

Version 2.0

Juli, 2024



PEMROGRAMAN MOBILE

Modul 3 Materi - Authentication, Messaging,
and Backend Services

Tim Penyusun:

Ali Sofyan, S.kom, M.kom

Ahmad Naufal

Marchanda Balqis

LAB. INFORMATIKA

Universitas Muhammadiyah Malang

PEMROGRAMAN MOBILE

PERSIAPAN MATERI

Mahasiswa diharapkan mempelajari materi sebelum mengerjakan tugas, materi yang tercakup antara lain:

1. Flutter dan Dart
2. Authentication, Messaging, and Backend Services dengan Firebase

TUJUAN

Mahasiswa diharapkan mampu untuk melakukan dan memahami:

1. Mampu memahami penggunaan Firebase pada Framework Flutter
2. Mampu memahami serta mengimplementasikan penggunaan Firebase Authentication pada aplikasi Framework Flutter
3. Mampu memahami serta mengimplementasikan penggunaan Firebase Messaging pada aplikasi Framework Flutter
4. Mampu memahami serta mengimplementasikan penggunaan Cloud Firestore pada aplikasi Framework Flutter

PERSIAPAN HARDWARE & SOFTWARE

1. Laptop atau PC
2. IDE Android Studio atau Visual Studio Code
3. Flutter SDK: [Flutter 3.16.9](#)

MATERI POKOK

Firestore

Firestore adalah platform pengembangan aplikasi yang dibuat oleh Google yang menyediakan berbagai plugin backend seperti autentikasi, hosting, penyimpanan file, analitik, dan masih banyak lagi. Firestore memudahkan developer untuk membuat aplikasi yang scalable tanpa harus mengelola infrastruktur backend secara manual. Berikut adalah beberapa plugin yang disediakan oleh Firestore:

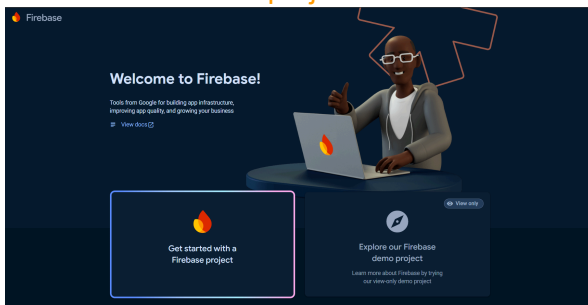
1. **Firestore Authentication:** Memudahkan autentikasi pengguna melalui berbagai metode seperti email dan kata sandi, Google Sign-In, Facebook Login, dll.

2. **Firestore Realtime Database:** Database yang dihosting di cloud yang disimpan sebagai JSON dan disinkronkan secara realtime ke setiap klien yang terhubung.
3. **Cloud Firestore:** Database cloud NoSQL yang fleksibel dan dapat diskalakan untuk menyimpan dan menyinkronkan data.
4. **Firestore Cloud Messaging (FCM):** Layanan untuk mengirimkan notifikasi dan pesan ke perangkat pengguna.
5. **Firestore Analytics:** Alat analitik gratis yang memberikan wawasan tentang penggunaan aplikasi dan interaksi pengguna.
6. **Firestore Hosting:** Hosting web cepat dan aman untuk aplikasi web dan konten statis.
7. **Firestore Cloud Storage:** Penyimpanan file yang dapat diakses secara langsung dari Firestore SDK.
8. **Firestore Crashlytics:** Alat untuk pelaporan crash real-time yang membantu Anda melacak dan memperbaiki masalah aplikasi.
9. **Machine Learning:** Jika menggunakan model TensorFlow Lite khusus, Firestore ML dapat membantu memastikan pengguna selalu menggunakan versi model khusus terbaik yang tersedia.
10. **Performance Monitoring:** Layanan yang membantu mendapatkan wawasan tentang karakteristik kinerja aplikasi iOS, Android, dan web.

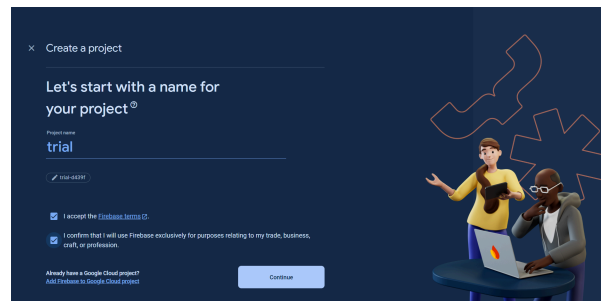
Setup Firebase

Untuk setup Firebase kunjungi [Firestore Console](#) dan login dengan akun Google Anda lalu klik "Mulai" untuk memulai.

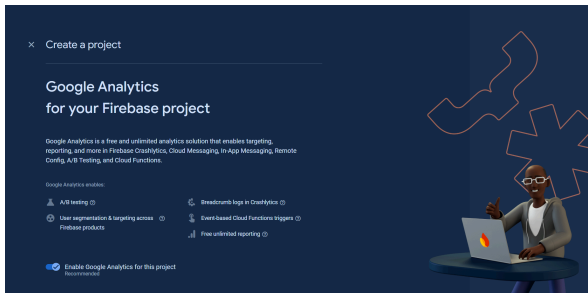
1. Pada halaman Firestore Console, buat project baru dengan memilih **Get started with a Firestore project**



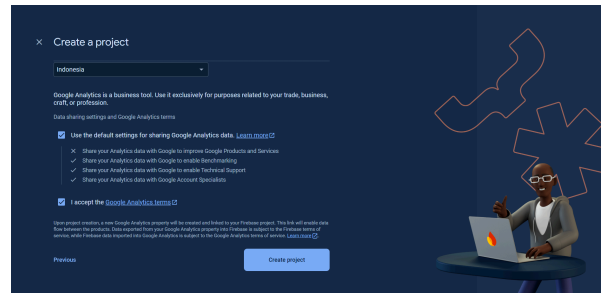
2. Beri nama project. Contohnya seperti **trial**, lalu klik **Continue**



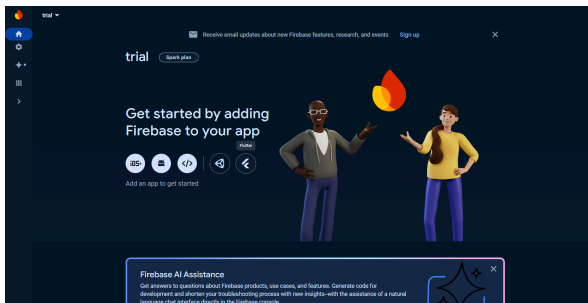
3. Pada langkah konfigurasi Google Analytics, klik **Continue**



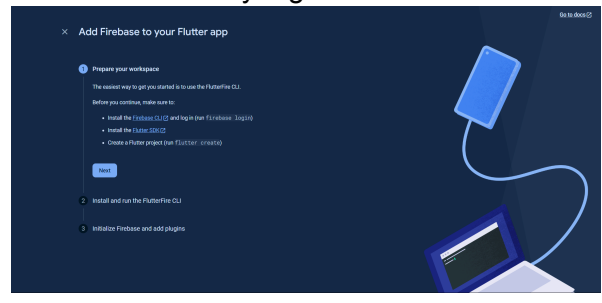
4. Selanjutnya centang checkbox yang tersedia lalu pilih **Create Project**. Tunggu hingga project selesai dibuat dan klik **Continue**



5. Setelah project siap, akan diarahkan ke halaman utama Firebase. Pilih **logo Flutter** untuk memulai integrasi dengan project Flutter Anda



6. Pada halaman ini berisi petunjuk untuk mengintegrasikan Firebase ke dalam project Flutter Anda. Ikuti langkah-langkah yang dijelaskan dalam dokumentasi yang sudah disediakan



7. Untuk mengintegrasikan Firebase ke dalam project Flutter, instal **Firebase CLI**. Pertama, pastikan Anda telah menginstal **Node.js**, lalu buka terminal pada direktori project Flutter dan jalankan:

```
npm install -g firebase-tools
```

Flag **-g** digunakan untuk menginstal Firebase CLI secara global, sehingga dapat digunakan di semua direktori project. Pastikan sudah login dengan akun Google yang sesuai dengan proyek Firebase yang Anda buat sebelumnya. Jika akun yang digunakan tidak sesuai, Anda bisa logout dengan perintah di bawah ini dan kemudian login kembali.

```
firebase logout
```

8. Login kembali ke Firebase dengan akun Google

```
firebase login
```

9. Instal **FlutterFire CLI** untuk mengatur Firebase di project Flutter

```
dart pub global activate flutterfire_cli
```

10. Pada direktori utama project Flutter, lakukan konfigurasi Firebase

```
flutterfire configure
```

11. Setelah itu, biasanya Anda akan diminta untuk menjawab beberapa pertanyaan terkait konfigurasi Firebase pada proyek Flutter

Pertanyaan: Select a Firebase project to configure your Flutter applications with

Sesuaikan dengan nama project yang sudah Anda buat sebelumnya
(**nama_project**)

Pertanyaan: Which platforms should your configuration support?

Pilih platform (**Android, iOS, Web**) dengan menekan tombol spasi untuk memilih dan enter setelah selesai memilih.

12. Tunggu proses instalasi Firebase selesai, jika berhasil pada terminal akan menampilkan keterangan berikut:

```
i Firebase android app com.example.trial is not registered on Firebase project trial-d439f.
i Registered a new Firebase android app on Firebase project trial-d439f.
i Firebase ios app com.example.trial is not registered on Firebase project trial-d439f.
i Registered a new Firebase ios app on Firebase project trial-d439f.
i Firebase macos app com.example.trial registered.
i Firebase web app trial (web) is not registered on Firebase project trial-d439f.
i Registered a new Firebase web app on Firebase project trial-d439f.
i Firebase windows app trial (windows) is not registered on Firebase project trial-d439f.
i Registered a new Firebase windows app on Firebase project trial-d439f.
```

13. Setelah konfigurasi berhasil, beberapa file akan ditambahkan dalam project, seperti:

```
google-service.json
```

File ini berada pada folder **android/app**. Berguna untuk mengkonfigurasi aplikasi Android agar dapat

	terhubung dengan Firebase.
<code>GoogleService-Info.plist</code>	File ini terletak di dalam folder <code>ios/Runner</code> . Berguna untuk mengkonfigurasi aplikasi iOS agar dapat terhubung dengan Firebase.
<code>firebase_app_is_file.json</code>	File ini terletak di dalam folder <code>ios</code> . Berguna sebagai sumber informasi Firebase untuk aplikasi iOS.
<code>firebase_options.dart</code>	File ini terletak di dalam folder <code>lib</code> . Berfungsi sebagai kelas yang mendefinisikan konfigurasi Firebase di dalam project Flutter.

14. Setelah itu lakukan konfigurasi Android, buka file `build.gradle` pada folder `android/app` lalu ubah konfigurasi `minSdkVersion`.

```
android {
    defaultConfig {
        //...
        minSdkVersion 19
    }
}
```

15. Tambahkan plugin Google Services, buka file `build.gradle` pada folder `android` pastikan repository Maven Google sudah ditambahkan, dan tambahkan baris berikut di dependencies:

```
buildscript {
    repositories {
        // Periksa apakah sudah menyertakan Maven Google atau tidak.
        google() // Google's Maven repository
    }
    dependencies {
        // ...
        // Tambahkan baris di bawah ini untuk menyertakan plugin Google
        Services.
        classpath 'com.google.gms:google-services:4.3.13' // Google Services
        plugin
    }
}

allprojects {
    // ...
    repositories {
        // Periksa apakah sudah menyertakan Maven Google atau tidak.
        google() // Google's Maven repository
    }
}
```

```

    // ...
  }
}

```

16. Aktifkan plugin Google Services, buka file build.gradle pada folder **android/app** tambahkan code berikut:

```



apply plugin: 'com.android.application'
// Tambahkan baris di bawah ini untuk menerapkan plugin Google Services.
apply plugin: 'com.google.gms.google-services' // Google Services plugin

android {
    // ...
}

```

Instalasi Plugin

Menambahkan plugin **firebase_core** pada file **pubspec.yaml**

- Buka file **pubspec.yaml** pada project
 -  nama_project
 -  pubspec.yaml
- Tambahkan path direktori pada **pubspec.yaml**. Ubah dari code sebelum (kiri) menjadi sesudah (kanan) lalu save

```

dependencies:
  flutter:
    sdk: flutter
  cupertino_icons: ^1.0.2

```

```

dependencies:
  flutter:
    sdk: flutter
  # Tambahkan ini
  firebase_core: ^2.15.1
  cupertino_icons: ^1.0.2

```

- Setelah plugin **firebase_core** ditambahkan, project Flutter sudah siap untuk menggunakan Firebase
- Tambahkan kode inisialisasi Firebase di dalam fungsi **main()** pada file **main.dart**

```

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

```

- Setelah semua langkah di atas selesai, jalankan project Flutter seperti biasa

Firestore Authentication

[Firestore Authentication](#) adalah layanan dari Firestore yang memungkinkan Anda untuk menambahkan sistem autentikasi pengguna ke dalam aplikasi dengan cara yang cepat dan aman. Layanan ini memudahkan proses pendaftaran, login, dan pengelolaan akun pengguna. Firestore Authentication menyediakan berbagai metode autentikasi yang dapat digunakan untuk memenuhi kebutuhan aplikasi seperti email dan sandi, Google, Facebook, Twitter, dll [FlutterFire-Auth Documentation](#). Fitur utama Firestore Authentication:

- Pendaftaran (Sign Up):** Pengguna dapat membuat akun baru dengan menggunakan email dan kata sandi. Alternatif lainnya adalah dengan menggunakan akun media sosial yang sudah ada, seperti Google, Facebook, atau Twitter.
- Masuk (Sign In):** Pengguna yang sudah memiliki akun dapat masuk ke aplikasi dengan menggunakan informasi autentikasi yang benar (misalnya, email dan kata sandi).
- Pemulihan Kata Sandi (Password Reset):** Jika pengguna lupa kata sandi mereka, Firestore Authentication menyediakan fitur untuk mereset kata sandi melalui email.
- Verifikasi Email:** Setelah pengguna mendaftar, Anda dapat mengirimkan email verifikasi untuk memastikan alamat email mereka valid dan aktif.
- Otentikasi Dua Faktor (Two-Factor Authentication):** Untuk meningkatkan keamanan, Firestore Authentication mendukung otentikasi dua faktor. Ini menambahkan lapisan tambahan dengan meminta pengguna untuk memasukkan kode tambahan yang dikirimkan ke perangkat mereka.

Metode authentication yang dapat digunakan:

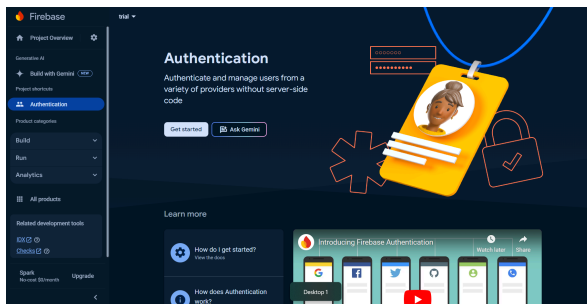
- Email dan Kata Sandi (Email and Password):** Pengguna dapat mendaftar dan masuk menggunakan alamat email dan kata sandi yang mereka buat.
- Google Sign-In:** Pengguna dapat menggunakan akun Google mereka untuk autentikasi.
- Facebook Login:** Pengguna dapat menggunakan akun Facebook mereka untuk autentikasi.
- Twitter Login:** Pengguna dapat menggunakan akun Twitter mereka untuk autentikasi.
- Autentikasi Nomor Telepon (Phone Number Authentication):** Pengguna dapat mendaftar dan masuk dengan menggunakan nomor telepon mereka. Firestore akan mengirimkan kode verifikasi melalui SMS.

6. **Autentikasi Kustom (Custom Authentication):** Jika Anda sudah memiliki sistem autentikasi yang ada, Anda dapat mengintegrasikan sistem tersebut dengan Firebase Authentication menggunakan metode autentikasi kustom.

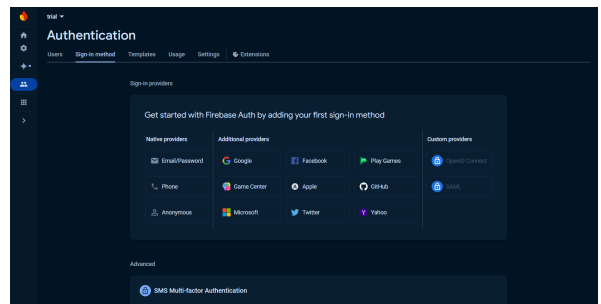
Setup Authentication Email dan Password

Untuk mengaktifkan metode autentikasi pada project Firebase, ikuti langkah-langkah berikut:

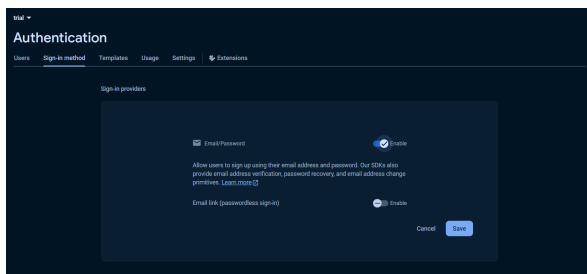
1. Klik menu **build** pada navbar lalu pilih **authentication**. Setelah muncul seperti gambar dibawah klik **Get Started**



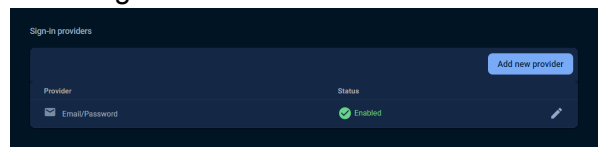
2. Setelah itu pada **tab Sign-in method** ada beberapa metode authentication yang dapat digunakan. Pilih metode **Email/Password**



3. **Enable** switch yang pertama, lalu **Save**





4. Provider/metode lain dapat ditambahkan dengan klik **Add New Provider**



Instalasi Firebase Authentication Plugin

Sebelum mengimplementasikan authentication dalam project Flutter, perlu menambahkan plugin `firebase_auth` ke dalam project.

1. Buka file `pubspec.yaml` pada project
 -  nama_project
 -  `pubspec.yaml`
2. Tambahkan path direktori pada `pubspec.yaml`. Ubah dari code sebelum (kiri) menjadi sesudah (kanan) lalu save

```
dependencies:
  flutter:
    sdk: flutter
  cupertino_icons: ^1.0.2
```

```
dependencies:
  flutter:
    sdk: flutter

  firebase_core: ^2.15.1
  # Tambahkan ini
  firebase_auth: ^4.7.3

  cupertino_icons: ^1.0.2
```

3. Setelah plugin `firebase_auth` ditambahkan, Firebase Authentication sudah dapat digunakan

Register Email dan Password

Setelah berhasil menginstal plugin `firebase_auth`, selanjutnya membuat fitur registrasi pengguna.

1. Buat file baru dengan nama `auth_controller.dart`. File ini akan digunakan untuk mengatur proses authentication menggunakan state management GetX. Pada contoh code berikut metode `createUserWithEmailAndPassword` digunakan untuk membuat akun baru dengan email dan kata sandi yang diinputkan oleh pengguna.

```
class AuthController extends GetxController {
  final FirebaseAuth _auth = FirebaseAuth.instance;
  RxBool isLoading = false.obs;

  Future<void> registerUser(String email, String password) async {
    try {
      isLoading.value = true;
      await _auth.createUserWithEmailAndPassword(
        email: email,
        password: password,
```

```

    );
    Get.snackbar('Success', 'Registration successful',
        backgroundColor: Colors.green);
    Get.off(LoginPage()); // Navigasi ke halaman Login
  } catch (error) {
    Get.snackbar('Error', 'Registration failed: $error',
        backgroundColor: Colors.red);
  } finally {
    isLoading.value = false;
  }
}
}
}

```

2. Selanjutnya, buat tampilan halaman registrasi dengan membuat file `register_page.dart`. Berikut adalah kode contoh untuk `register_page.dart`:

```

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

class _RegisterPageState extends State<RegisterPage> {
  final AuthController _authController = Get.put(AuthController());
  final TextEditingController _emailController = TextEditingController();
  final TextEditingController _passwordController = TextEditingController();

  @override
  void dispose() {
    _emailController.dispose();
    _passwordController.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Register'),
      ),
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          crossAxisAlignment: CrossAxisAlignment.center,
          children: [

```

```

TextField(
  controller: _emailController,
  decoration: InputDecoration(labelText: 'Email'),
),
TextField(
  controller: _passwordController,
  obscureText: true,
  decoration: InputDecoration(labelText: 'Password'),
),
 SizedBox(height: 16),
  Obx(() {
    return ElevatedButton(
      onPressed: _authController.isLoading.value
        ? null
        : () {
            _authController.registerUser(
              _emailController.text,
              _passwordController.text,
            );
          },
      child: _authController.isLoading.value
        ? CircularProgressIndicator()
        : Text('Register'),
    );
  }),
],
),
),
);
}
}

```

3. Setelah berhasil melakukan registrasi, Anda dapat memverifikasi pengguna baru di Firebase Console. Pengguna yang terdaftar akan muncul di daftar pengguna pada halaman Authentication di Firebase.

Login Email dan Password

Setelah berhasil mengimplementasikan fitur registrasi, langkah berikutnya adalah membuat fitur login. Fitur ini memungkinkan pengguna yang sudah terdaftar untuk masuk ke dalam aplikasi menggunakan email dan password yang sudah mereka daftarkan.

1. Membuat fungsi login pada `auth_controller.dart`

Dalam file `auth_controller.dart`, kita perlu menambahkan fungsi baru untuk proses login. Fungsi ini akan memverifikasi kredensial yang dimasukkan oleh pengguna melalui Firebase Authentication.

```
class AuthController extends GetxController {
  final FirebaseAuth _auth = FirebaseAuth.instance;
  RxBool isLoading = false.obs;

  // Fungsi untuk registrasi pengguna
  Future<void> registerUser(String email, String password) async {
    try {
      isLoading.value = true;
      await _auth.createUserWithEmailAndPassword(
        email: email,
        password: password,
      );
      Get.snackbar('Success', 'Registration successful',
        backgroundColor: Colors.green);
      Get.off(LoginPage()); // Beralih ke halaman Login setelah registrasi
      berhasil
    } catch (error) {
      Get.snackbar('Error', 'Registration failed: $error',
        backgroundColor: Colors.red);
    } finally {
      isLoading.value = false;
    }
  }

  // Fungsi untuk Login pengguna
  Future<void> loginUser(String email, String password) async {
    try {
      isLoading.value = true;
      await _auth.signInWithEmailAndPassword(
        email: email,
        password: password,
      );
      Get.snackbar('Success', 'Login successful',
        backgroundColor: Colors.green);
    }
  }
}
```

```

    } catch (error) {
      Get.snackbar('Error', 'Login failed: $error',
        backgroundColor: Colors.red);
    } finally {
      isLoading.value = false;
    }
  }
}

```

Ketika pengguna mencoba login, fungsi `signInWithEmailAndPassword()` akan dipanggil untuk memvalidasi email dan password yang dimasukkan. Jika login berhasil, pengguna akan mendapatkan notifikasi sukses, sedangkan jika gagal, notifikasi kesalahan akan ditampilkan.

- Setelah fungsi login, langkah berikutnya adalah membuat tampilan halaman login. Di sini, pengguna akan memasukkan email dan password mereka. Buat file `login_page.dart` untuk membuat tampilan halaman login.

```

class LoginPage extends StatefulWidget {
  @override
  State<LoginPage> createState() => _LoginPageState();
}

class _LoginPageState extends State<LoginPage> {
  final AuthController _authController = Get.put(AuthController());
  final TextEditingController _emailController = TextEditingController();
  final TextEditingController _passwordController = TextEditingController();

  @override
  void dispose() {
    _emailController.dispose();
    _passwordController.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Login'),
      ),
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,

```

```

crossAxisAlignment: CrossAxisAlignment.center,
children: [
  TextField(
    controller: _emailController,
    decoration: InputDecoration(labelText: 'Email'),
  ),
  TextField(
    controller: _passwordController,
    obscureText: true,
    decoration: InputDecoration(labelText: 'Password'),
  ),
  SizedBox(height: 16),
  Obx(() {
    return ElevatedButton(
      onPressed: _authController.isLoading.value
        ? null
        : () {
            _authController.loginUser(
              _emailController.text,
              _passwordController.text,
            );
          },
      child: _authController.isLoading.value
        ? CircularProgressIndicator()
        : Text('Login'),
    );
  }),
],
),
);
}
}

```

Logout

Setelah menyelesaikan fitur registrasi dan login, langkah berikutnya adalah menambahkan fungsi logout agar pengguna bisa keluar dari akun mereka.

1. Tambahkan Fungsi Logout pada file `auth_controller.dart`, tambahkan sebuah fungsi bernama `logout` untuk menangani proses keluar dari akun.

```
class AuthController extends GetxController {
  // ...
  void logout() async {
    await _auth.signOut();
    Get.offAll(LoginPage()); // Menghapus semua halaman dari stack dan
    kembali ke halaman Login.
  }
}
```

Fungsi `logout` ini memanfaatkan metode `signOut` dari Firebase untuk mengeluarkan pengguna dari aplikasi. Setelah proses `logout` berhasil, kita menggunakan `Get.offAll()` untuk membersihkan semua halaman sebelumnya dan mengarahkan pengguna kembali ke halaman `login`.



2. Selanjutnya, buat tombol di halaman yang diinginkan untuk memanggil fungsi `logout`.

```
class HomePage extends StatelessWidget {
  final AuthController _authController = Get.put(AuthController());

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Home'),
        actions: [
          IconButton(
            icon: Icon(Icons.exit_to_app),
            onPressed: () {
              _authController.logout(); // Panggil fungsi logout ketika
              tombol ditekan.
            },
          ),
        ],
      ),
    );
  }
}
```


Shared Preferences

Shared Preferences adalah sebuah metode untuk menyimpan dan mengambil data sederhana dalam bentuk pasangan kunci-nilai pada penyimpanan lokal perangkat di aplikasi Flutter. Metode ini sering digunakan untuk menyimpan informasi seperti pengaturan aplikasi, preferensi pengguna, atau data kecil lain yang perlu dipertahankan antara sesi penggunaan aplikasi. Dengan SharedPreferences, Anda bisa menyimpan data dengan tipe seperti int, double, bool, dan string secara mudah dan cepat tanpa memerlukan database. SharedPreferences sangat berguna, misalnya, untuk menyimpan status login pengguna atau data lain yang berkaitan dengan pengguna setelah login. Untuk menggunakan SharedPreferences, perlu menambahkan plugin `shared_preferences` pada file `pubspec.yaml`.

1. Buka file `pubspec.yaml` pada project
 -  nama_project
 -  `pubspec.yaml`
2. Tambahkan path direktori pada `pubspec.yaml`. Ubah dari code sebelum (kiri) menjadi sesudah (kanan) lalu save

```
dependencies:
  flutter:
    sdk: flutter
  cupertino_icons: ^1.0.2
```

```
dependencies:
  flutter:
    sdk: flutter

  firebase_core: ^2.15.1
  firebase_auth: ^4.7.3
  # Tambahkan ini
  shared_preferences: ^2.2.0

  cupertino_icons: ^1.0.2
```

3. Setelah menyimpan perubahan, `shared_preferences` sudah dapat digunakan

Implementasi Shared Preferences pada Sistem Login

Dalam sistem login, Shared Preferences digunakan untuk menyimpan status login pengguna, sehingga pengguna tidak perlu login ulang setiap kali membuka aplikasi.

1. Pada file `auth_controller.dart`, tambahkan Shared Preferences untuk menyimpan token autentikasi pengguna, dan tambahkan juga fungsi `checkLoginStatus()` untuk memeriksa status login saat aplikasi dibuka

```
class AuthController extends GetxController {
  final FirebaseAuth _auth = FirebaseAuth.instance;
  final SharedPreferences _prefs = Get.find<SharedPreferences>();
  RxBool isLoading = false.obs;
  RxBool isLoggedIn = false.obs;

  @override
  void onInit() {
    super.onInit();
    checkLoginStatus(); // Periksa status login saat controller
diinisialisasi
  }

  Future<void> checkLoginStatus() async {
    isLoggedIn.value = _prefs.containsKey('user_token');
  }

  // Fungsi untuk registrasi pengguna
  // ...

  Future<void> loginUser(String email, String password) async {
    try {
      isLoading.value = true;
      await _auth.signInWithEmailAndPassword(
        email: email,
        password: password,
      );
      _prefs.setString('user_token', _auth.currentUser!.uid); // Simpan
token pengguna
      Get.snackbar('Success', 'Login berhasil',
        backgroundColor: Colors.green);
      isLoggedIn.value = true; // Set status login menjadi true
      Get.offAllNamed('/home'); // Navigasi ke halaman Home dan hapus semua
halaman sebelumnya
    } catch (error) {
      Get.snackbar('Error', 'Login gagal: $error',
        backgroundColor: Colors.red);
    }
  }
}
```

```

    } finally {
      isLoading.value = false;
    }
  }

  void logout() {
    _prefs.remove('user_token');
    isLoggedIn.value = false;
    _auth.signOut();
    Get.offAllNamed('/login');
  }
}

```

2. Inisialisasi Shared Preferences di fungsi `main()` pada file `main.dart` agar bisa digunakan di seluruh aplikasi

```

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform,
  );
  await Get.putAsync(() async => await SharedPreferences.getInstance());
  runApp(MyApp());
}

```

3. Pada class `MyApp` dalam `main.dart`, atur navigasi aplikasi berdasarkan status login pengguna. Dengan menggunakan `initialRoute`, Anda dapat menentukan apakah pengguna langsung masuk ke halaman `HomePage` atau `LoginPage` ketika aplikasi pertama kali dibuka

```

class MyApp extends StatelessWidget {
  MyApp({super.key});
  final AuthController _authController = Get.put(AuthController());

  @override
  Widget build(BuildContext context) {
    return GetMaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        useMaterial3: true,
      ),
      initialRoute: _authController.isLoggedIn.value ? '/home' : '/login',
      getPages: [

```

```

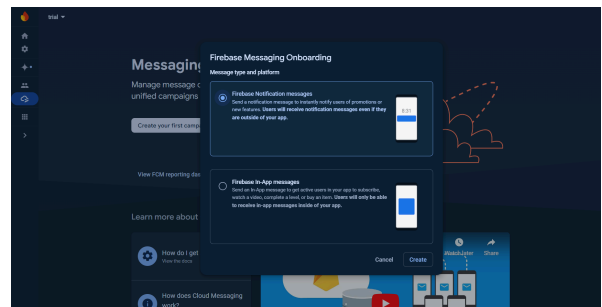
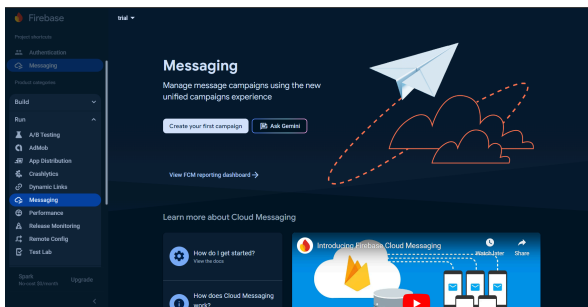
    GetPage(name: '/login', page: () => LoginPage()),
    GetPage(name: '/home', page: () => HomePage()),
  ],
);
}
}

```

Firestore Messaging

[Firestore Cloud Messaging \(FCM\)](#) adalah layanan dari Firebase yang memungkinkan Anda untuk mengirimkan notifikasi dan pesan langsung ke perangkat pengguna. Layanan ini cocok untuk mengelola komunikasi antara server dan klien dalam aplikasi mobile atau web. FCM mendukung pengiriman notifikasi push serta pesan data, dengan berbagai fitur yang memungkinkan untuk mengatur dan menyesuaikan cara pengiriman notifikasi dan pesan tersebut.

1. Masuk ke **Firestore Console**, pilih menu **run** pada sidebar lalu **messaging**. Setelah itu, Klik **Create Your First Campaign** untuk memulai
2. Akan muncul 2 opsi pengiriman pesan. Pada tutorial ini pilih **Firestore Notification Messages**. Setelah memilih, akan diarahkan ke halaman **Compose Notification** di mana kita bisa membuat dan menyesuaikan notifikasi yang ingin dikirimkan



Installation

Untuk mulai menggunakan Firestore Messaging, kita perlu menambahkan plugin **firebase_messaging** ke dalam **pubspec.yaml**.

1. Buka file **pubspec.yaml** pada project
 - nama_project
 - pubspec.yaml

2. Tambahkan path direktori pada `pubspec.yaml`. Ubah dari code sebelum (kiri) menjadi sesudah (kanan) lalu save

<pre>dependencies: flutter: sdk: flutter cupertino_icons: ^1.0.2</pre>	<pre>dependencies: flutter: sdk: flutter firebase_core: ^2.15.1 firebase_auth: ^4.7.3 shared_preferences: ^2.2.0 # Tambahkan ini firebase_messaging: ^14.6.6 cupertino_icons: ^1.0.2</pre>
--	--

3. Setelah menyimpan perubahan, `firebase_messaging` sudah dapat digunakan

Handler Background dan Terminated Message

1. Buat file baru bernama `notification_handler.dart` yang akan digunakan untuk menangani pesan yang diterima ketika aplikasi berjalan di background atau terminated

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

Future<void> _firebaseMessagingBackgroundHandler(RemoteMessage message)
async {
  print('Pesan diterima di background: ${message.notification?.title}');
}

class FirebaseMessagingHandler {
  final FirebaseMessaging _firebaseMessaging = FirebaseMessaging.instance;

  Future<void> initPushNotification() async {
    NotificationSettings settings = await
    _firebaseMessaging.requestPermission(
      alert: true,
      announcement: false,
      badge: true,
      carPlay: false,
      criticalAlert: false,
      provisional: false,
      sound: true,
    );

    print('Izin yang diberikan pengguna: ${settings.authorizationStatus}');
```

```

// Mendapatkan token FCM
_firebaseMessaging.getToken().then((token) {
  print('FCM Token: $token');
});

// Saat aplikasi dalam keadaan terminated
FirebaseMessaging.instance.getInitialMessage().then((message) {
  print("Pesan saat aplikasi terminated:
${message!.notification?.title}");
});

// Saat aplikasi dalam keadaan background
FirebaseMessaging.onBackgroundMessage(_firebaseMessagingBackgroundHandler);
}
}

```

2. Setelah membuat handler, panggil fungsi `initPushNotification()` di dalam fungsi `main()` pada file `main.dart`

```

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform,
  );
  await Get.putAsync(() async => await SharedPreferences.getInstance());
  await FirebaseMessagingHandler().initPushNotification();
  runApp(const MyApp());
}

```

Handler Foreground Message

Handler foreground message digunakan untuk membuat notifikasi ketika aplikasi berjalan dalam kondisi foreground atau aplikasi sedang digunakan. Dalam kondisi ini, notifikasi yang datang perlu ditangani secara khusus karena tidak akan secara otomatis muncul sebagai banner notifikasi seperti ketika aplikasi berada di background atau terminated.



Untuk menangani notifikasi di foreground, kita perlu menggunakan Firebase Messaging bersama Flutter Local Notifications. Firebase Messaging menangkap pesan yang masuk, sementara Flutter Local Notifications digunakan untuk menampilkan notifikasi tersebut di perangkat.

Flutter Local Notifications

Flutter Local Notifications adalah library yang digunakan untuk mengelola dan menampilkan notifikasi langsung pada perangkat pengguna. Library ini memungkinkan aplikasi untuk memberikan notifikasi tanpa perlu bergantung pada server eksternal seperti Firebase.

Installation

Untuk menginstal Flutter Local Notifications, tambahkan dependensi ke dalam file `pubspec.yaml`.

1. Buka file `pubspec.yaml` pada project
 -  nama_project
 -  `pubspec.yaml`
2. Tambahkan path direktori pada `pubspec.yaml`. Ubah dari code sebelum (kiri) menjadi sesudah (kanan) lalu save

```
dependencies:
  flutter:
    sdk: flutter
  cupertino_icons: ^1.0.2
```

```
dependencies:
  flutter:
    sdk: flutter

  firebase_core: ^2.15.1
  firebase_auth: ^4.7.3
  shared_preferences: ^2.2.0
  firebase_messaging: ^14.6.6
  # Tambahkan ini
  flutter_local_notifications: ^15.1.0+1

  cupertino_icons: ^1.0.2
```

3. Setelah menyimpan perubahan, `flutter_local_notifications` sudah dapat digunakan

Implementasi Firebase Messaging dan Local Notifications

Untuk menangani notifikasi ketika aplikasi berjalan di foreground, Perlu setup handler dalam file `notification_handler.dart`.

1. Pada file `notification_handler.dart` tambahkan handler yang menangani notifikasi ketika aplikasi berada di kondisi foreground (saat aplikasi sedang aktif). Dilakukan dengan mendefinisikan channel notifikasi untuk Android dan menggunakan `FlutterLocalNotificationsPlugin` dan buat fungsi `initLocalNotification()` untuk menampilkan notifikasi secara lokal.

```
class FirebaseMessagingHandler {
  final FirebaseMessaging _firebaseMessaging = FirebaseMessaging.instance;

  // Inisialisasi kanal notifikasi untuk Android
  final _androidChannel = const AndroidNotificationChannel(
    'channel_notification',
    'High Importance Notification',
    description: 'Used For Notification',
    importance: Importance.defaultImportance,
  );

  // Inisialisasi plugin notifikasi lokal
  final _localNotification = FlutterLocalNotificationsPlugin();

  Future<void> initPushNotification() async {
    // Izin notifikasi dari pengguna
    ...

    // Dapatkan token FCM
    ...

    // Handler untuk notifikasi ketika aplikasi di foreground
    FirebaseMessaging.onMessage.listen((message) {
      final notification = message.notification;
      if (notification == null) return;

      // Tampilkan notifikasi lokal
      _localNotification.show(
        notification.hashCode,
        notification.title,
        notification.body,
        NotificationDetails(
          android: AndroidNotificationDetails(
```



```

        _androidChannel.id,
        _androidChannel.name,
        channelDescription: _androidChannel.description,
        icon: '@drawable/ic_launcher'
    )
  ),
  payload: jsonEncode(message.toMap()),
);
print('Pesan diterima saat aplikasi di foreground:
${message.notification?.title}');
});

// Handler ketika pesan dibuka dari notifikasi
FirebaseMessaging.onMessageOpenedApp.listen((RemoteMessage message) {
  print('Pesan dibuka dari notifikasi: ${message.notification?.title}');
});
}

Future initLocalNotification() async {
  const ios = DarwinInitializationSettings();
  const android = AndroidInitializationSettings('@drawable/ic_launcher');
  const settings = InitializationSettings(android: android, iOS: ios);
  await _localNotification.initialize(settings);
}
}

```

2. Memanggil fungsi `initLocalNotification()` di dalam fungsi `main()` pada aplikasi, Untuk memastikan bahwa local notifications siap digunakan sebelum aplikasi mulai berjalan.

```

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform,
  );
  await Get.putAsync(() async => await SharedPreferences.getInstance());
  await FirebaseMessagingHandler().initPushNotification();
  await FirebaseMessagingHandler().initLocalNotification();
  runApp(const MyApp());
}

```

- Untuk memastikan notifikasi bisa ditampilkan dengan benar di Android, kita perlu menambahkan pengaturan untuk notification channel di dalam file **AndroidManifest.xml**. Ini dilakukan dengan menambahkan **meta-data** di dalam elemen **<application>**.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
  <application
    ...
    <activity
      ...
    </activity>
    <meta-data

      android:name="com.google.firebase.messaging.default_notification_channel_id"
      android:value="channel_notification" />
    ...
  </application>
</manifest>
```

- Setelah semua setup selesai, jalankan aplikasi seperti biasa dan lakukan pengujian notifikasi melalui halaman **Compose Notification** di Firebase, dengan kondisi aplikasi berada di foreground. Pastikan aplikasi berjalan dan notifikasi muncul sesuai dengan yang diharapkan.

Push Local Notification

Kita akan menampilkan notifikasi sederhana menggunakan **flutter_local_notifications**. Notifikasi sederhana ini berisi judul, isi pesan, dan ikon notifikasi yang ditampilkan di perangkat pengguna.

1. Menampilkan Notifikasi Sederhana

Buat fungsi **showNotification** yang akan mengatur semua parameter notifikasi, seperti **AndroidNotificationDetails** untuk Android dan **DarwinNotificationDetails** untuk iOS. Konfigurasi notifikasi menggunakan objek **NotificationDetails** yang menggabungkan pengaturan untuk Android dan iOS. Panggil metode **show()** dari objek **flutterLocalNotificationsPlugin** untuk menampilkan notifikasi.

```
Future<void> showNotification(FlutterLocalNotificationsPlugin
flutterLocalNotificationsPlugin) async {
  var androidPlatformChannelSpecifics = AndroidNotificationDetails(
    _channelId,
    _channelName,
    channelDescription: _channelDesc,
```

```

        importance: Importance.max,
        priority: Priority.high,
        ticker: 'ticker',
    );

    var iOSPlatformChannelSpecifics = DarwinNotificationDetails(); //
    Pengaturan notifikasi untuk iOS

    var platformChannelSpecifics = NotificationDetails(
        android: androidPlatformChannelSpecifics,
        iOS: iOSPlatformChannelSpecifics,
    );

    await flutterLocalNotificationsPlugin.show(
        0, // ID unik untuk notifikasi
        'plain title',
        'plain body',
        platformChannelSpecifics,
        payload: 'plain notification',
    );
}

```

2. Menampilkan Notifikasi dengan Progress Indicator di Android

Notifikasi dengan **progress indicator** sangat berguna saat kita ingin menampilkan status proses yang sedang berjalan, seperti pengunduhan file. Buat loop untuk mensimulasikan progres dari suatu proses, misalnya unduhan. Gunakan parameter **showProgress**, **maxProgress**, dan **progress** di **AndroidNotificationDetails** untuk mengatur tampilan progres di notifikasi. Setiap kali progres diperbarui, panggil **show()** untuk memperbarui tampilan notifikasi.

```

Future<void> showProgressNotification(FlutterLocalNotificationsPlugin
flutterLocalNotificationsPlugin) async {
    var maxProgress = 5; // Jumlah total progres (misal 5 tahap)

    for (var i = 0; i <= maxProgress; i++) {
        await Future.delayed(Duration(seconds: 1), () async {
            var androidPlatformChannelSpecifics = AndroidNotificationDetails(
                _channelId,
                _channelName,
                channelDescription: _channelDesc,
                channelShowBadge: false, // Tidak menampilkan badge (angka) di ikon
                aplikasi
                importance: Importance.max,
                priority: Priority.high,
            );
            flutterLocalNotificationsPlugin.show(
                i,
                'Progress $i',
                'Progress $i',
                androidPlatformChannelSpecifics,
                payload: 'Progress $i',
            );
        });
    }
}

```

```

        onlyAlertOnce: true,
        showProgress: true,
        maxProgress: maxProgress,
        progress: i,
    );

    var platformChannelSpecifics = NotificationDetails(android:
androidPlatformChannelSpecifics);

    await flutterLocalNotificationsPlugin.show(
        0,
        'progress notification title',
        'progress notification body',
        platformChannelSpecifics,
        payload: 'item x',
    );
  });
}
}



```

Firestore Cloud Firestore

[Cloud Firestore](#) adalah layanan database berbasis cloud yang disediakan oleh Firebase, bagian dari Google Cloud Platform. Firestore digunakan untuk menyimpan, menyinkronkan, dan mengelola data dalam aplikasi secara real-time yang merupakan database NoSQL dan menggunakan konsep collections. Lalu, di dalam collections tersebut kita bisa menyimpan documents dan di dalam documents kita bisa menyimpan data.

Instalasi Firestore Cloud Firestore

Sebelum mengimplementasikan cloud firestore dalam project Flutter, perlu menambahkan plugin [cloud firestore](#) ke dalam project dan pastikan firebase telah terinstall pada aplikasi kalian.

1. Buka file **pubspec.yaml** pada project
 -  nama_project
 -  pubspec.yaml
2. Tambahkan path direktori pada **pubspec.yaml**. Ubah dari code sebelum (kiri) menjadi sesudah (kanan) lalu save

```
dependencies:
  flutter:
```

```
dependencies:
  flutter:
```

```

sdk: flutter
cupertino_icons: ^1.0.2

```

```

sdk: flutter
firebase_core: ^2.15.1
firebase_auth: ^4.7.3
shared_preferences: ^2.2.0
flutter_local_notifications: ^15.1.0+1
# Tambahkan ini
cloud_firestore: ^5.4.4
cupertino_icons: ^1.0.2

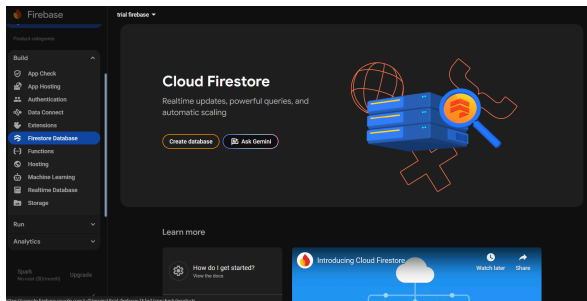
```

- Setelah plugin **cloud firestore** ditambahkan, Firebase Cloud Firestore sudah dapat digunakan

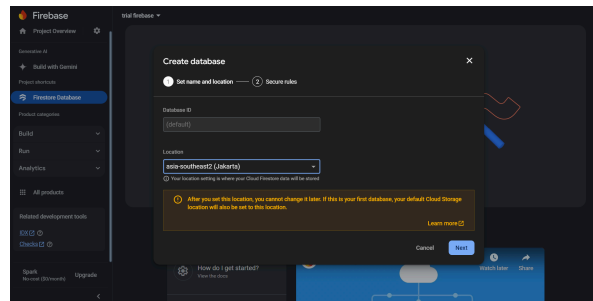
Setup Firebase Cloud Firestore

Untuk setup Firebase Cloud Firestore kunjungi [Firebase Console](#) dan gunakan project yang sudah ada atau buat project baru.

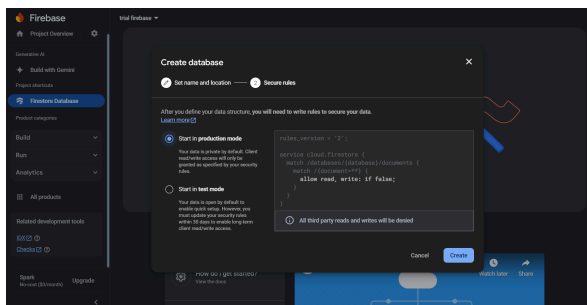
- Klik menu **build** pada navbar lalu pilih **firebase database**. Setelah muncul seperti gambar dibawah klik **Create Database**



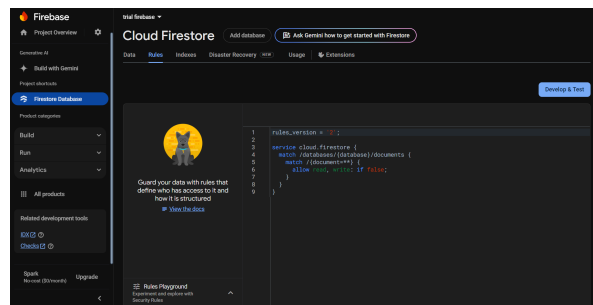
- Setelah itu pilih lokasi yang terdekat dengan lokasi kalian saat membuat database. Pada tutorial kali ini menggunakan **asia-southeast2 (Jakarta)**



- Pilih **Start in Production Mode** lalu klik **create**



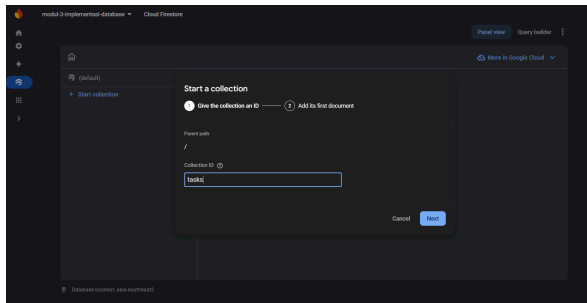
- Jika sudah, masuk ke menu **rules**. Ubah bagian **false** menjadi **true** agar database dapat digunakan lalu klik **publish**



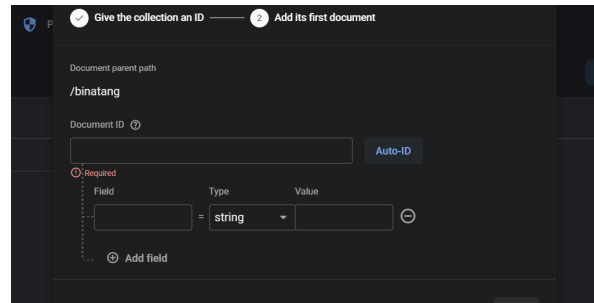
Implementasi Firebase Cloud Firestore

Setelah berhasil menginstal plugin **cloud firestore** selanjutnya membuat collections untuk menyimpan dokumen dan data.

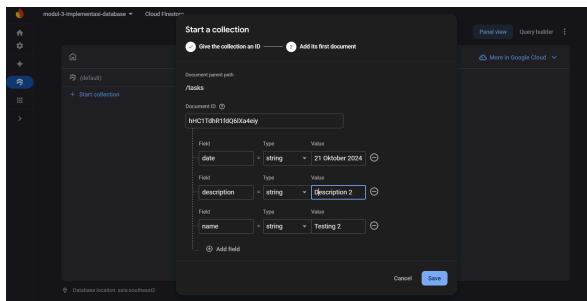
1. Buat collections baru dengan nama **tasks**



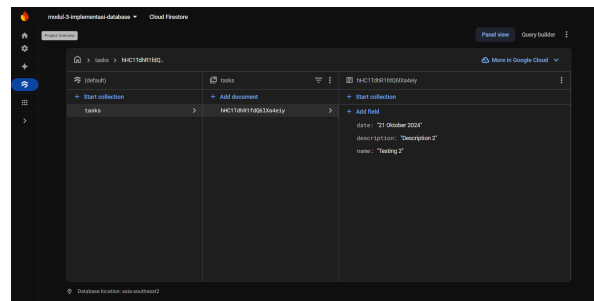
2. Setelah itu, untuk **Document ID** kalian klik **Auto-ID**



3. Isi data yang akan disimpan dalam dokumen sesuai dengan kebutuhan aplikasi



4. Jika sudah selesai, maka tampilan akan seperti gambar dibawah



5. Sebelum kita membuat tampilannya ada beberapa file pendukung yang perlu kita buat yaitu **app_color.dart** dan **widget_background.dart** agar tampilan aplikasi menjadi lebih menarik. Source code **app_color.dart** :

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

class AppColor {
  final Color colorPrimary = Color(0xFFFFBE4D4);
  final Color colorSecondary = Color(0xFFEC81B7);
  final Color colorTertiary = Color(0xFF425195);
}
```

Source code `widget_background.dart` :

```
import 'package:flutter/material.dart';
import 'package:flutter_firestore_todo/app_color.dart';

class WidgetBackground extends StatelessWidget {
  final AppColor appColor = AppColor();

  @override
  Widget build(BuildContext context) {
    return Stack(
      children: <Widget>[
        Positioned(
          top: -64,
          right: -128,
          child: Container(
            width: 256.0,
            height: 256.0,
            decoration: BoxDecoration(
              borderRadius: BorderRadius.circular(9000),
              color: appColor.colorTertiary,
            ),
          ),
        ),
        Positioned(
          top: -164,
          right: -8.0,
          child: Container(
            width: 256.0,
            height: 256.0,
            decoration: BoxDecoration(
              borderRadius: BorderRadius.circular(9000),
              backgroundBlendMode: BlendMode.hardLight,
              color: Colors.redAccent.withOpacity(0.8),
            ),
          ),
        ),
      ],
    );
  }
}
```

6. Setelah itu, langkah selanjutnya membuat tampilan halaman utama aplikasi dengan merubah pada file `home_screen.dart`.

```
class HomeScreen extends StatefulWidget {
  @override
  _HomeScreenState createState() => _HomeScreenState();
}

class _HomeScreenState extends State<HomeScreen> {
  final GlobalKey<ScaffoldState> scaffoldState = GlobalKey<ScaffoldState>();
  final Firestore firestore = Firestore.instance;
  final AppColor appColor = AppColor();

  @override
  Widget build(BuildContext context) {
    MediaQueryData mediaQueryData = MediaQuery.of(context);
    double widthScreen = mediaQueryData.size.width;
    double heightScreen = mediaQueryData.size.height;

    return Scaffold(
      key: scaffoldState,
      backgroundColor: appColor.colorPrimary,
      body: SafeArea(
        child: Stack(
          children: <Widget>[
            WidgetBackground(),
            _buildWidgetListTodo(widthScreen, heightScreen, context),
          ],
        ),
      ),
      floatingActionButton: FloatingActionButton(
        child: Icon(
          Icons.add,
          color: Colors.white,
        ),
        onPressed: () async {
          bool result = await Navigator.push(context, MaterialPageRoute(builder:
(context) => CreateTaskScreen(isEdit: false)));
          if (result != null && result) {
            scaffoldState.currentState.showSnackBar(SnackBar(
              content: Text('Task has been created'),
            ));
            setState(() {});
          }
        },
        backgroundColor: appColor.colorTertiary,
      ),
    );
  }
}
```



```

    ),
  );
}

Container _buildWidgetListTodo(double widthScreen, double heightScreen,
BuildContext context) {
  return Container(
    width: widthScreen,
    height: heightScreen,
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: <Widget>[
        Padding(
          padding: const EdgeInsets.only(left: 16.0, top: 16.0),
          child: Text(
            'Todo List',
            style: Theme.of(context).textTheme.title,
          ),
        ),
        Expanded(
          child: StreamBuilder<QuerySnapshot>(
            stream: firestore.collection('tasks').orderBy('date').snapshots(),
            builder: (BuildContext context, AsyncSnapshot<QuerySnapshot>
snapshot) {
              if (!snapshot.hasData) {
                return Center(child: CircularProgressIndicator());
              }
              return ListView.builder(
                padding: EdgeInsets.all(8.0),
                itemCount: snapshot.data.documents.length,
                itemBuilder: (BuildContext context, int index) {
                  DocumentSnapshot document = snapshot.data.documents[index];
                  Map<String, dynamic> task = document.data;
                  String strDate = task['date'];
                  return Card(
                    child: ListTile(
                      title: Text(task['name']),
                      subtitle: Text(
                        task['description'],
                        maxLines: 1,
                        overflow: TextOverflow.ellipsis,
                      ),
                    ),
                    isThreeLine: false,
                    leading: Column(
                      mainAxisAlignment: MainAxisAlignment.center,
                      children: <Widget>[

```

```

        Container(
          width: 24.0,
          height: 24.0,
          decoration: BoxDecoration(
            color: appColor.colorSecondary,
            shape: BoxShape.circle,
          ),
          child: Center(
            child: Text(
              '${int.parse(strDate.split(' ')[0])}',
              style: TextStyle(color: Colors.white),
            ),
          ),
        ),
        SizedBox(height: 4.0),
        Text(
          strDate.split(' ')[1],
          style: TextStyle(fontSize: 12.0),
        ),
      ],
    ),
    trailing: PopupMenuButton(
      itemBuilder: (BuildContext context) {
        return List<PopupMenuEntry<String>>()
          ..add(PopupMenuItem<String>(
            value: 'edit',
            child: Text('Edit'),
          ))
          ..add(PopupMenuItem<String>(
            value: 'delete',
            child: Text('Delete'),
          ));
      },
      // CodeLab
      onSelected: (String value) async {
        if (value == 'edit') {
          // TODO: fitur edit task
        } else if (value == 'delete') {
          // TODO: fitur hapus task
        }
      },
      child: Icon(Icons.more_vert),
    ),
  ),
);
},

```

```

        );
      },
    ),
  ],
),
);
}
}

```

7. Setelah merubah tampilan home, buat file baru dengan nama `create_task_screen.dart`

```

import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/material.dart';
import 'package:flutter_firestore_todo/app_color.dart';
import 'package:flutter_firestore_todo/widget_background.dart';
import 'package:intl/intl.dart';

class CreateTaskScreen extends StatefulWidget {
  final bool isEdit;
  final String documentId;
  final String name;
  final String description;
  final String date;

  CreateTaskScreen({
    @required this.isEdit,
    this.documentId = '',
    this.name = '',
    this.description = '',
    this.date = '',
  });

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

class _CreateTaskScreenState extends State<CreateTaskScreen> {
  final GlobalKey<ScaffoldState> scaffoldState = GlobalKey<ScaffoldState>();
  final Firestore firestore = Firestore.instance;
  final AppColor appColor = AppColor();
  final TextEditingController controllerName = TextEditingController();
  final TextEditingController controllerDescription = TextEditingController();
  final TextEditingController controllerDate = TextEditingController();

```

```

double widthScreen;
double heightScreen;
DateTime date = DateTime.now().add(Duration(days: 1));
bool isLoading = false;

@override
void initState() {
  if (widget.isEdit) {
    date = DateFormat('dd MMMM yyyy').parse(widget.date);
    controllerName.text = widget.name;
    controllerDescription.text = widget.description;
    controllerDate.text = widget.date;
  } else {
    controllerDate.text = DateFormat('dd MMMM yyyy').format(date);
  }
  super.initState();
}

@override
Widget build(BuildContext context) {
  MediaQueryData mediaQueryData = MediaQuery.of(context);
  widthScreen = mediaQueryData.size.width;
  heightScreen = mediaQueryData.size.height;

  return Scaffold(
    key: scaffoldState,
    backgroundColor: appColor.colorPrimary,
    resizeToAvoidBottomInset: false,
    body: SafeArea(
      child: Stack(
        children: <Widget>[
          WidgetBackground(),
          Container(
            width: widthScreen,
            height: heightScreen,
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.start,
              children: <Widget>[
                _buildWidgetFormPrimary(),
                SizedBox(height: 16.0),
                _buildWidgetFormSecondary(),
                isLoading
                  ? Container(
                      color: Colors.white,
                      padding: const EdgeInsets.all(16.0),
                      child: Center(

```

```

        child: CircularProgressIndicator(
          valueColor:
AlwaysStoppedAnimation<Color>(appColor.colorTertiary),
        ),
      ),
    ),
    : _buildWidgetButtonCreateTask(),
  ],
),
),
],
),
),
);
}

Widget _buildWidgetFormPrimary() {
  return Padding(
    padding: const EdgeInsets.all(12.0),
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: <Widget>[
        GestureDetector(
          onTap: () {
            Navigator.pop(context);
          },
          child: Icon(
            Icons.arrow_back,
            color: Colors.grey[800],
          ),
        ),
        SizedBox(height: 16.0),
        Text(
          widget.isEdit ? 'Edit\nTask' : 'Create\nNew Task',
          style: Theme.of(context).textTheme.display1.merge(
            TextStyle(color: Colors.grey[800]),
          ),
        ),
        SizedBox(height: 16.0),
        TextField(
          controller: controllerName,
          decoration: InputDecoration(
            labelText: 'Name',
          ),
          style: TextStyle(fontSize: 18.0),
        ),
      ],
    ),
  );
}

```

```

    ],
  ),
);
}

Widget _buildWidgetFormSecondary() {
  return Expanded(
    child: Container(
      decoration: BoxDecoration(
        color: Colors.white,
        borderRadius: BorderRadius.only(
          topLeft: Radius.circular(24.0),
          topRight: Radius.circular(24.0),
        ),
      ),
    ),
    padding: const EdgeInsets.symmetric(horizontal: 16.0, vertical: 8.0),
    child: Column(
      children: <Widget>[
        TextField(
          controller: controllerDescription,
          decoration: InputDecoration(
            labelText: 'Description',
            suffixIcon: Column(
              mainAxisAlignment: MainAxisAlignment.end,
              children: <Widget>[
                Icon(Icons.description),
              ],
            ),
          ),
          style: TextStyle(fontSize: 18.0),
        ),
        SizedBox(height: 16.0),
        TextField(
          controller: controllerDate,
          decoration: InputDecoration(
            labelText: 'Date',
            suffixIcon: Column(
              mainAxisAlignment: MainAxisAlignment.end,
              children: <Widget>[
                Icon(Icons.today),
              ],
            ),
          ),
          style: TextStyle(fontSize: 18.0),
          readOnly: true,
          onTap: () async {

```

```

        DateTime today = DateTime.now();
        DateTime datePicker = await showDatePicker(
            context: context,
            initialDate: date,
            firstDate: today,
            lastDate: DateTime(2021),
        );
        if (datePicker != null) {
            date = datePicker;
            controllerDate.text = DateFormat('dd MMMM yyyy').format(date);
        }
    },
),
],
),
),
);
}

```

```

Widget _buildWidgetButtonCreateTask() {
    return Container(
        width: double.infinity,
        color: Colors.white,
        padding: const EdgeInsets.symmetric(horizontal: 16.0, vertical: 8.0),
        child: RaisedButton(
            color: appColor.colorTertiary,
            child: Text(widget.isEdit ? 'UPDATE TASK' : 'CREATE TASK'),
            textColor: Colors.white,
            shape: RoundedRectangleBorder(
                borderRadius: BorderRadius.circular(4.0),
            ),
            onPressed: () async {
                String name = controllerName.text;
                String description = controllerDescription.text;
                String date = controllerDate.text;
                if (name.isEmpty) {
                    _showSnackBarMessage('Name is required');
                    return;
                } else if (description.isEmpty) {
                    _showSnackBarMessage('Description is required');
                    return;
                }
                setState(() => isLoading = true);
                if (widget.isEdit) {
                    DocumentReference documentTask =
                    firestore.document('tasks/${widget.documentId}');

```

```

        firestore.runTransaction((transaction) async {
            DocumentSnapshot task = await transaction.get(documentTask);
            if (task.exists) {
                await transaction.update(
                    documentTask,
                    <String, dynamic>{
                        'name': name,
                        'description': description,
                        'date': date,
                    },
                );
                Navigator.pop(context, true);
            }
        });
    } else {
        CollectionReference tasks = firestore.collection('tasks');
        DocumentReference result = await tasks.add(<String, dynamic>{
            'name': name,
            'description': description,
            'date': date,
        });
        if (result.documentID != null) {
            Navigator.pop(context, true);
        }
    }
},
),
);
}

void _showSnackBarMessage(String message) {
    scaffoldState.currentState.showSnackBar(SnackBar(
        content: Text(message),
    ));
}
}

```

Dokumentasi Firebase

Silahkan membaca [dokumentasi firebase ini](#) untuk penggunaan plugin firebase lainnya.

CODELAB

TUGAS 1

1. Lakukan setup dan instalasi firebase, pastikan muncul keterangan bahwa berhasil menginisiasi Firebase di Terminal
2. Buatlah halaman registrasi sederhana beserta controller untuk menangani input email dan password dari user menggunakan Firebase Authentication.
3. Jika user berhasil melakukan registrasi, email yang terdaftar akan tampil di halaman Authentication pada Firebase.

TUGAS 2

1. Lengkapi code `// TODO:` berikut pada file `home_screen.dart` (nama file sesuaikan dengan yang kalian buat)

```
onSelected: (String value) async {
  if (value == 'edit') {
    // TODO: fitur edit task
  } else if (value == 'delete') {
    // TODO: fitur hapus task
  }
},
```

KRITERIA & DETAIL PENILAIAN

Bobot Penilaian Modul 3 Materi (15%)

Berhasil melakukan instalasi firebase serta terdapat keterangan berhasil dalam terminal	10
Berhasil mengimplementasikan fitur registrasi dengan input email dan password	40
Email yang terdaftar terlihat pada halaman Authentication pada Firebase	10
Berhasil melengkapi code TODO dan mengimplementasikan CRUD dengan Cloud Firestore	40
Total	100