

Programmazione Mobile

Nicola Noviello

nicola.noviello@unimol.it

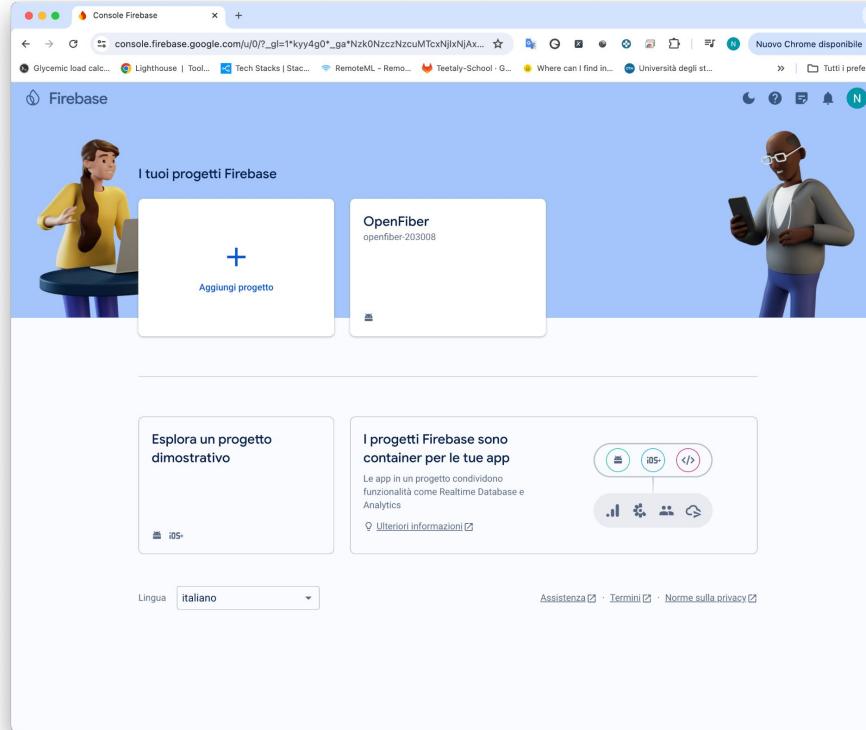
Corso di Laurea in Informatica
Dipartimento di Bioscienze e Territorio
Università degli Studi del Molise
Anno 2023/2024

Lezione: Integrazione di Firebase

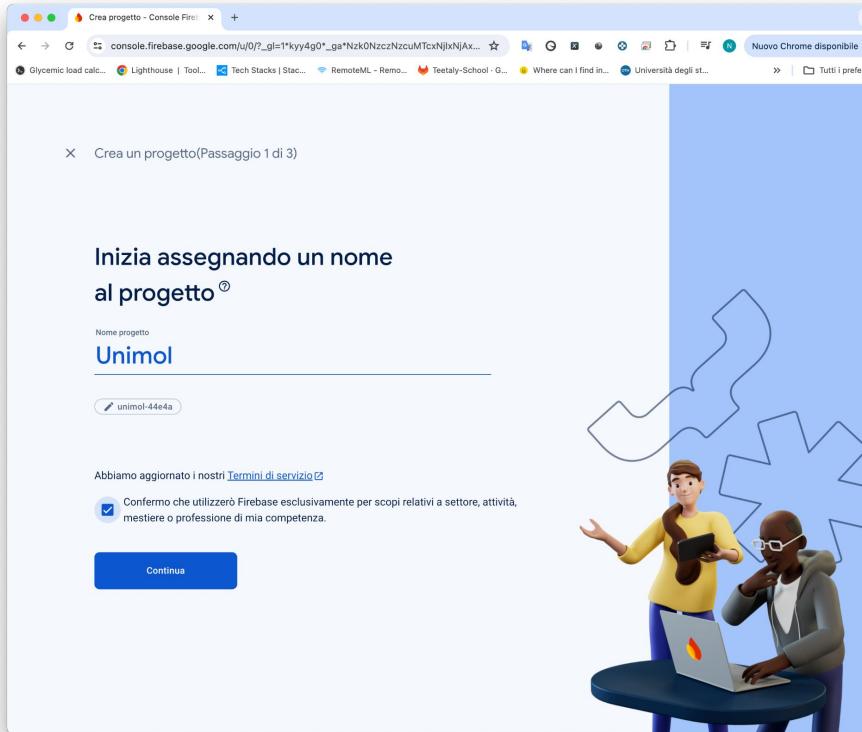
- Creazione di un progetto Firebase
- Gestione dell'autenticazione
- Salvataggio informazioni in remoto
- Salvataggio file in remoto
- Notifiche push

Creazione di un nuovo Progetto Firebase

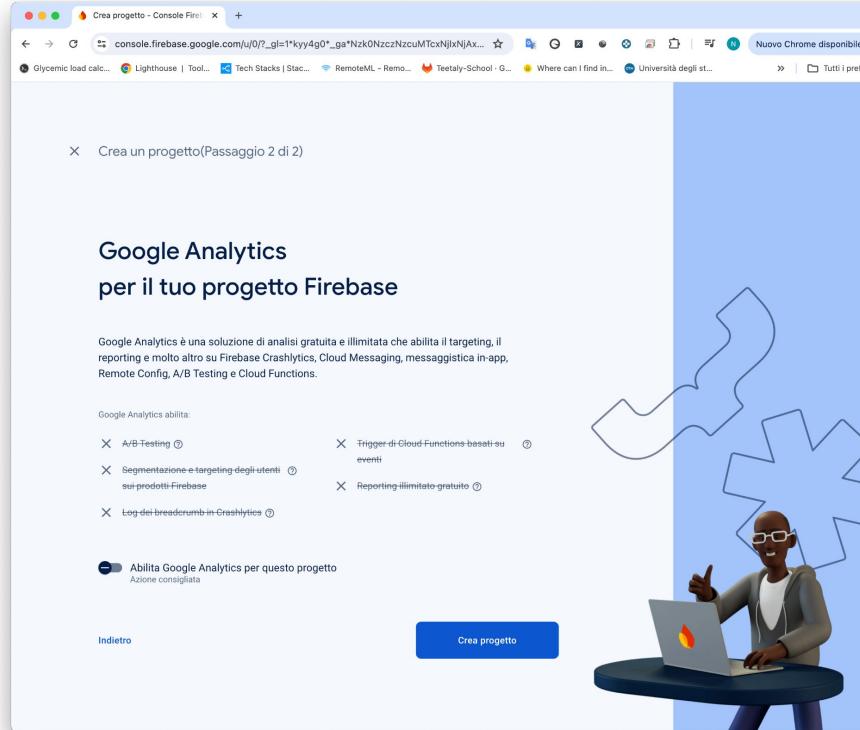
Creazione di un nuovo progetto su Firebase



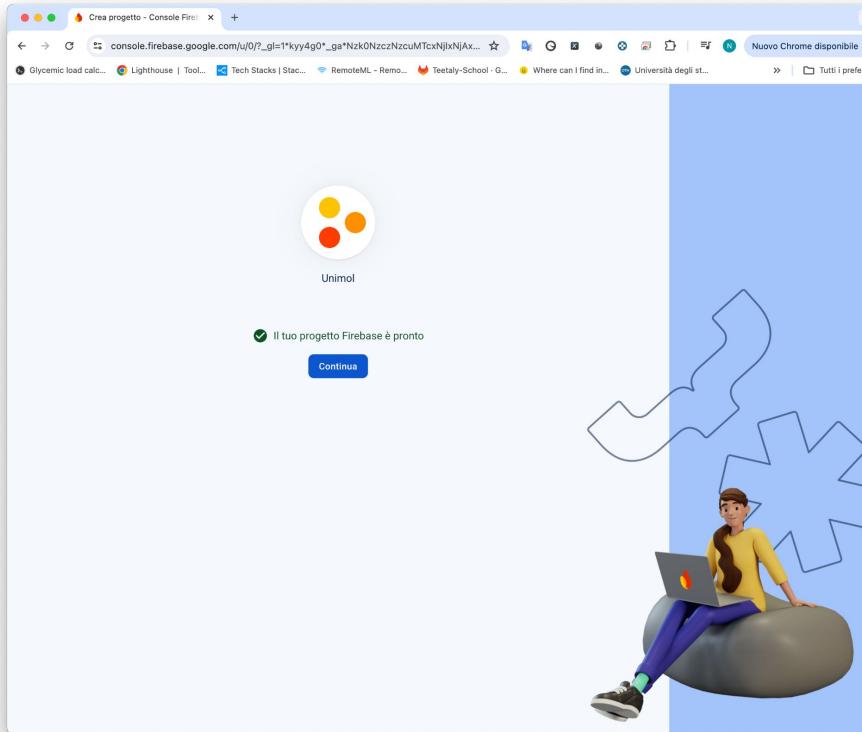
Creazione di un nuovo progetto su Firebase



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 (NEW), Scorsizi delle impostazioni, Authentication (selected), App Hosting (NEW), Data Connect (NEW), Creazione, Esecuzione, Analisi, and Tutti i prodotti.

The main content area is titled "Authentication". The "Metodo di accesso" tab is selected, showing the "Provider di accesso" section. It displays 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 it allows users to add an extra layer of security to their accounts via SMS. 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 button at the bottom right says "Esegui upgrade per abilitare".

At the bottom of the page, a footer bar shows the URL <https://console.firebaseio.google.com/u/0/project/unimol-44e4a/authentication/users>.

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), Scrittore di progetto, Authentication (selected), Novità, App Hosting (NEW), Data Connect (NEW), Creazione, Esecuzione, Analisi, and Tutti i prodotti. 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 is a section titled "Avanzate" (Advanced) containing information about "Autenticazione a più fattori tramite SMS" (Multi-factor authentication via SMS). It states: "Permetti ai tuoi utenti di aggiungere un ulteriore livello di sicurezza al proprio account. Una volta abilitato, integrato e configurato, gli utenti potranno accedere ai propri account in due fasi tramite un SMS." A link "Ulteriori informazioni" is provided. A note at the bottom says: "★ 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 button "Esegui upgrade per abilitare" is available.

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.

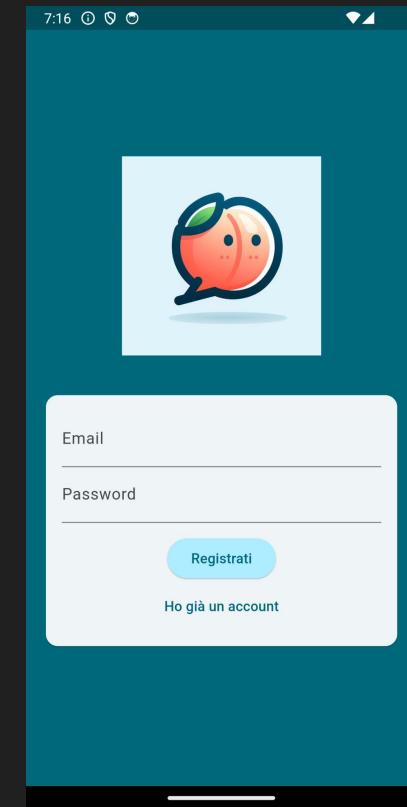
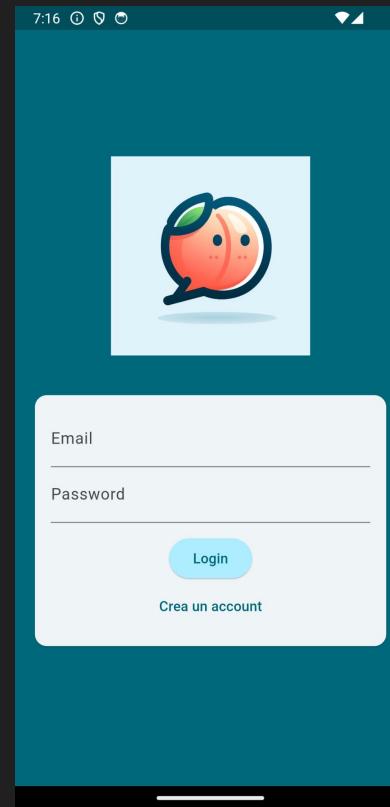
```
...  
    child: Image.asset('assets/images/chat.jpeg'),  
)  
Card(  
margin: const EdgeInsets.all(20),  
child: SingleChildScrollView  
child: Padding(  
padding: const EdgeInsets.all(16),  
child: Form(  
child: Column(  
mainAxisSize: MainAxisSize.min,  
children: [  
    TextFormField(  
decoration:  
        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),
                ...
              )
            ],
          ),
        ),
      ),
    );
  }
}
```

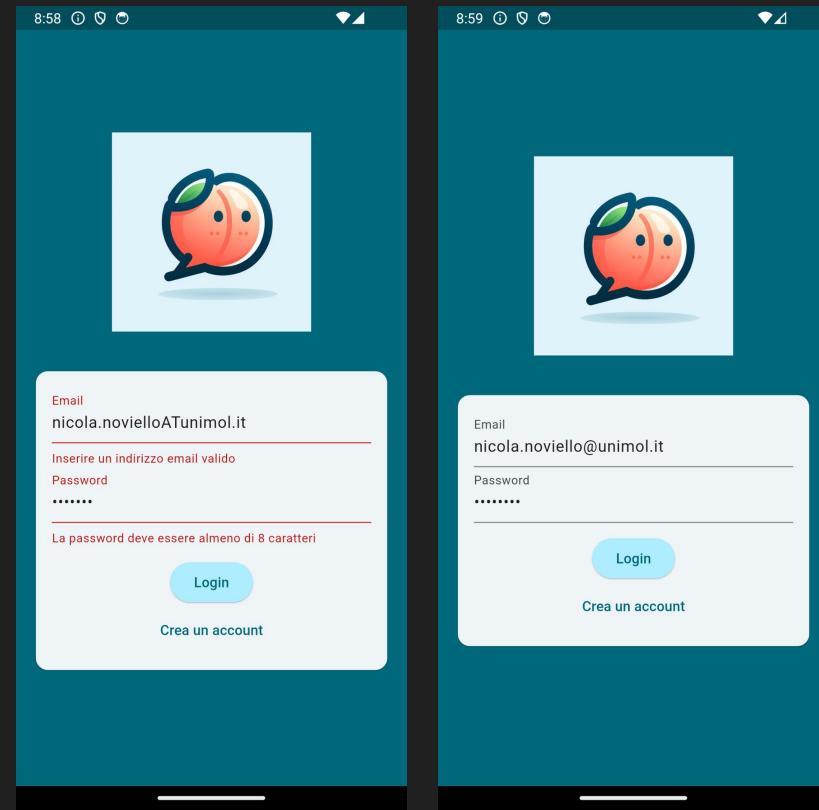
GlobalKey è un tipo di chiave in Flutter che fornisce un riferimento univoco a un widget specifico attraverso l'intero albero dei widget. È utile quando si desidera accedere allo stato di un widget da un'altra parte dell'app.

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

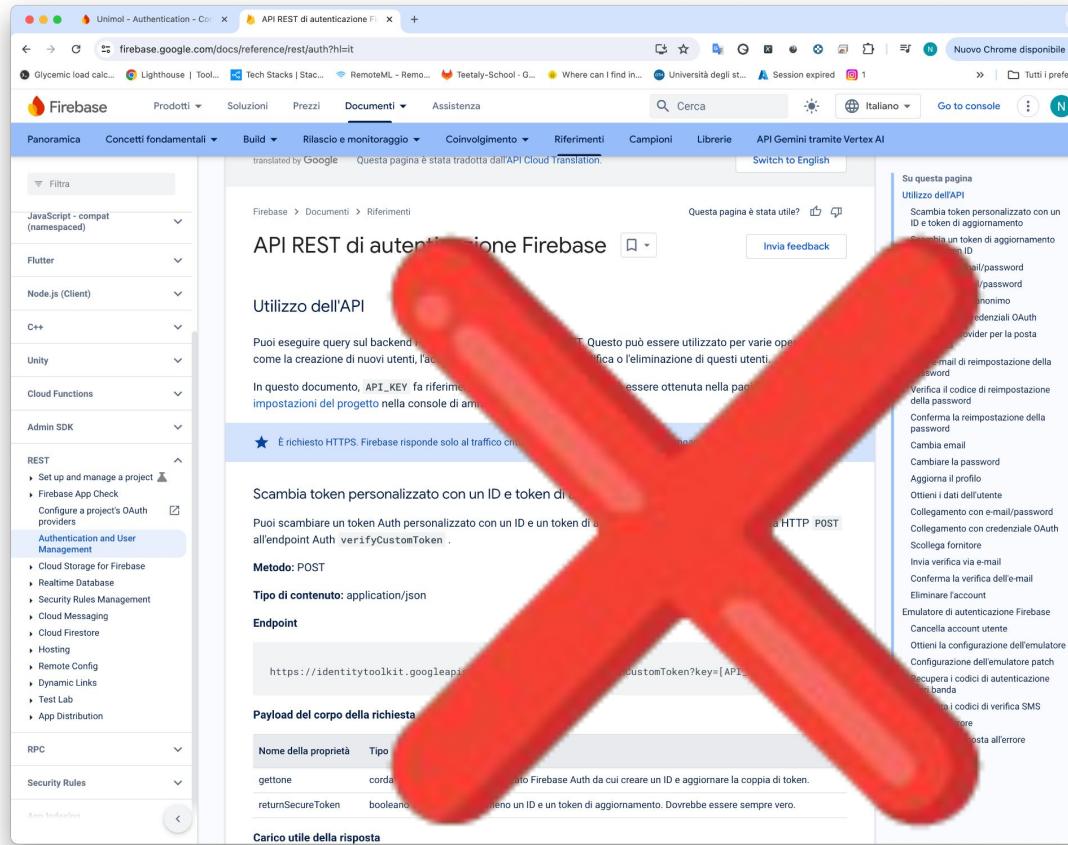
Nome della proprietà	Tipo	Descrizione
gettoken	corda	Un token personalizzato Firebase Auth da cui creare un ID e aggiornare la coppia di token.
returnSecureToken	booleano	Se restituire o meno un ID e un token di aggiornamento. Dovrebbe essere sempre vero.

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 web browser 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();

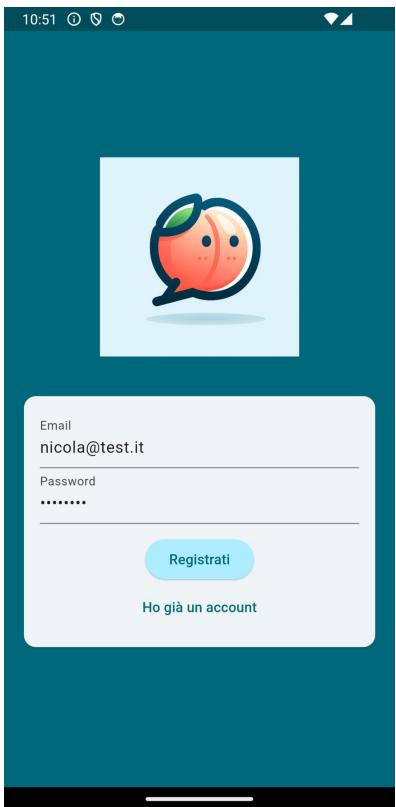
    if (!isValid) {
      return;
    }

    _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'),
            ),
          );
        }
      }
    }
  }
}
```

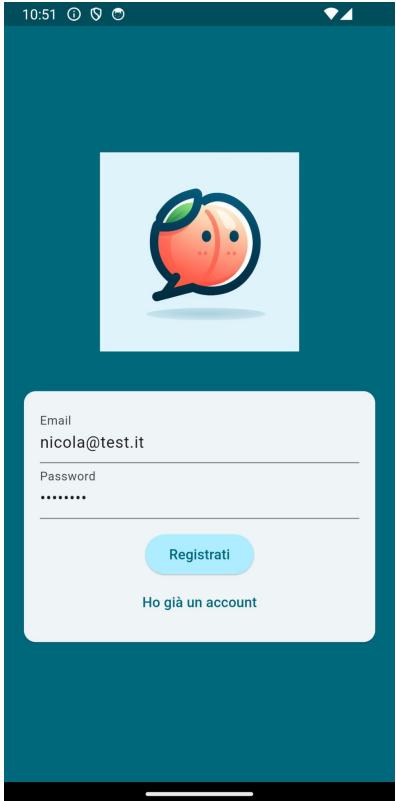
- Il codice controlla se l'utente sta cercando di accedere o di registrarsi attraverso la variabile `_isLogin`, al momento il codice per la login non è ancora implementato;
- Se `_isLogin` è false, allora l'utente sta cercando di registrarsi. Il codice tenta di creare un nuovo utente con l'email e la 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) x
```

Il risultato

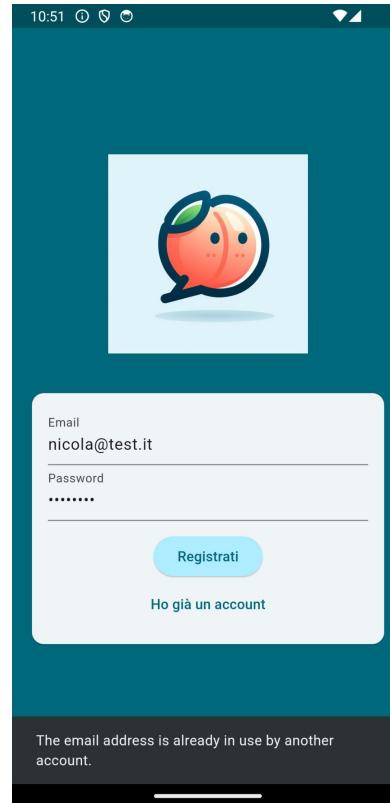


A screenshot of the Firebase Authentication console in a web browser. The URL is console.firebaseio.google.com/u/0/project/unimol-44e4a/authentication/users. The page shows a table of users:

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?



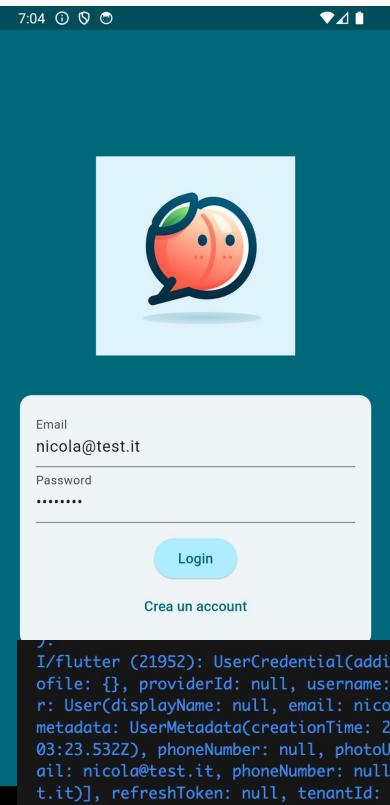
Login dell'utente

Login in AuthScreen

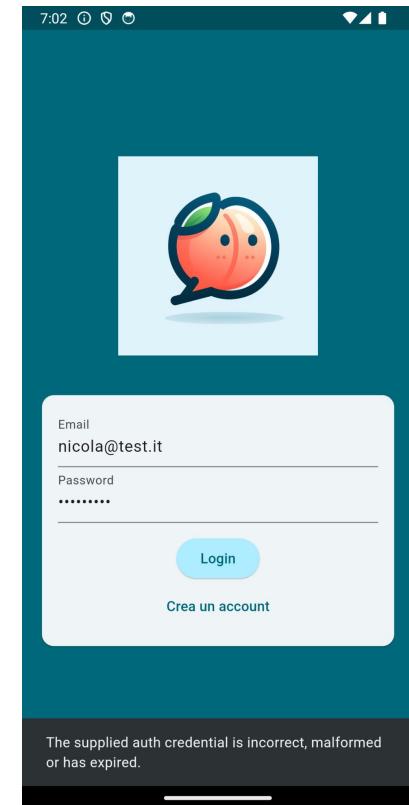
```
try {
  if (_isLogin) {
    final userCredentials = await _firebase.signInWithEmailAndPassword(
      email: _enteredEmail, password: _enteredPassword);
    print(userCredentials);
  } else {
    final userCredentials = await _firebase.createUserWithEmailAndPassword(
      email: _enteredEmail, password: _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.