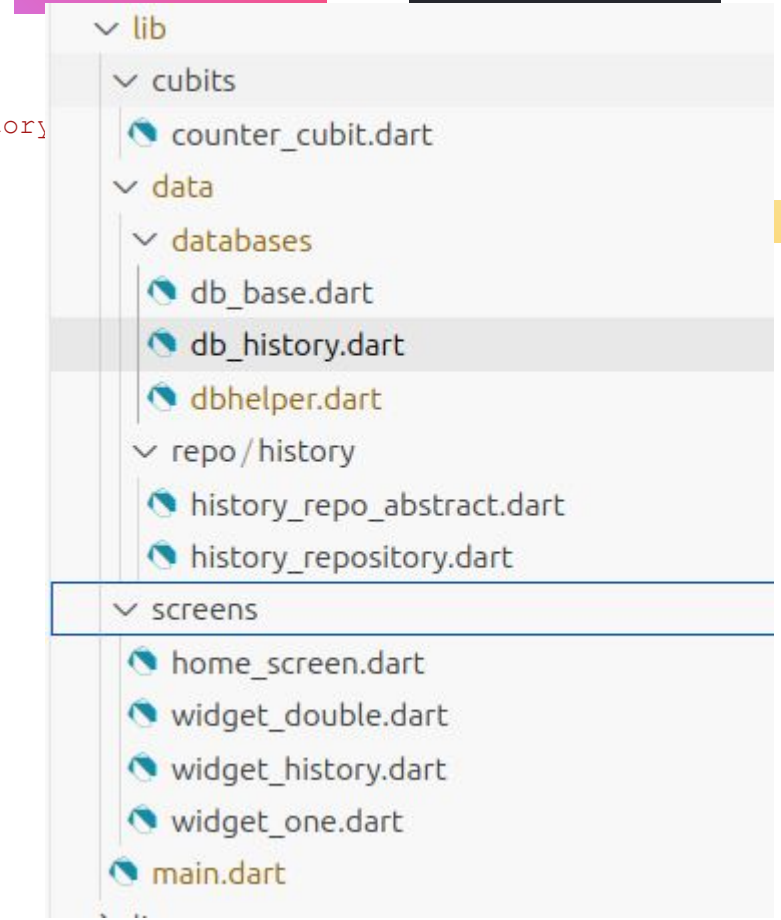
history_repository.dart

```dart
import '../../databases/db_history.dart';
import 'history_repo_abstract.dart';

class HistoryRepository extends HistoryRepositoryBase {
 final DB_HISTORY = DBHistoryTable();

 @override
 Future<List<String>> getData() async {
   List<Map> obj = await DB_HISTORY.getRecords();
   List<String> result = [];
   obj.forEach((item) {
     result.add(item['message'] + ' ' + item['create_date']);
   });
   return result;
 }

 @override
 Future<bool> insertData(String item) async {
   DB_HISTORY.insertRecord({
     'message': item,
     'create_date': DateTime.now().toString(),
   });
   return true;
 }
}
```



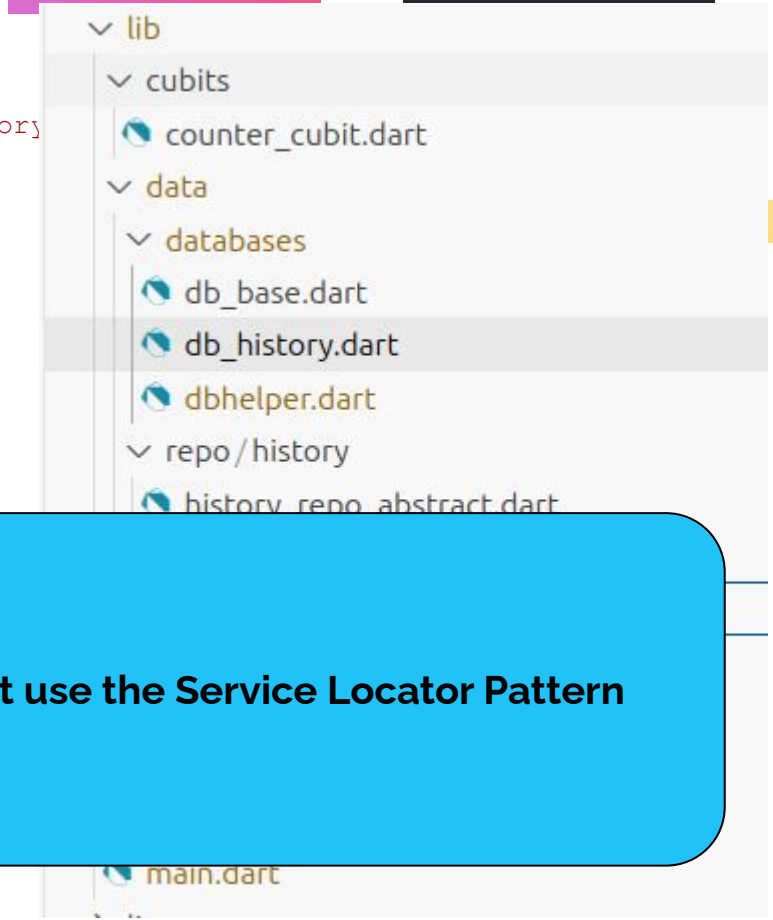
counter_cubit.dart

```dart
class CounterCubit extends Cubit<Map<String, dynamic>> {
 final repo = HistoryRepositoryBase.getInstance();
 CounterCubit() : super({'status': 'loading', 'count': 0, 'history
   loadData();
 }
 Future<bool> loadData({int increment = 1}) async {
   Map<String, dynamic> data = {'status': 'loading',
     'count': increment,
     'history': state['history'],
   };
   emit(data);
   await Future.delayed(Duration(seconds: 3));
   try {
     final records = await repo.getData();
     Map<String, dynamic> data2 = {
       'status': 'done','count': increment,'history': records};
     emit(data2);
   } catch (e, stack) {print('$e --> $stack');}
   return true;
 }

 Future<bool> increment() async {
   await repo.insertData('Incremented');
   loadData(increment: state['count'] + 1);
   return true;
 }
}
```



counter_cubit.dart

```dart
class CounterCubit extends Cubit<Map<String, dynamic>> {
 final repo = HistoryRepositoryBase.getInstance();
 CounterCubit() : super({'status': 'loading', 'count': 0, 'history
   loadData();
 }
 Future<bool> loadData({int increment = 1}) async {
   Map<String, dynamic> data = {'status': 'loading',
     'count': increment,
     'history': state['history'],
   };
   emit(data);
   await Future.delayed(Duration(seconds: 3));
   try {
     final records = await repo.getData();
     Map<String, dynamic> data2 = {
       'status': 'done','count': increment,'histor
     emit(data2);
   } catch (e, stack) {print('$e --> $stack');}
   return true;
 }

 Future<bool> increment() async {
   await repo.insertData('Incremented');
   loadData(increment: state['count'] + 1);
   return true;
 }
}
```



Or Just use the Service Locator Pattern

Section 2

Architecture & Technologies for Integration

External Databases & Storage

External Databases & Storage

- **Local Storage is enough ?**
  - For a small calculator
  - Or the snake game
  - Or personal note taking
  - Or other very very limited few apps

External Databases & Storage

- **Local Storage is enough ?**
  - For a s
  - Or the
  - Or per
  - Or oth

How do you know how many active users of your offline app ?

External Databases & Storage

- **Why the need for External Databases & Storage :**
  - Data Persistence Across Different Devices
  - Collaboration, Sharing and Multi-users
  - Backup and Recovery
  - Scalability and performance
  - Integration with other systems (AI..)
  - Business needs
  - Analytics

External Databases & Storage

- **Types of Data Formats to be stored externally :**
  - **Files & Binary Data**
    - *Examples :*
      - *Images, Files, videos*
    - *Important Notes*
      - *NEVER store binary files inside an SQL or NoSQL Engines.*

External Databases & Storage

- **Types of Data Formats to be stored externally :**
  - **NoSQL data**
    - *Data is stored in a flexible format within documents containing simple JSON data - Key-Values*
  - Best used for :
    - Semi-structured or rapidly changing data structure
    - Apps that don't need complex joins
    - Apps without **frequent**/extensive write operations at row level.

External Databases & Storage

- **Types of Data Formats to be stored externally :**
  - **Structured Relational Data**
    - *Tables for different entities related together with foreign keys…*
    - *Data organized in tables, rows, columns, with relationships (foreign keys..)*
  - **Best for :**
    - Highly structured data
    - Need for consistency, transactions, and complex queries.

External Databases & Storage

- **Technology for : Files & Binary Data**
  - **Cloud-Based Solution** : The provider will manage the infrastructure and you may pay for the storage + transfer OUT+IN
  - Providers:
    - Firebase Storage / Google Cloud Storage
    - Supabase Storage
    - AWS S3
    - DigitalOcean Spaces
    - Azure Blob Storage
    - Cloudinary

External Databases & Storage

- **Technology for : Files & Binary Data**
  - **Self-Hosted Solution** : You manage the infrastructure/servers/network connection
  - Software Solutions:
    - *MinIO*
    - *Supabase*

External Databases & Storage

- **Technology for : Files & Binary Data**
  - **Content Delivery Networks** : The same file will be stored/copied to different locations to speed up the delivery of your file to different users across the globe.
    - Technology Providers:
      - *Cloudflare R2*
      - *Bunny.net*
      - *AWS CloudFront*
      - *Fastly*
      - *Google Cloud CDN*

External Databases & Storage

- **Technology for ： NoSQL data**
  - Cloud-Based Solution:
    - Firebase Firestore
    - MongoDB
    - AWS DynamoDB

External Databases & Storage

- **Technology for ： Relational Structured Databases**
  - Cloud-Based Providers :
    - Supabase (PostgreSQL-based)
    - Neon (Serverless PostgreSQL)
    - PlanetScale (MySQL-based)

External Databases & Storage

- **Traditional Databases Access vs. Real-time Databases**
  - Traditional databases :
    - Mobile App → Request data → DB Server → Give data → Mobile Refresh UI based on the data given.
  - Real-time Databases
    - Automatically synchronizes data changes across all connected **clients instantly**, without requiring manual refresh or polling

External Databases & Storage

- **Real-time Databases**
  - Best use for
    - Chat Apps
    - Collaborative Apps
    - Live data (Dashboards, analytics..)
    - Multiplayer games
    - Location tracking...

External Databases & Storage

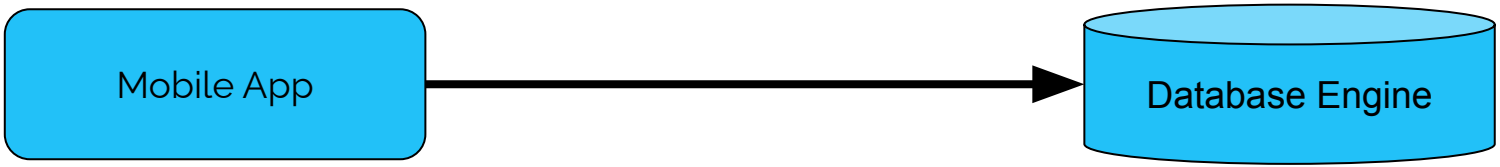
- **Real-time Databases**
  - Technology Providers:
    - Firebase Realtime Database
    - Supabase

External Databases & Storage

- How to access a database from a mobile app ?

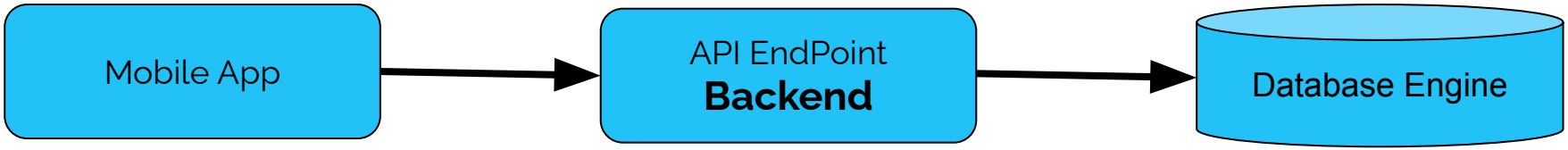
External Databases & Storage

- **How to access a database from a mobile app ?**
  - **Direct Database Access**



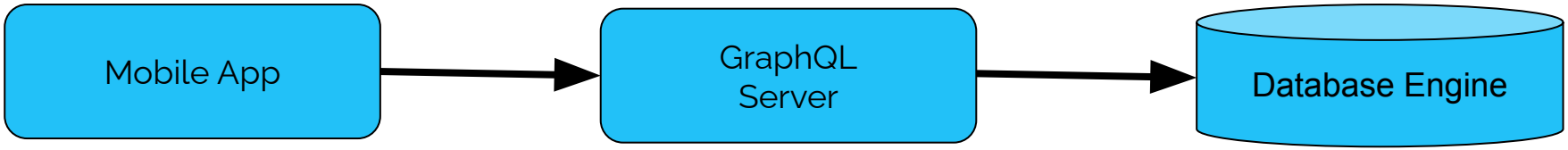
External Databases & Storage

- **How to access a database from a mobile app ?**
  - **Access to Database via EndPoint API**



External Databases & Storage

- **How to access a database from a mobile app ?**
  - **Access to Database via GraphQL**



External Databases & Storage

- **How to access a database from a mobile app ?**
  - *Scalability*
  - *Security*
  - *Flexibility*
  - *Business Logic*
  - *Sovereignty*
  - *Development Time*
  - *Extensibility*

External Databases & Storage

	Direct Access	Via Backend
Scalability	Expensive	Controllable, Easy
Security	You embed credentials	Secure
Flexibility	Not flexible	Very Flexible
Business Logic	Must be on Mobile App	Can be done on Backend
Sovereignty	NO	You can change
Extensibility	No	You can always add…
Development Time	Excellent	Takes time

Ext Services: Cloud Storage

Firebase Services : Introduction

- **Firebase is :**
  - **“**a Backend-as-a-Service (BaaS) app development platform that provides hosted backend services such as a realtime database, cloud storage, authentication, crash reporting, machine learning, remote configuration, and hosting for your static files.**“**

https://docs.flutter.dev/data-and-backend/firebase?

Firebase Services : Introduction

- **Services Provided by Firebase :**
  - Messaging
  - Remote Config
  - Database (NoSQL)
  - File Storage
  - Authentication
  - Machine Learning
  - Analytics
  - Functions
  -

Available plugins

Product	Plugin name	iOS	Android	Web	Other Apple (macOS, etc.)
Analytics	`firebase_analytics`	✓	✓	✓	beta
App Check	`firebase_app_check`	✓	✓	✓	beta
Authentication	`firebase_auth`	✓	✓	✓	beta
Cloud Firestore	`cloud_firestore`	✓	✓	✓	beta
Cloud Functions	`cloud_functions`	✓	✓	✓	beta
Cloud Messaging	`firebase_messaging`	✓	✓	✓	beta
Cloud Storage	`firebase_storage`	✓	✓	✓	beta
Crashlytics	`firebase_crashlytics`	✓	✓		beta
Dynamic Links	`firebase_dynamic_links`	✓	✓		
In-App Messaging	`firebase_in_app_messaging`	✓	✓		
Firebase installations	`firebase_app_installations`	✓	✓	✓	beta
ML Model Downloader	`firebase_ml_model_downloader`	✓	✓		beta
Performance Monitoring	`firebase_performance`	✓	✓	✓	
Realtime Database	`firebase_database`	✓	✓	✓	beta
Remote Config	`firebase_remote_config`	✓	✓	✓	beta

List of all firebase plugins for flutter : https://firebase.google.com/docs/flutter/setup?platform=ios#available-plugins

Firebase Services : Introduction

- **Services Provided by Firebase :**
  - **Messaging**
  - **Remote Config**
  - **Database (NoSQL)**
  - **File Storage**
  - Authentication
  - Machine Learning
  - **Analytics**
  - Functions
  -

Available plugins

Product	Plugin name	iOS	Android	Web	Other Apple (macOS, etc.)
Analytics	firebase_analytics	✓	✓	✓	beta
App Check	firebase_app_check	✓	✓	✓	beta
Authentication	firebase_auth	✓	✓	✓	beta
Cloud Firestore	cloud_firestore	✓	✓	✓	beta
Cloud Functions	cloud_functions	✓	✓	✓	beta
Cloud Messaging	firebase_messaging	✓	✓	✓	beta
Cloud Storage	firebase_storage	✓	✓	✓	beta
Crashlytics	firebase_crashlytics	✓	✓		beta
Dynamic Links	firebase_dynamic_links	✓	✓		
In-App Messaging	firebase_in_app_messaging	✓	✓		
Firebase installations	firebase_app_installations	✓	✓	✓	beta
ML Model Downloader	firebase_ml_model_downloader	✓	✓		beta
Performance Monitoring	firebase_performance	✓	✓	✓	
Realtime Database	firebase_database	✓	✓	✓	beta
Remote Config	firebase_remote_config	✓	✓	✓	beta

List of all firebase plugins for flutter : https://firebase.google.com/docs/flutter/setup?platform=ios#available-plugins

Firebase Services : Introduction

- **Steps to get Started**
  - Install the Firebase CLI :
    - https://firebase.google.com/docs/cli#install-cli-mac-linux
  - Link your Firebase Account
    - **firebase login**
  - Follow all instructions at:

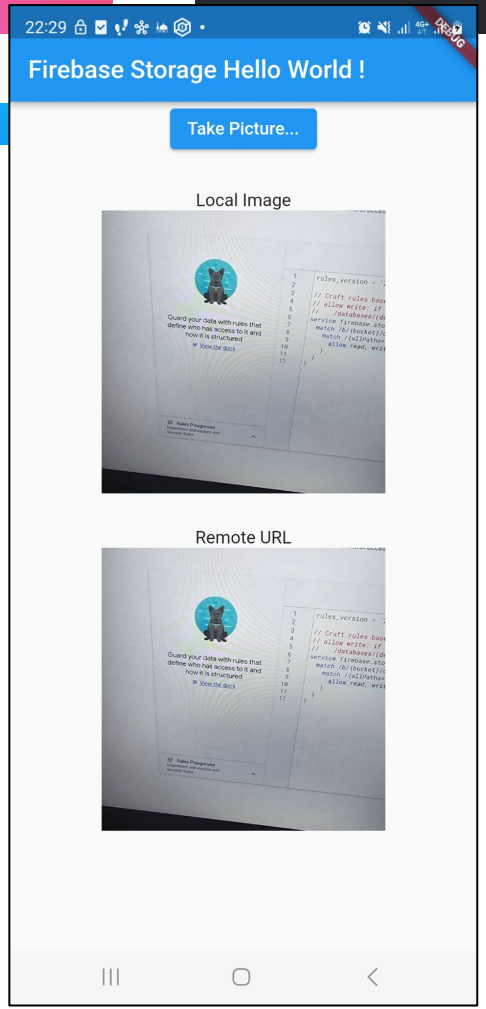
    https://firebase.google.com/docs/flutter/setup?platform=ios

External Services: Firebase Cloud Storage

- **Firebase Cloud Storage**
  - “built on fast and secure Google Cloud infrastructure for app developers who need to store and serve user-generated content, such as images or videos.”
  - Plugins to install
    - **flutter pub add firebase_core**
    - **flutter pub add firebase_storage**

External Services: Firebase Cloud Storage

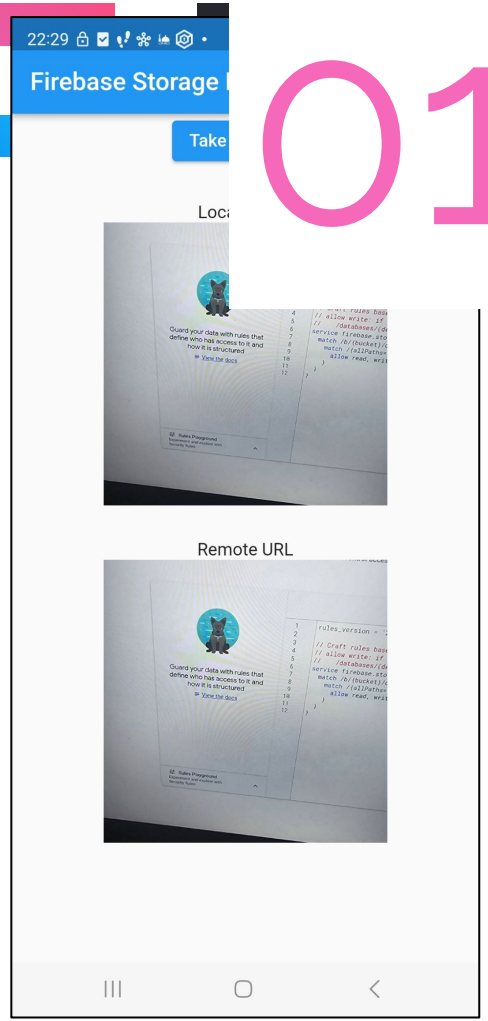
- **Firebase Cloud Storage : MVP (Hello World !)**
  - MVP :
    - Simple Image Picker
    - Take and show a photo stored locally
    - Upload **immediately** to Firebase
    - Get a Remote URL for the image
    - Show it on the UI



External Services: Firebase Cloud Storage

01

- **Firebase Cloud Storage : MVP (Hello World !)**
  - Create a project inside the Firebase Console
    - https://console.firebase.google.com
  -

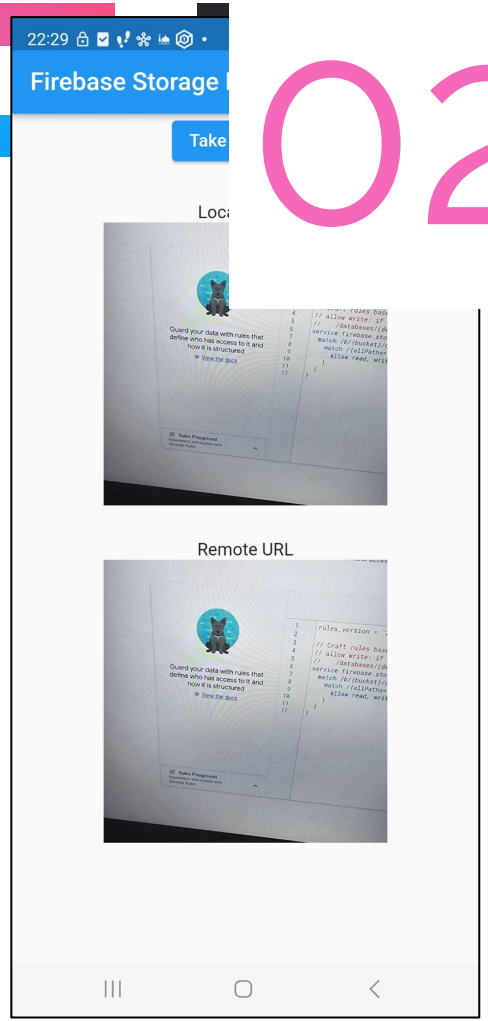


External Services: Firebase Cloud Storage

02

- **Firebase Cloud Storage : MVP (Hello World !)**
  - Install Firebase CLI and FlutterFire CLI
    - **npm install -g firebase-tools**
    - **dart pub global activate flutterfire_cli**
    - **firebase login**

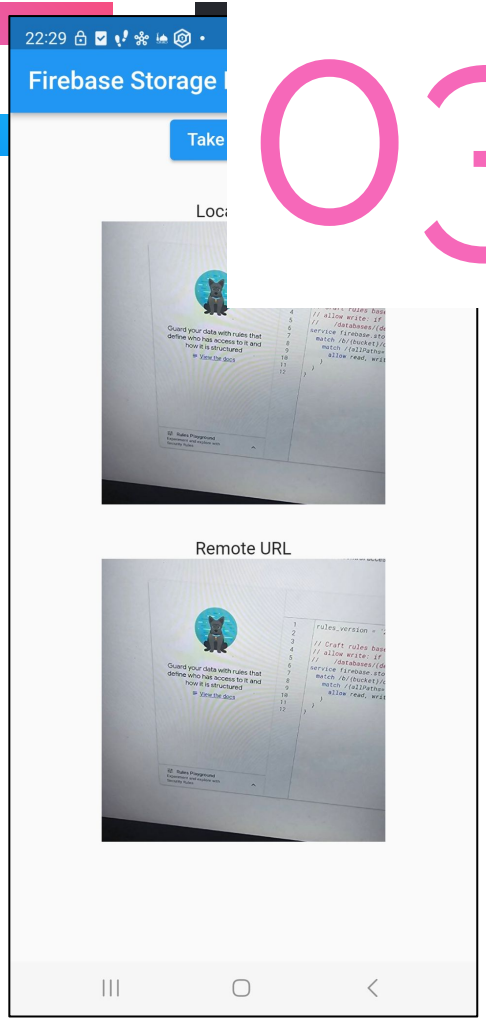
ONE TIME ONLY



External Services: Firebase Cloud Storage

03

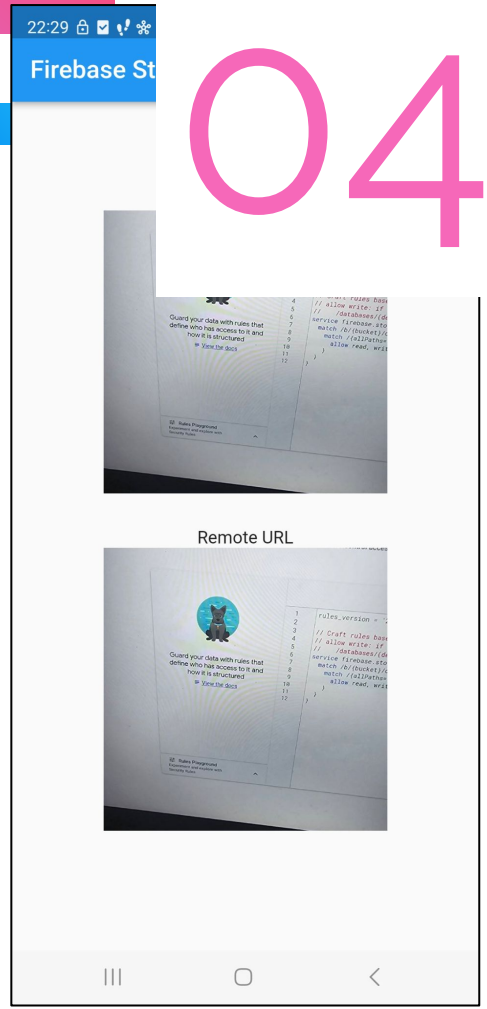
- **Firebase Cloud Storage : MVP (Hello World !)**
  - Add Flutter Dependencies
    - **flutter pub add firebase_core**
    - **flutter pub add firebase_storage**
  - For taking photos, we need also the image_picker
    - **flutter pub add image_picker**



External Services: Firebase Cloud Storage

04

- **Firebase Cloud Storage : MVP (Hello World !)**
  - Configure Firebase for Flutter
    - **flutterfire configure**
      - *select your Firebase project*
      - *Select platforms (choose Android and iOS, or just Android for simplicity)*
      - *Automatically create* ***firebase_options.dart*** *file*
      - *Configure your app with Firebase*



EXPLORER

FIREBASE_CLOUD_STORAGE_HELLO

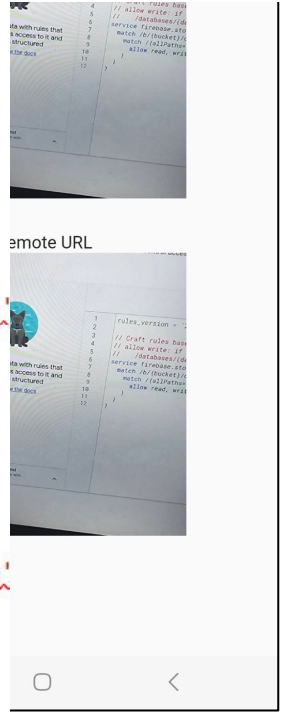
- .dart_tool
- .idea
- android
- ios
- lib
  - firebase_options.dart 9+
  - main.dart
- linux
  - flutter
  - runner
  - .gitignore
  - CMakeLists.txt
- macos
- web
- windows
- .gitignore
- .metadata
- analysis_options.yaml
- firebase_cloud_storage_hello.iml
- firebase.json
- pubspec.lock
- pubspec.yaml
- README.md

firebase_options.dart 9+

lib > firebase_options.dart > ...

```
17  class DefaultFirebaseOptions {
18    static FirebaseOptions get currentPlatform {
34              'you can reconfigure this by running the FlutterFire CLI
35            );
36          default:
37            throw UnsupportedError(
38              'DefaultFirebaseOptions are not supported for this platf
39            );
40      }
41    }
42
43    static const FirebaseOptions web = FirebaseOptions(
44      apiKey: 'AIzaSyAQ_Bh7fKSIlye-3W3M9LZDIWFPxO8QNMc',
45      appId: '1:126420766772:web:b9197e0f96a375944c3a61',
46      messagingSenderId: '126420766772',
47      projectId: 'ensia-hello-storage-cloud',
48      authDomain: 'ensia-hello-storage-cloud.firebaseapp.com',
49      storageBucket: 'ensia-hello-storage-cloud.firebasestorage.app'
50    );
51
52    static const FirebaseOptions android = FirebaseOptions(
53      apiKey: 'AIzaSyBQMwklEOsCTxbrR79wEPQ9IumfIt6Qf1c',
54      appId: '1:126420766772:android:8def1f8a5eeb973f4c3a61',
55      messagingSenderId: '126420766772',
56      projectId: 'ensia-hello-storage-cloud',
57      storageBucket: 'ensia-hello-storage-cloud.firebasestorage.app'
58    );
59
60    static const FirebaseOptions ios = FirebaseOptions(
61      apiKey: 'AIzaSyAUmaY-B8z2_ouYo42xew-T1TgXiXZbLmQ',
```

04

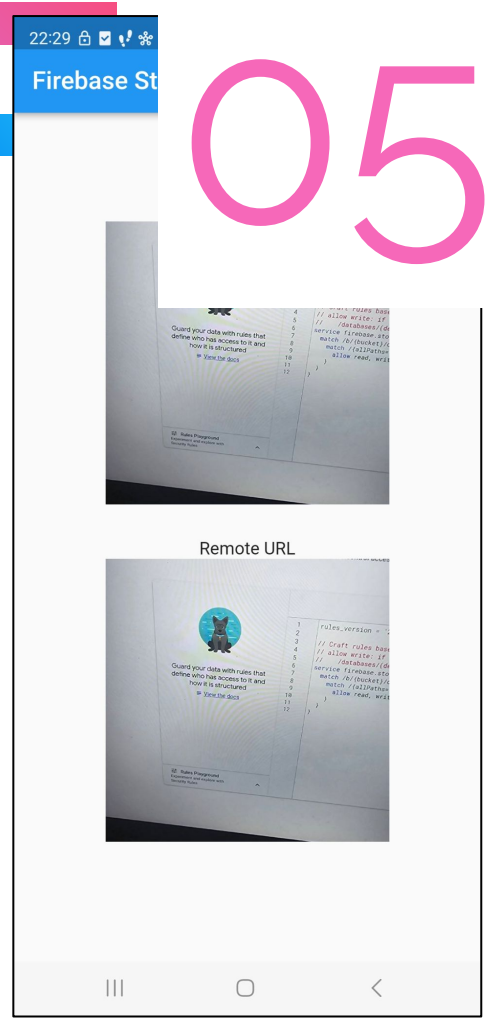


External Services: Firebase Cloud Storage

05

main.dart

```dart
import 'firebase_options.dart';

final storage = FirebaseStorage.instance;
final storageRef = FirebaseStorage.instance.ref();

Future<bool> my init app() async {
 await Firebase.initializeApp(
   options: DefaultFirebaseOptions.currentPlatform,
 );
 return true;
}

void main() async {
 WidgetsFlutterBinding.ensureInitialized();
 await my_init_app();
 runApp(const MainApp());
}
```

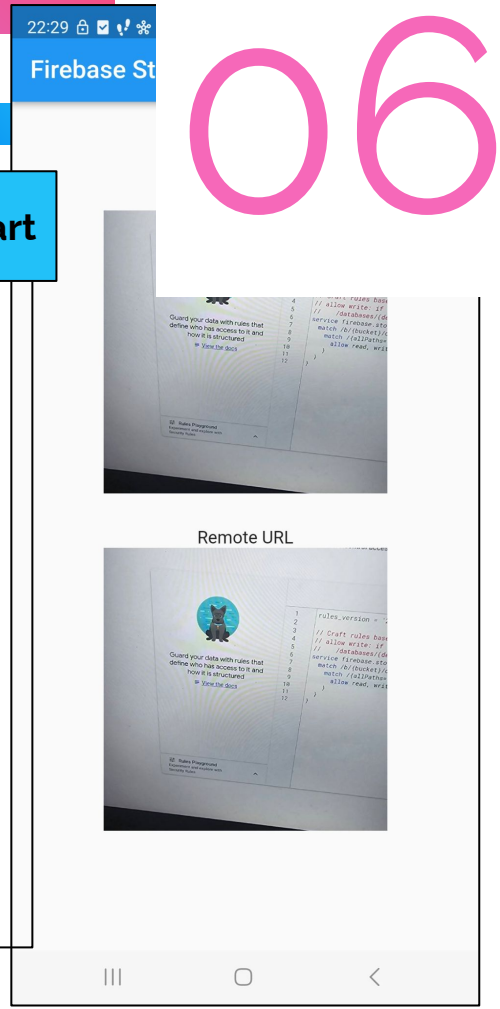


Firebase Services : Cloud Storage

06

screens/home.dart

```
ElevatedButton(
    onPressed: () async {
      String? path = await onImageButtonPressed(
        ImageSource.camera,
        context,
      );
      if (path != null) {
        local_path = path;
        setState(() {});

        upload_to_firebase_storage();
      }
    },
    child: Text("Take Picture..."),
```

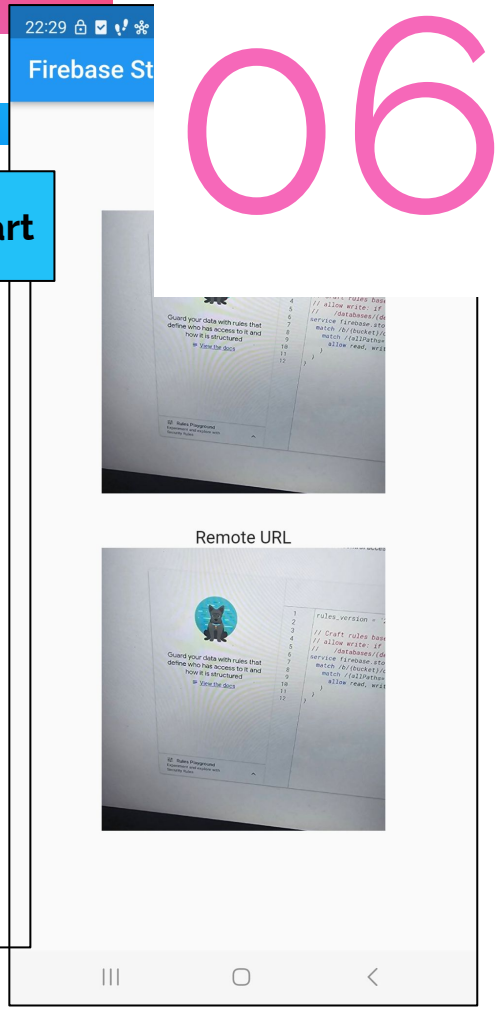


Firebase Services : Cloud Storage

06

screens/home.dart

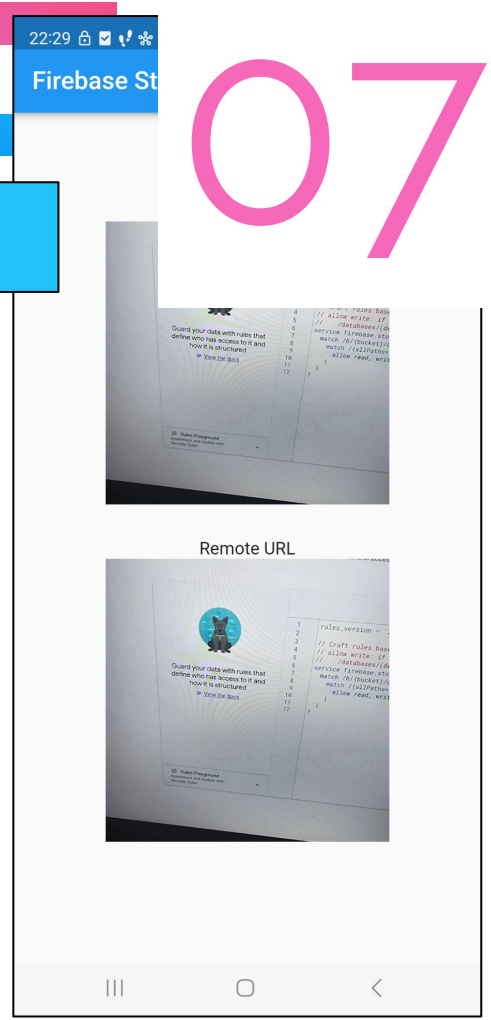
```
Future<bool> upload_to_firebase_storage() async {
  try {
    String filename = 'abc.jpg';
    final spaceRef = storageRef.child("myimages/$filename");
    await spaceRef.putFile(File(local_path));
    final imageUrl = remote_url = await spaceRef.getDownloadURL();
    setState(() {});
  } catch (e) {
    print("Exception $e");
  }
  return true;
}
```



Firebase Services : Cloud Storage

07

IMPORTANT ..



- **Firebase Cloud Storage**
  - Login to Firebase Console and Initialize the bucket
  - Grant the security rules to write, read to **true**

Firebase

Product Information Collection

Project Overview

Project shortcuts

Storage

Firestore Database

Remote Config

Authentication

Product categories

Build

Release & Monitor

Analytics

Engage

All products

Storage

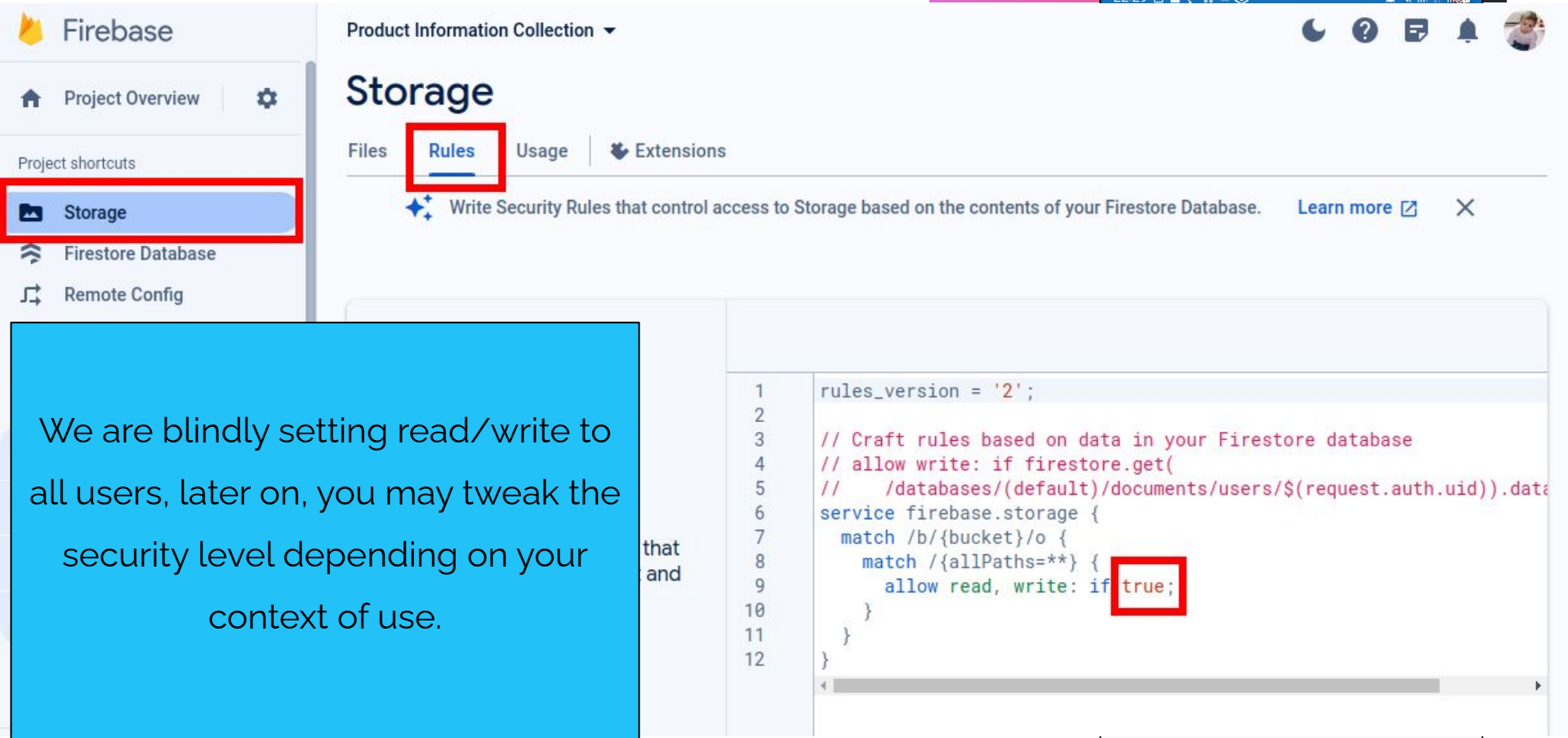
Files | Rules | Usage | Extensions

Write Security Rules that control access to Storage based on the contents of your Firestore Database. Learn more

Guard your data with rules that define who has access to it and how it is structured

View the docs

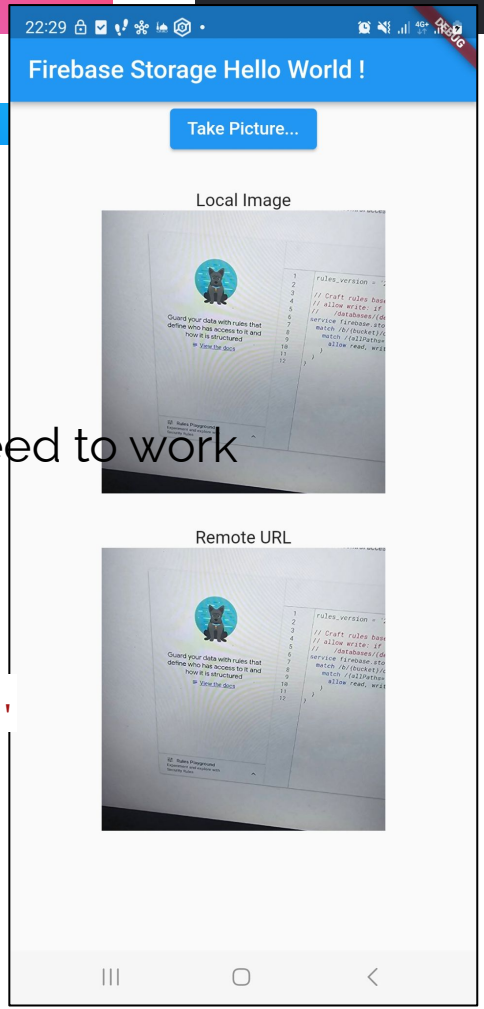
```
1  rules_version = '2';
2
3  // Craft rules based on data in your Firestore database
4  // allow write: if firestore.get(
5  //    /databases/(default)/documents/users/$(request.auth.uid)).data
6  service firebase.storage {
7    match /b/{bucket}/o {
8      match /{allPaths=**} {
9        allow read, write: if true;
10     }
11   }
12 }
```



We are blindly setting read/write to all users, later on, you may tweak the security level depending on your context of use.

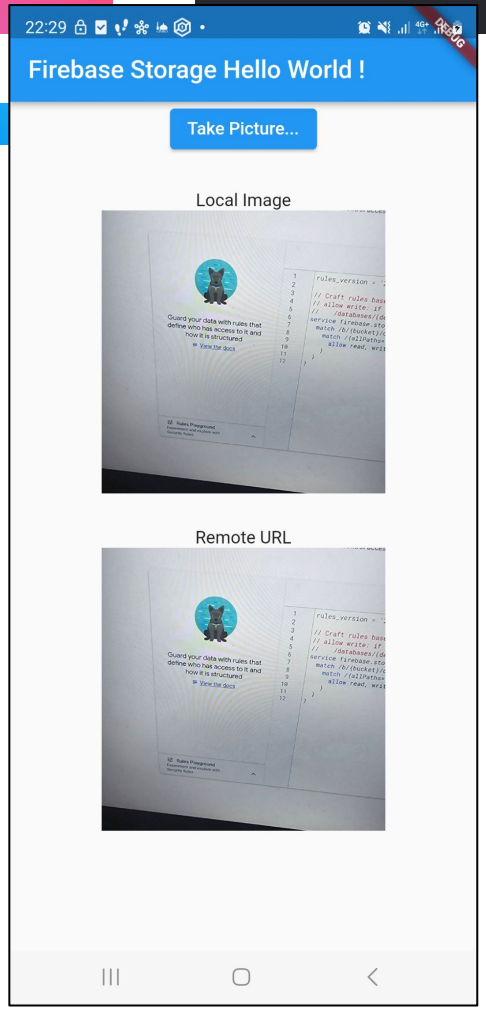
Firebase Services : Cloud Storage

- **Firebase Cloud Storage**
  - Errors & Failure when building ? Try (not guaranteed to work
    - Update the Google Play Services
      - android/build.gradle
        - `classpath 'com.google.gms:google-services:4.4.0'`
    - Increase the minSdk
      - android/app/src/build.gradle
        - `minSdkVersion 19`



Firebase Services : Cloud Storage

- **Warning**
  - We don't upload directly to backed
  - This was an easy hello world to show you how to upload.
  - In real cases, use the principles taught in the previous lecture:
    - *Store upload tasks into a local DB*
    - *Upload using a Cron Job*
    - *Use State Management when possible*



External Services: Firestore

Firebase Services : Cloud Firestore Database

- **Cloud Firestore Database :**
  - Flexible, scalable database for mobile, web, and server development.
  - It is Like Firebase Realtime Database, it keeps your data in sync across client apps through realtime listeners.
  - It offers offline support for mobile and web so you can build responsive apps that work regardless of network latency or Internet connectivity.

Firebase Services : Cloud Firestore Database

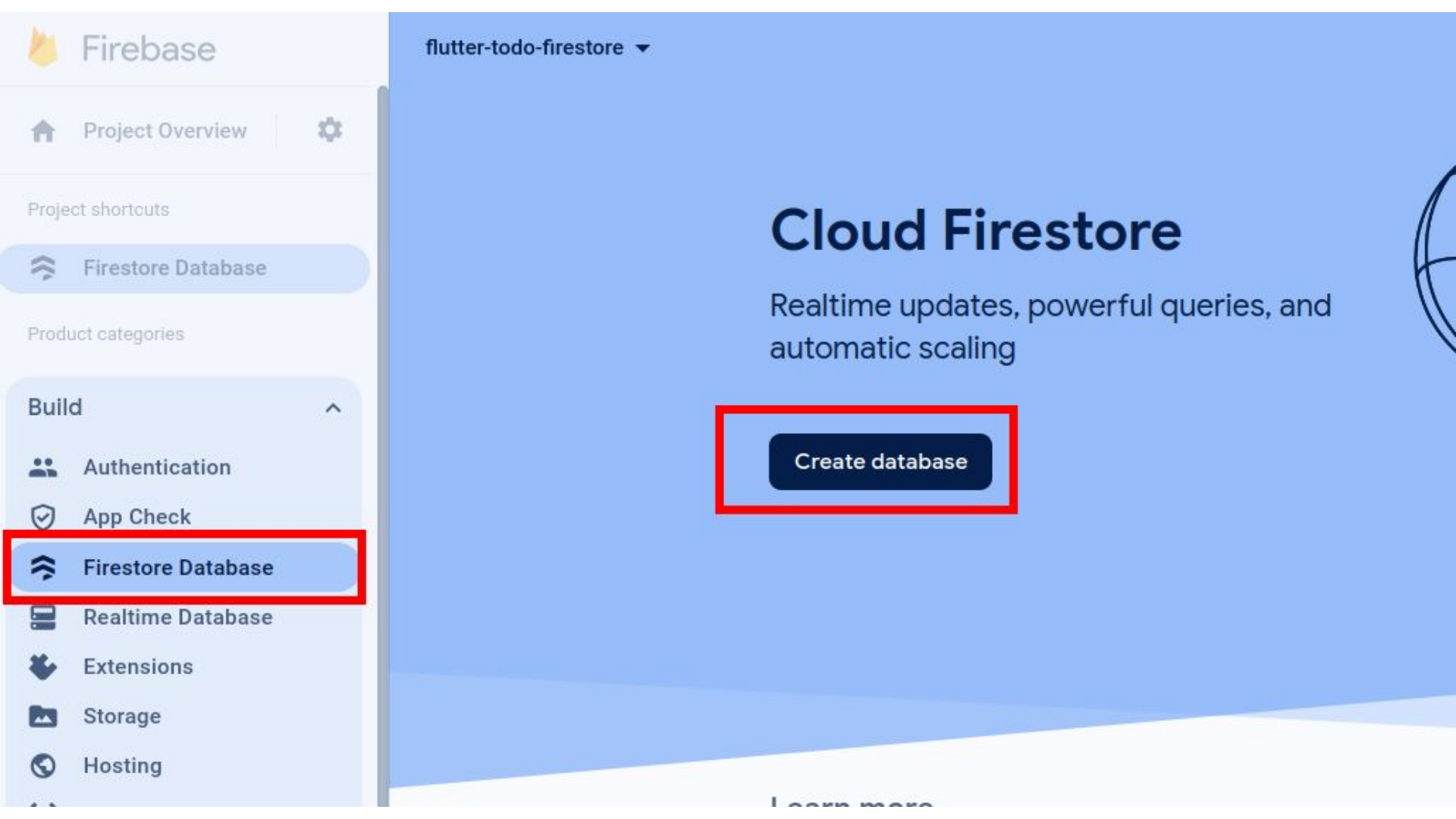
- **Cloud Firestore Database :**
  - Storage is structured as :
    - List of Collections
      - Each Collection contains Document
        - Document is a simple JSON text/Map.

Firebase Services : Cloud Firestore Database

- **Cloud Firestore Database :**
  - Getting Started : Case Study : To-do app (No need for user authentication)
    - Decide whether Firestore is relevant to your mobile app.
    - Can integrate with other software components : Backend ?
    - Define the schema/structuring of the data.
    - Define the security policy to get to your data.
    - Integrate with your App.

Firebase Services : Cloud Firestore Database

- **Cloud Firestore Database :**
  - Integration with the App :
    - **flutterfire configure**
    - **flutter pub add firebase_core**
    - **flutter pub add cloud_firestore**


Firebase
flutter-todo-firestore
Project Overview
Project shortcuts
Firestore Database
Product categories
Build
Authentication
App Check
Firestore Database
Realtime Database
Extensions
Storage
Hosting
Cloud Firestore
Realtime updates, powerful queries, and automatic scaling
Create database

flutter-todo-firestore

Cloud Firestore

Data | Rules | Indexes | Usage | Extensions

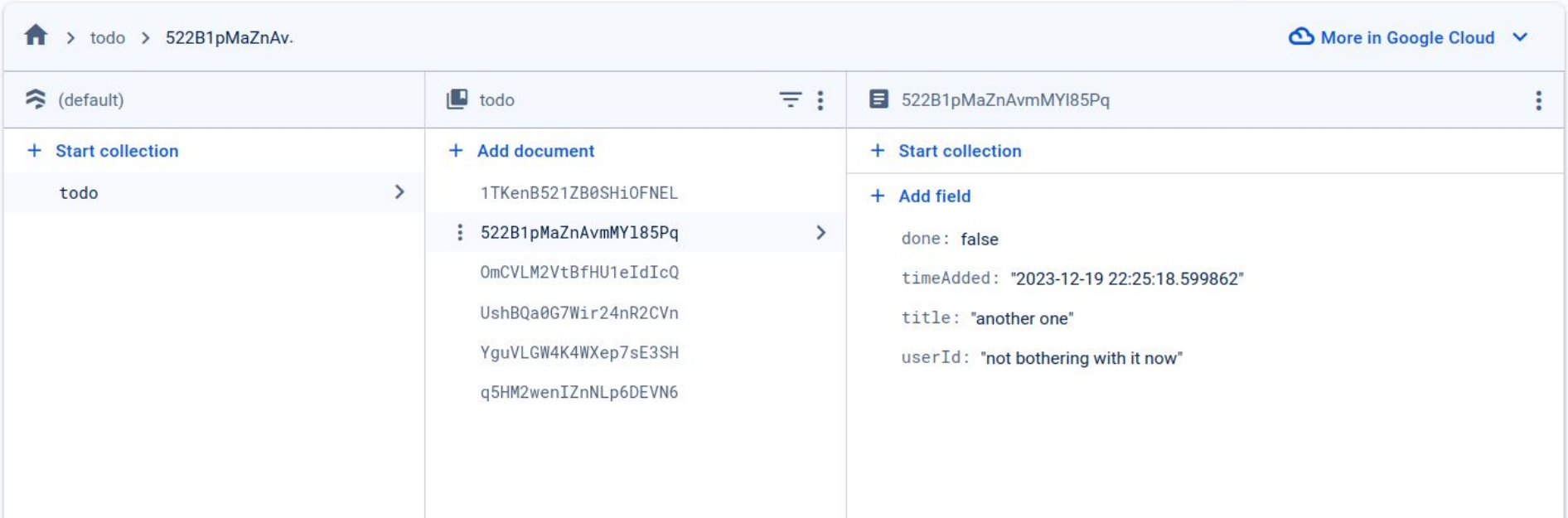
Today • 10:09 PM

Today • 10:08 PM

```
1  rules_version = '2';
2
3  service cloud.firestore {
4    match /databases/{database}/documents {
5      match /{document=**} {
6        allow read, write: if true;
7      }
8    }
9  }
```

Firebase Services : Cloud Firestore Database

- Cloud Firestore Database :

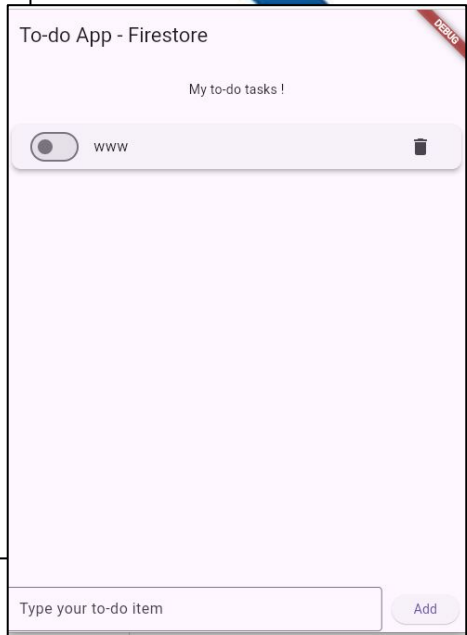


Firebase Services : Cloud Firestore Database

01

main.dart

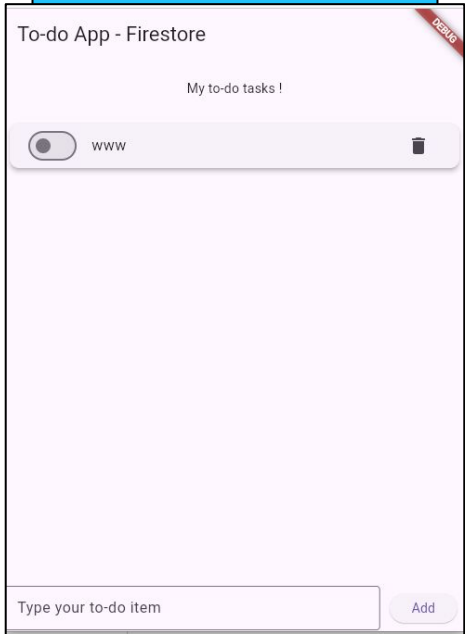
```dart
import 'firebase_options.dart';

final fireStore = FirebaseFirestore.instance;
Future<bool> my init app() async {
 await Firebase.initializeApp(options: DefaultFirebaseOptions.currentPlatform);
 return true;
}
void main() async {
 WidgetsFlutterBinding.ensureInitialized();
 await my_init_app();
 runApp(const MainApp());
}

class MainApp extends StatelessWidget {
 const MainApp({super.key});
 @override
 Widget build(BuildContext context) {
   return MaterialApp(home: HomeScreen());
 }
}
```



```dart
@override
Widget build(BuildContext context) {
  Future<List<Map>> firestore_data = get_data_from_firebase();
  return Scaffold(
    appBar: AppBar(title: const Text('To-do App - Firestore')),
    body: Center(
      child: Column(
          ...
          Expanded(
            child: FutureBuilder(
              future: firestore data,
              builder: buildToListWidget,
            ),
          ),
          ...
                ElevatedButton(
                  onPressed: () async {
                    _tx_title_controller.text = '';
                    await fireStore.collection('todo').add({
                      'title':  tx title_value,
                      'done': false,
                      'userId': 'not bothering with it now',
                      'timeAdded': DateTime.now().toString(),
                     });
                    setState(() {});
                   },
                  child: const Text("Add"),
                ),
```

02

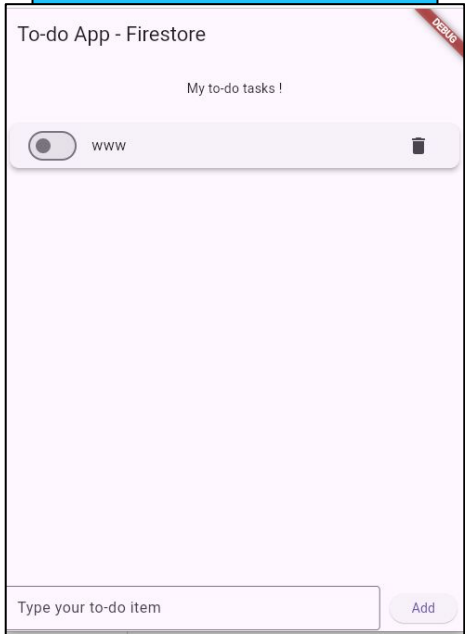
homescreen.dart



```dart
Future<List<Map>> get_data_from_firebase() async {
  List<Map> firestore_data = [];
  QuerySnapshot<Map<String, dynamic>> tmp = await fireStore
      .collection("todo")
      .get();
  for (var doc in tmp.docs) {
    Map map = doc.data();
    map['id'] = doc.id;
    firestore_data.add(map);
  }
  return firestore_data;
}
```

03

homescreen.dart

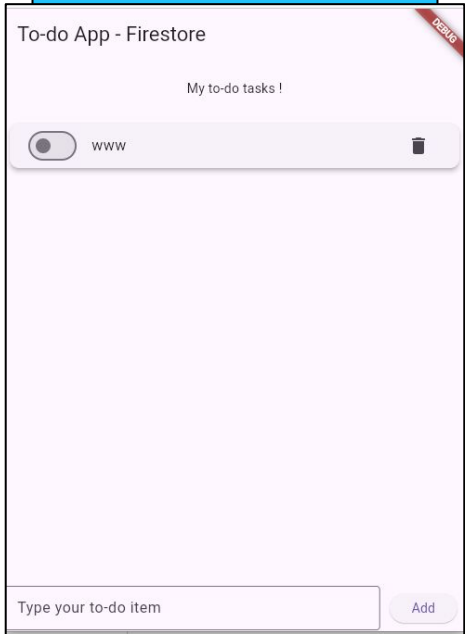


```
Future<List<Map>> get_data_from_firebase() async {
  List<Map> firestore_data = [];
  QuerySnapshot<Map<String, dynamic>> tmp = await fireStore
      .collection("todo")
      .get();
  for (var doc in tmp.docs) {
    Map map = doc.data();
    map['id'] = doc.id;
    firestore_data.add(map);
  }
  return firestore_data;
}
```

Implementation was minimal, no design pattern, everything is assembled into a single file ...

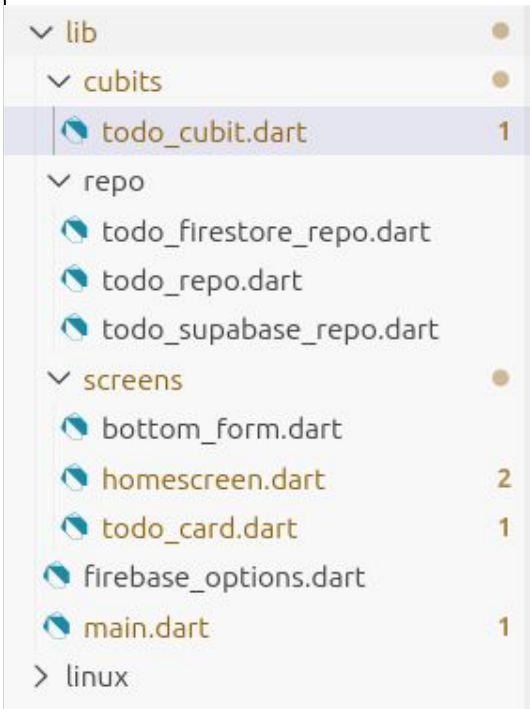
03

homescreen.dart



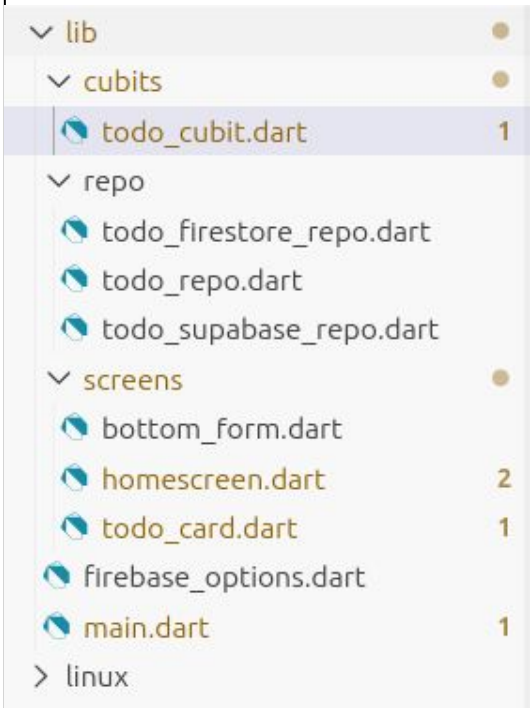
main.dart

```dart
Future<bool> init my app() async {
 final sl = GetIt.instance;
 if (1 == 2) {
   //to be done later
 } else {
   sl.registerLazySingleton<TodoRepo>(() => TodoFirestoreRepo());
   await Firebase.initializeApp(
     options: DefaultFirebaseOptions.currentPlatform,
   );
 }
 WidgetsFlutterBinding.ensureInitialized();
 return true;
}

void main() async {
 await init my app();
 runApp(const MainApp());
}

class MainApp extends StatelessWidget {
 const MainApp({super.key});

 @override
 Widget build(BuildContext context) {
   return MultiBlocProvider(
     providers: [BlocProvider(create: (context) => TodoCubit())],
     child: MaterialApp(home: HomeScreen()),
   );
 }
}
```



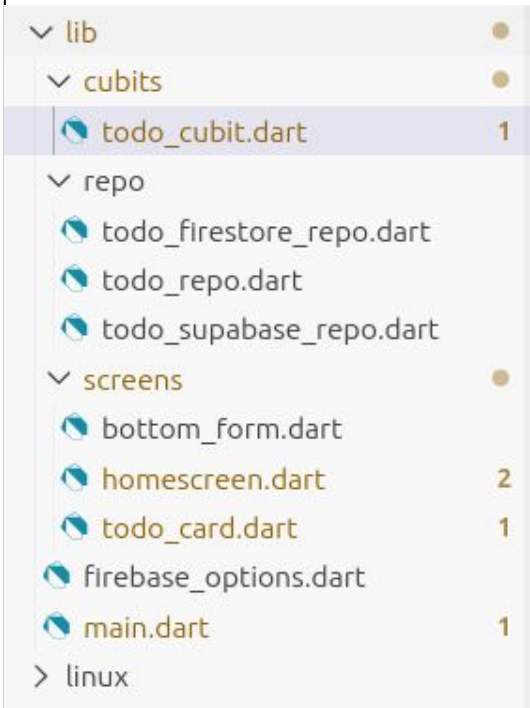
todo_repo.dart

```dart
abstract class TodoRepo {
 Future<List<Map<String, dynamic>>> getData();
 Future<bool> insertRecord(String title);
 Future<bool> deleteRecord(String id);
}
```



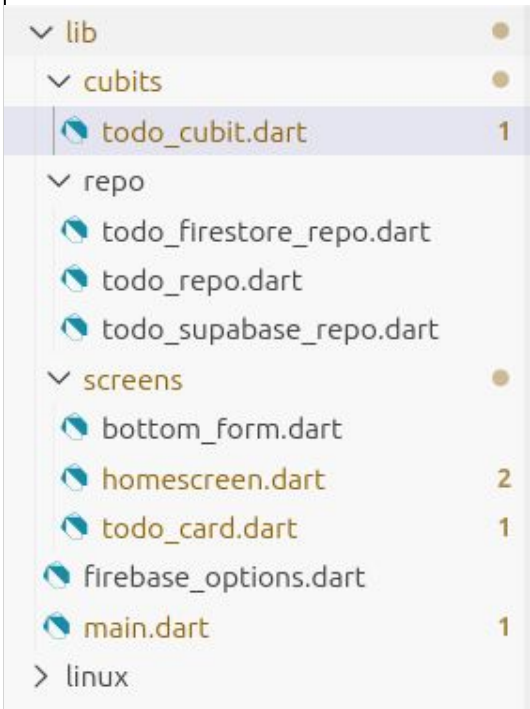
todo_repo.dart

```dart
class TodoFirestoreRepo extends TodoRepo {
 Future<List<Map<String, dynamic>>> getData() async {
   final fireStore = FirebaseFirestore.instance;
   List<Map<String, dynamic>> firestore_data = [];
   QuerySnapshot<Map<String, dynamic>> tmp = await fireStore
       .collection("todo")
       .get();
   for (var doc in tmp.docs) {
     Map<String, dynamic> map = doc.data();
     map['id'] = doc.id;
     firestore_data.add(map);
   }
   return firestore_data;
 }

 Future<bool> deleteRecord(String id) async {
   final fireStore = FirebaseFirestore.instance;
   await fireStore.collection('todo').doc(id).delete();
   return true;
 }

 Future<bool> insertRecord(String title) async {
   final fireStore = FirebaseFirestore.instance;
   await fireStore.collection('todo').add({
     'title': title,
     'done': false,
     'userId': 'not bothering with it now',
     'timeAdded': DateTime.now().toString(),
   });
   return true;
 }
```



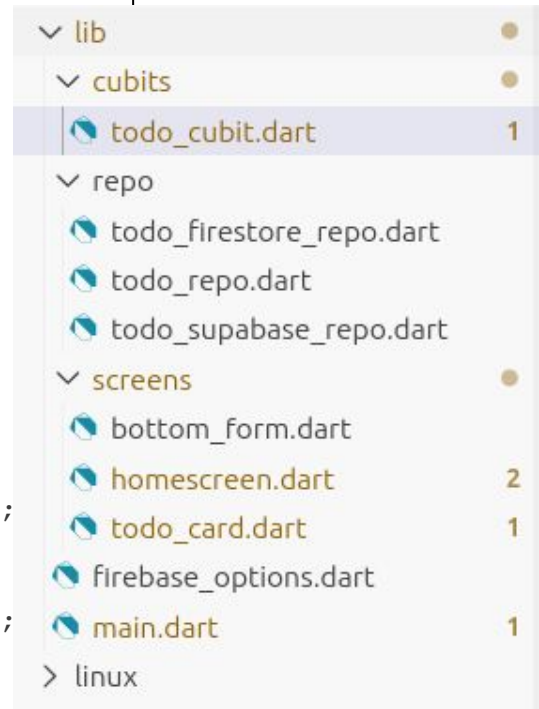
todo_cubit.dart

```dart
class TodoCubit extends Cubit<Map<String, dynamic>> {
 late final TodoRepo todoRepo;
 TodoCubit() : super({'state': 'loading', 'data': [], 'message': ''}) {
   todoRepo = GetIt.I<TodoRepo>();
   loadData();
 }
 Future<bool> loadData() async {
   print('Loading data');
   emit({...state, 'state': 'loading'});
   try {
     var new_data = await todoRepo.getData();
     emit({'state': 'done', 'data': new_data, 'message': ''});
   } catch (e, stack) {
     print('error $e $stack');
     emit({'state': 'error', 'data': [], 'message': e.toString()});
   }
   return true;
 }
 Future<bool> insertRecord(String title) async {
   emit({...state, 'state': 'loading'});
   try {
     var new_data = await todoRepo.insertRecord(title);
     loadData();
   } catch (e, stack) {
     print('error $e $stack');
     emit({'state': 'error', 'data': [], 'message': e.toString()});
   }
   return true;
 }
 Future<bool> deleteRecord(String id) async {
   emit({...state, 'state': 'loading'});
```



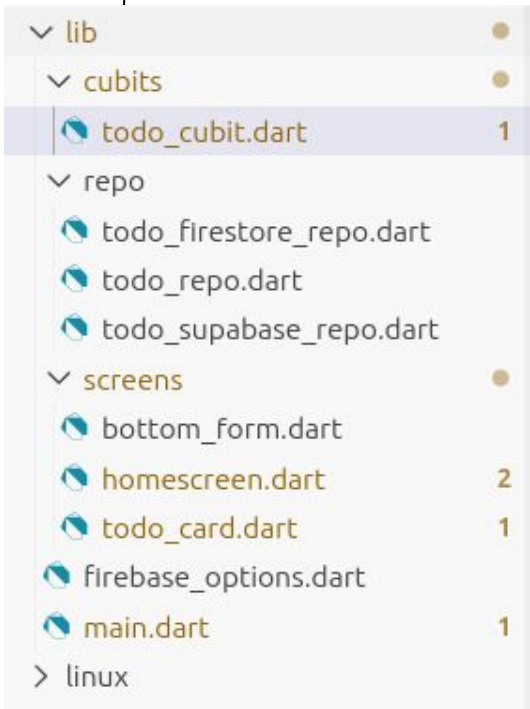
homescreen.dart

```dart
class  HomeScreenState extends State<HomeScreen> {
 @override
 Widget build(BuildContext context) {
   return Scaffold(
     appBar: AppBar(title: const Text('Organized Simple To-do App')),
     body: Center(
       child: Column(
         crossAxisAlignment: CrossAxisAlignment.center,
         children: [
           const SizedBox(height: 20),
           const Text('My to-do tasks !'),
           const SizedBox(height: 20),
           Expanded(
             child: BlocConsumer<TodoCubit, Map<String, dynamic>>(
                listener: (context, state) {},
                builder: (context, data) {
                  if (data['state'] == 'loading') {
                    return const Center(child: CircularProgressIndicator());
                  }
                  if (data['state'] == 'error') {
                    return Center(child: Text('Error: ${data['message']}'));
                  }

                  return ListView.builder(
                    itemCount: data['data'].length,
                    itemBuilder: (context, index) {
                      return TodoCard(data: data['data'][index]);
                    },
                  );
                },
```



bottom_form.dart

```dart
class  BottomFormState extends State<BottomForm> {
 final _tx_title_controller = TextEditingController();
 @override
 Widget build(BuildContext context) {
   return SizedBox(
     height: 50,
     width: MediaQuery.of(context).size.width,
     child: Row(
       children: <Widget>[
         Expanded(
           child: TextFormField(
             decoration: const InputDecoration(
               border: OutlineInputBorder(
                 borderSide: BorderSide(color: Colors.teal),
               ),
               labelText: 'Type your to-do item',
             ),
             keyboardType: TextInputType.text,
             controller: _tx_title_controller,
           ),
         ),
         const SizedBox(width: 10),
         ElevatedButton(
           onPressed: () async {
             String title = _tx_title_controller.text;
             context.read<TodoCubit>().insertRecord(title);
             _tx_title_controller.text = '';
           },
           child: const Text("Add"),
         ),
         const SizedBox(width: 10),
```



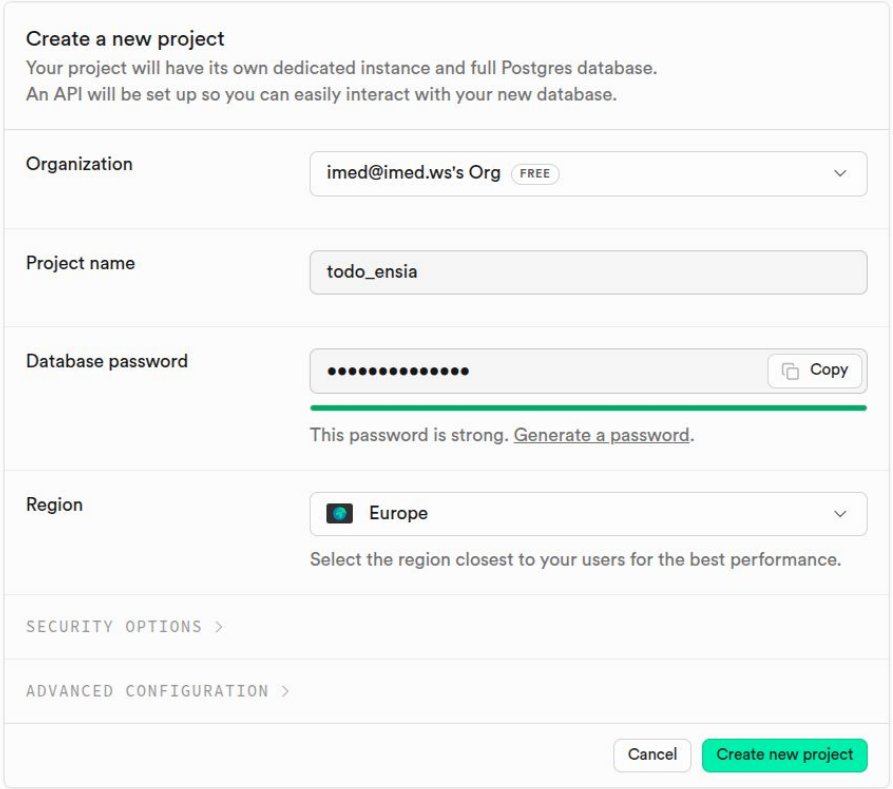
External Services: Supabase

External Services: Supabase / Relational

- **Use of Relational Databases:**
  - As data has relation among them, relational database engine the recommended solution
  - Using NoSQL is only recommended when processing large data which is **already structured**
  - **PostgreSQL** another free relational database with its better performance compared to other dbms.
  - Supabase is another cloud provider offering free PostgreSQL hosting (To get started, when you grow, you pay..).

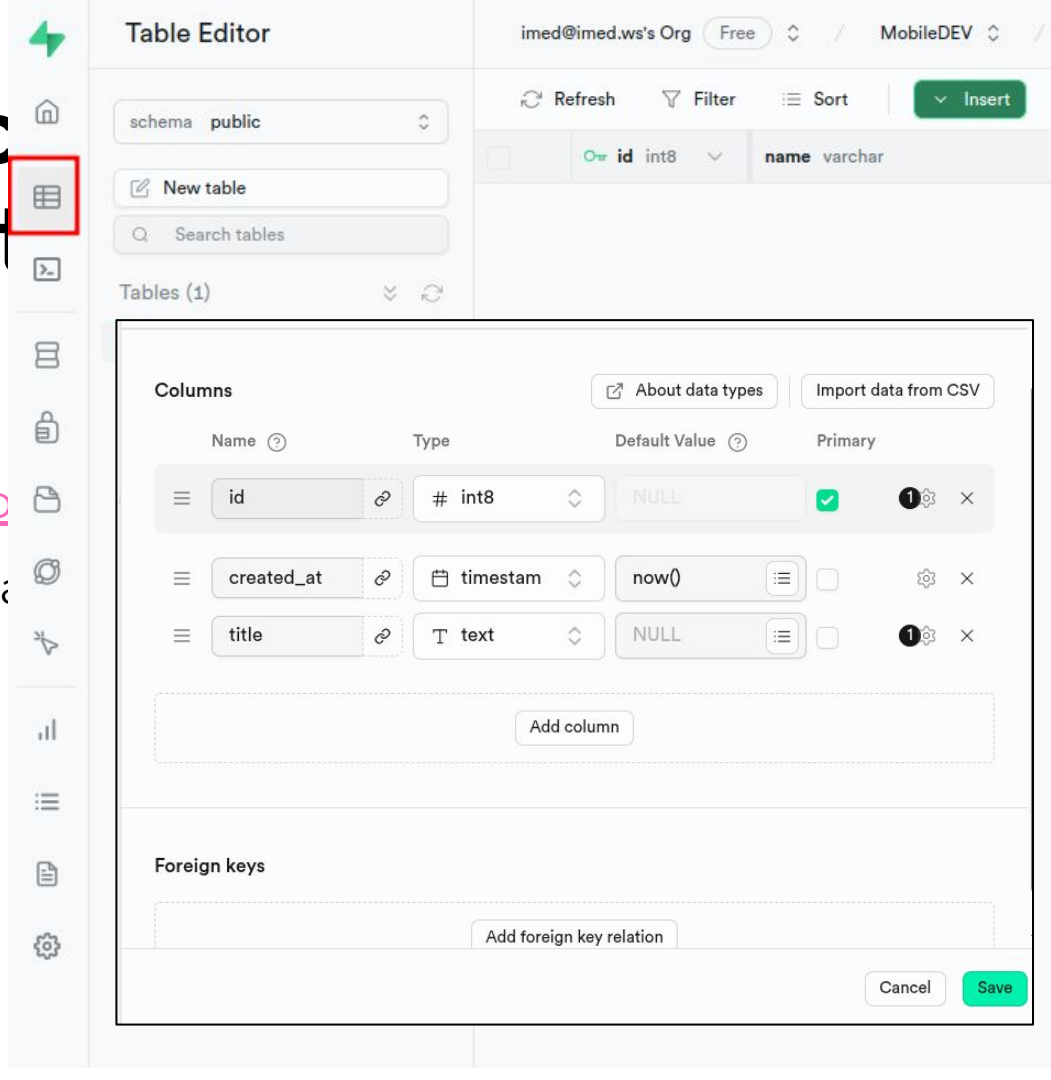
External Services: Supabase / Relati

- **Steps to get Started :**
  1. Signup at www.supabase.com
  2. Create a Project and set a Password, Please keep the password at a safe place



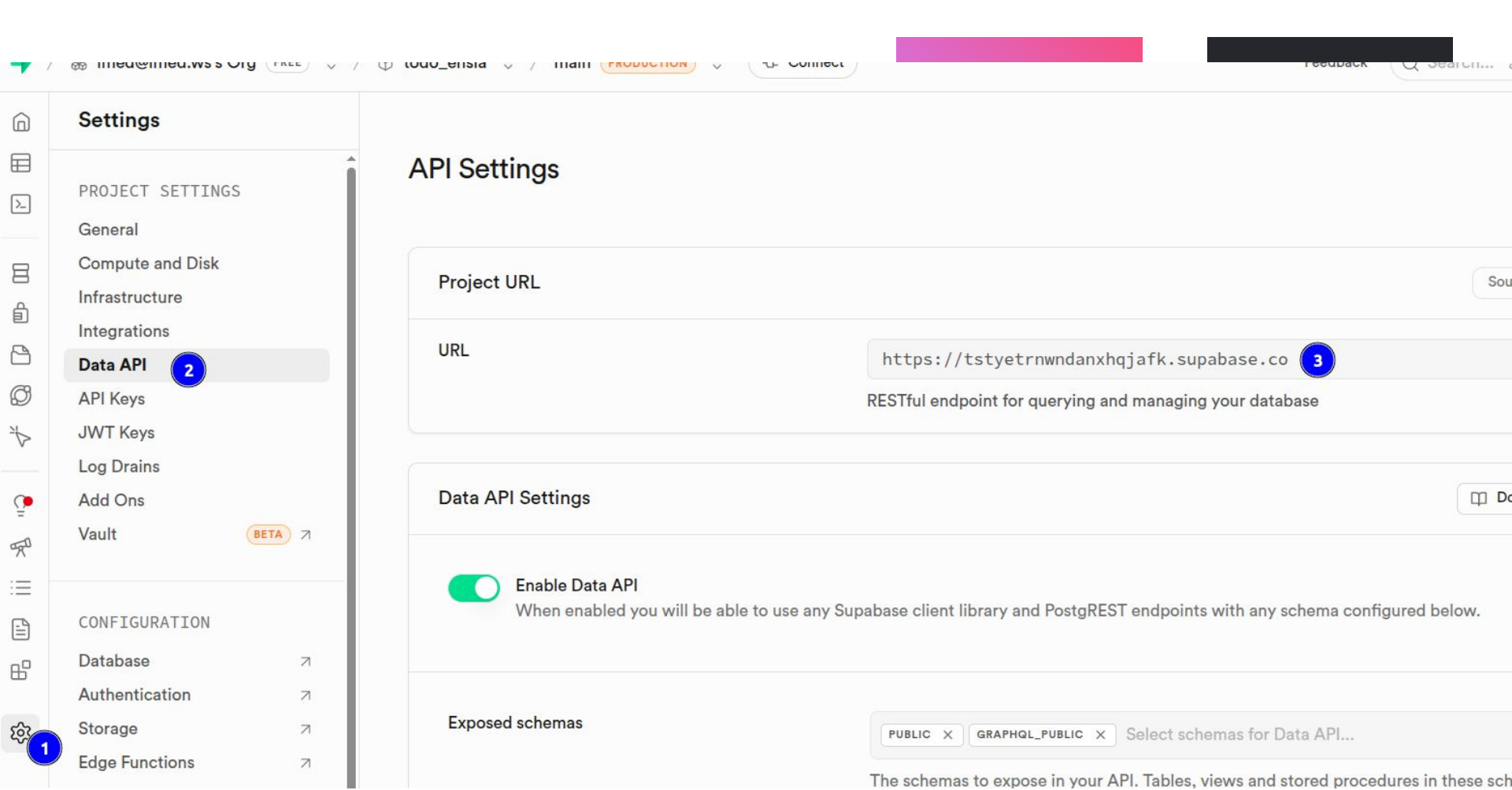
External Servic
Supabase / Relat

- **Steps to get Started :**
  1. Signup at www.supabase.co
  2. Create a Project and set a Pa password at a safe place
  3. **Open the Table Editor Create Tables, insert data...**



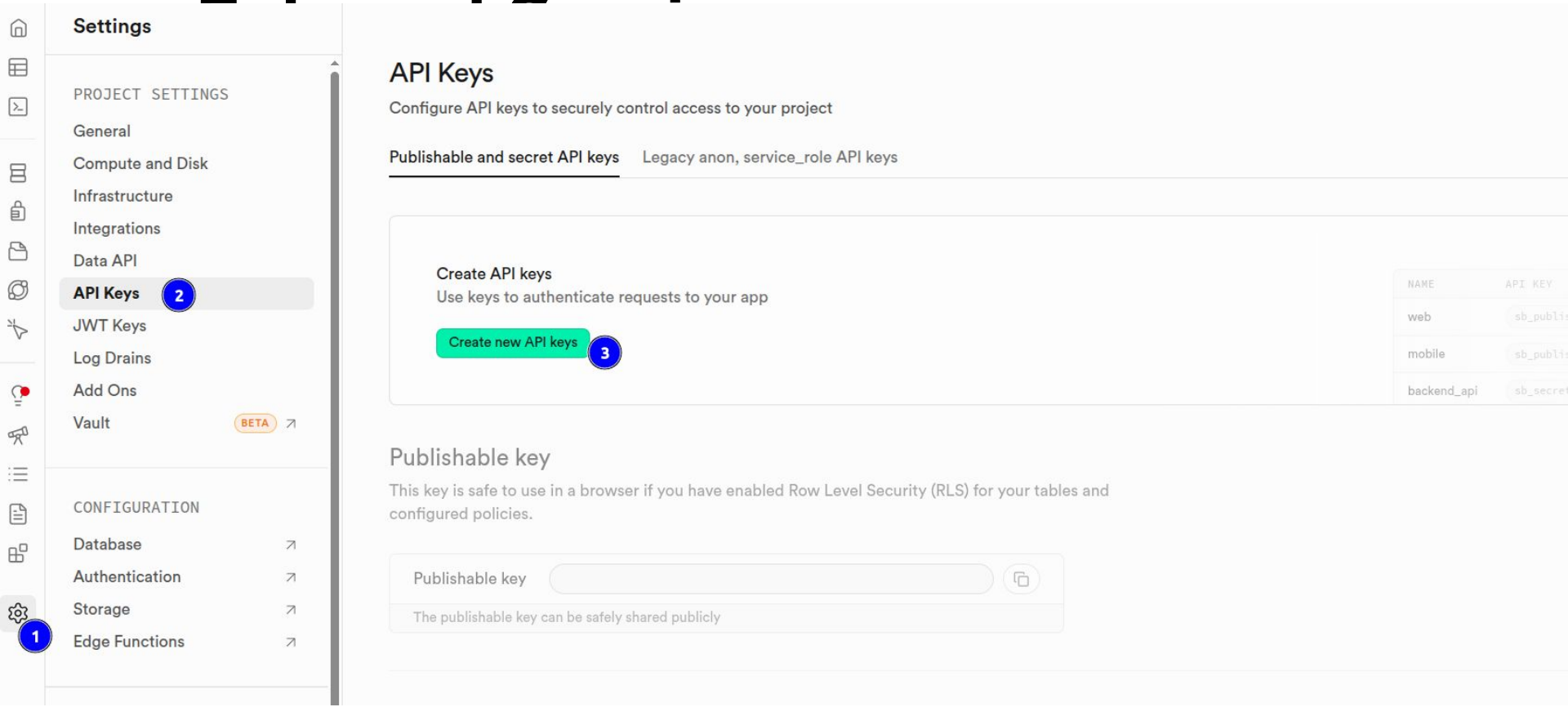
External Services: Supabase / Relational

- **Steps to get Started :**
  1. Signup at www.supabase.com
  2. Create a Project and set a Password, Please keep the password safe..
  3. Open the Table Editor Create Tables, insert data...
  4. **Copy the URL for the Data API**


Settings
PROJECT SETTINGS
General
Compute and Disk
Infrastructure
Integrations
Data API
API Keys
JWT Keys
Log Drains
Add Ons
Vault
BETA
CONFIGURATION
Database
Authentication
Storage
Edge Functions
1
2
API Settings
Project URL
URL
https://tstyetrnwndanxhqjafk.supabase.co
3
RESTful endpoint for querying and managing your database
Data API Settings
Enable Data API
When enabled you will be able to use any Supabase client library and PostgREST endpoints with any schema configured below.
Exposed schemas
PUBLIC
GRAPHQL_PUBLIC
Select schemas for Data API...
The schemas to expose in your API. Tables, views and stored procedures in these sch

External Services: Supabase / Relational

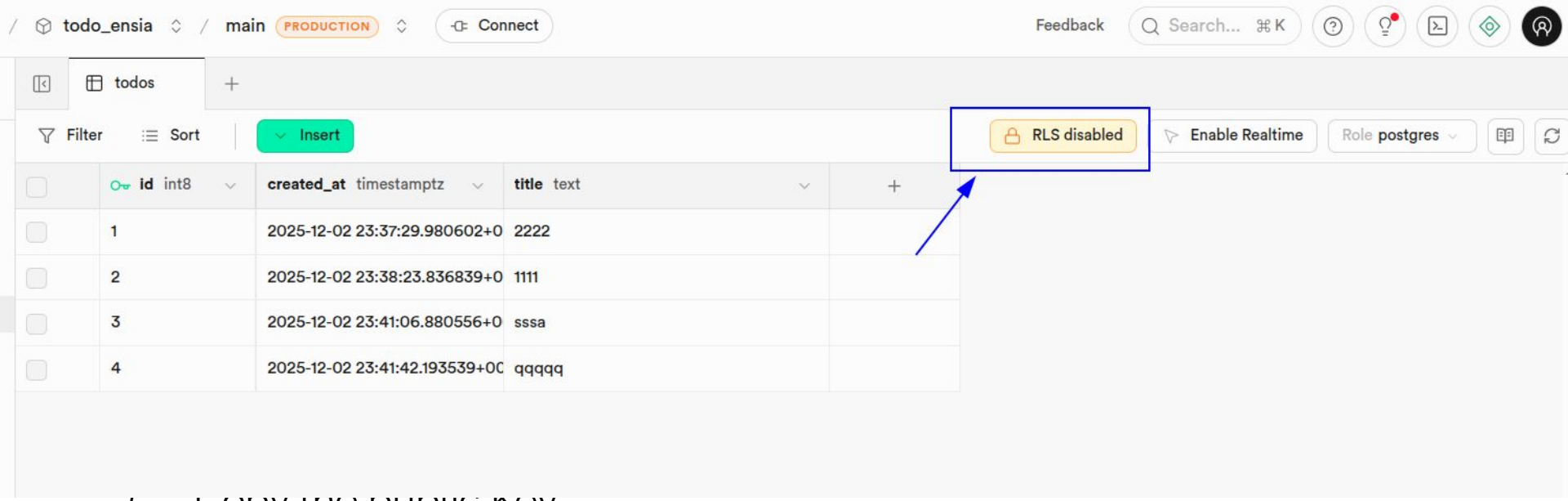
- **Steps to get Started :**
  1. Signup at www.supabase.com
  2. Create a Project and set a Password, Please keep the password safe..
  3. Open the Table Editor Create Tables, insert data...
  4. Copy the URL for the Data API
  5. **Copy the public Key**


Settings
PROJECT SETTINGS
General
Compute and Disk
Infrastructure
Integrations
Data API
API Keys
2
JWT Keys
Log Drains
Add Ons
Vault
BETA
CONFIGURATION
Database
Authentication
Storage
Edge Functions
1
API Keys
Configure API keys to securely control access to your project
Publishable and secret API keys
Legacy anon, service_role API keys
Create API keys
Use keys to authenticate requests to your app
Create new API keys
3
NAME
API KEY
web
mobile
backend_api
Publishable key
This key is safe to use in a browser if you have enabled Row Level Security (RLS) for your tables and configured policies.
Publishable key
The publishable key can be safely shared publicly

External Services: Supabase / Relational

- **Steps to get Started :**
  1. Signup at www.supabase.com
  2. Create a Project and set a Password, Please keep the password safe..
  3. Open the Table Editor Create Tables, insert data...
  4. Copy the URL for the Data API
  5. Copy the public Key
  6. Work on the RLS Policy **(For my case, i have disabled it at my own risk)**

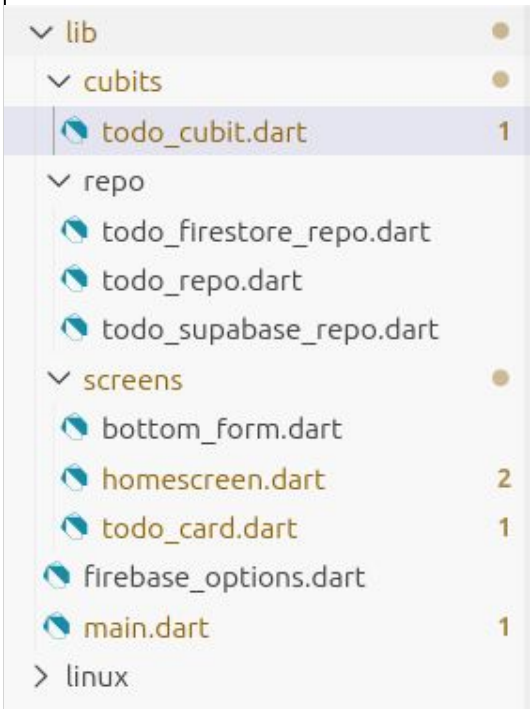
External Services: Supabase / Relational



5. Copy the public key
6. Work on the RLS Policy **(For my case, i have disabled it at my own risk)**

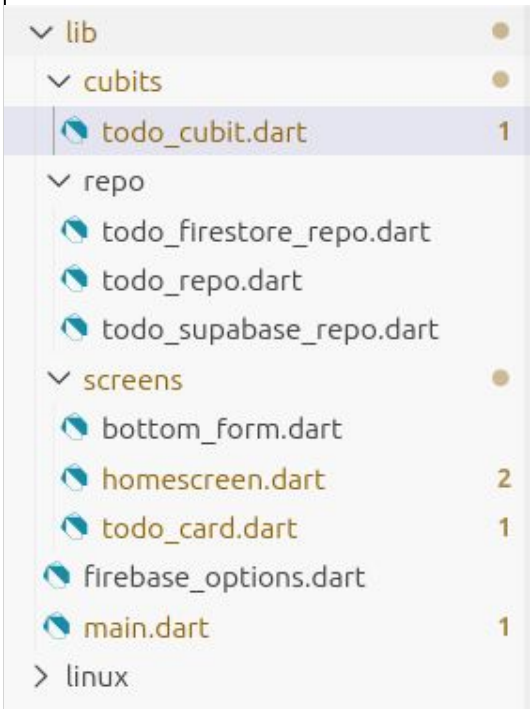
main.dart

```dart
Future<bool> init my app() async {
 final sl = GetIt.instance;
 if (1 == 1) {
   await Supabase.initialize(
     url: 'https://tstyetrnwndanxhqjafk.supabase.co',
     anonKey: 'sb_publishable_Z82M6XDWOxZ2lFdtFapd9Q_TIIP2wab',
   );
   sl.registerLazySingleton<TodoRepo>(() => TodoSupabaseRepo());
 } else {
   sl.registerLazySingleton<TodoRepo>(() => TodoFirestoreRepo());
   await Firebase.initializeApp(
     options: DefaultFirebaseOptions.currentPlatform,
   );
 }
 WidgetsFlutterBinding.ensureInitialized();
 return true;
}
void main() async {
 await init my app();
 runApp(const MainApp());
}
class MainApp extends StatelessWidget {
 const MainApp({super.key});
 @override
 Widget build(BuildContext context) {
   return MultiBlocProvider(
     providers: [BlocProvider(create: (context) => TodoCubit())],
     child: MaterialApp(home: HomeScreen()),
   );
 }
}
```



todo_supabase_repo.dart

```dart
class TodoSupabaseRepo extends TodoRepo {
 Future<List<Map<String, dynamic>>> getData() async {
   try {
     final supabase = Supabase.instance.client;
     List<Map<String, dynamic>> response = await supabase
         .from('todos')
         .select()
         .order('created_at', ascending: false);
     print('_____ $response');
     return response;
   } catch (e, stack) {
     print('Error fetching data from Supabase: $e $stack');
     return [];
   }
 }

 Future<bool> deleteRecord(String id) async {
   return true;
 }

 Future<bool> insertRecord(String title) async {
   final supabase = Supabase.instance.client;
   await supabase.from('todos').insert({'title': title});
   return true;
 }
}
```



External Databases &

Personally, I always prefer access to the DB to be done via an EndPoint Server where extra business logic can be performed. This involves:

- Curation of Data,
- Validation
- Advanced Business Logic (Machine Learning..)
- Communication with Other services.
- Better control of security
- Better extensibility and easy to roll new features
- Easy to change the providers, technologies
  - ? Just the complexity you will be overloading the Endpoint, deploy more instances to overcome the problem

Development Time	Excellent	Takes time

External Databases &

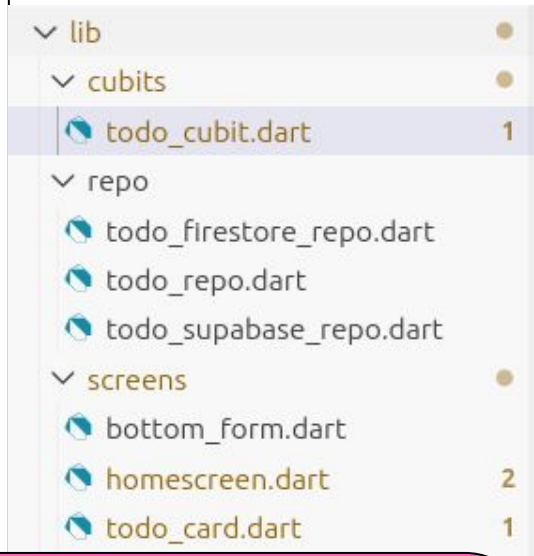
Direct Database Access from your mobile app :

- For simple and disposable apps having no intention to maintain them in the future.
- What happens if you are forced to change the database provider ? or just a database table **even a column name** inside the database table ?
- Security ? it is too complex to enforce and master. What happens if we can declasse your app and find your keys ? we will flood your database with requests and you end up paying a large bill.

Development Time	Excellent	Takes time

```dart
class TodoSupabaseRepo extends TodoRepo {
 Future<List<Map<String, dynamic>>> getData() async {
   try {
     final supabase = Supabase.instance.client;
     List<Map<String, dynamic>> response = await supabase
         .from('todos')
         .select()
         .order('created_at', ascending: false);
     print('_____ $response');
     return response;
   } catch (e, stack) {
     print('Error fetching data from Supabase: $e $stack');
     return [];
   }
 }

 Future<bool> deleteRecord(String id) async {
   return true;
 }

 Future<bool> insertRecord(String title) async {
   final supabase = Supabase.instance.client;
   await supabase.from('todos').insert({'title': title});
   return true;
 }
}
```

todo_supabase_repo.dart



Will not be accepted in the mobile dev project final version

Backends

Backends for a Mobile App

- **What's a Backends :**
  - The component where mobile apps connects to do:
    - Store/Get data
    - Delegate business logic to be conducted
  - Simply,
    - It is like the kitchen where food gets cooked.
  - The User sees the menu (mobile app) and should never be given access to the kitchen where all the business logic is conducted.

Backends for a Mobile App

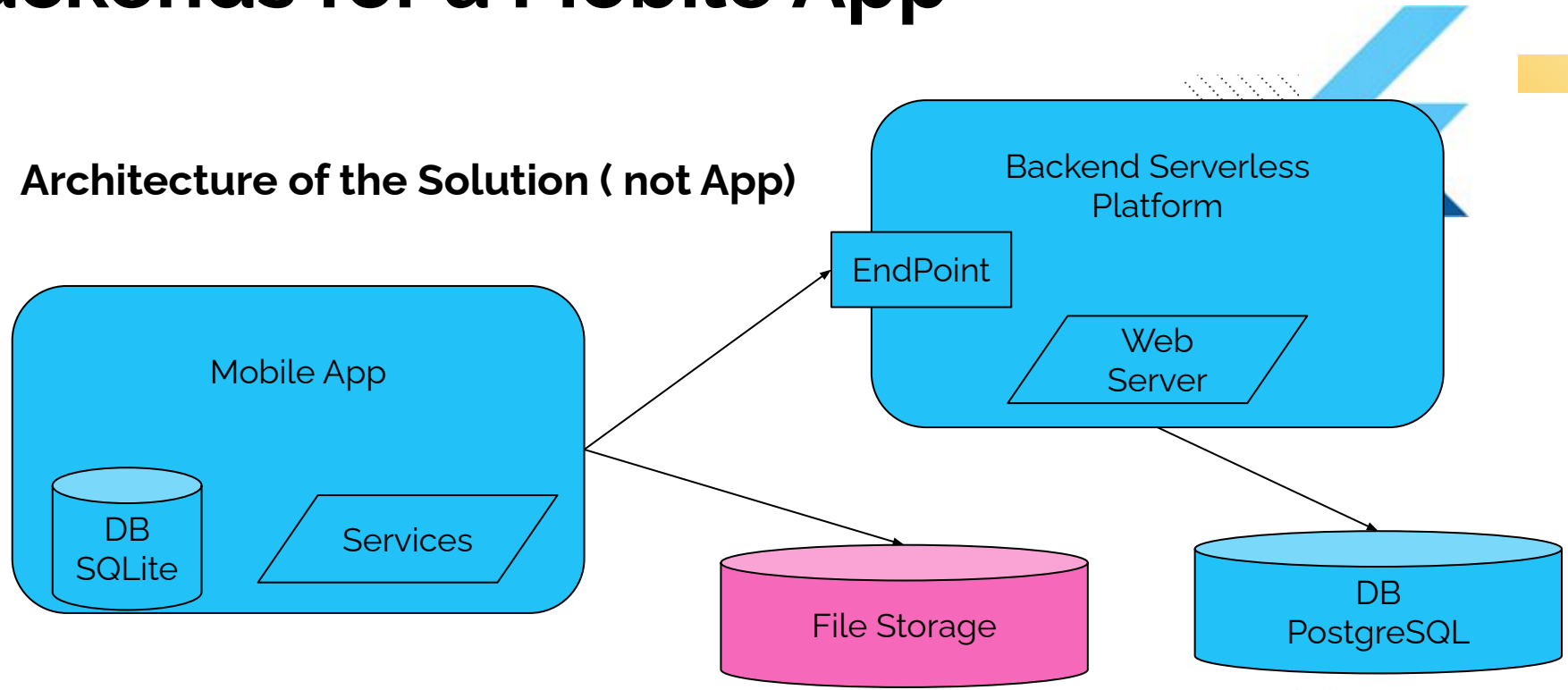
- **Types of Backends :**
  - **Backend-as-a-Service (Baas)**
    - Pre-built backend services that handle common backend needs without writing server code. Think of it as "backend in a box."
  - Examples:
    - Firebase
    - Supase

Backends for a Mobile App

- **Types of Backends :**
  - **Custom Backend**
    - The backend business logic is coded either from scratch or on top of an existing framework including:
      - *Flask / Django*
      - *Node.js + Express*
      - *Laravel*
      - *Spring Boot*
      - *Go*

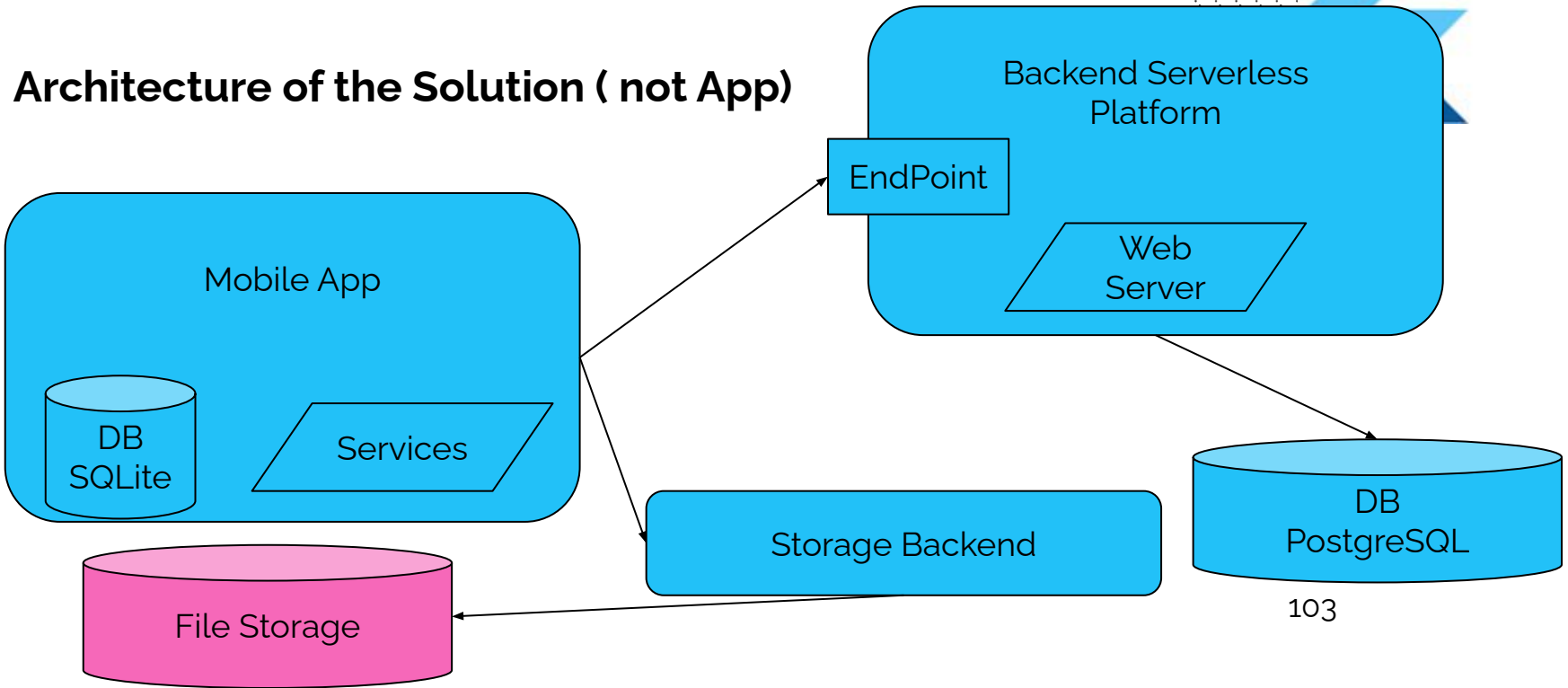
Backends for a Mobile App

- **Architecture of the Solution (not App)**



Serverless Technology for Mobile App Backends

- **Architecture of the Solution (not App)**



Backends for a Mobile App

- **How to run it : Server-Centric approach**
  - Get a dedicated Server or even virtual …
  - You need to manage it yourself
    - Upgrades
    - Security
  - Scaling, can be difficult to main
  - But, you have control over everything.
  - Cost : Cannot say (it may cost $5/month ..)

Backends for a Mobile App

- **Introduction to Serverless Technologies**
  - Even though, it is called serverless, the server is always there but:
    - Provided when there is a request from a user
    - You pay usually per request / processing duration / bandwidth transferred.
  - Serverless Technologies provided by the hosting provider where they have the infrastructure to deploy dynamically servers/frameworks on demand.

Backends for a Mobile App

- **Introduction to Serverless Technologies**
  - **Benefits..**
    - Maintenance of the technology taken care by the infrastructure provider
    - Scaling is always being taken care of. When more users, the hosting provider will deploy more resources dynamically, (but be prepared to pay more)
  - **Drawbacks..**
    - You have no control
    - Dependent on a technology provider

Serverless Technology for Mobile App Backends

- **Improvement to the previous implementation:**
  - Use Python,PHP, Java,,,,for Backend instead :
    - Existing Framework : **Flask** (Or even Django)
    - Link with PostgreSQL provided by Supabase
  - Use Better Storage Facilities : S3 Storage or Firebase Storage
  - Integrate Firebase Services
  - Users' Authentication.

Serverless Technology for Mobile App Backends

- **Hello World in Flask**

```
from flask import Flask
import json

app = Flask(__name__)

@app.route('/categories.get')
def get_categories():
    data=[{'id':1,'name':'Food'},{'id':2,'name':'Beverage'}]
    return json.dumps(data)


@app.route('/')
def index():
    return 'Welcome ENSIA Students from Flask!'

if __name__ == "__main__":
    app.run(port=8080)
```

This is the route which **binds URL** to a function to perform a given business logic

__name__ is a special variable when the script is invoked directly (not imported), the __name is set as "__main__"

Serverless Technology for Mobile App Backends

- **Getting Variables from URLs**

```
from flask import Flask
import json

app = Flask(__name__)

@app.route('/categories.getById/<int:cat_id>')
def get_category_byId(cat_id):
    // some business logic here…
   return json.dumps(data)


@app.route('/')
def index():
   return 'Welcome ENSIA Students from Flask!'

if __name__ == "__main__":
   app.run(port=8080)
```

You can enclose URL variables inside **<type:variable_name>** or simply **<variable_name>**

Serverless Technology for Mobile App Backends

- **Getting GET/POST Variables**

```
from flask import Flask , request
import json

app = Flask(__name__)

@app.route('/user.login', methods=['GET', 'POST'] )
def users_login():
   username=request.form.get('username')
   password=request.form.get('password')
   //some more business logic
   return json.dumps(data)

@app.route('/')
def index():
   return 'Welcome ENSIA Students from Flask!'

if __name__ == "__main__":
   app.run(port=8080)
```

GET or POST variables can be accessed via the special variable :

request.form.get(VAR_NAME)

Serverless Technology for Mobile App Backends

- **Getting Binary Files from Uploads**

```
from flask import Flask
import json

app = Flask(__name__)

@app.route('/photo.upload', methods=['POST'] )
def users_login():
   if request.method=='POST':
         file=request.files.get('file_name')

   //some more business logic
   return json.dumps(data)

@app.route('/')
def index():
   return 'Welcome ENSIA Students from Flask!'

if __name__ == "__main__":
   app.run(port=8080)
```

Binary Data Files can be accessed via the special variable :

request.files.get(FILE_NAME)

Serverless Technology for Mobile App Backends

- **Linking with Flask**

```
from flask import Flask, request
import json
from supabase import create_client, Client

app = Flask(__name__)

url="https://yvnctxteoqxvamhbzpvd.supabase.co"
key="eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJlZiI6Inl2bmN0eHRlb3F4dmFtaGJ6cHZkIiwicm9sZSI6ImFub24iLCJpYXQiOjE3MDIzMDE4ODQsImV4cCI6MjAxNzg3Nzg4NH0.4AoGpxSQF3-T4b_dJ2B5ZfJY1pukT7Gu8xbKq8pN9gA"
supabase: Client = create_client(url, key)

@app.route('/todo.get')
def api_categories_get():
    response = supabase.table('todos').select("*").execute()
    return json.dumps(response.data)
```

Test and Run always Locally FIRST

flask --app api/index run

```python
from flask import Flask, request
import json
import psycopg2

app = Flask(__name__)
#This is the very stupid way to store private/confidential data inside GIT
url="db.yvnctxteoqxvamhbzpvd.supabase.co"
password="Your Initial Password here…"


@app.route('/categories.get')
def api_categories_get():
    conn=False
    try:
        conn = psycopg2.connect("dbname='postgres' user='postgres' host='"+url+"' password='"+password+"'")
    except Exception as error:
        print("I am unable to connect to the database")
        return 'cannot connect to database'+str(error)

    curs=conn.cursor()
    curs.execute("select id,title from todo")
    data=[]
    for record in curs:
        print(record)
        data.append({'id':record[0],'name':record[1]})

    return json.dumps(data)
```

Background Services

Section 3

Case Study : MedBox - Integration with External Services

Case Study : Using External Services

- **List of Doctors in Algeria :**
  - User Story :
    - As an user, i can add a medical record by choosing the doctor, adding a date, upload file.
  - For choosing the doctor ? :
    - We provide the user with a simple text field ?
    - We prefill it most doctors in Algeria
    - We ask the user to fill it on the Go ?