

Programmazione Mobile

Nicola Noviello

nicola.noviello@unimol.it

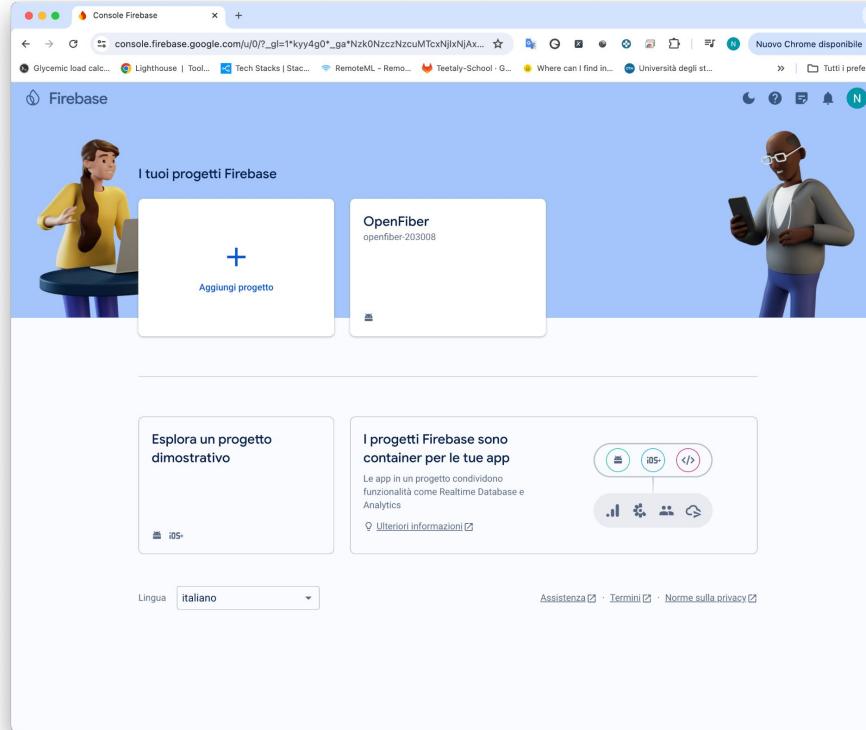
Corso di Laurea in Informatica
Dipartimento di Bioscienze e Territorio
Università degli Studi del Molise
Anno 2023/2024

Lezione: Integrazione con Firebase

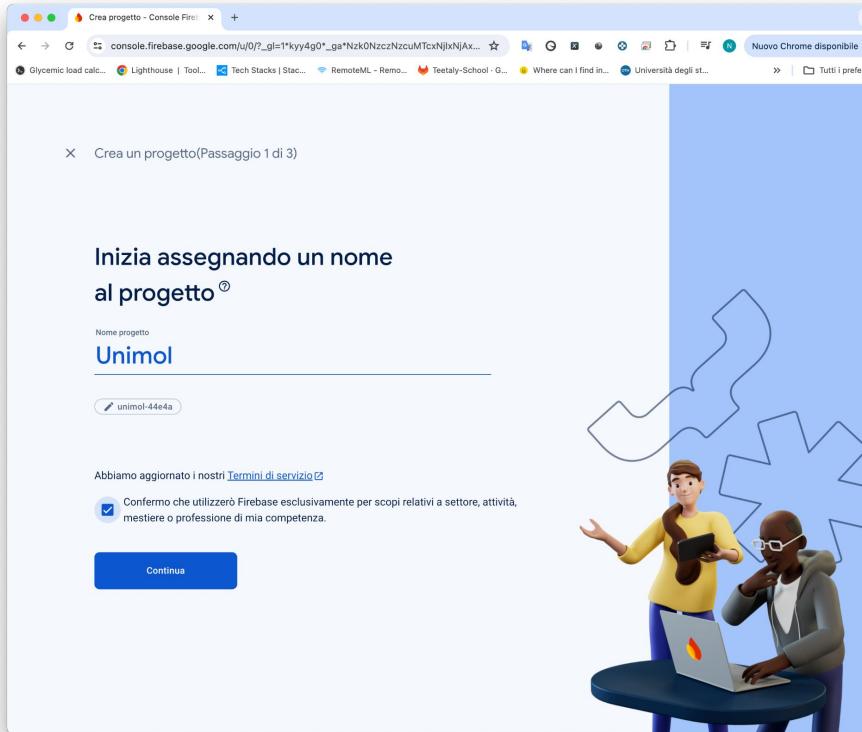
- Creazione di un progetto Firebase
- Gestione dell'autenticazione
- Salvataggio informazioni in remoto
- Salvataggio file in remoto
- Notifiche push

Creazione di un nuovo Progetto Firebase

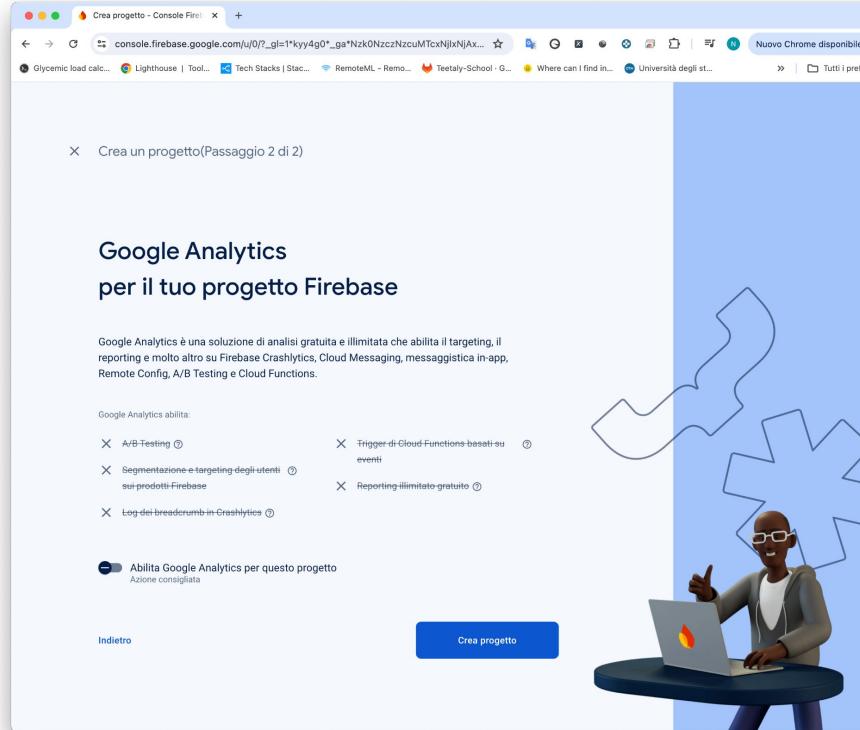
Creazione di un nuovo progetto su Firebase



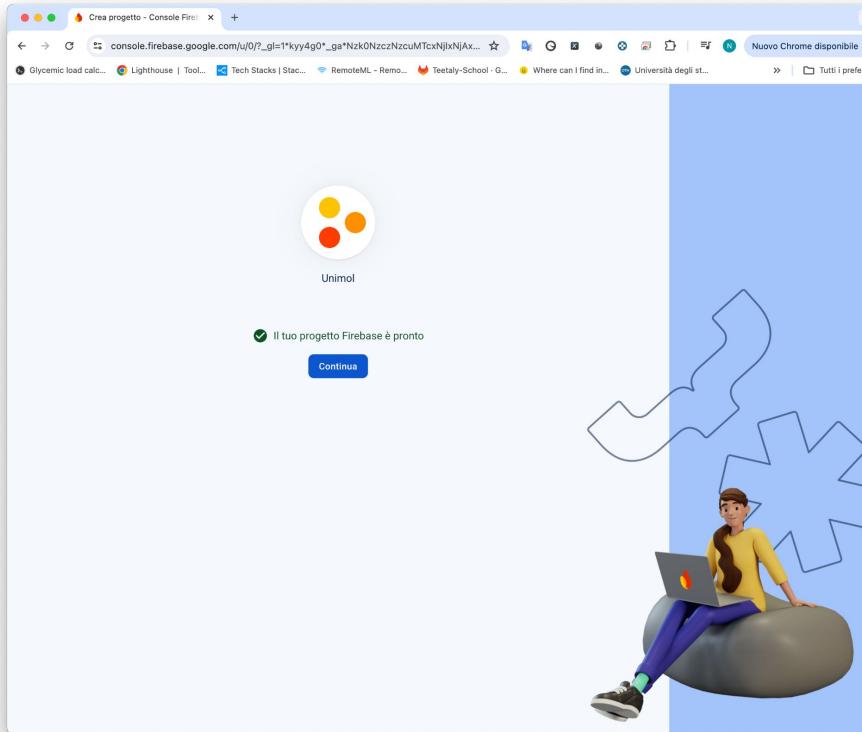
Creazione di un nuovo progetto su Firebase



Creazione di un nuovo progetto su Firebase



Creazione di un nuovo progetto su Firebase



Autenticazione su Firebase

The screenshot shows the Firebase console interface for managing authentication providers. The left sidebar lists various services: Panoramica del progetto, Build with Gemini, Data Connect, App Hosting, and others under Creazione, Esecuzione, and Analisi. The main content area is titled "Authentication". The "Metodo di accesso" tab is selected, showing a grid of provider icons:

Provider nativi	Provider aggiuntivi	Provider personalizzati
Email/password	Google, Facebook, Play Giochi	OpenID Connect
Telefono	Game Center, Apple, GitHub	SAML
Anonimo	Microsoft, Twitter, Yahoo	

Below this, the "Avanzate" section contains a box for "Autenticazione a più fattori tramite SMS". It explains that this feature allows users to add an extra layer of security via SMS. A note states that MFA and other advanced features are available with Identity Platform. A blue button at the bottom right says "Esegui upgrade per abilitare".

At the bottom of the page, it says "Nessun costo 0 \$/mese" and "Esegui l'upgrade". The URL https://console.firebaseio.google.com/u/0/project/unimol-44e4a/authentication/users is visible at the bottom.

Autenticazione su Firebase

The screenshot shows the Firebase console interface for managing authentication providers. The left sidebar lists various services: Panoramica del progetto, IA generativa, Build with Gemini (NEW), Scorsiziote di progetto, Authentication (selected), App Hosting (NEW), and Data Connect (NEW). The main content area is titled "Authentication" and shows the "Metodo di accesso" tab selected. It displays a table of providers:

Provider	Stato
Email/password	Abilitato

Below the table, there's a section titled "Avanzate" (Advanced) with a sub-section for "Autenticazione a più fattori tramite SMS" (Multi-factor authentication via SMS). It explains that enabling this feature adds an extra layer of security for users. A note states: "MFA e altre funzionalità avanzate sono disponibili con Identity Platform, la soluzione completa per l'identità dei clienti di Google Cloud, realizzata in partnership con Firebase. Questo upgrade è disponibile per i piani Spark e Blaze." A blue button at the bottom right of this section says "Esegui upgrade per abilitare".

Creazione di un progetto Flutter

main.dart

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

import 'package:fireb/screens/auth.dart';

void main() {
  runApp(const App());
}

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'FlutterChat',
      theme: ThemeData().copyWith(
        colorScheme:
          ColorScheme.fromSeed(seedColor: Color.fromARGB(255, 223, 243, 250)),
      ),
      home: const AuthScreen(),
    );
}
}
```

AuthScreen

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

class AuthScreen extends StatefulWidget {
  const AuthScreen({super.key});

  @override
  State<AuthScreen> createState() {
    return _AuthScreenState();
  }
}

class _AuthScreenState extends State<AuthScreen> {
  var _isLogin = true;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Theme.of(context).colorScheme.primary,
      body: Center(
        child: SingleChildScrollView(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              Container(
                margin: const EdgeInsets.only(
                  top: 30,
                  bottom: 20,
                  left: 20,
                  right: 20,
                ),
                width: 200,
                child: Image.asset('assets/images/chat.jpeg'),
              ),
              Card(
                margin: const EdgeInsets.all(20),
                ...
              )
            ],
          ),
        ),
      ),
    );
  }
}
```

- `_AuthScreenState` ha una variabile `_isLogin` che determina se l'utente sta cercando di accedere o registrarsi;
- Il metodo `build` di `_AuthScreenState` restituisce un widget `Scaffold` che rappresenta la struttura di base della schermata di autenticazione;
- Il corpo del `Scaffold` è un `Center` che contiene un `SingleChildScrollView`. Questo permette alla schermata di scorrere se il contenuto supera l'altezza dello schermo;
- Il `SingleChildScrollView` contiene una `Column` con due figli: un `Container` per un'immagine e una `Card` per il modulo di autenticazione.

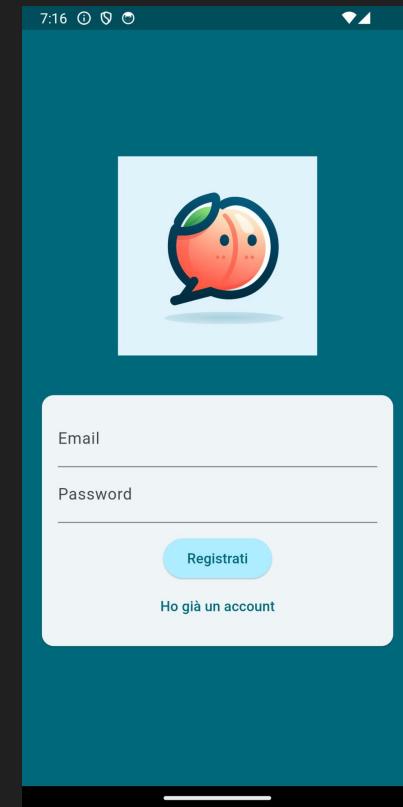
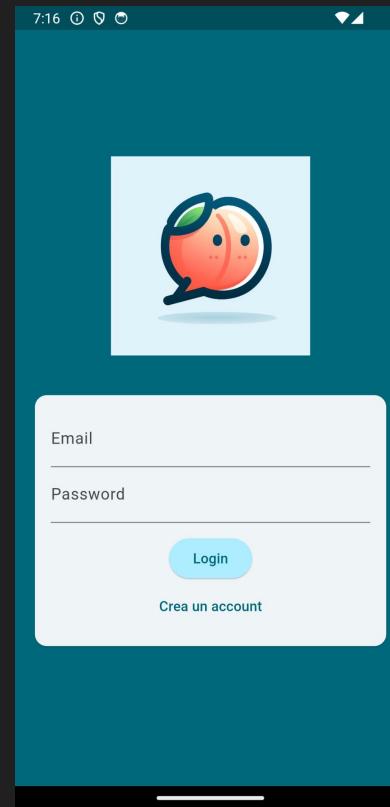
```
...  
    child: Image.asset('assets/images/chat.jpeg'),  
)  
Card(  
margin: const EdgeInsets.all(20),  
child: SingleChildScrollView  
child: Padding(  
padding: const EdgeInsets.all(16),  
child: Form(  
child: Column(  
mainAxisSize: MainAxisSize.min,  
children: [  
    TextFormField(  
decoration:  
        const InputDecoration(labelText: 'Email'),  
keyboardType: TextInputType.emailAddress,  
autocorrect: false,  
textCapitalization: TextCapitalization.none,  
),  
TextFormField(  
decoration:  
        const InputDecoration(labelText: 'Password'),  
obscureText: true,  
),  
const SizedBox(height: 12),  
ElevatedButton(  
onPressed: () {},  
style: ElevatedButton.styleFrom(  
    backgroundColor: Theme.of(context)  
        .colorScheme  
        .primaryContainer,  
),  
child: Text(_isLogin ? 'Login' : 'Registrati'),  
),  
TextButton(  
onPressed: () {  
    setState(() {  
        isLogin = !_isLogin;  
    });  
},  
child: Text(isLogin  
    ? 'Crea un account'  
    : 'Ho già un account'),  
),  
],  
),  
)  
,
```

- Il **SingleChildScrollView** contiene una **Column** con due figli: un **Container** per un'immagine e una **Card** per il modulo di autenticazione;
- Il modulo di autenticazione è un **Form** che contiene due **TextFormField** per l'email e la password, un **ElevatedButton** per inviare il modulo, e un **TextButton** per cambiare tra login e registrazione;
- Quando l'utente preme il **TextButton**, il metodo **setState** viene chiamato per invertire il valore di **_isLogin** e ricostruire l'interfaccia utente con il nuovo stato.

```
    child: Image.asset('assets/images/chat.jpeg'),  
),  
Card(  
margin: const EdgeInsets.all(20),  
child: SingleChildScrollView(  
  child: Padding(  
    padding: const EdgeInsets.all(16),  
    child: Form(  
      child: Column(  
        mainAxisSize: MainAxisSize.min,  
        children: [  
          TextFormField(  
            decoration:  
              const InputDecoration(labelText: 'Email'),  
            keyboardType: TextInputType.emailAddress,  
            autocorrect: false,  
            textCapitalization: TextCapitalization.none,  
          ),  
          TextFormField(  
            decoration:  
              const InputDecoration(labelText: 'Password'),  
            obscureText: true,  
          ),  
          const SizedBox(height: 12),  
          ElevatedButton(  
            onPressed: () {},  
            style: ElevatedButton.styleFrom(  
              backgroundColor: Theme.of(context)  
                .colorScheme  
                .primaryContainer,  
            ),  
            child: Text(_isLogin ? 'Login' : 'Registrati'),  
          ),  
          TextButton(  
            onPressed: () {  
              setState(() {  
                _isLogin = !_isLogin;  
              });  
            },  
            child: Text( isLogin  
              ? 'Crea un account'  
              : 'Ho già un account'),  
          ),  
        ],  
      ),  
    ),  
  ),  
)
```

Interessante: Il modo come sono costruiti i TextFormField

```
        child: Image.asset('assets/images/chat.jpeg'),  
    ),  
    Card(  
        margin: const EdgeInsets.all(20),  
        child: SingleChildScrollView(  
            child: Padding(  
                padding: const EdgeInsets.all(16),  
                child: Form(  
                    child: Column(  
                        mainAxisAlignment: MainAxisAlignment.min,  
                        children: [  
                            TextFormField(  
                                decoration:  
                                    const InputDecoration(labelText: 'Email'),  
                                keyboardType: TextInputType.emailAddress,  
                                autocorrect: false,  
                                textCapitalization: TextCapitalization.none,  
                            ),  
                            TextFormField(  
                                decoration:  
                                    const InputDecoration(labelText: 'Password'),  
                                obscureText: true,  
                            ),  
                            const SizedBox(height: 12),  
                            ElevatedButton(  
                                onPressed: () {},  
                                style: ElevatedButton.styleFrom(  
                                    backgroundColor: Theme.of(context)  
                                        .colorScheme  
                                        .primaryContainer,  
                                ),  
                                child: Text(_isLogin ? 'Login' : 'Registrati'),  
                            ),  
                            TextButton(  
                                onPressed: () {  
                                    setState(() {  
                                        _isLogin = !_isLogin;  
                                    });  
                                },  
                                child: Text(_isLogin  
                                    ? 'Crea un account'  
                                    : 'Ho già un account'),  
                            ),  
                        ],  
                    ),  
                ),  
            ),  
        ),  
    ),
```



Validazione dei form

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

class AuthScreen extends StatefulWidget {
  const AuthScreen({super.key});

  @override
  State<AuthScreen> createState() {
    return _AuthScreenState();
  }
}

class _AuthScreenState extends State<AuthScreen> {
  final _form = GlobalKey<FormState>();

  var _isLogin = true;
  var _enteredEmail = '';
  var _enteredPassword = '';

  void submit() async {
    final isValid = _form.currentState!.validate();

    if (!isValid) {
      return;
    }

    _form.currentState!.save();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Theme.of(context).colorScheme.primary,
      body: Center(
        child: SingleChildScrollView(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              Container(
                margin: const EdgeInsets.only(
                  top: 30,
                  bottom: 20,
                  left: 20,
                  right: 20,
                ),
                width: 200,
                child: Image.asset('assets/images/chat.jpeg'),
              ),
              Card(
                margin: const EdgeInsets.all(20),
                ...
              )
            ],
          ),
        ),
      ),
    );
  }
}
```

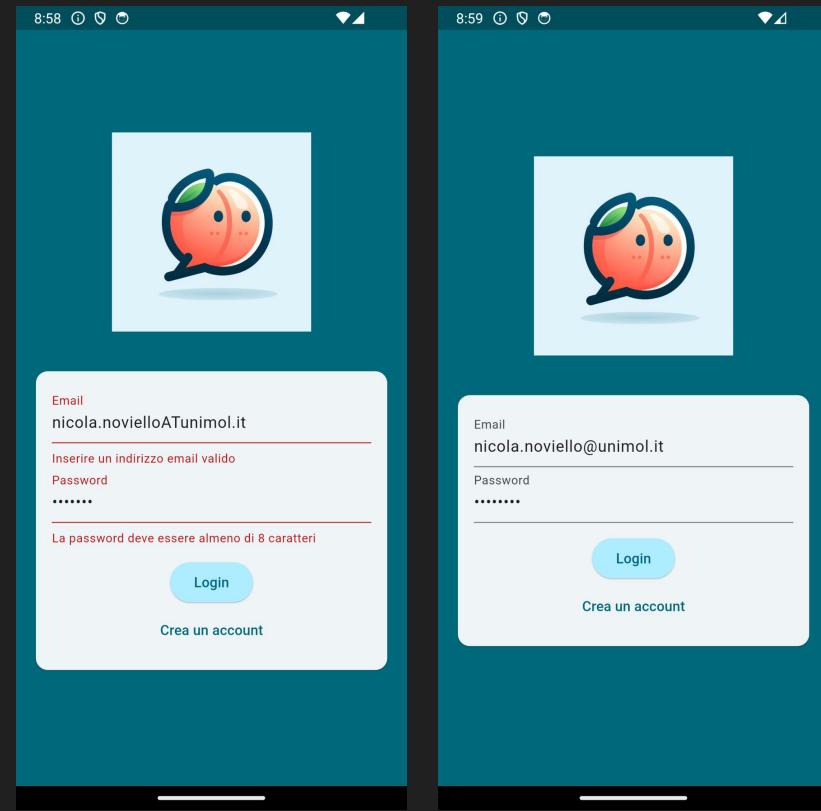
GlobalKey è un tipo di chiave in Flutter che fornisce un riferimento univoco a un widget specifico attraverso l'intero albero dei widget. È utile quando si desidera accedere allo stato di un widget da un'altra parte dell'app.

Nel codice, utilizzando **GlobalKey<FormState>** per creare un riferimento al Form, abbiamo la possibilità di chiamare i metodi validate e save sullo stato del modulo.

Il tutto viene fatto a fronte di una validazione implementata nei singoli **TextFormField**.

```
Card(
  margin: const EdgeInsets.all(20),
  child: SingleChildScrollView(
    child: Padding(
      padding: const EdgeInsets.all(16),
      child: Form(
        key: form,
        child: Column(
          mainAxisSize: MainAxisSize.min,
          children: [
            TextFormField(
              decoration:
                const InputDecoration(labelText: 'Email'),
              keyboardType: TextInputType.emailAddress,
              autocorrect: false,
              textCapitalization: TextCapitalization.none,
              validator: (value) {
                if (value == null ||
                    value.trim().isEmpty ||
                    !value.contains('@')) {
                  return 'Inserire un indirizzo email valido';
                }
                return null;
              },
              onSaved: (value) {
                _enteredEmail = value!;
              },
            ),
            TextFormField(
              decoration:
                const InputDecoration(labelText: 'Password'),
              obscureText: true,
              validator: (value) {
                if (value == null || value.trim().length < 8) {
                  return 'La password deve essere almeno di 8 caratteri';
                }
                return null;
              },
              onSaved: (value) {
                _enteredPassword = value!;
              },
            ),
            const SizedBox(height: 12),
            ElevatedButton(
              onPressed: _submit,
              style: ElevatedButton.styleFrom(
                backgroundColor: Theme.of(context)
                  .colorScheme
                  .primaryContainer,
              ),
              child: Text(_isLogin ? 'Login' : 'Registrati'),
            ),
            TextButton(
              onPressed: () {
                Navigator.pushNamed(context, '/password');
              },
            ),
          ],
        ),
      ),
    ),
  ),
)
```

Integrata la validazione attraverso il parametro **validator** dei **TextField**



Configurazione Firebase sul progetto

Autenticazione con API REST Firebase?

The screenshot shows a browser window displaying the Firebase REST Authentication API documentation. The URL is <https://firebase.google.com/docs/reference/rest/auth?hl=it>. The page is in Italian and has been translated by Google. It includes a sidebar with navigation links for various Firebase products like JavaScript, Flutter, Node.js, C++, Unity, Cloud Functions, Admin SDK, REST, RPC, and Security Rules. The main content area is titled "API REST di autenticazione Firebase" and contains sections for "Utilizzo dell'API", "Scambia token personalizzato con un ID e token di aggiornamento", and "Payload del corpo della richiesta". A sidebar on the right lists related topics such as "Utilizzo dell'API", "Scambia token personalizzato con un ID e token di aggiornamento", and "Invia feedback".

API REST di autenticazione Firebase

Utilizzo dell'API

Puoi eseguire query sul backend Firebase Auth tramite un'API REST. Questo può essere utilizzato per varie operazioni come la creazione di nuovi utenti, l'accesso a quelli esistenti e la modifica o l'eliminazione di questi utenti.

In questo documento, `API_KEY` fa riferimento alla chiave API Web, che può essere ottenuta nella pagina delle impostazioni del progetto nella console di amministrazione.

★ È richiesto HTTPS. Firebase risponde solo al traffico crittografato in modo che i tuoi dati rimangano al sicuro.

Scambia token personalizzato con un ID e token di aggiornamento

Puoi scambiare un token Auth personalizzato con un ID e un token di aggiornamento inviando una richiesta HTTP POST all'endpoint Auth `verifyCustomToken`.

Modo: POST

Tipo di contenuto: application/json

Endpoint

`https://identitytoolkit.googleapis.com/v1/accounts:signInWithCustomToken?key=[API_KEY]`

Payload del corpo della richiesta

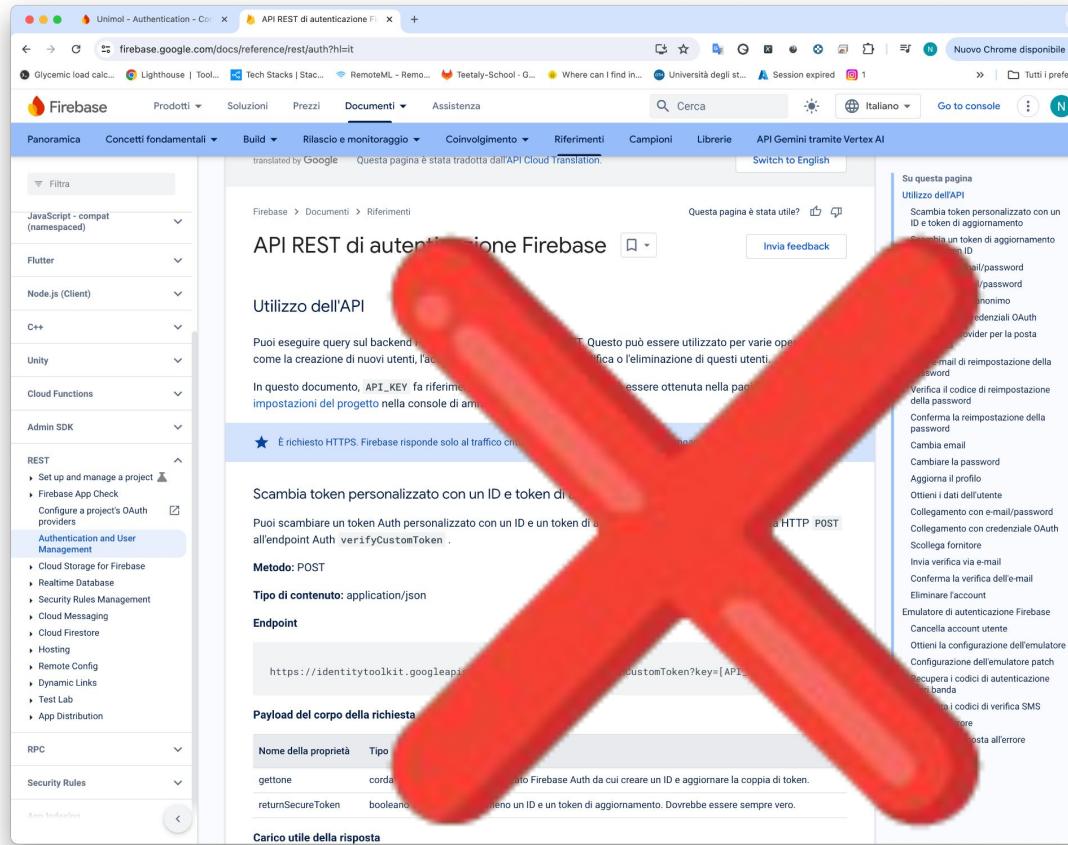
Nome della proprietà	Tipo	Descrizione
gettoken	corda	Un token personalizzato Firebase Auth da cui creare un ID e aggiornare la coppia di token.
returnSecureToken	booleano	Se restituire o meno un ID e un token di aggiornamento. Dovrebbe essere sempre vero.

Carico utile della risposta

Su questa pagina

- Utilizzo dell'API
- Scambia token personalizzato con un ID e token di aggiornamento
- Scambia un token di aggiornamento con un token ID
- Iscriviti con email/password
- Accedi con email/password
- Accedi in modo anonimo
- Accedi con le credenziali OAuth
- Recepirà i provider per la posta elettronica
- Invia e-mail di reimpostazione della password
- Verifica il codice di reimpostazione della password
- Conferma la reimpostazione della password
- Cambia email
- Cambiare la password
- Aggiorna il profilo
- Ottieni i dati dell'utente
- Collegamento con e-mail/password
- Collegamento con credenziali OAuth
- Scopri fornitore
- Invia verifica via e-mail
- Conferma la verifica dell'e-mail
- Eliminare l'account
- Emulatore di autenticazione Firebase
- Cancella account utente
- Ottieni la configurazione dell'emulatore
- Configurazione dell'emulatore patch
- Recepirà i codici di autenticazione fuori banda
- Recepirà i codici di verifica SMS
- Risposta di errore
- Formato di risposta all'errore

Autenticazione con API REST Firebase?



Usiamo l'SDK (Software Development Kit) apposita

The screenshot shows a browser window displaying the Firebase documentation at <https://firebase.google.com/docs/flutter/setup?hl=it&platform=ios>. The page title is "Aggiungi Firebase alla tua app Flutter". The left sidebar is titled "Concetti fondamentali" and includes sections for "Inizia a utilizzare Firebase", "Flutter", "Gestisci i progetti Firebase", "Plattforme e framework", "Prototipazione e test con Emulator Suite", and "Assistenza per l'IA con Gemini in Firebase". The main content area has a heading "Aggiungi Firebase alla tua app Flutter" with a subtitle "Add Firebase to your Flutter app: The Fast Way". It features a video thumbnail showing a person using a laptop with the title "Add Firebase to your Flutter app: The Fast Way". Below the video, there is a section titled "Prerequisiti" with a bulleted list of requirements:

- Installa il tuo editor o IDE preferito.
- Configura un dispositivo Apple fisico o utilizza un simulatore per eseguire la tua app.
- Vuoi utilizzare la messaggistica cloud?
- Assicurati che la tua app Flutter sia destinata alle seguenti versioni della piattaforma o successive:
 - iOS11
 - macOS 10.13
- Installa Flutter per il tuo sistema operativo specifico, incluso quanto segue:
 - SDK Flutter
 - Biblioteche di supporto
 - Software e SDK specifici della piattaforma
- Accedi a [Firebase](#) utilizzando il tuo account Google.

At the bottom of the page, it says: "Se non disponi già di un'app Flutter, puoi completare il [Get Started: Test Drive](#) per creare una nuova app Flutter utilizzando il tuo editor o IDE preferito."

Comandi da eseguire UNA SOLA VOLTA sulla macchina

Last login: Sun May 12 08:55:56 on ttys008

```
nico@Mac-mini-di-Nico ~ % curl -sL https://firebase.tools | bash
Password:
-- Checking for existing firebase-tools on PATH...
-- Checking your machine type...
-- Downloading binary from https://firebase.tools/bin/macos/latest
#####
##### 100.0%#####
#####
## Setting permissions on binary...
## Checking your PATH variable...
-- firebase-tools@13.9.0 is now installed
-- All Done!
nico@Mac-mini-di-Nico ~ % firebase login
I Firebase optionally collects CLI and Emulator Suite usage and error reporting information to help improve our products. Data is collected in accordance with Google's privacy policy (https://policies.google.com/privacy) and is not used to identify you.
? Allow Firebase to collect CLI and Emulator Suite usage and error reporting information? Yes
I To change your data collection preference at any time, run `firebase logout` and log in again.
Visit this URL on this device to log in:
https://accounts.google.com/o/oauth2/auth?client\_id=xxx
Waiting for authentication...
✓ Success! Logged in as nico.novello@gmail.com
nico@Mac-mini-di-Nico ~ % dart pub global activate flutterfire_cli
Downloading packages... (20.2s)
+ ansi_styles 0.3.2+1
...
+ yaml 3.1.2
Building package executables... (1.7s)
Built flutterfire_cli:flutterfire.
Installed executable flutterfire.
Warning: Pub installs executables into $HOME/.pub-cache/bin, which is not on your path.
You can fix that by adding this to your shell's config file (.zshrc, .bashrc, .bash_profile, etc.):
  export PATH="$PATH:$HOME/.pub-cache/bin"
Activated flutterfire_cli 1.0.0.
nico@Mac-mini-di-Nico ~ % vim .zshrc
nico@Mac-mini-di-Nico ~ % cat .zshrc
export PATH="$PATH:/Users/nico/flutter/bin"
export PATH="$PATH:$HOME/.pub-cache/bin"
nico@Mac-mini-di-Nico ~ %
```

Comandi da eseguire su ogni progetto

```
nico@Mac-mini-di-Nico ~ % cd flutter-proj/fireb
```

```
nico@Mac-mini-di-Nico fireb % flutterfire configure
```

ℹ Found 2 Firebase projects.

✓ Select a Firebase project to configure your Flutter application with · `unimol-44e4a (Unimol)`

? Which platforms should your configuration support (use arrow keys & space to select) · ✓ Which platforms should your configuration support (use arrow keys & space to select)? · `android`

? Which Android application id (or package name) do you want to use for this configuration? · ✓ Which Android application id (or package name) do you want to use for this configuration, e.g. '`com.example.app`'? · `com.example.appfireunimol`

ℹ Firebase `android` app `com.example.appfireunimol` is not registered on Firebase project `unimol-44e4a`.

ℹ Registered a new Firebase `android` app on Firebase project `unimol-44e4a`.

Firebase configuration file `lib.firebaseio_options.dart` generated successfully with the following Firebase apps:

Platform Firebase App Id

android xxx

Learn more about using this file and next steps from the documentation:

> <https://firebase.google.com/docs/flutter/setup>

```
nico@Mac-mini-di-Nico fireb % flutter pub add firebase_core
```

Resolving dependencies...

Downloading packages...

+ `firebase_core` 2.31.0

...

```
nico@Mac-mini-di-Nico fireb %
```

Comandi da eseguire su ogni progetto

Inoltre è utile eseguire anche “**flutter pub add firebase_auth**” se si vuole usare l’SDK specifico per l’autenticazione.

Aggiorno il main.dart

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

import 'firebase_options.dart';
import 'package:fireb/screens/auth.dart';

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

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'FlutterChat',
      theme: ThemeData().copyWith(
        colorScheme:
          ColorScheme.fromSeed(seedColor: Color.fromARGB(255, 223, 243, 250)),
      ),
      home: const AuthScreen(),
    );
}
```

Registrazione dell'utente

Registrazione in AuthScreen

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

final _firebase = FirebaseAuth.instance;

class AuthScreen extends StatefulWidget {
  const AuthScreen({super.key});

  @override
  State<AuthScreen> createState() {
    return _AuthScreenState();
  }
}

class _AuthScreenState extends State<AuthScreen> {
  final _form = GlobalKey<FormState>();

  var _isLogin = true;
  var _enteredEmail = '';
  var _enteredPassword = '';

  void _submit() async {
    final isValid = _form.currentState!.validate();

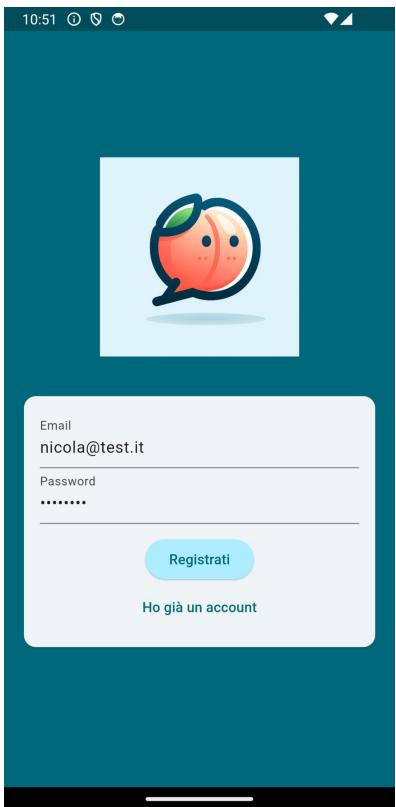
    if (!isValid) {
      return;
    }

    _form.currentState!.save();

    if (_isLogin) {
      // log users in
    } else {
      try {
        final userCredentials = await _firebase.createUserWithEmailAndPassword(
          email: _enteredEmail, password: _enteredPassword);
        print(userCredentials);
      } on FirebaseAuthException catch (error) {
        if (error.code == 'email-already-in-use') {
          // ...
        }
        if (mounted) {
          ScaffoldMessenger.of(context).clearSnackBars();
          ScaffoldMessenger.of(context).showSnackBar(
            SnackBar(
              content: Text(error.message ?? 'Autenticazione fallita'),
            ),
          );
        }
      }
    }
  }
}
```

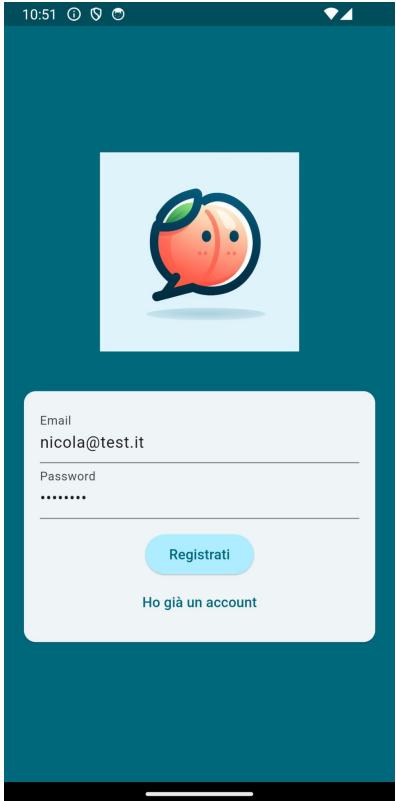
- Il codice controlla se l'utente sta cercando di accedere o di registrarsi attraverso la variabile `_isLogin`, al momento il codice per la login non è ancora implementato;
- Se `_isLogin` è false, allora l'utente sta cercando di registrarsi. Il codice tenta di creare un nuovo utente con l'email e la password fornite (`_enteredEmail` e `_enteredPassword`) utilizzando il metodo `createUserWithEmailAndPassword` di Firebase;
- Se la creazione dell'utente ha successo, le credenziali dell'utente vengono stampate sulla console;
- Se si verifica un errore durante la creazione dell'utente (ad esempio, se l'email fornita è già in uso), l'errore viene catturato e gestito nel blocco `catch`. In particolare, se l'errore è che l'email è già in uso, saranno implementate delle funzioni specifiche;
- Se l'errore è di qualsiasi altro tipo, il codice mostra un messaggio di errore all'utente utilizzando un widget `SnackBar`. Il messaggio di errore è il messaggio dell'eccezione catturata, o la stringa '**Autenticazione fallita**' se il messaggio dell'eccezione è null.

Il risultato



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Filter (e.g. test, exclude|escape) Flutter (Pixel_3a_API_34, )
```

Il risultato

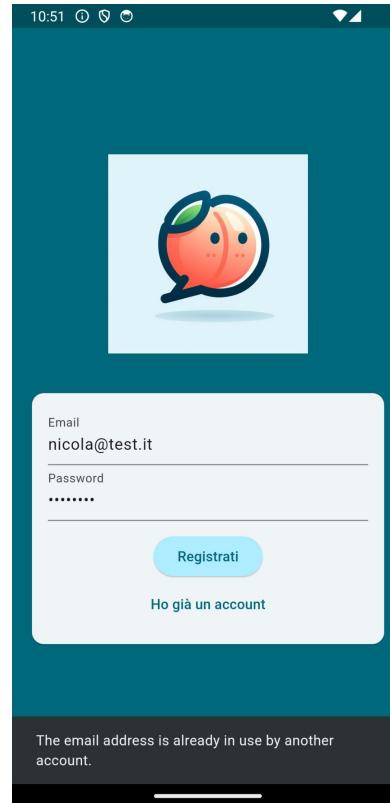


A screenshot of the Firebase Authentication console in a web browser. The URL is console.firebaseio.google.com/u/0/project/unimol-44e4a/authentication/users. The page shows a table of users:

Identificatore	Provider	Data di creazione	Accesso eseguito	UID utente
nicola@test.it	✉️	20 mag 2024	20 mag 2024	QzFCG0vsJGebonvvM4mh0gq...

At the bottom of the page, there is a message: "Spark Nessun costo 0 \$/mese Esegui l'upgrade".

E se provo a registrarmi con la stessa email?



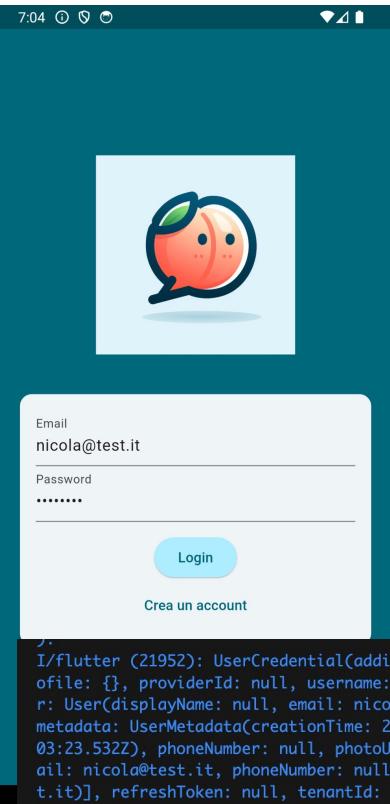
Login dell'utente

Login in AuthScreen

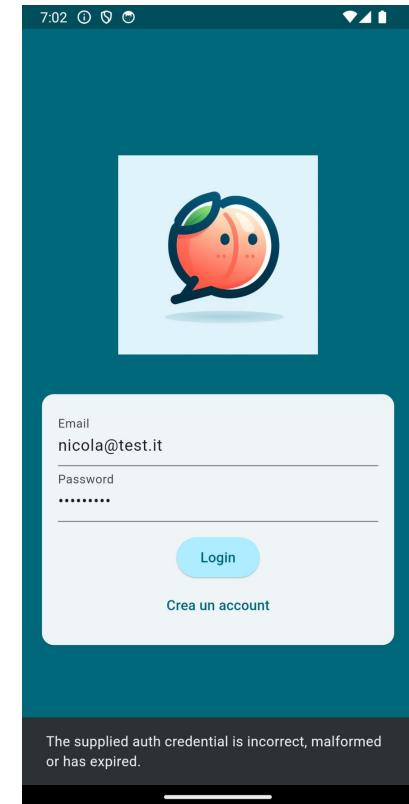
```
try {
  if (_isLogin) {
    final userCredentials = await _firebase.signInWithEmailAndPassword(
      email: _enteredEmail, password: _enteredPassword);
    print(userCredentials);
  } else {
    final userCredentials = await _firebase.createUserWithEmailAndPassword(
      email: _enteredEmail, password: _enteredPassword);
    print(userCredentials);
  }
} on FirebaseAuthException catch (error) {
  if (error.code == 'email-already-in-use') {
    // ...
  }
  if (mounted) {
    ScaffoldMessenger.of(context).clearSnackBars();
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
        content: Text(error.message ?? 'Autenticazione fallita'),
      ),
    );
  }
}
```

- Il codice controlla usa il metodo **signInWithEmailAndPassword** per effettuare la login con le credenziali dell'utente.
- il **try** è stato spostato su tutto il blocco, poiché le eccezioni di entrambe le funzionalità (registrazione e login) sono analoghe,

Il risultato



Se volessi gestire
l'errore in italiano?



Persistenza dei token derivanti dalla login

Nuova schermata ChatScreen

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

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('PescheChat'),
      ),
      body: const Center(
        child: Text('Hai già effettuato il login!'),
      ),
    );
  }
}
```

- Al momento presenta solo un widget Text
- Sarà mostrato agli utenti che hanno già effettuato il login

Aggiorno il main.dart

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

import 'firebase_options.dart';
import 'package:fireb/screens/auth.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:fireb/screens/chat.dart';

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

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'PescheChat',
      theme: ThemeData().copyWith(
        colorScheme:
          ColorScheme.fromSeed(seedColor: Color.fromARGB(255, 223, 243, 250)),
      ),
      home: StreamBuilder(
        stream: FirebaseAuth.instance.authStateChanges(),
        builder: (ctx, snapshot) {
          if (snapshot.hasData) {
            return const ChatScreen();
          }
          return const AuthScreen();
        },
      ),
    );
  }
}
```

- **StreamBuilder** è un widget che prende uno **Stream** come input e ricostruisce la sua interfaccia utente ogni volta che riceve un nuovo evento da quello Stream. In questo caso, lo Stream è fornito dal metodo **authStateChanges** di **FirebaseAuth.instance**, che emette un evento ogni volta che lo stato di autenticazione cambia (ad esempio, quando un utente si autentica o si disconnette);
- Il widget **StreamBuilder** utilizza una funzione builder per costruire la sua interfaccia utente. Questa funzione viene chiamata ogni volta che lo Stream emette un nuovo evento. La funzione builder prende due argomenti: un **BuildContext** e uno **AsyncSnapshot**, che contiene i dati dell'ultimo evento emesso dallo Stream;
- La funzione builder in questo caso controlla se lo snapshot ha dei dati. Se ha dei dati, significa che un utente è attualmente autenticato, quindi restituisce il widget **ChatScreen**. Se non ha dati, significa che nessun utente è attualmente autenticato, quindi restituisce il widget **AuthScreen**;
- Uno Stream è simile a un Future ma la differenza principale è che **StreamBuilder** può gestire una serie di eventi nel tempo, mentre **FutureBuilder** gestisce un singolo evento futuro. In altre parole StreamBuilder viene usato quando si vuole rispondere a più eventi nel tempo (come i cambiamenti dello stato di autenticazione) mentre FutureBuilder si attende un risultato che può essere una risposta positiva o un errore.

Nuova schermata SplashScreen

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

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('PescheChat'),
      ),
      body: const Center(
        child: Text('Caricamento'),
      ),
    );
  }
}
```

- Presenta solo un widget Text, ma può essere personalizzato in autonomia;
- Sarà mostrato quando le verifiche di Firebase sull'autenticazione non sono immediate.

Aggiorno il main.dart per gestire la SplashScreen

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

import 'firebase_options.dart';
import 'package:fireb/screens/auth.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:fireb/screens/chat.dart';
import 'package:fireb/screens/splash.dart';

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

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'PescheChat',
      theme: ThemeData().copyWith(
        colorScheme:
          ColorScheme.fromSeed(seedColor: Color.fromARGB(255, 223, 243, 250)),
      ),
      home: StreamBuilder(
        stream: FirebaseAuth.instance.authStateChanges(),
        builder: (ctx, snapshot) {
          if (snapshot.connectionState == ConnectionState.waiting) {
            return const SplashScreen();
          }

          if (snapshot.hasData) {
            return const ChatScreen();
          }

          return const AuthScreen();
        },
      ),
    );
  }
}
```

- Cosa fa il codice evidenziato?

Logout in ChatScreen

- Funziona sia quando l'utente si slogga ma anche se l'utente viene eliminato da Firebase

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

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('PescheChat'),
        actions: [
          IconButton(
            onPressed: () {
              FirebaseAuth.instance.signOut();
            },
            icon: Icon(
              Icons.exit_to_app,
              color: Theme.of(context).colorScheme.primary,
            ),
          ),
        ],
      ),
      body: const Center(
        child: Text('Hai già effettuato il login!'),
      ),
    );
  }
}
```



Esercizio

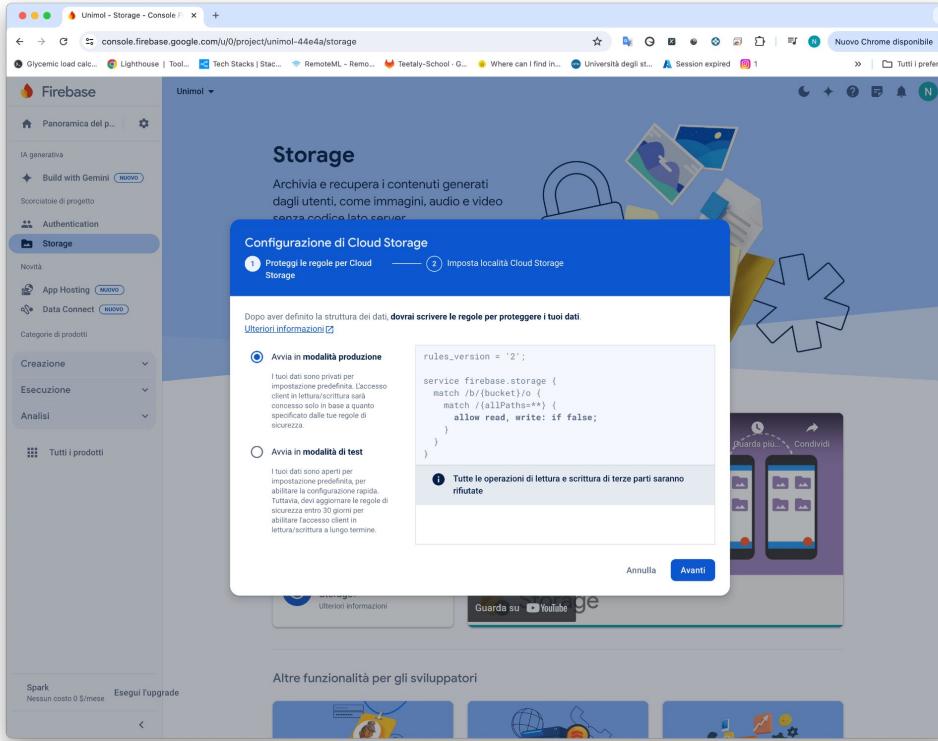
Il codice login.zip contiene il progetto svolto fino ad ora. Provate a configurare Firebase e verificate se, una volta eliminato un utente da firebase, il sistema di logout automatico funziona. **Attenzione: il progetto contiene solo i file della cartella lib e gli asset, il resto va costruito con i comandi visti fino ad ora.**

Storage su Firebase

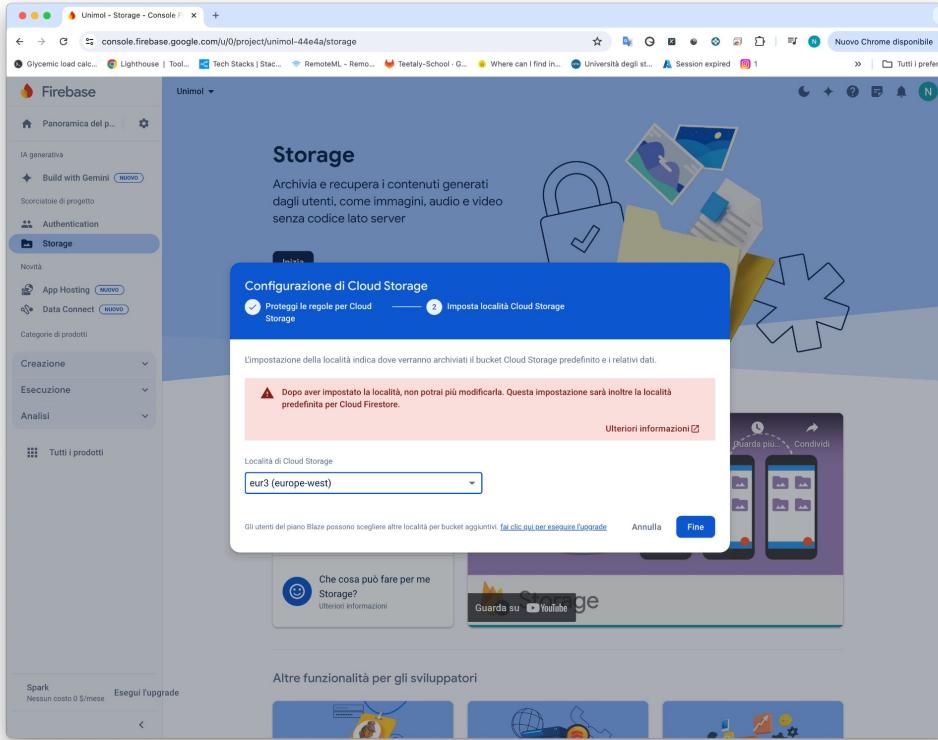
Storage su Firebase

The screenshot shows the Firebase Storage console interface for the project "unimol-44e4a". The left sidebar lists various services: Panoramica del progetto, IA generativa, Build with Gemini, Scorrivole di progetto, Authentication, Storage (selected), Novità, App Hosting, Data Connect, Creazione, Esecuzione, Analisi, and Tutti i prodotti. The main content area is titled "Storage" and contains the following text: "Archivia e recupera i contenuti generati dagli utenti, come immagini, audio e video senza codice lato server". Below this is a large illustration of a yellow folder containing documents and a lock icon. A call-to-action button labeled "Inizia" is present. To the right of the illustration is a section titled "Ulteriori informazioni" with three items: "Che cosa devo fare per iniziare?", "Come funziona Storage?", and "Che cosa può fare per me Storage?". Below this is a video thumbnail titled "Introducing Cloud Storage for Firebase" with a "Guarda più tardi" button. At the bottom, there's a section for "Altre funzionalità per gli sviluppatori" featuring three icons. At the very bottom, it says "Spark Nessun costo 0 \$/mese" and "Esegui l'upgrade".

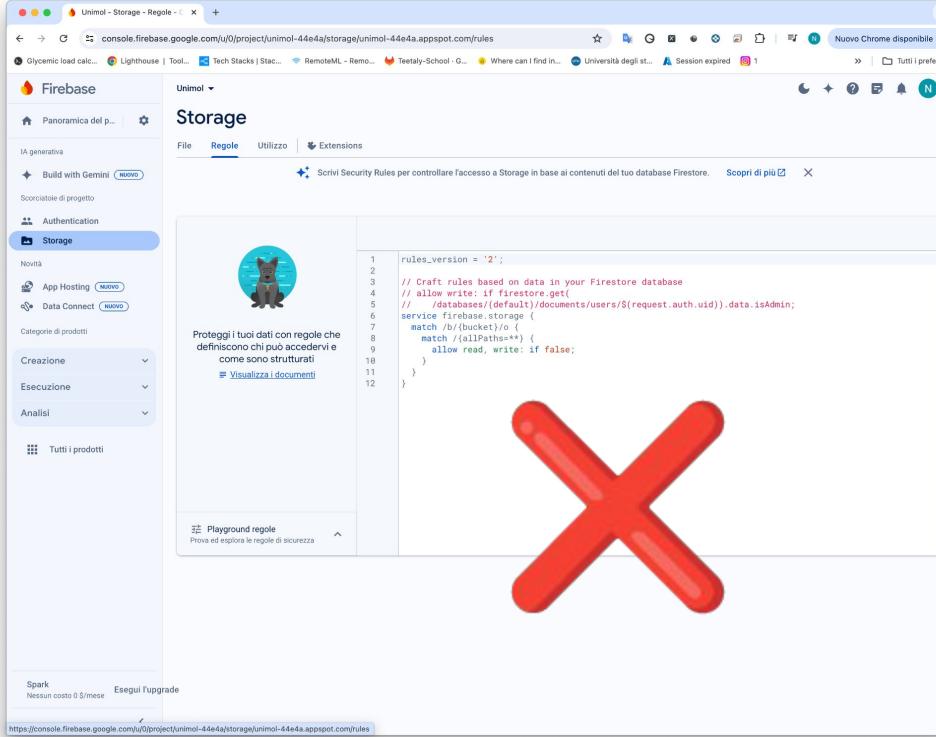
Storage su Firebase



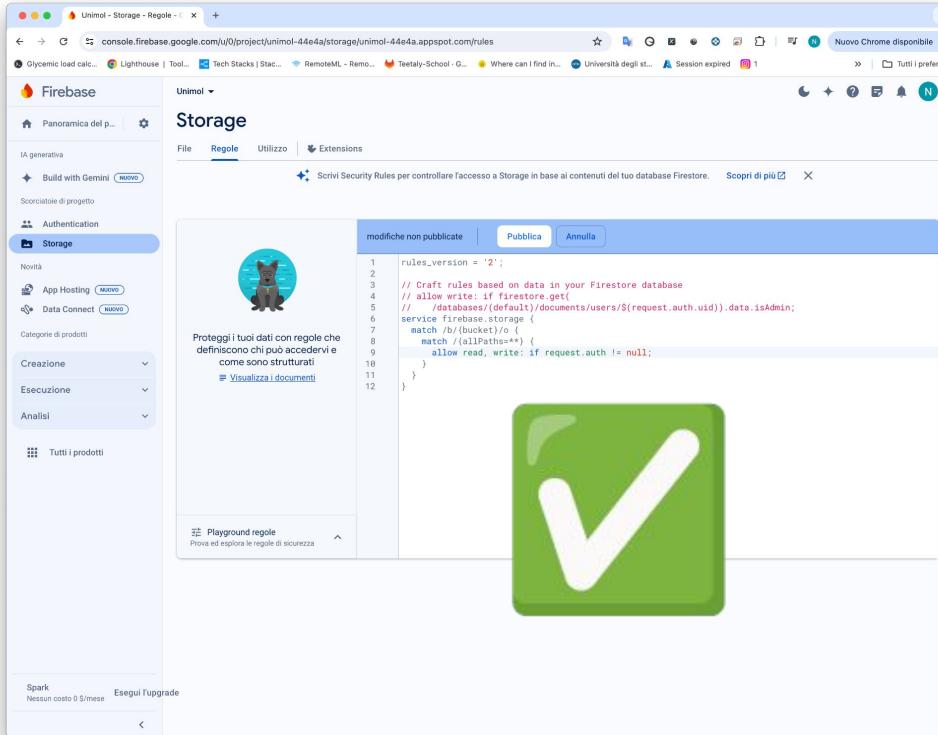
Storage su Firebase



Sicurezza sullo Storage



Sicurezza sullo Storage



https://pub.dev/packages/firebase_storage

The screenshot shows the pub.dev package page for `firebase_storage` version 11.7.5. At the top, there's a header with the pub.dev logo, a search bar, and a sign-in button. Below the header, the package name `firebase_storage` is displayed along with its version `11.7.5`. It indicates that it was published 14 days ago and is Dart 3 compatible. The package is categorized under `SDK`, `FLUTTER`, `PLATFORM`, `ANDROID`, `IOS`, `MACOS`, `WEB`, and `WINDOWS`. It has 1.5K stars and 1545 likes. On the right side, there are sections for the publisher (`firebase.google.com`), metadata (described as a Flutter plugin for Firebase Cloud Storage), homepage, repository, contributing, documentation, API reference, license (BSD-3-Clause), dependencies (listing `firebase_core`, `firebase_core_platform_interface`, `firebase_storage_platform_interface`, `firebase_storage_web`, and `flutter`), and issues and feedback.

Dovendo anche
caricare le immagini
forse abbiamo bisogno
anche di un altro
Package...

Aggiungiamo un nuovo widget: UserImagePicker

Concettualmente, a livello di organizzazione progettuale, è possibile pensare di dividere i widget dagli screen. Di fatto si tratta sempre di Widget, ma i primi hanno una funzionalità specifica o possono essere inclusi nei secondi. I secondi rappresentano una schermata singola.

A livello di organizzazione di progetto questa suddivisione (widget, screen, model, provider, etc.) può aiutare nella leggibilità del codice.

UserImagePicker

```
import 'dart:io';

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

class UserImagePicker extends StatefulWidget {
  const UserImagePicker({
    super.key,
    required this.onPickImage,
  });

  final void Function(File pickedImage) onPickImage;

  @override
  State<UserImagePicker> createState() {
    return _UserImagePickerState();
  }
}

class _UserImagePickerState extends State<UserImagePicker> {
  File? _pickedImageFile;

  void _pickImage() async {
    final pickedImage = await ImagePicker().pickImage(
      source: ImageSource.camera,
      imageQuality: 50,
      maxWidth: 150,
    );
    if (pickedImage == null) {
      return;
    }
    setState(() {
      _pickedImageFile = File(pickedImage.path);
    });
    widget.onPickImage(_pickedImageFile!);
  }

  @override
  ...
```

- Questo widget permette all'utente di scattare una foto con la fotocamera del dispositivo e la mostra in un **CircleAvatar**;
- **UserImagePicker** prende un parametro obbligatorio **onPickImage**, che è una funzione che viene chiamata ogni volta che l'utente scatta una foto. Questa funzione prende come argomento il file dell'immagine scattata;
- La classe **_UserImagePickerState** definisce una variabile di stato **_pickedImageFile**, che tiene traccia del file dell'ultima immagine scattata dall'utente.
- Il metodo **_pickImage** è responsabile dell'apertura della fotocamera del dispositivo e permettere all'utente di scattare una foto, basandosi sul funzionamento di **image_picker**;
- Se l'utente scatta una foto, il metodo **_pickImage** aggiorna **_pickedImageFile** con il file dell'immagine scattata e chiama **onPickImage** con il file dell'immagine.

UserImagePicker

```
@override
Widget build(BuildContext context) {
  return Column(
    children: [
      CircleAvatar(
        radius: 40,
        backgroundColor: Colors.grey,
        foregroundImage:
          _pickedImageFile != null ? FileImage(_pickedImageFile!) : null,
      ),
      TextButton.icon(
        onPressed: _pickImage,
        icon: const Icon(Icons.image),
        label: Text(
          'Aggiungi un\' Immagine',
          style: TextStyle(
            color: Theme.of(context).colorScheme.primary,
          ),
        ),
      ),
    ],
  );
}
```

- Il metodo build di **_UserImagePickerState** definisce l'interfaccia utente del widget UserImagePicker. Questa interfaccia utente consiste in un **CircleAvatar** che mostra l'immagine scattata dall'utente (o uno sfondo di colore grigio se nessuna immagine è stata scattata) e un **TextButton** che permette all'utente di scattare una foto quando viene premuto.

Integriamo il codice in AuthScreen

```
import 'package:flutter/material.dart';
import 'dart:io';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:fireb/widgets/user_image_picker.dart';

final _firebase = FirebaseAuth.instance;

class AuthScreen extends StatefulWidget {
  const AuthScreen({super.key});

  @override
  State<AuthScreen> createState() {
    return _AuthScreenState();
  }
}

class _AuthScreenState extends State<AuthScreen> {
  final _form = GlobalKey<FormState>();

  var _isLogin = true;
  var _enteredEmail = '';
  var _enteredPassword = '';
  File? _selectedImage;

  void _submit() async {
    final isValid = _form.currentState!.validate();

    if (!isValid || !_isLogin && _selectedImage == null) {
      // mostriamo un messaggio di errore
      // se le credenziali non sono valide
      // se prova a registrarsi senza aver inserito un'immagine
      return;
    }

    _form.currentState!.save();
    try {
      if (_isLogin) {
        final userCredentials = await _firebase.signInWithEmailAndPassword(
          email: _enteredEmail, password: _enteredPassword);
        print(userCredentials);
      } else {
        final userCredentials = await _firebase.createUserWithEmailAndPassword(
          email: _enteredEmail, password: _enteredPassword);
        print(userCredentials);
      }
    } on FirebaseAuthException catch (error) {
      if (error.code == 'email-already-in-use') {
        // ...
      }
    }
  }
}
```

- Il controllo **!isValid** verifica se un determinato stato o un insieme di condizioni è valido. La variabile **isValid** è un booleano che viene impostato a *true* se tutti i campi di input (come email e password) sono validi e a *false* altrimenti. Se **isValid** è *false*, il codice non procede oltre;
- Il controllo **_isLogin && _selectedImage == null** verifica se l'utente sta cercando di registrarsi (indicato da **_isLogin** che è *false*) e se non ha selezionato un'immagine (indicato da **_selectedImage** che è *null*). Se entrambe queste condizioni sono vere, il codice non procede oltre;
- Questo controllo impedisce all'utente di procedere se i campi di input non sono validi o se l'utente sta cercando di registrarsi ma non ha selezionato un'immagine. Se una di queste condizioni è vera, al momento non fa niente ma potrà essere implementato un messaggio di errore (come indicato dal commento) e il codice esce dalla funzione corrente con un **return**.

```

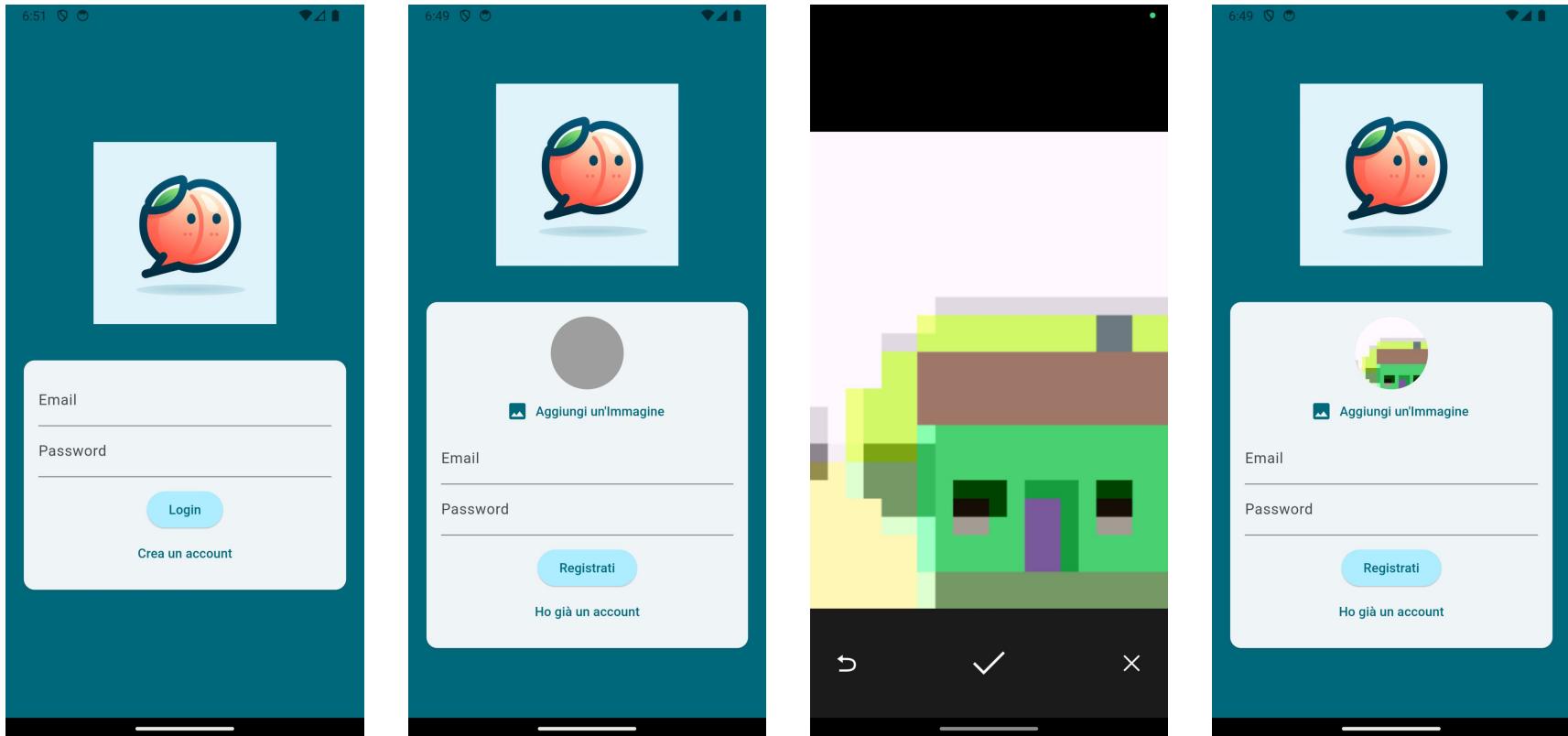
if (mounted) {
  ScaffoldMessenger.of(context).clearSnackBars();
  ScaffoldMessenger.of(context).showSnackBar(
    SnackBar(
      content: Text(error.message ?? 'Autenticazione fallita'),
    ),
  );
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: Theme.of(context).colorScheme.primary,
    body: Center(
      child: SingleChildScrollView(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Container(
              margin: const EdgeInsets.only(
                top: 30,
                bottom: 20,
                left: 20,
                right: 20,
              ),
              width: 200,
              child: Image.asset('assets/images/chat.jpeg'),
            ),
            Card(
              margin: const EdgeInsets.all(20),
              child: SingleChildScrollView(
                child: Padding(
                  padding: const EdgeInsets.all(16),
                  child: Form(
                    key: _form,
                    child: Column(
                      mainAxisAlignment: MainAxisAlignment.min,
                      children: [
                        if (!_isLoggedIn)
                          UserImagePicker(
                            onPickImage: (pickedImage) {
                              _selectedImage = pickedImage;
                            },
                          ),
                        TextFormField(
                          decoration:
                            const InputDecoration(labelText: 'Email'),
                          keyboardType: TextInputType.emailAddress,
                          autocorrect: false,
                          textCapitalization: TextCapitalization.none,
                          validator: (value) {
                            if (value == null ||
                                value.trim().isEmpty ||
                                !value.contains('@')) {
                              return 'Inserire un indirizzo email valido';
                            }
                          }
                        )
                      ],
                    ),
                  ),
                ),
              ),
            ),
          ],
        ),
      ),
    ),
  );
}

```

- Se `_isLoggedIn` è false (cioè l'utente sta cercando di registrarsi), il codice crea un nuovo widget **UserImagePicker**. Questo widget permette all'utente di selezionare un'immagine, da utilizzare come immagine del profilo;
- Il widget **UserImagePicker** prende un parametro chiamato **onPickImage**, che è una funzione che viene chiamata quando l'utente seleziona un'immagine. In questo caso, la funzione assegna l'immagine selezionata alla variabile `_selectedImage`.

Il risultato





Upload di
un'immagine
Firebase

Integriamo il codice in AuthScreen

```
import 'package:flutter/material.dart';
import 'dart:io';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:widgets/user_image_picker.dart';
import 'package:firebase_storage/firebase_storage.dart';

final _firebase = FirebaseAuth.instance;

class AuthScreen extends StatefulWidget {
  const AuthScreen({super.key});

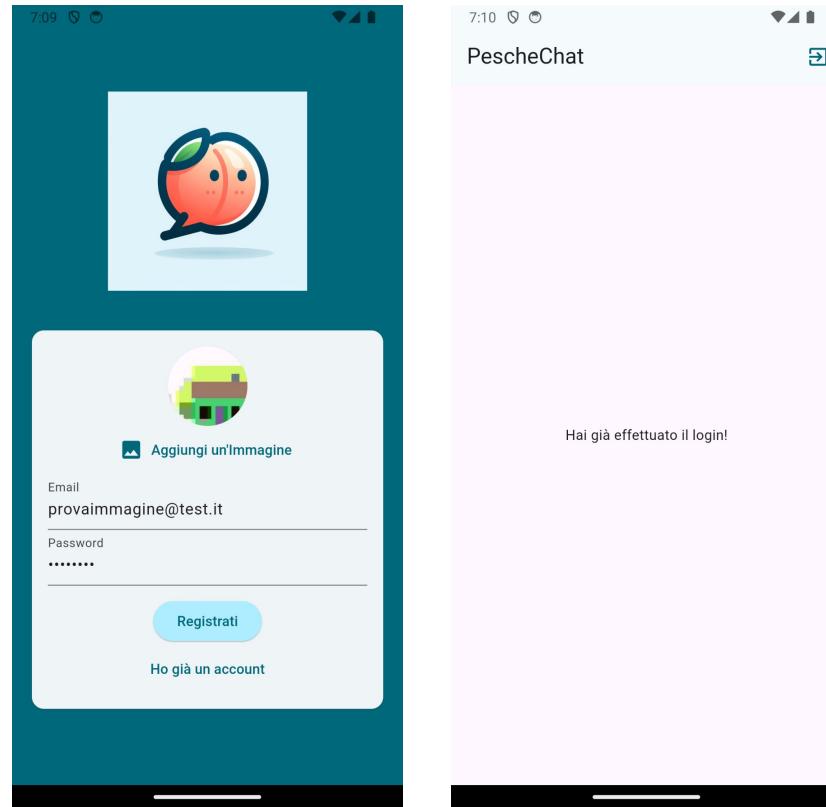
  - try {
    if (_isLogin) {
      final userCredentials = await _firebase.signInWithEmailAndPassword(
          email: _enteredEmail, password: _enteredPassword);
    } else {
      final userCredentials = await _firebase.createUserWithEmailAndPassword(
          email: _enteredEmail, password: _enteredPassword);
      final storageRef = FirebaseStorage.instance
          .ref()
          .child('user_images')
          .child('${userCredentials.user!.uid}.jpg');

      await storageRef.putFile(_selectedImage!);
      final imageUrl = await storageRef.getDownloadURL();
      print(imageUrl);
    }
  } on FirebaseAuthException catch (error) {
```

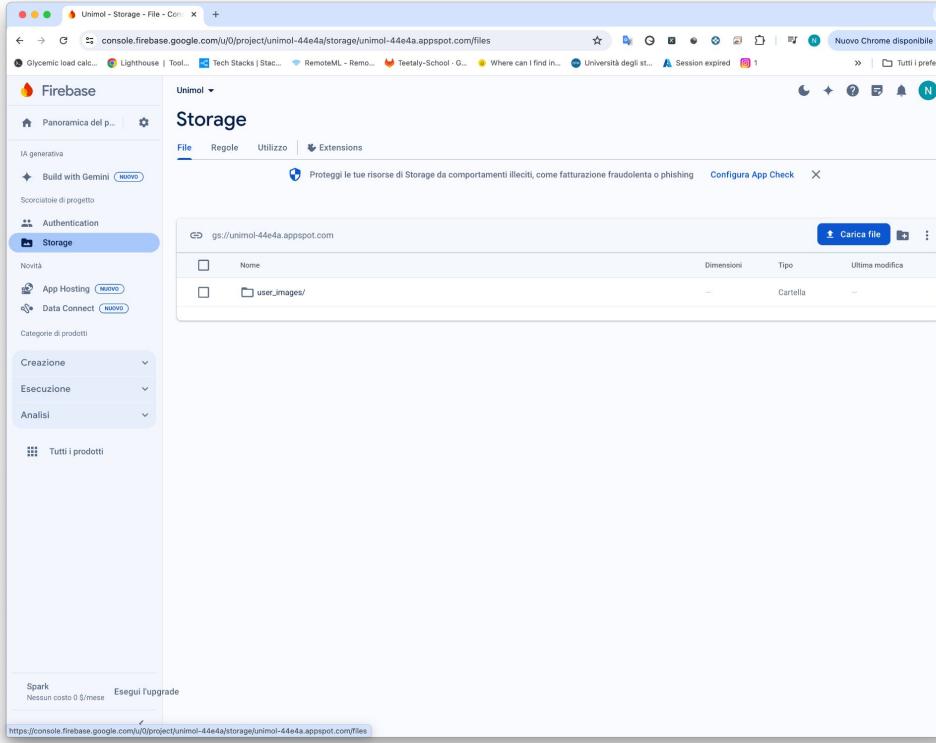
Il codice integrato carica un'immagine su Firebase Storage e ottiene un URL per scaricare l'immagine e lo fa in questo modo:

- **FirebaseStorage.instance.ref().child('user_images').child('\${userCredentials.user!.uid}.jpg');** crea un riferimento a un file nel bucket di storage di Firebase. Il file si trova nella directory 'user_images' e il suo nome è l'ID dell'utente con l'estensione '.jpg'. L'ID dell'utente viene ottenuto da userCredentials.user!.uid;
- **await storageRef.putFile(_selectedImage!);** Carica l'immagine selezionata (_selectedImage) al file a cui il riferimento (storageRef) punta. L'operazione di caricamento è asincrona, quindi usa await per aspettare che l'operazione sia completata prima di procedere;
- **final imageUrl = await storageRef.getDownloadURL();** - Dopo che l'immagine è stata caricata, viene acquisito l'URL per scaricare l'immagine dal bucket di storage. Questo URL sarà utilizzato per visualizzare l'immagine in un secondo momento.

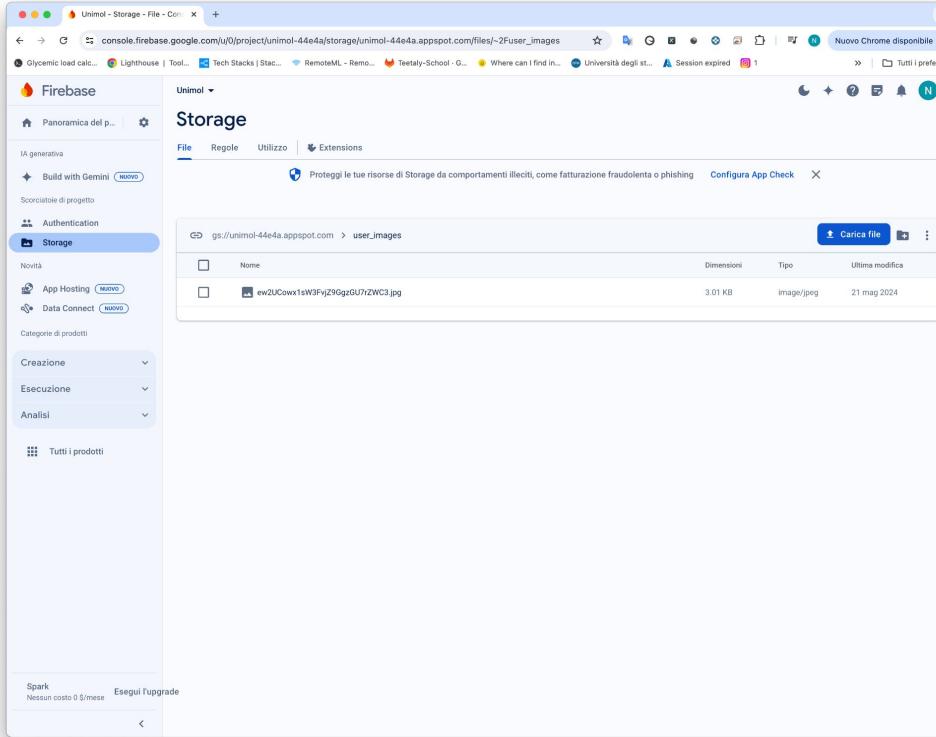
Il risultato



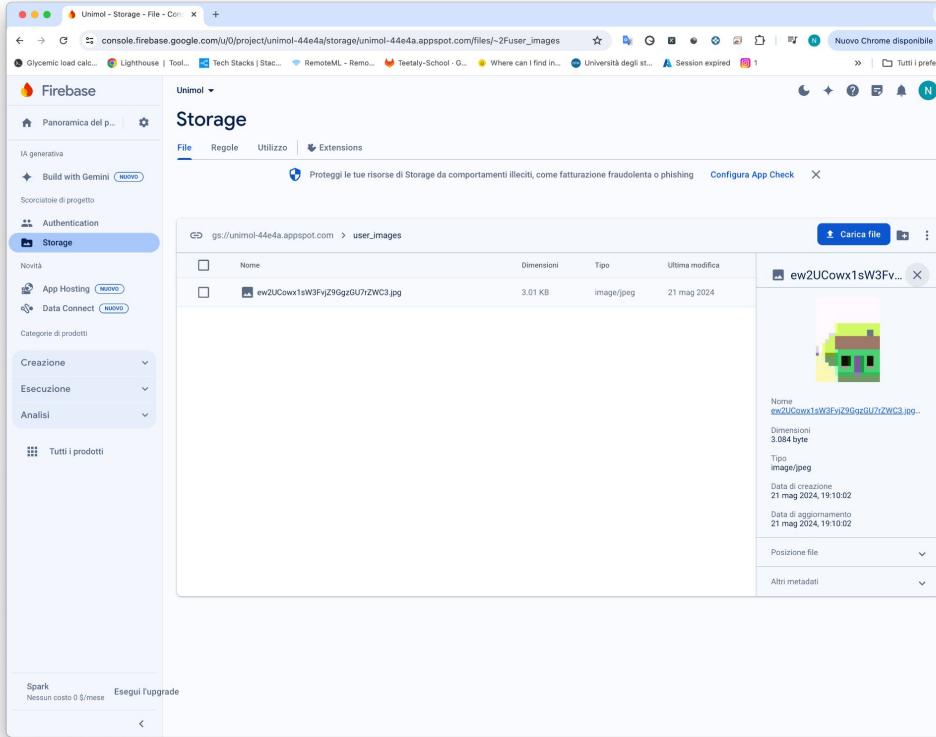
Il risultato



Il risultato



Il risultato



Il risultato

The screenshot shows the Firebase Authentication console for the project "Unimol". The left sidebar lists various services: IA generativa, Build with Gemini (selected), Scorciatoie di progetto (Authentication, Storage, App Hosting, Data Connect), Categorie di prodotti (Creazione, Esecuzione, Analisi), and Tutti i prodotti.

The main page displays the "Authentication" section under "Unimol". It includes tabs for Utenti, Metodo di accesso, Modelli, Utilizzo, Impostazioni, and Extensions. A search bar at the top allows filtering by email, phone number, or UID. An "Aggiungi utente" button is visible.

A table lists the users:

Identificatore	Provider	Data di creazione	Accesso eseguito	UID utente
provaimmagine@test.it	✉️	21 mag 2024	21 mag 2024	ew2UCowx1sW3FyjZ9GgzGU7...
nicola@test.it	✉️	20 mag 2024	21 mag 2024	QzFCGcvslGebonvvM4ml0gq...

At the bottom, it says "Spark Nessun costo \$/mese" and "Esegui l'upgrade".

Esercizio

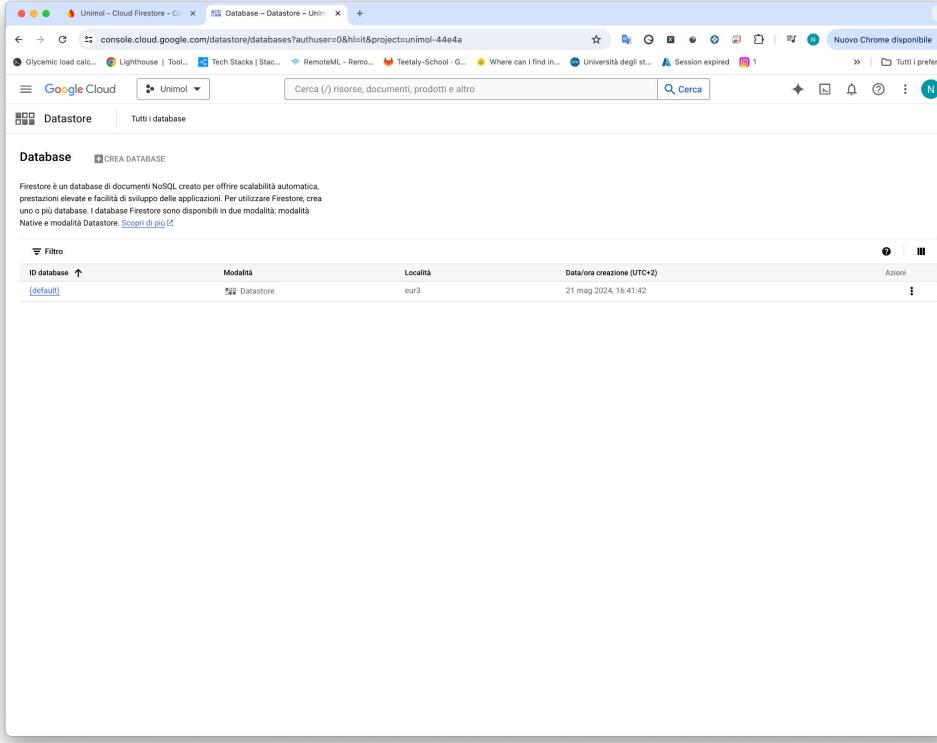
Il codice image.zip contiene il progetto visto fin d'ora con una piccola modifica.
Provate a fare una build e ditemi cosa ho integrato!

Gestione di un Database remoto

Database Documentale: Cloud Firestore

The screenshot shows the Firebase Cloud Firestore dashboard. On the left, a sidebar lists various services: Panoramica del progetto, IA generativa, Build with Gemini (nuovo), Scorsiziore di progetto, Authentication, Storage, **Firebase Database**, Novità, App Hosting (nuovo), Data Connect (nuovo), Categorie di prodotti, Creazione, Esecuzione, Analisi, and Tutti i prodotti. The main content area is titled "Cloud Firestore" and features a large illustration of a stack of servers with a magnifying glass focusing on one. Below the illustration, a message states: "Questo progetto è configurato per utilizzare Cloud Firestore in modalità Datastore. Questa modalità è accessibile solo da Google Cloud Console." A "Vai a Google Cloud Console" button is present. To the right of the message, there's a section titled "Ulteriori informazioni" with three items: "Che cosa devo fare per iniziare?", "Quanto costerà Cloud Firestore?", and "Che cosa può fare per me Cloud Firestore?". Below this is a video thumbnail titled "Introducing Cloud Firestore" with a "Guarda su YouTube" button. At the bottom, there's a section titled "Altre funzionalità per gli sviluppatori" with three small icons representing different developer tools.

Database Documentale: Cloud Firestore



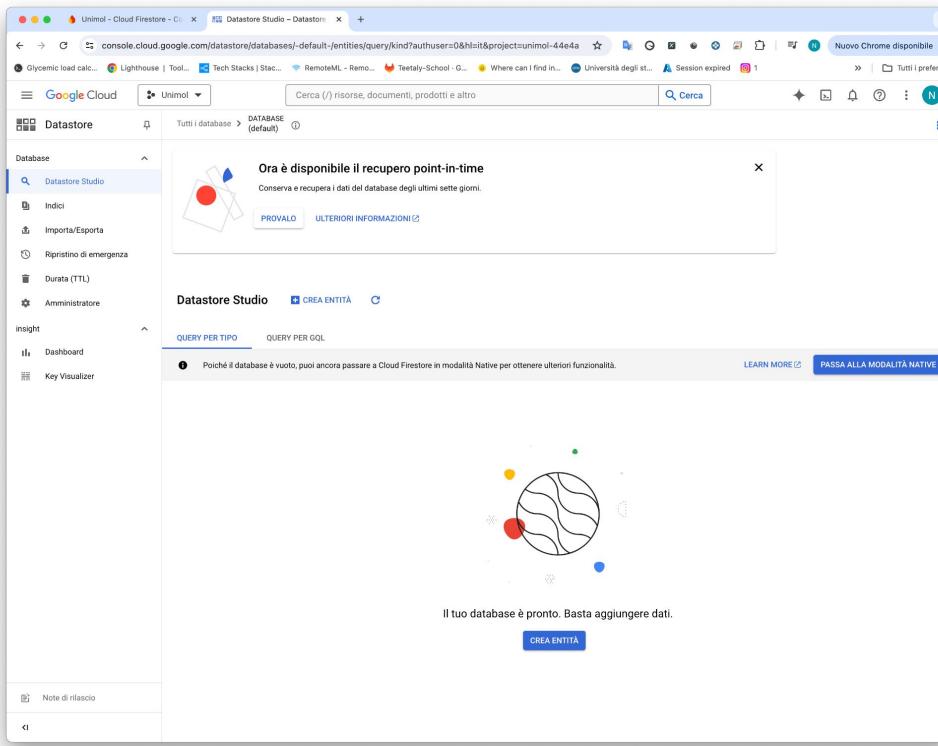
The screenshot shows the Google Cloud Platform interface for managing Firestore databases. The browser window title is "Unimol - Cloud Firestore - Unimol - Database - Firestore - Unimol". The URL is "console.cloud.google.com/firestore/databases?authuser=0&hl=it&project=unimol-44e4a". The page header includes the Google Cloud logo, the project name "Unimol", a search bar, and a "Cerca" button.

The main content area is titled "Database" and features a "CREA DATABASE" button. Below this, there is a brief description of Firestore: "Firestore è un database di documenti NoSQL creato per offrire scalabilità automatica, prestazioni elevate e facilità di sviluppo delle applicazioni. Per utilizzare Firestore, crea uno o più database e database Firestore sono disponibili in due modalità: modalità Native e modalità Datastore." A link "Scopri di più" is provided.

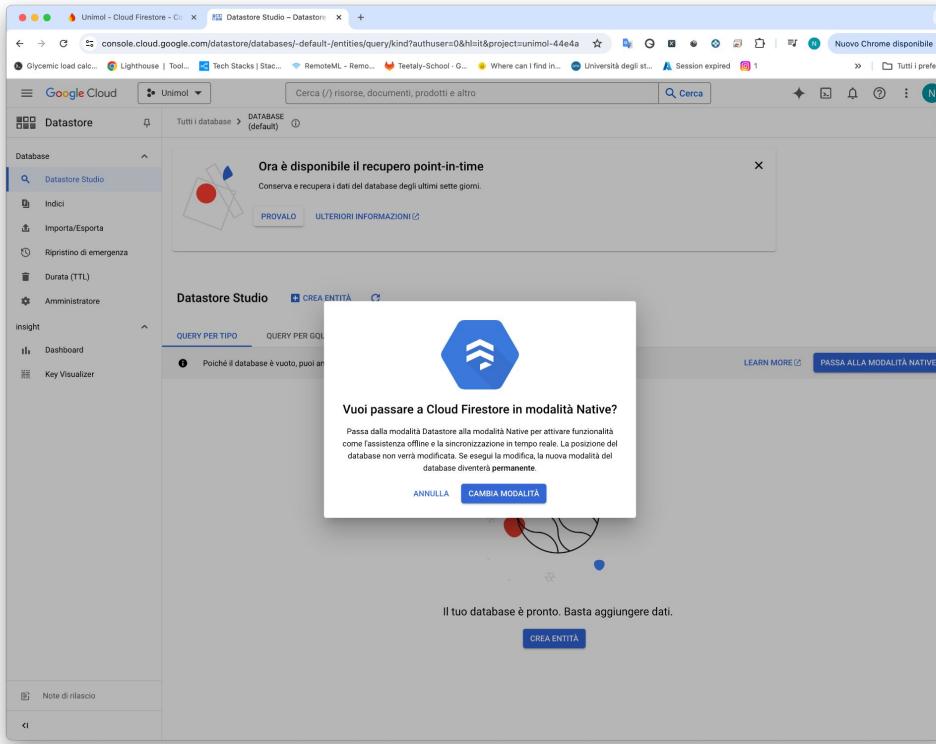
A table lists the existing database configuration:

Filtro	ID database	Modalità	Località	Data/ora creazione (UTC+2)	Azioni
	(default)	Datastore	eur3	21 mag 2024, 16:41:42	⋮

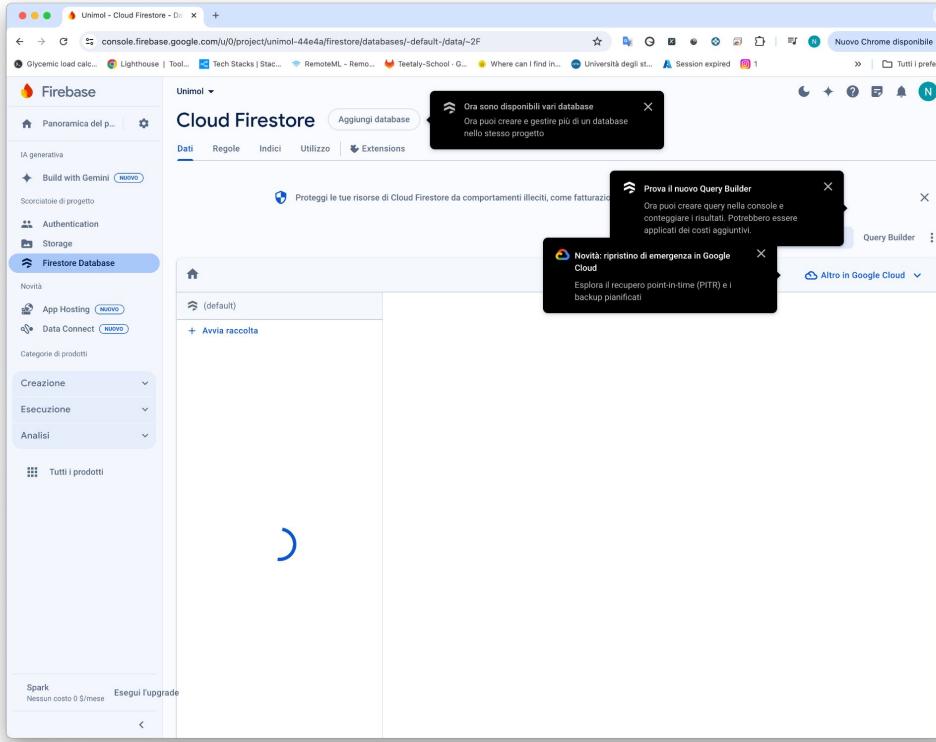
La modalità Native garantisce SDK per Flutter



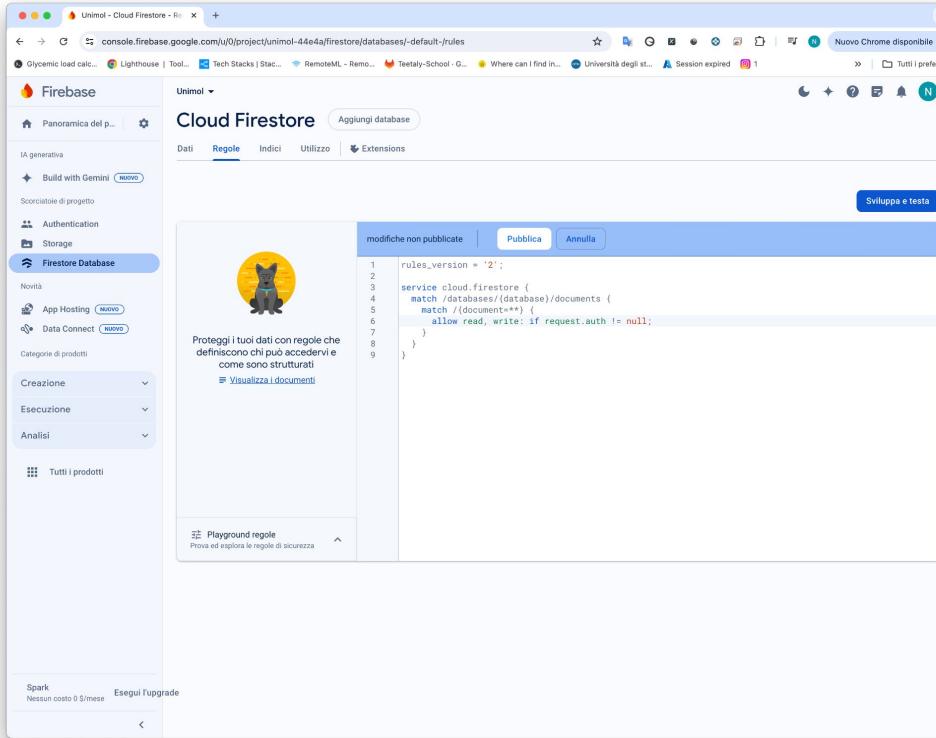
La modalità Native garantisce SDK per Flutter



Database Documentale: Cloud Firestore



Importante: Cambiare le regole



https://pub.dev/packages/cloud_firestore

The screenshot shows the pub.dev package page for `cloud_firestore` version 4.17.4. The page has a dark header with the Flutter logo and a search bar. Below the header, the package name `cloud_firestore` and version `4.17.4` are displayed. A timestamp indicates it was published 2 hours ago. The package is marked as Dart 3 compatible. It supports the `SDK`, `FLUTTER`, and `PLATFORM` platforms, and is available for `ANDROID`, `IOS`, `MACOS`, `WEB`, and `WINDOWS`. The download count is 3.3K. The popularity is 100%, with 3353 likes and 130 pub points. The publisher is listed as `firebase.google.com`. The metadata section describes the plugin as a Flutter plugin for Cloud Firestore, a cloud-hosted, noSQL database with live synchronization and offline support on Android and iOS. It links to the homepage, repository on GitHub, and contributing guidelines. The documentation section includes links to the API reference and license information (BSD-3-Clause). The dependencies listed are `cloud_firestore_platform_interface`, `cloud_firestore_web`, `collection`, `firebase_core`, `firebase_core_platform_interface`, `flutter`, and `meta`. The page also features sections for Getting Started, Usage, and Issues and feedback.

cloud_firestore 4.17.4

Published 2 hours ago • [@firebase.google.com](#) (Dart 3 compatible)

SDK FLUTTER PLATFORM ANDROID IOS MACOS WEB WINDOWS 3.3K

3353 130 100%

Likes Pub Points Popularity

Publisher [@firebase.google.com](#)

Metadata

Flutter plugin for Cloud Firestore, a cloud-hosted, noSQL database with live synchronization and offline support on Android and iOS.

Homepage Repository (GitHub) View/report issues Contributing

Documentation API reference

License BSD-3-Clause (LICENSE)

Dependencies

cloud_firestore_platform_interface, cloud_firestore_web, collection, firebase_core, firebase_core_platform_interface, flutter, meta

Cloud Firestore Plugin for Flutter

A Flutter plugin to use the [Cloud Firestore API](#).

To learn more about Firebase Cloud Firestore, please visit the [Firebase website](#)

pub v0.17.4

Getting Started

To get started with Cloud Firestore for Flutter, please [see the documentation](#).

Usage

To use this plugin, please visit the [Firestore Usage documentation](#)

Issues and feedback

Implementiamo il salvataggio di un profilo in AuthScreen

All'interno di AuthScreen procediamo a creare un campo username che sarà disponibile solo in fase di registrazione e, una volta effettuata la registrazione, andiamo a salvare l'url con le immagini, l'id e lo username nel database remoto Firestore.

```

import 'package:flutter/material.dart';
import 'dart:io';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:fireb/widgets/user_image_picker.dart';
import 'package:firebase_storage/firebase_storage.dart';
import 'package:cloud_firestore/cloud_firestore.dart';

final _firebase = FirebaseAuth.instance;

class AuthScreen extends StatefulWidget {
  const AuthScreen({super.key});

  @override
  State<AuthScreen> createState() {
    return _AuthScreenState();
  }
}

class _AuthScreenState extends State<AuthScreen> {
  final _form = GlobalKey<FormState>();

  var _isLogin = true;
  var _enteredEmail = '';
  var _enteredPassword = '';
  var _enteredUsername = '';
  File? _selectedImage;
  var _isAuthenticating = false;

  void _submit() async {
    final isValid = _form.currentState!.validate();

    if (!isValid || !_isLogin && _selectedImage == null) {
      return;
    }

    _form.currentState!.save();
    try {
      setState(() {
        _isAuthenticating = true;
      });
      if (_isLogin) {
        final userCredentials = await _firebase.signInWithEmailAndPassword(
          email: _enteredEmail,
          password: _enteredPassword);
      } else {
        final userCredentials = await _firebase.createUserWithEmailAndPassword(
          email: _enteredEmail,
          password: _enteredPassword);
        final storageRef = FirebaseStorage.instance
          .ref()
          .child('user_images')
          .child('${userCredentials.user!.uid}.jpg');

        await storageRef.putFile(_selectedImage!);
        final imageUrl = await storageRef.getDownloadURL();

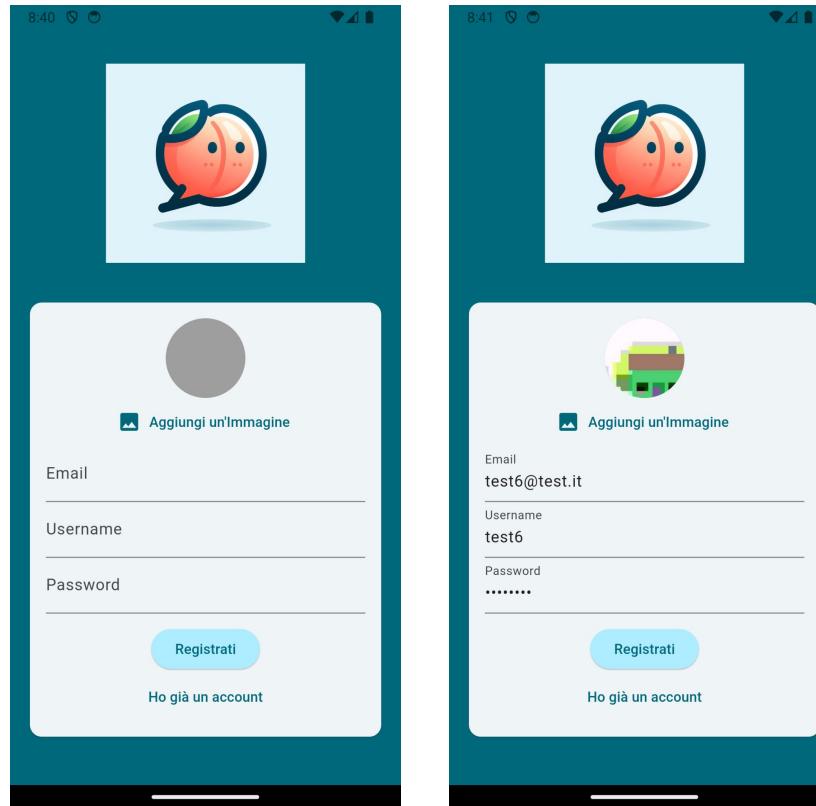
        await FirebaseFirestore.instance
          .collection('users')
          .doc(userCredentials.user!.uid)
          .set({
            'username': _enteredUsername,
            'email': _enteredEmail,
            'image_url': imageUrl,
          });
      }
    } on FirebaseAuthException catch (error) {
      if (error.code == 'email-already-in-use') {
        // ...
      }
    }
  }
}

```

- Viene creato un nuovo documento nella collezione 'users' di Firestore. L'ID del documento è l'ID dell'utente, che viene ottenuto da **userCredentials.user!.uid** mentre l'username, l'email e l'image_url vengono acquisite dalle rispettive variabili.
- L'operazione è asincrona, quindi usa await per aspettare che il documento sia stato creato.

- Come per l'email e per la password, viene definito un campo username che sia lungo almeno 5 caratteri.

Il risultato



Il risultato

The screenshot shows the Firebase Cloud Firestore interface in a web browser. The left sidebar lists various services: IA generativa, Build with Gemini (NEW), Authentication, Storage, and Firestore Database (which is selected). The main area displays the 'Cloud Firestore' dashboard for the project 'Unimol'. A banner at the top right encourages protecting resources from illegal behaviors like fraudulent billing and phishing. Below the banner, there's a 'Visualizzazione riquadro' (Grid View) button and a 'Query Builder' button. The main content area shows a collection named 'users' with two documents: 'HNOfm0sxqel83rpvkQbjSjmtD2' and 'RGHv9rt6tnM083yMKelpxv9LNeh1'. At the bottom of the page, there's a note about the 'Spark' plan and an 'Esegui Upgrade' (Upgrade Now) button.

Spark
Nessun costo 0 \$/mese Esegui Upgrade

Il risultato

The screenshot shows the Firebase Cloud Firestore interface in a web browser. The left sidebar navigation includes 'Panoramica del progetto', 'IA generativa', 'Build with Gemini', 'Authentication', 'Storage', and 'Firestore Database' (which is selected). Under 'Categorie di prodotti', there are sections for 'Creazione', 'Esecuzione', and 'Analisi'. At the bottom of the sidebar, there are links for 'Spark' (costo 0 \$/mese) and 'Esegui Upgrade'.

The main content area displays the 'Cloud Firestore' dashboard for the 'Unimol' project. It features tabs for 'Dati', 'Regole', 'Indici', 'Utilizzo', and 'Extensions'. A banner at the top says 'Proteggi le tue risorse di Cloud Firestore da comportamenti illeciti, come fatturazione fraudolenta o phishing' and includes a 'Configura App Check' button.

The 'users' collection is shown with a single document named 'HNOFmXOsxqeL83rpvkqBjSjmwTD2'. The document details are:

- email: "test6@test.it"
- image_url: "https://firebasestorage.googleapis.com/v0/b/unimol-44e4a.appspot.com/o/user_images%2FHNOFmXOsxqeL83rpvkqBjSjmwTD2%2fmediaToken%2f72b933a-f157-4d3d-ae38-f305977faea"
- username: "test6"

Predisposizione di una chat

Un nuovo widget: NewMessage

```
import 'package:flutter/material.dart';
class NewMessage extends StatefulWidget {
  const NewMessage({super.key});
  @override
  State<NewMessage> createState() {
    return _NewMessageState();
  }
}
class _NewMessageState extends State<NewMessage> {
  var _messageController = TextEditingController();
  @override
  void dispose() {
    _messageController.dispose();
    super.dispose();
  }
  void _submitMessage() {
    final enteredMessage = _messageController.text;
    if (enteredMessage.trim().isEmpty) {
      return;
    }
    // codice per inviare a firebase
    _messageController.clear();
  }
  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: const EdgeInsets.only(left: 15, right: 1, bottom: 14),
      child: Row(
        children: [
          Expanded(
            child: TextField(
              controller: _messageController,
              textCapitalization: TextCapitalization.sentences,
              autocorrect: true,
              enableSuggestions: true,
              decoration: const InputDecoration(labelText: 'Invia un messaggio...'),
            ),
            IconButton(
              color: Theme.of(context).colorScheme.primary,
              icon: const Icon(Icons.send),
              onPressed: _submitMessage,
            ),
          ],
        ),
      );
  }
}
```

- Questo widget è utilizzato per creare un campo di input e un pulsante che permettono all'utente di inviare un nuovo messaggio;
- La classe **_NewMessageState** definisce una variabile di stato **_messageController**, che è un **TextEditingController**. Questo controller permette di controllare il testo e la selezione del campo di input;
- Il metodo **dispose** viene chiamato quando il widget **NewMessage** viene rimosso dall'albero dei widget. Questo metodo si assicura di liberare le risorse utilizzate dal **TextEditingController** quando il widget non è più necessario;
- Il metodo **_submitMessage** viene chiamato quando l'utente preme il pulsante di invio. Questo metodo controlla se il messaggio inserito dall'utente è vuoto e, se non lo è eseguirà delle operazioni che implementeremo nei prossimi passaggi e successivamente pulisce il campo di input;
- Il metodo **build** di **_NewMessageState** definisce l'interfaccia utente del widget **NewMessage**. Questa interfaccia utente consiste in un **Padding** che contiene una **Row** con un **TextField** e un **IconButton**. Il **TextField** permette all'utente di inserire un messaggio e il **IconButton** permette all'utente di inviare il messaggio. Quando l'utente preme il **IconButton**, viene chiamato il metodo **_submitMessage**.

Un nuovo widget: ChatMessages

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

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

  @override
  Widget build(BuildContext context) {
    return const Center(
      child: Text('Nessun messaggio da visualizzare!'),
    );
  }
}
```

Integrazione di entrambi i widget all'interno dello screen ChatScreen

```
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:fireb/widgets/chat_messages.dart';
import 'package:fireb/widgets/new_message.dart';

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('PescheChat'),
        actions: [
          IconButton(
            onPressed: () {
              FirebaseAuth.instance.signOut();
            },
            icon: Icon(
              Icons.exit_to_app,
              color: Theme.of(context).colorScheme.primary,
            ),
          ),
        ],
      ),
      body: const Column(
        children: [
          Expanded(
            child: ChatMessages(),
          ),
          NewMessage(),
        ],
      ),
    );
  }
}
```

- Dopo la login vengono visualizzati all'interno di una **Column** i due widget appena creati.

Invio a Firestore dei messaggi di chat

Implementazione della funzione che permette l'invio dei messaggi a Firestore
attraverso il widget NewMessage

```

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

class NewMessage extends StatefulWidget {
  const NewMessage({super.key});

  @override
  State<NewMessage> createState() {
    return _NewMessageState();
  }
}

class _NewMessageState extends State<NewMessage> {
  var _messageController = TextEditingController();

  @override
  void dispose() {
    _messageController.dispose();
    super.dispose();
  }

  void _submitMessage() async {
    final enteredMessage = _messageController.text;

    if (enteredMessage.trim().isEmpty) {
      return;
    }

    FocusScope.of(context).unfocus();
    _messageController.clear();

    final user = FirebaseAuth.instance.currentUser!;
    final userData = await FirebaseFirestore.instance
        .collection('users')
        .doc(user.uid)
        .get();

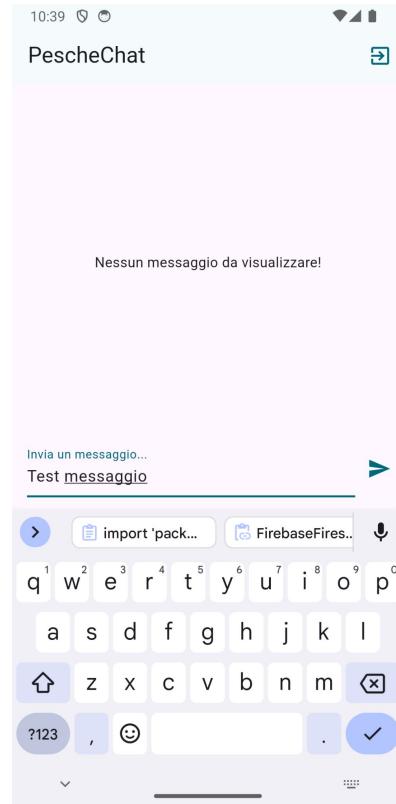
    FirebaseFirestore.instance.collection('chat').add({
      'text': enteredMessage,
      'createdAt': Timestamp.now(),
      'userId': user.uid,
      'username': userData.data()!['username'],
      'userImage': userData.data()!['image_url'],
    });
  }

  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: const EdgeInsets.only(left: 15, right: 1, bottom: 14),
      child: Row(
        children: [
          Expanded(
            child: TextField(
              controller: _messageController,
              textCapitalization: TextCapitalization.sentences,
              autocorrect: true,
              enableSuggestions: true,
              decoration: const InputDecoration(labelText: 'Invia un messaggio...'),
            ),
          ),
          IconButton(
            color: Theme.of(context).colorScheme.primary,
            icon: const Icon(
              Icons.send,
            ),
            onPressed: _submitMessage,
          ),
        ],
      );
  }
}

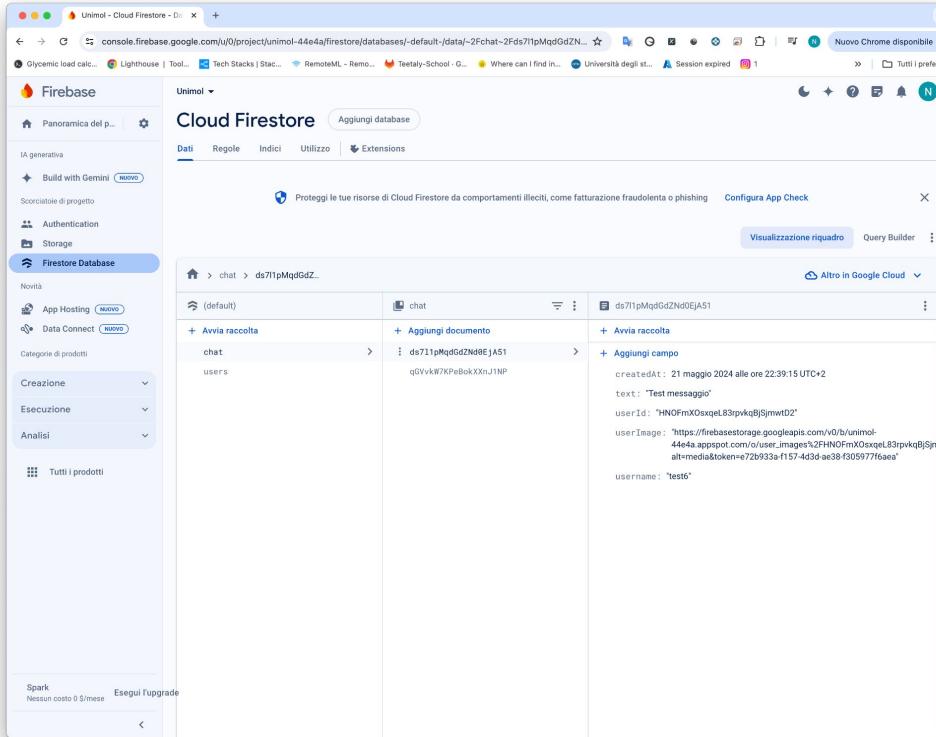
```

- **if (enteredMessage.trim().isEmpty) { return; } :** Se il messaggio inserito dall'utente è vuoto (dopo aver rimosso gli spazi bianchi all'inizio e alla fine) esce immediatamente;
- **FocusScope.of(context).unfocus(); :** Rimuove il focus dal campo di input, in sostanza la tastiera si chiude;
- **_messageController.clear(); :** Viene ripulito il texfield di input;
- **final user = FirebaseAuth.instance.currentUser!;** : Si procede a recuperare l'utente corrente da Firebase Authentication e successivamente si acquisiscono i singoli dati;
- **FirebaseFirestore.instance.collection('chat').add({...}); :** Viene aggiunto un nuovo documento alla collezione 'chat' in Firestore. Il documento contiene il testo del messaggio, l'ora corrente, l'ID dell'utente, il nome utente e l'URL dell'immagine dell'utente.

Il risultato



Il risultato



The screenshot shows the Firebase Cloud Firestore interface. On the left, the sidebar includes sections for IA generativa, Build with Gemini, Authentication, Storage, and Firestore Database (which is selected). The main area displays a document in the 'chat' collection under the 'users' subcollection. The document ID is 'ds71pMqdGdZ...'. The data within the document is as follows:

```
chat: {  
    qGVvkW7KPeBokXXnJNP:  
        createdat: "21 maggio 2024 alle ore 22:39:15 UTC+2"  
        text: "Test messaggio"  
        userId: "HNOfmXOsxqeL83rpvkqBjSmwtd2"  
        userImage: "https://firebasestorage.googleapis.com/v0/b/unimol-44e4a.appspot.com/user_images%2FHNOfmXOsxqeL83rpvkqBjSmwtd2.jpg?alt=media&ttoken=e72b933a-f157-4d3d-ae38-f305977f6aea"  
        username: "test6"  
}
```

Leggo da Firestore i messaggi inviati

Implementiamo all'interno del widget ChatMessages la lettura da Firestore della collection 'chat'.

```

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

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

  @override
  Widget build(BuildContext context) {
    final authenticatedUser = FirebaseAuth.instance.currentUser!;

    return StreamBuilder(
      stream: FirebaseFirestore.instance
        .collection('chat')
        .orderBy(
          'createdAt',
          descending: true,
        )
        .snapshots(),
      builder: (ctx, chatSnapshots) {
        if (chatSnapshots.connectionState == ConnectionState.waiting) {
          return const Center(
            child: CircularProgressIndicator(),
          );
        }

        if (!chatSnapshots.hasData || chatSnapshots.data!.docs.isEmpty) {
          return const Center(
            child: Text('Non ci sono messaggi. Scrivi qualcosa!'),
          );
        }

        if (chatSnapshots.hasError) {
          return const Center(
            child: Text('C\'è un errore, dobbiamo indagare...'),
          );
        }

        final loadedMessages = chatSnapshots.data!.docs;

        return ListView.builder(
          itemCount: loadedMessages.length,
          itemBuilder: (ctx, index) {
            return Text(
              loadedMessages[index].data()['text'],
            );
          },
        );
      },
    );
  }
}

```

- Il metodo `build` di `ChatMessages` definisce l'interfaccia utente del widget. Questa interfaccia utente consiste in uno **StreamBuilder** che ascolta gli aggiornamenti della collection Firestore chiamata 'chat';
- Quando i dati della chat vengono caricati, lo **StreamBuilder** costruisce una **ListView** dei messaggi. Ogni messaggio è rappresentato da un widget **Text** che visualizza il testo del messaggio;
- Se i dati della chat non sono ancora disponibili (cioè, quando la connessione è in attesa), viene visualizzato un **CircularProgressIndicator**;
- Se non ci sono messaggi o c'è un errore vengono visualizzati dei messaggi.

Il risultato





Un po' di stile...

Un nuovo Widget: MessageBubble

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

class MessageBubble extends StatelessWidget {
  const MessageBubble.first({
    super.key,
    required this.userImage,
    required this.username,
    required this.message,
    required this.isMe,
    required this.isSME,
  }) : isFirstInSequence = true;

  const MessageBubble.next({
    super.key,
    required this.message,
    required this.isMe,
  }) : isFirstInSequence = false,
        userImage = null,
        username = null;

  final bool isFirstInSequence;
  final String? userImage;
  final String? username;
  final String message;
  final bool isMe;

  @override
  Widget build(BuildContext context) {
    final theme = Theme.of(context);

    return Stack(
      children: [
        if (userImage != null)
          Positioned(
            top: 15,
            right: isMe ? 0 : null,
            child: CircleAvatar(
              backgroundImage: NetworkImage(
                userImage!,
              ),
              backgroundColor: theme.colorScheme.primary.withOpacity(180),
              radius: 23,
            ),
          ),
        Container(
          margin: const EdgeInsets.symmetric(horizontal: 46),
          child: Row(
            mainAxisAlignment: MainAxisSizeAlignment.isMe ? MainAxisSize.end : MainAxisSize.start,
            crossAxisAlignment: CrossAxisAlignmentAlignment.isMe ? CrossAxisAlignmentAlignment.end : CrossAxisAlignmentAlignment.start,
            children: [
              Column(
                crossAxisAlignment: CrossAxisAlignmentAlignment.isMe ? CrossAxisAlignmentAlignment.end : CrossAxisAlignmentAlignment.start,
                children: [
                  if (isFirstInSequence) const SizedBox(height: 18),
                  if (username != null) Padding(
                    padding: const EdgeInsets.only(
                      left: 13,
                      right: 13,
                    ),
                    child: Text(
                      username!,
                      style: const TextStyle(
                        fontWeight: FontWeight.bold,
                        color: Colors.black87,
                      ),
                    ),
                  ),
                ],
              ),
            ],
          ),
        ),
      ],
    );
  }
}
```

- Il costruttore **MessageBubble.first** viene utilizzato per creare un **MessageBubble** che rappresenta il primo messaggio in una sequenza di messaggi da un utente. Questo costruttore richiede un'immagine dell'utente, un nome utente, un messaggio e un flag **isMe** che indica se il messaggio è stato inviato dall'utente corrente;
 - Il costruttore **MessageBubble.next** viene utilizzato per creare un **MessageBubble** che rappresenta un messaggio successivo nella stessa sequenza di messaggi. Questo costruttore richiede solo un messaggio e il flag **isMe**;
 - Il metodo **build** di **MessageBubble** definisce l'interfaccia utente del widget. Questa interfaccia utente consiste in uno **Stack** che contiene un **CircleAvatar** (se l'immagine dell'utente è disponibile) e un **Container** che contiene il nome utente (se disponibile) e il testo del messaggio;
 - Il **CircleAvatar** viene posizionato in alto a destra se il messaggio è stato inviato dall'utente corrente, altrimenti viene posizionato in alto a sinistra;
 - Il **Container** contiene una **Row** con una **Column** di widget. La **Column** contiene un **SizedBox** (se il messaggio è il primo della sequenza), un **Text** con il nome utente (se disponibile) e un altro **Container** con il testo del messaggio (slide successiva).

Un nuovo Widget: MessageBubble

```
...  
  
Container(  
  decoration: BoxDecoration(  
    color: isMe  
      ? Colors.grey[300]  
      : theme.colorScheme.secondary.withOpacity(200),  
    borderRadius: BorderRadius.only(  
      topLeft: !isMe && isFirstInSequence  
        ? Radius.zero  
        : const Radius.circular(12),  
      topRight: isMe && isFirstInSequence  
        ? Radius.zero  
        : const Radius.circular(12),  
      bottomLeft: const Radius.circular(12),  
      bottomRight: const Radius.circular(12),  
    ),  
  constraints: const BoxConstraints(maxWidth: 200),  
  padding: const EdgeInsets.symmetric(  
    vertical: 10,  
    horizontal: 14,  
  ),  
  margin: const EdgeInsets.symmetric(  
    vertical: 4,  
    horizontal: 12,  
  ),  
  child: Text(  
    message,  
    style: TextStyle(  
      height: 1.3,  
      color: isMe  
        ? Colors.black87  
        : theme.colorScheme.onSecondary,  
    ),  
    softWrap: true,  
  ),  
,  
,  
,  
,  
,  
,  
);
```

- Il **Container** con il testo del messaggio ha un colore di sfondo diverso a seconda che il messaggio sia stato inviato dall'utente corrente o no. Inoltre, il bordo superiore sinistro o destro del Container è piatto se il messaggio è il primo della sequenza.

Adeguo ChatMessage per usare lo stile di MessageBubble

```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:fireb/widgets/message_bubble.dart';

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

  @override
  Widget build(BuildContext context) {
    ...

    final loadedMessages = chatSnapshots.data!.docs;

    return ListView.builder(
      padding: const EdgeInsets.only(
        bottom: 40,
        left: 13,
        right: 13,
      ),
      reverse: true,
      itemCount: loadedMessages.length,
      itemBuilder: (ctx, index) {
        final chatMessage = loadedMessages[index].data();
        final nextChatMessage = index + 1 < loadedMessages.length
          ? loadedMessages[index + 1].data()
          : null;

        final currentMessageUserId = chatMessage['userId'];
        final nextMessageUserId =
          nextChatMessage != null ? nextChatMessage['userId'] : null;
        final nextUserIsSame = nextMessageUserId == currentMessageUserId;

        if (nextUserIsSame) {
          return MessageBubble.next(
            message: chatMessage['text'],
            isMe: authenticatedUser.uid == currentMessageUserId,
          );
        } else {
          return MessageBubble.first(
            userImage: chatMessage['userImage'],
            username: chatMessage['username'],
            message: chatMessage['text'],
            isMe: authenticatedUser.uid == currentMessageUserId,
          );
        }
      },
    );
  }
}
```

- **reverse: true** : Questo comando fa sì che la ListView inizi dal fondo e scorra verso l'alto così da replicare l'esperienza utente delle app di chat più comuni;
- **itemBuilder: (ctx, index) {...}** : La funzione che viene chiamata per ogni elemento della lista. La funzione riceve il contesto del widget e l'indice dell'elemento corrente, e deve restituire il widget che rappresenta l'elemento. In questo caso, la funzione restituisce un widget **MessageBubble** per dare lo stile ad un singolo messaggio della chat;
- All'interno della funzione **itemBuilder**, viene determinato se il messaggio corrente e il messaggio successivo sono dello stesso utente. Se lo sono, viene creato un **MessageBubble.next**, altrimenti viene creato un **MessageBubble.first**. Questo permette di raggruppare i messaggi consecutivi dello stesso utente.

Il risultato



Pausa

Il codice chat.zip contiene il codice con i file dart necessari a costruire la chat.

Notifiche Push

Messaging

The screenshot shows the Firebase Cloud Firestore console for the project "Unimol". The left sidebar has a red box around the "Messaging" item under the "Creazione" section. A red arrow points from this box to the main content area.

Cloud Firestore (Aggiungi database)

Dati Regole Indici Utilizzo Extensions

Protegli le tue risorse di Cloud Firestore da comportamenti illeciti, come fatturazione fraudolenta o phishing Configura App Check

Visualizzazione riquadro Query Builder

(default) chat ds71pMqdGdZ...

+ Avvia raccolta + Aggiungi documento + Avvia raccolta

chat users

+ Aggiungi campo

AN05LFGW3KK7wKQqf9bB
Fz2zf9mgfYtadL54rZK
ds71pMqdGdZNd0EjA51
FPTvex1SxATBKtNTcdc
1UoJhfhbxcvhdUgRF9B09
qGVvk#72PeBoKXNj1NP

createdAt: 21 maggio 2024 alle ore 22:39:15 UTC+2
text: "Test messaggio"
userId: "HNOFmxFoxsqel83rpvkxBjSjmwtD2"
userImage: "https://storage.googleapis.com/o/user_images%2FHNOFmxFoxsqel83rpvkxBjSjmwtD2?alt=media&token=e72b933a-f157-4d3d-ae38-f0305977feaea"
username: "test6"

Spark Nessun codice 0 / 1 mese Esegui l'upgrade

Messaging

The screenshot shows the Firebase console interface for the 'Messaging' service. The left sidebar lists various services: Panoramica del progetto, IA generativa (Build with Gemini), Scrittore di progetto, Authentication, Storage, Firestore Database, **Messaging**, Novità, App Hosting (nuovo), Data Connect (nuovo), Categorie di prodotti, Creazione, Esecuzione, Analisi, and Tutti i prodotti. The main content area is titled 'Messaging' and features a large illustration of a paper airplane flying over clouds. Below the illustration is a button labeled 'Crea la tua prima campagna'. A callout box states: 'La mirea degli eventi chiave e la maggior parte delle opzioni di targeting richiedono Google Analytics, che al momento non è abilitato per questo progetto.' To the right of the callout is a link to 'Abilita Google Analytics'. Below this section, there's a heading 'Learn more about Cloud Messaging' followed by three cards: 'Che cosa devo fare per iniziare?' (Visualizza i documenti), 'Come funziona Cloud Messaging?' (Visualizza i documenti), and 'Che cosa può fare per me Cloud Messaging?' (Ulteriori informazioni). To the right of these cards is a video thumbnail titled 'Introducing Firebase Cloud Messaging' with a play button and a 'Guarda più' link. At the bottom of the main content area is a link to 'Learn more about In-App Messaging'.

<https://firebase.google.com/docs/cloud-messaging/flutter/client>

The screenshot shows a browser window displaying the Firebase Cloud Messaging documentation for Flutter. The URL in the address bar is <https://firebase.google.com/docs/cloud-messaging/flutter/client>. The page title is "Set up a Firebase Cloud Messaging client app on Flutter". The left sidebar contains navigation links for Remote Config, A/B Testing, Analytics, Cloud Messaging (with sub-links for Introduction, Migrate to the HTTP v1 API, Concepts and best practices, iOS+, Android, Flutter, C++, Unity, Web (JavaScript), Server environments), In-App Messaging, Dynamic Links, Google AdMob, Google Ads, and App Indexing. The main content area starts with a message: "Thanks for tuning in to Google I/O: Watch content on-demand." It then provides instructions for setting up a FCM client on Flutter, mentioning platform-specific setup and requirements for iOS+ (including enabling app capabilities in Xcode). It also covers the upload of APNs authentication keys and project settings. A sidebar on the right includes "Info", "Chat", and "Go to console".

Set up a Firebase Cloud Messaging client app on Flutter

Platform-specific setup and requirements

iOS+

Enable app capabilities in Xcode

Before your application can start to receive messages, you must enable push notifications and background modes in your Xcode project.

1. Open your Xcode project workspace (`ios/Runner.xcworkspace`).
2. Enable push notifications [🔗](#).
3. Enable the `Background fetch` and the `Remote notifications` background execution modes [🔗](#).

Upload your APNs authentication key

Before you use FCM, upload your APNs certificate to Firebase. If you don't already have an APNs certificate, create one in the [Apple Developer Member Center](#).

1. Inside your project in the Firebase console, select the gear icon, select `Project Settings`, and then select the `Cloud Messaging` tab.
2. Select the `Upload Certificate` button for your development certificate, your production certificate, or both. At least one is required.
3. For each certificate, select the `.p12` file, and provide the password, if any. Make sure the bundle ID for this certificate matches the bundle ID of your app. Select `Save`.

Se si vogliono usare le Notifiche Push su iOS bisogna seguire con attenzione i vari step descritti qui

Flutter plugin

The screenshot shows a browser window displaying the Firebase Cloud Messaging documentation for Flutter. The URL is firebase.google.com/docs/cloud-messaging/flutter/client. The page is titled "Set up a Firebase Cloud Messaging client".

The left sidebar contains a navigation menu for various Firebase services, with "Cloud Messaging" expanded. Under "Cloud Messaging", the "Flutter" section is selected, showing sub-options: "Set up a Flutter client" (which is highlighted), "Send a test message", "Receive messages", and "Subscribe to topics". Other sections include "C++", "Unity", "Web (JavaScript)", and "Server environments".

The main content area provides instructions for setting up the FCM plugin:

1. Install and initialize the Firebase plugins for Flutter if you haven't already done so.
2. From the root of your Flutter project, run the following command to install the plugin:
`flutter pub add firebase_messaging`
3. Once complete, rebuild your Flutter application:
`flutter run`

A red box highlights the command `flutter pub add firebase_messaging`.

Below this, there is a section titled "Access the registration token" with a sub-section about retrieving the token:

To send a message to a specific device, you need to know that device's registration token. Because you'll need to enter the token in a field in the Notifications console to complete this tutorial, make sure to copy the token or securely store it after you retrieve it.

To retrieve the current registration token for an app instance, call `getToken()`. If notification permission has not been granted, this method will ask the user for notification permissions. Otherwise, it returns a token or rejects the future due to an error.

⚠ Warning: In iOS SDK 10.4.0 and higher, it is a requirement that the APNs token is available before making API requests. The APNs token is not guaranteed to have been received before making FCM plugin API requests.

At the bottom, there is a code snippet for setting up permission requests:

```
// You may set the permission requests to "provisional" which allows the user to choose what type
// of notifications they would like to receive once the user receives a notification.
final notificationSettings = await FirebaseMessaging.instance.requestPermission(provisional: true);

// For apple platforms, ensure the APNs token is available before making any FCM plugin API calls
final apnsToken = await FirebaseMessaging.instance.getAPNSToken();
if (apnsToken != null) {
  // APNs token is available, make FCM plugin API requests...
}
```

Come implemento le modifiche nel codice?

Procediamo nel configurare la logica delle notifiche push all'interno del widget ChatScreen. Abbiamo usato ChatScreen perché abbiamo la certezza che gli utenti che accederanno a questo widget saranno già loggati. Non esiste una regola su dove andare ad inserire la registrazione delle notifiche push, questo dipende da applicazione ad applicazione.

Per configurazione delle notifiche push si intende dover integrare una funzione che di fatto chiede all'utente il permesso di voler ricevere delle notifiche e registra un token univoco legato al suo dispositivo, quindi per permettere il funzionamento delle notifiche dobbiamo convertire ChatScreen da StatelessWidget ad uno StatefulWidget perché è necessario usare il metodo initState.

ChatScreen con topic (collection 'chat')

```
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:fireb/widgets/chat_messages.dart';
import 'package:fireb/widgets/new_message.dart';

import 'package:firebase_messaging/firebase_messaging.dart';

class ChatScreen extends StatefulWidget {
  const ChatScreen({super.key});

  @override
  State<ChatScreen> createState() => _ChatScreenState();
}

class _ChatScreenState extends State<ChatScreen> {
  void setupPushNotifications() async {
    final fcm = FirebaseMessaging.instance;

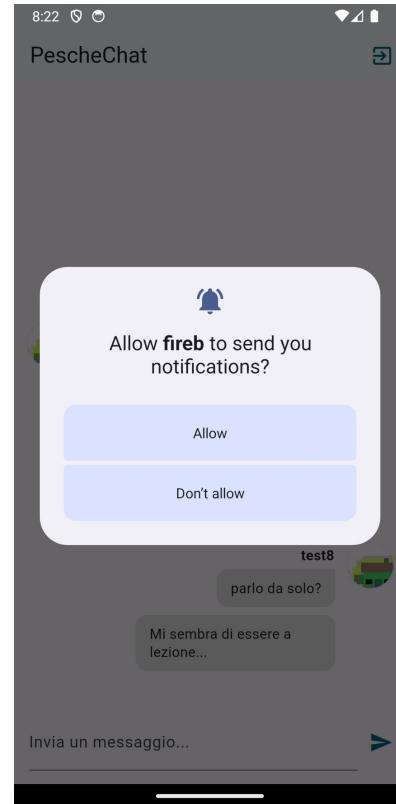
    await fcm.requestPermission();
    fcm.subscribeToTopic('chat');
  }

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

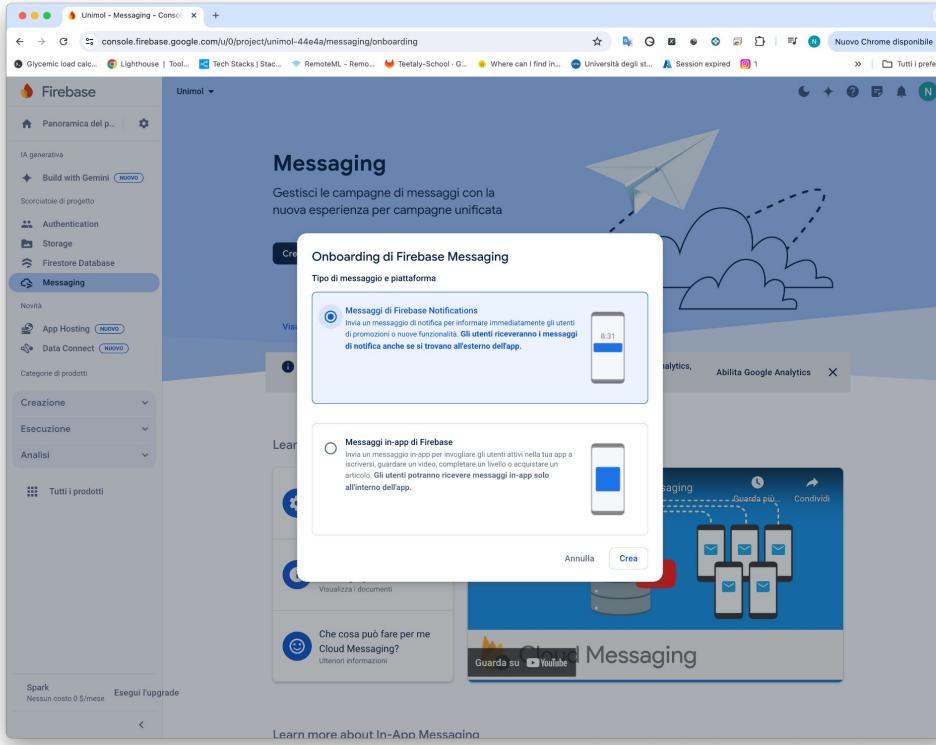
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('PescheChat'),
        actions: [
          IconButton(
            onPressed: () {
              FirebaseAuth.instance.signOut();
            },
            icon: Icon(
              Icons.exit_to_app,
              color: Theme.of(context).colorScheme.primary,
            ),
          ),
        ],
      ),
    );
  }
}
```

- **setupPushNotifications** è un metodo asincrono che configura le notifiche push. Utilizza l'istanza di FirebaseMessaging per gestire le notifiche;
- **await fcm.requestPermission();** : Richiede all'utente il permesso di ricevere notifiche push. Poiché la richiesta di permesso può richiedere del tempo, viene utilizzata la parola chiave await per aspettare che la richiesta sia completata prima di procedere;
- **fcm.subscribeToTopic('chat');** : L'app viene sottoscritta al topic 'chat' su Firebase Cloud Messaging. Quando un messaggio viene inviato a questo topic, tutti i device sottoscritti riceveranno una notifica push. Il topic corrisponde alla collection precedentemente creata.
- **setupPushNotifications** viene eseguito nell'initState, quindi non appena il widget viene creato;

Stoppare e rieseguire la build dell'App



Test notifica



Test notifica

The screenshot shows the Firebase Cloud Messaging 'Componi notifica' (Compose notification) interface. On the left, there's a sidebar with project navigation (Panoramica del progetto, IA generativa, Scorciatoie di progetto, Novità, Categorie di prodotti), a dropdown for the project (Unimol), and sections for Creazione, Esecuzione, and Analisi. At the bottom of the sidebar, it says 'Spark' and 'Nessun costo 0 \$/mese' with a link to 'Esegui l'upgrade'. The main area is titled 'Notifica' and contains the following fields:

- Notifica:** Nuovo messaggio dalla chat di pesche.
- Target:** Segmento utenti (Argomento selected).
 - Argomento del messaggio: chat <1000
- Programmazione:** Invia ora.
- Opzioni aggiuntive (facoltativo):** Buttons for 'Salva come bozza' and 'Rivedi'.

A yellow banner at the top right says 'La misura degli eventi chiave e la maggior parte delle opzioni di targeting richiedono Google Analytics, che al momento non è abilitato per questo progetto' with a 'Abilita Google Analytics' button.

Test notifica

The screenshot shows the Firebase Cloud Messaging 'Componi notifica' (Compose notification) interface. On the left, there's a sidebar with project navigation (Panoramica del progetto, IA generativa, Scorciatoie di progetto, Novità, Categorie di prodotti), and dropdown menus for Creazione, Esecuzione, and Analisi. At the bottom of the sidebar, it says 'Spark' and 'Nessun costo 0 \$/mese' with a link to 'Esegui l'upgrade'. The main area is titled 'Unimol - Componi notifica' and shows a step-by-step process for creating a notification:

- Notifica:** Nuovo messaggio dalla chat di pesche.
- Target:** Segmento utenti (Argomento: chat). An input field contains the word "chat". A blue button labeled "Avanti" is below the input field.
- Programmazione:** Invia ora.
- Opzioni aggiuntive (facoltativo):** This section is currently empty.

At the bottom right of the main form are two buttons: "Salva come bozza" and "Rivedi". Above the main form, a yellow banner displays a warning message: "La misura degli eventi chiave e la maggior parte delle opzioni di targeting richiedono Google Analytics, che al momento non è abilitato per questo progetto" and a "Abilita Google Analytics" button with a red 'X' icon.

Test notifica

The screenshot shows the Firebase Cloud Messaging 'Componi notifica' (Compose notification) interface. On the left, there's a sidebar with project settings like 'Build with Gemini' (NEW), 'Scorciatoie di progetto' (Authentication, Storage, Firestore Database, Messaging), and sections for 'Creazione', 'Esecuzione', and 'Analisi'. The main area is titled 'Componi notifica' and contains the following steps:

- Notifica**: Nuovo messaggio dalla chat di pesche.
- Target**: Ircimi all'argomento chat.
- Programmazione**: Invia agli utenti idonei:
- Ora: Ora
- Avanti:
- Opzioni aggiuntive (facoltativo)**

At the bottom right are 'Salva come bozza' and 'Rivedi' buttons. A yellow banner at the top right says 'Abilita Google Analytics' with a red 'X' button. The URL in the address bar is `console.firebaseio.google.com/u/0/project/unimol-44e4a/notification/compose`.

Test notifica

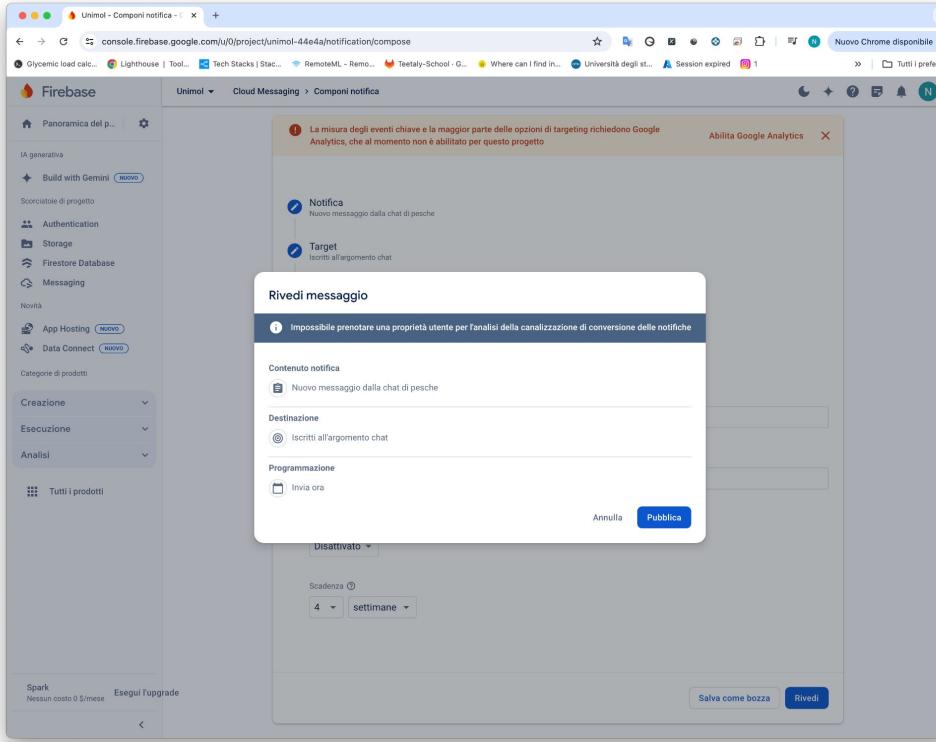
The screenshot shows the Firebase Cloud Messaging 'Componi notifica' (Compose notification) interface. The left sidebar lists various services: IA generativa, Build with Gemini (NEW), Scorrivole di progetto (Authentication, Storage, Firestore Database, Messaging), Novità, App Hosting (NEW), Data Connect (NEW), Categorie di prodotti (Creazione, Esecuzione, Analisi), and Tutti i prodotti.

The main panel is titled 'Notifica' and contains the following fields:

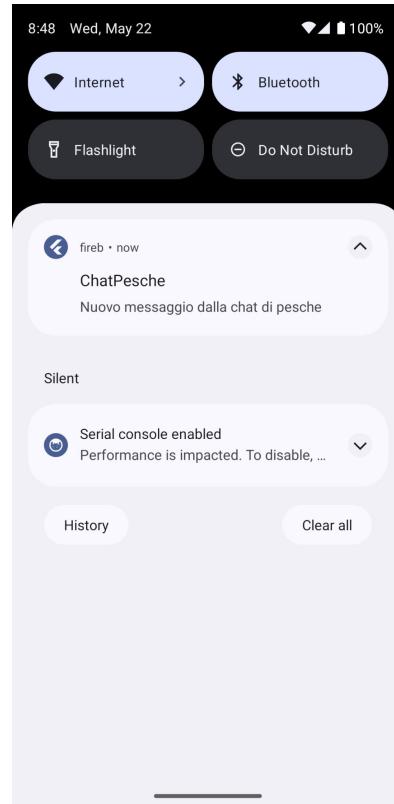
- Notifica:** Nuovo messaggio dalla chat di pesche.
- Target:** Iscritti all'argomento chat.
- Programmazione:** Invia ora.
- Opzioni aggiuntive (facoltativo):** Canale di notifica Android (empty input field), Dati personalizzati (Chiave: empty input field, Valore: empty input field).
- Suoneria:** Disattivato.
- Scadenza:** 4 settimane.

At the bottom, it says 'Spark Nessun costo \$/mese' and 'Esegui l'upgrade'. There are buttons for 'Salva come bozza' and 'Rivedi'.

Test notifica



Il risultato



Fine

Il codice notifiche.zip contiene il codice con i file dart necessari a costruire la chat con le notifiche push su singolo topic.