



Empowering Digital Skills For The Jobs Of The Future



by



The background of the slide features a series of overlapping, wavy bands in various shades of blue and purple. These bands create a sense of movement and depth, framing the central text. The colors range from deep indigo and navy blue to lighter, almost white, lavender and pale blue tones.

**http e firebase**

# Docente



Claudia Infante



[claudia.infante@bcsoft.net](mailto:claudia.infante@bcsoft.net)

# Sommario

- Recuperare dati da internet
- http package
- Serialization
- Firebase in Flutter
- Creazione progetto
- Configurazione dell'applicazione
- Autenticazione
- Creazione database

# Recuperare dati da internet

L'HTTP (Hypertext Transfer Protocol) svolge un ruolo fondamentale nel recupero dei dati da un'API (Application Programming Interface).

# Recuperare dati da internet

HTTP è il protocollo utilizzato dai server Web per comunicare con i client, come i browser Web o le applicazioni.

Quando si recuperano dati da un'API, un client invia una richiesta HTTP al server API specificando il tipo di azione che desidera eseguire (ad esempio, ottenere dati, pubblicare dati, aggiornare dati, cancellare dati).

# Recuperare dati da internet

Il server API elabora la richiesta e invia una risposta HTTP al client con i dati richiesti o un messaggio di errore se la richiesta non può essere soddisfatta .

# Recuperare dati da internet

Le richieste HTTP vengono effettuate utilizzando metodi specifici, come GET, POST, PUT e DELETE, che corrispondono a diverse azioni che possono essere eseguite sui dati API.



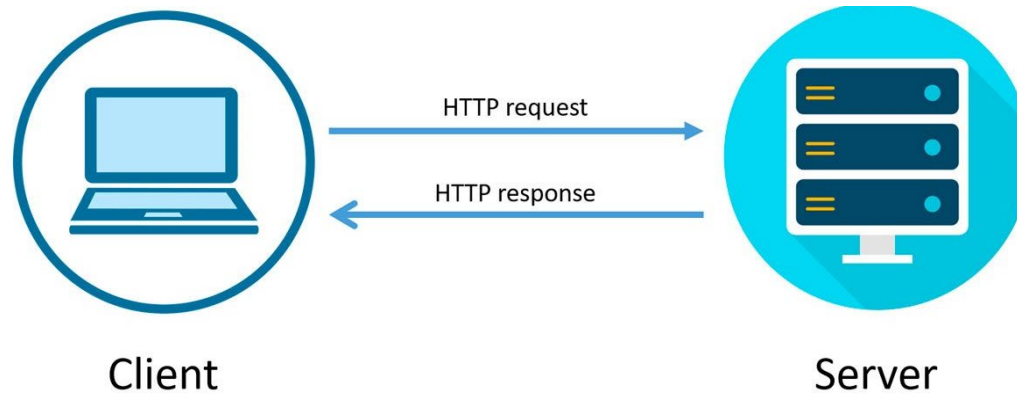
# Recuperare dati da internet

La risposta del server include in genere un codice di stato (ad esempio, 200 per una richiesta riuscita, 404 per un errore non trovato) e i dati richiesti dal client .

# Recuperare dati da internet

In generale, l'HTTP funge da livello di comunicazione che consente ai client di interagire con le API e di recuperare i dati su Internet.

Garantisce il trasferimento sicuro e affidabile delle informazioni tra il client e il server API.



# http package

In flutter, l'utilizzo del protocollo http è correlato alla dipendenza del pacchetto http, che andrà inserito nel file di configurazione pubspec.yaml.

```
dependencies:  
  http: ^1.2.1
```

# http package

Una volta aggiunto, lanciare il comando *flutter packages get* nella cartella del progetto, in modo da scaricare il pacchetto e renderlo utilizzabile .

# Serialization

I dati ricevuti dovranno poi essere serializzati, ovvero convertiti seguendo una formattazione dei dati standard che può essere passata nel network di comunicazione.

# Serialization

I principali formati sono XML e JSON.

# JSON Serialization

In Flutter, esistono due opzioni di serializzazione : manuale e automatica utilizzando un pacchetto.

# JSON Serialization

Il processo di serializzazione manuale si basa sull'utilizzo di classi non troppo robuste che rappresenteranno il modello dei dati ricevuti.

```
// json
{
  "userId": 1,
  "id": 1,
  "title": "delectus aut autem",
  "completed": false
}
```

```
// class
class Todo {
  final int userId;
  final int id;
  final String title;
  bool completed;

  Todo(
    this.userId,
    this.id,
    this.title,
    this.completed,
  );
}
```



# JSON Serialization

Successivamente andrà definito il metodo che trasformerà i dati ricevuti. Convenzionalmente si definisce un factory method per la classe chiamato `fromJson()`.

# JSON Serialization

Questo metodo prenderà una Mappa come argomento e creerà un nuovo oggetto da quella mappa.

# JSON Serialization

La risorsa da recuperare è la seguente

```
[
  {
    "id": 1,
    "todo": "Do something nice for someone I care
about",
    "completed": true,
    "userId": 26
  },
  {
    "id": 2,
    "todo": "Memorize the fifty states and their
capitals",
    "completed": false,
    "userId": 48
  },
]
```

# JSON Serialization

Si definisce una classe `Todo` che offra un modello per gli oggetti ricevuti

```
9  class Todo {  
10    final int userId;  
11    final int id;  
12    final String todo;  
13    bool completed;  
14    Todo(this.userId, this.id, this.todo, this.completed);  
15  factory Todo.fromJson(  
16    Map<String, dynamic> json,  
17  ) {  
18    return Todo(  
19      json['userId'] as int,  
20      json['id'] as int,  
21      json['todo'] as String,  
22      json['completed'] as bool  
23    );  
24  }  
25 }
```

# JSON Serialization

Si recupera la lista e la si mostra nella UI grazie al metodo `json.decode`

```
27 class MyApp extends StatefulWidget {
28   @override
29   _MyAppState createState() => _MyAppState();
30 }
31
32 class _MyAppState extends State<MyApp> {
33   List<Todo> todos = [];
34
35   @override
36   void initState() {
37     super.initState();
38     fetchTodos();
39   }
40
41   Future<void> fetchTodos() async {
42     final response =
43     await http.get(Uri.parse('https://mocki.io/v1/ed127878-607d-4040-a5fc-69cd8a11b405'));
44     if (response.statusCode == 200) {
45       List<dynamic> data = json.decode(response.body);
46       setState(() {
47         todos = data.map((item) => Todo.fromJson(item)).toList();
48       });
49     } else {
50       throw Exception('Failed to load todos');
51     }
52   }
53
54   void toggleTodoStatus(int index) {
55     setState(() {
56       todos[index].completed = !todos[index].completed;
57     });
58   }
```

# JSON Serialization

A questo punto  
sarà poi possibile  
mostrarli nella UI

```
59
60 @override
61 Widget build(BuildContext context) {
62   return MaterialApp(
63     home: Scaffold(
64       appBar: AppBar(
65         title: Text('Todo List'),
66       ), // AppBar
67       body: ListView.builder(
68         itemCount: todos.length,
69         itemBuilder: (context, index) {
70           return ListTile(
71             title: Text(todos[index].todo),
72             subtitle: Text('Completed: ${todos[index].completed}'),
73             leading: Checkbox(
74               value: todos[index].completed,
75               onChanged: (bool? value) {
76                 toggleTodoStatus(index);
77               },
78             ),
79           ); // ListTile
80         },
81       ), // ListView.builder
82     ); // Scaffold
83   ); // MaterialApp
84 }
85 }
86
```

# Firestore in Flutter

Firestore è una piattaforma di sviluppo web e mobile che fornisce una serie di servizi di backend per aiutare gli sviluppatori a creare applicazioni scalabili e di facile utilizzo. Offre servizi come l'autenticazione, il database in tempo reale, il cloud storage e l'hosting.



# Firestore

# Firestore in Flutter

In Flutter, Firestore è comunemente utilizzato perché fornisce un backend robusto e affidabile per memorizzare e recuperare i dati in tempo reale. Questo è particolarmente utile per le applicazioni che richiedono aggiornamenti in tempo reale e comunicazione tra gli utenti.



# Firestore in Flutter

Quando si tratta di effettuare richieste HTTP in Flutter, Firestore può essere utilizzato come alternativa alle richieste HTTP tradizionali, fornendo un modo più efficiente e affidabile per interagire con i servizi di backend.

# Firestore in Flutter

Gli SDK di Firebase offrono metodi integrati per la gestione delle richieste di rete, rendendo più facile per gli sviluppatori integrare i servizi Firebase nelle loro applicazioni Flutter senza dover gestire manualmente le richieste HTTP.

# Firestore in Flutter

Nel complesso, Firestore viene utilizzato in Flutter per la sua praticità, scalabilità e capacità di utilizzo in tempo reale, che possono aiutare gli sviluppatori a creare applicazioni di alta qualità e reattive.

# Firestore in Flutter

Il servizio di database offerto da Firebase si chiama Firestore. Si occupa di gestire la comunicazione tra l'applicazione e i dati dando modo di conservarli e lavorare con essi.

# Firestore in Flutter

In particolare si ha la possibilità di sottoscrivere i dati ed interagire. Quando il codice sottoscrive i dati in Firestore, il codice conosce quando i dati specificati cambiano e l'applicazione si aggiorna di conseguenza, creando applicazioni altamente reattive.

# Firestore in Flutter

Per configurare Firestore bisogna :

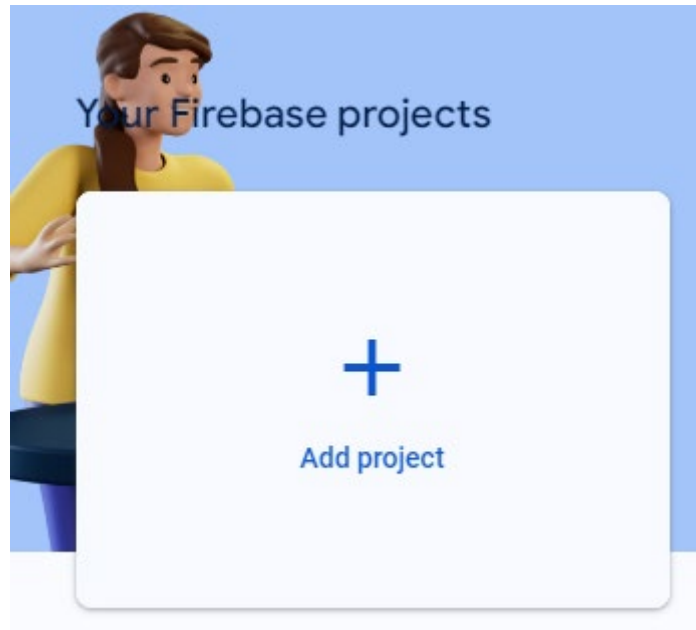
1. Possedere un account Firestore.
2. Creare un nuovo progetto in Firestore.
3. Aggiungere un database Firestore al progetto.
4. Registrare l'applicazione con Firestore.
5. Aggiornare le cartelle native.
6. Aggiungere Firestore e Firestore come dipendenze del file `pubspec.yaml`.
7. Utilizzarle.

# Creazione progetto in Firebase

Il primo passaggio si basa sulla creazione di un account gratuito sul sito [firebase.google.com](https://firebase.google.com).

# Creazione progetto in Firebase

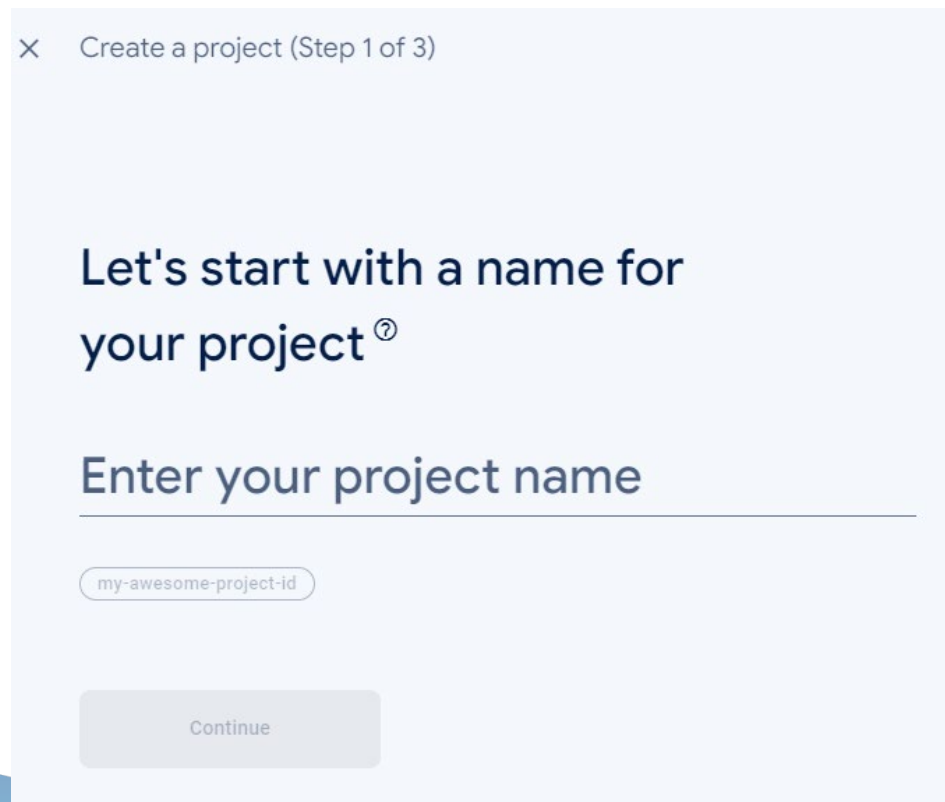
Successivamente creare un progetto cliccando nella console sulla voce Add project.





# Creazione progetto in Firebase

Indicare il nome del progetto e cliccare su continue.



The screenshot shows a modal dialog box titled "Create a project (Step 1 of 3)". The main text inside the dialog says "Let's start with a name for your project" followed by a question mark icon. Below this is a label "Enter your project name" above a text input field. The input field contains the text "my-awesome-project-id". At the bottom of the dialog is a "Continue" button.

X Create a project (Step 1 of 3)

Let's start with a name for  
your project<sup>?</sup>

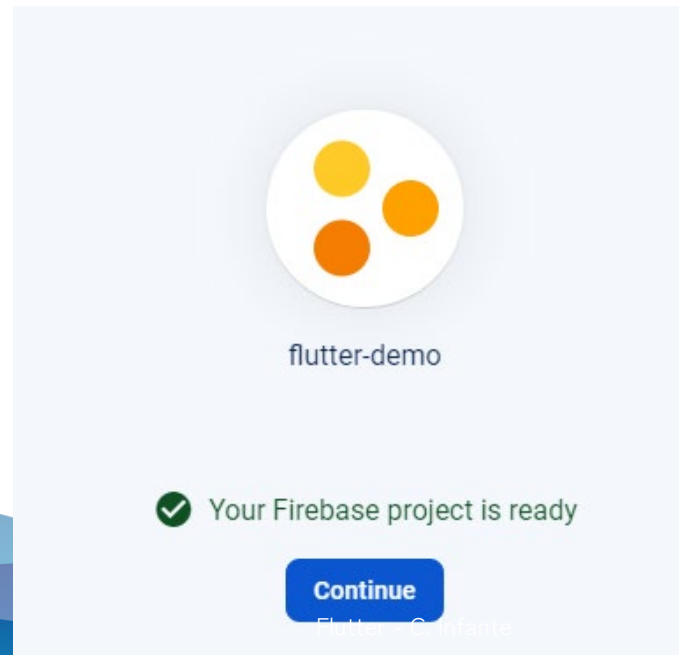
Enter your project name

my-awesome-project-id

Continue

# Creazione progetto in Firebase

Abilitare o meno l'accesso al tool di marketing Google Analytics a seconda delle esigenze. Firebase fornisce automaticamente le risorse per il progetto Firebase. Quando il processo è concluso, si accede alla pagina di riepilogo del progetto Firebase nella console di Firebase.

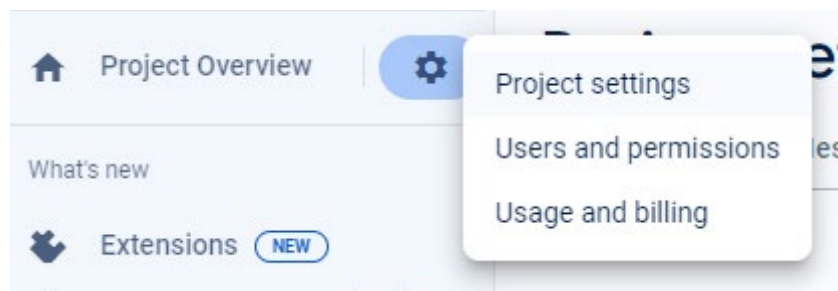


# Configurazione dell'applicazione

Prima di poter utilizzare i packages di Firebase bisognerà configurare le piattaforme utilizzate dal progetto. Il processo in passato cambiava in base al tipo di piattaforma nativa (iOS o Android).

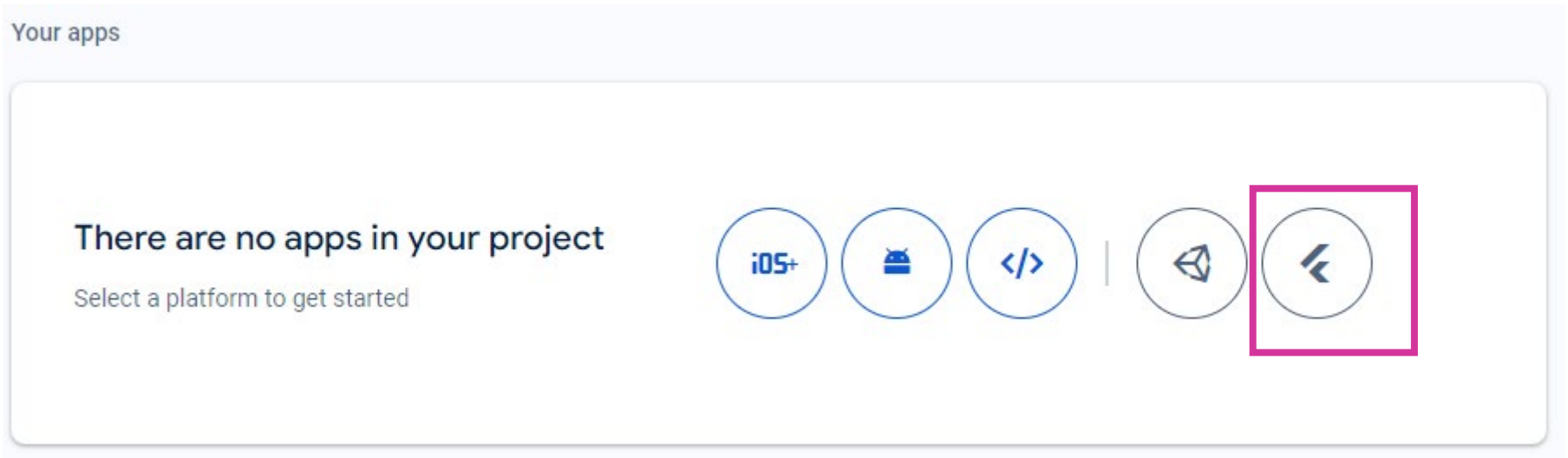
# Configurazione dell'applicazione

Nella console di Firebase, selezionare dal menu Project Overview. Cliccare poi sul pulsante delle impostazioni e poi su Project settings



# Configurazione dell'applicazione

Selezionare poi una piattaforma per cominciare oppure selezionare direttamente flutter.



# Configurazione dell'applicazione

Una volta selezionato, appariranno le indicazioni da seguire. Il primo è relativo alla installazione della CLI di firebase e di Flutter SDK.

## 1 Prepare your workspace

The easiest way to get you started is to use the FlutterFire CLI.

Before you continue, make sure to:

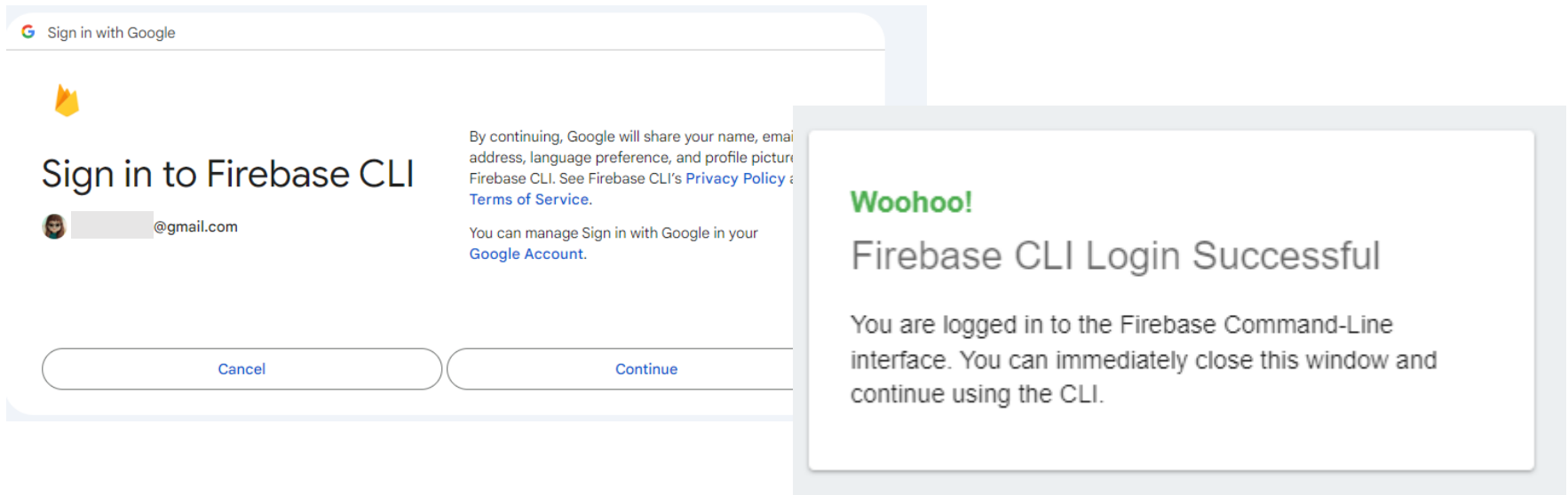
- Install the [Firebase CLI](#) and log in (run `firebase login`)
- Install the [Flutter SDK](#)
- Create a Flutter project (run `flutter create`)

# Configurazione dell'applicazione

La CLI di firebase è disponibile al [seguente link](#) oppure lanciando il comando: `npm install -g firebase-tools`.

# Configurazione dell'applicazione

Una volta installata, procedere al login lanciando il comando: `firebase login`. Si aprirà una pagina da cui sarà possibile scegliere l'account di autenticazione.





# Configurazione dell'applicazione

Nella seconda fase lanciare i seguenti comandi:

Da una folder qualsiasi : `dart pub global activate flutterfire_cli`

Dalla root del progetto : `flutterfire configure --project=idProgetto`

Queste operazioni registreranno automaticamente il file di configurazione nel progetto.

# Configurazione dell'applicazione

All'esecuzione verrà chiesto la piattaforma da configurare. Con lo space è possibile selezionare più di un'opzione o rimuoverle. Una volta selezionate, cliccare su invito

```
i Found 3 Firebase projects. Selecting project flutter-demo-13f08.  
? Which platforms should your configuration support (use arrow keys & space to select)? >  
✓ android  
✓ ios  
  macos  
✓ web
```

# Configurazione dell'applicazione

Nella terza fase, aggiungere il package di firebase a flutter

## 3 Initialize Firebase and add plugins

To initialize Firebase, call `Firebase.initializeApp` from the `firebase_core` package with the configuration from your new `firebase_options.dart` file:

```
import 'package:firebase_core/firebase_core.dart';  
import 'firebase_options.dart';  
  
// ...  
  
await Firebase.initializeApp(  
  options: DefaultFirebaseOptions.currentPlatform,  
);
```



Then, add and begin using the [Flutter plugins](#) for the Firebase products you'd like to use.

# Configurazione dell'applicazione

Aggiungere alle dipendenze del file pubspec.yaml  
firebase\_core con il comando : flutter pub add firebase\_core

```
dependencies:  
  firebase_core: ^2.28.0
```

Ora sarà possibile utilizzarlo grazie al seguente import :

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

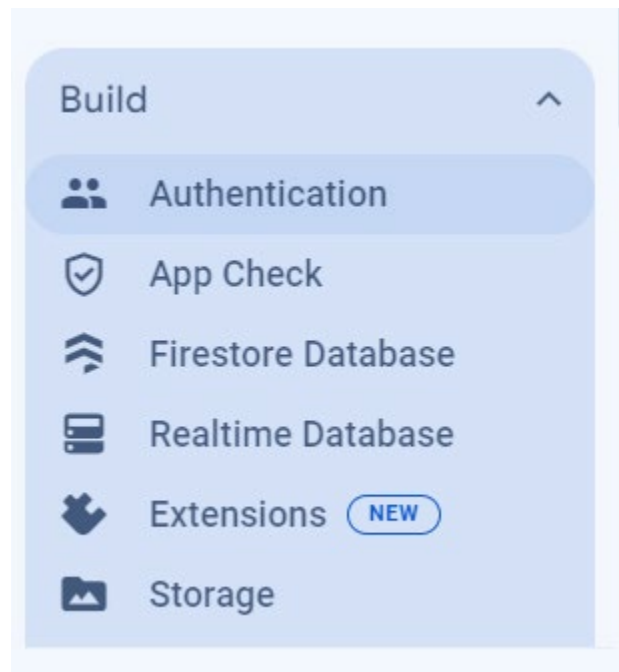
# Configurazione dell'applicazione

Nel main.dart apportare le seguenti modifiche

```
Run | Debug | Profile
7 void main() async {
8   WidgetsFlutterBinding.ensureInitialized();
9   await Firebase.initializeApp(options: DefaultFirebaseOptions.currentPlatform);
0   runApp(MyApp());
1 }
```

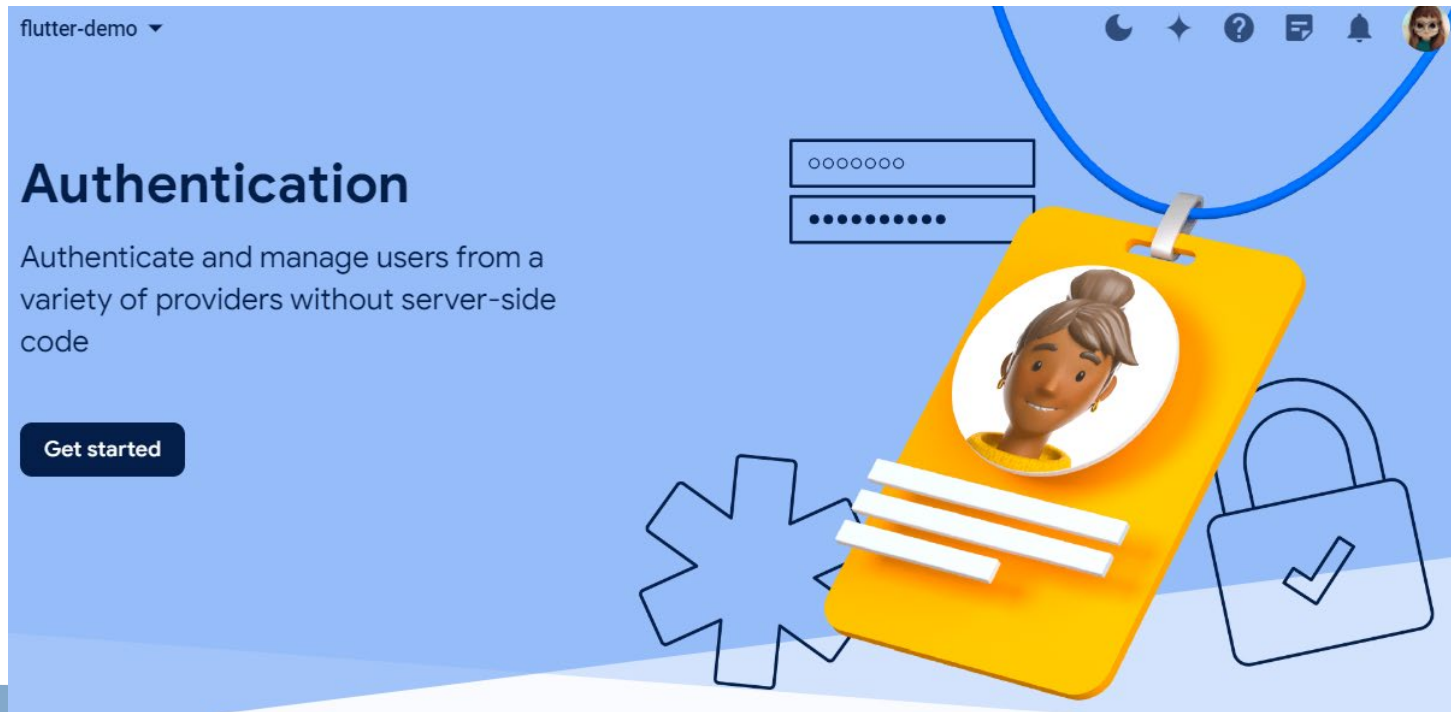
# Autenticazione in firebase

In Build selezionare Authentication



# Autenticazione in firebase

Cliccare sul bottone Get Started



# Autenticazione in firebase

Scegliere il provider Email/Password

Get started with Firebase Auth by adding your first sign-in method

## Native providers

 Email/Password

 Phone

 Anonymous

## Additional providers

 Google

 Facebook

 Play Games

 Game Center

 Apple

 GitHub

 Microsoft

 Twitter

 Yahoo



# Autenticazione in firebase

Abilitare l'impostazione per la registrazione tramite email

Sign-in providers

 Email/Password

 Enable

Allow users to sign up using their email address and password. Our SDKs also provide email address verification, password recovery, and email address change primitives. [Learn more](#)

Email link (passwordless sign-in)

 Enable

Cancel

Save

Add new provider

Provider

Status

 Email/Password

 Enabled

# Autenticazione in firebase

Creare un file di configurazione definendo una classe astratta in cui si invocherà il metodo `createUserWithEmailAndPassword()`

```
! pubspec.yaml  main.dart 2  auth_data.dart X  login.dart  auth_page.dart  build.gradle

lib > auth_data.dart > AuthenticationRemote
1  import 'package:firebase_auth/firebase_auth.dart';
2
3  abstract class AuthenticationSource{
4    | Future<void> register(String email, String password);
5  }
6
7  class AuthenticationRemote extends AuthenticationSource{
8    | @override
9    | Future<void> register(String email, String password)async{
10   |   await FirebaseAuth.instance.createUserWithEmailAndPassword(email: email.trim(), password: password.trim());
11   | }
12 }
```

# Autenticazione in firebase

A questo punto si potrà invocare il metodo nel bottone di un form di registrazione.

```
ElevatedButton(  
  onPressed: () {  
    AuthenticationRemote().register(email.text, password.text) ;  
  },  
)
```

# Autenticazione in firebase

L'utente comparirà anche all'interno di firebase

Search by email address, phone number, or user UID					Add user	↺	⋮
Identifier	Providers	Created ↓	Signed In	User UID			
abc@cd.it	✉	Apr 22, 2024	Apr 22, 2024	S9pMWntLxhRHAihPOPHk7D...			
					Rows per page:	50 ▼	1 – 1 of 1 < >

# Creazione de database

Si crei un database in firestore

**Cloud Firestore**

### Create database

1 Set name and location — 2 **Secure rules**

After you define your data structure, you will need to write rules to secure your data.  
[Learn more](#)

☐ Start in **production mode**

Your data is private by default. Client read/write access will only be granted as specified by your security rules.

☒ Start in **test mode**

Your data is open by default to enable quick setup. However, you must update your security rules within 30 days to enable long-term client read/write access.

```
rules_version = '2';

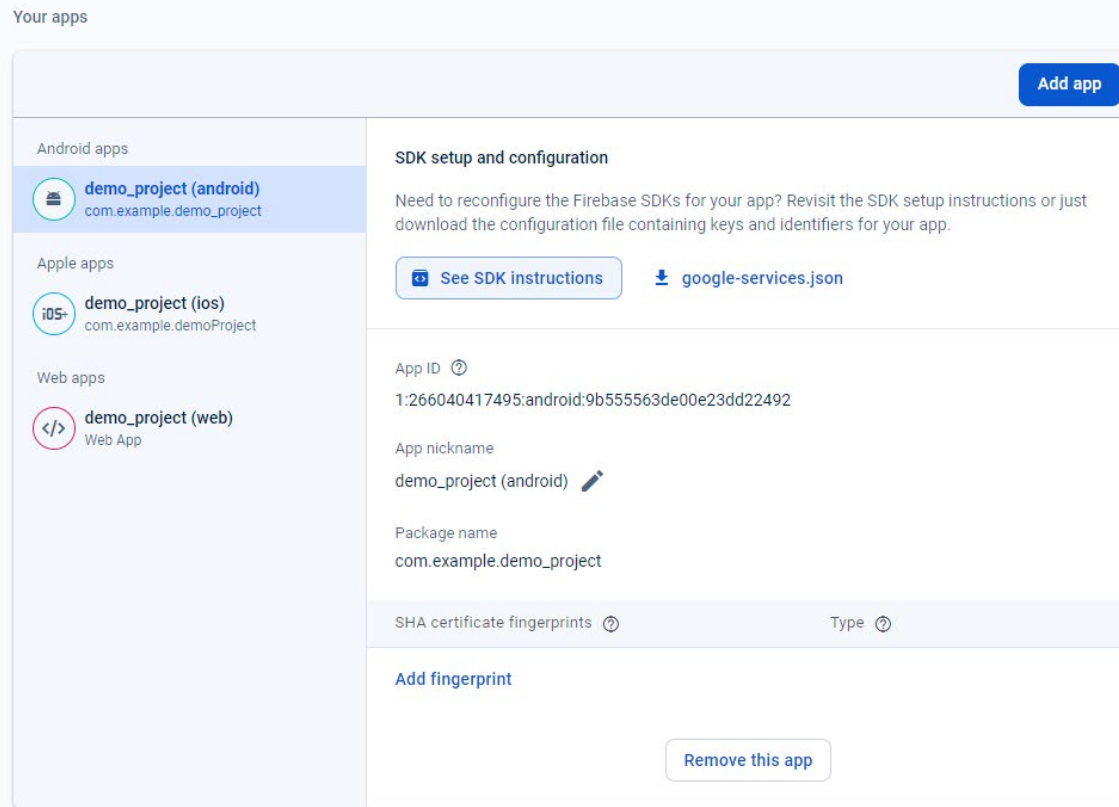
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write: if
        request.time < timestamp.date(2024, 5, 5);
    }
  }
}
```

**!** The default security rules for test mode allow anyone with your database reference to view, edit and delete all data in your database for the next 30 days

Cancel **Create**

# Creazione de database

In project setting selezionare le istruzioni di installazione della SDK per il setup del database nel progetto .



# Creazione de database

Scaricare il file json creato e inserirlo nella root directory di android

2


Download and then add config file

Instructions for Android Studio below | [Unity](#) [C++](#)

Download google-services.json

Switch to the **Project** view in Android Studio to see your project root directory.

Move your downloaded `google-services.json` file into your module (app-level) root directory.



google-services.json

Project

MyApplication [My Application]

.gradle

.idea

app

libs

src

.gitignore

build.gradle.kts

google-services.json

proguard-rules.pro

gradle

Next

# Creazione de database

Nel file build.gradle aggiornare plugin e dipendenze

```
plugins {  
    id "com.android.application"  
    id "kotlin-android"  
    id "dev.flutter.flutter-gradle-plugin"  
    id("com.google.gms.google-services") version "4.4.1" apply false  
}
```



# Domande e approfondimenti

# Attività

Integrare chiamate http nel progetto di Travel Tour (di cui si è definita la navigazione in precedenza) in modo da :

- Consentire l'accesso solo ad utenti autenticati utilizzando firebase;
- Mostrare dei dati recuperati tramite chiamata http o da firebase;