Firebase

Programmazione Mobile

A.A. 2021/22

M.O. Spata

Cosa è Firebase? [1]

- Firebase Realtime Database è un database ospitato su cloud che supporta più piattaforme Android, iOS e Web.
- Tutti i dati vengono archiviati in formato JSON e qualsiasi modifica ai dati si riflette immediatamente eseguendo la sincronizzazione su tutte le piattaforme e i dispositivi.
- Questo ci consente di creare facilmente app in tempo reale più flessibili con il minimo sforzo.

Tutte le informazioni che trovate in queste slides fanno riferimento a: [1] https://firebase.google.com/docs/android/setup

Firebase

- In parole più semplici, possiamo dire che un Firebase Realtime Database è un tipo di database che è ospitato su cloud, ovvero gira sul cloud ed è accessibile da qualsiasi piattaforma, ovvero è accessibile contemporaneamente da Android, iOS e Web.
- Quindi, creiamo il database una volta e lo usiamo su piattaforme diverse. I dati archiviati nel database sono sotto forma di database NoSQL e sono archiviati in formato JSON.
- La parte migliore è che ogni volta che c'è una modifica nel database, verrà immediatamente riflessa su tutti i dispositivi ad esso collegati.

Vantaggi dell’uso di Firebase

Di seguito sono riportati i vantaggi dell'utilizzo di Firebase Realtime Database nel nostro progetto:

- **In tempo reale**: i dati archiviati nel database in tempo reale di Firebase si rifletteranno in tempo reale, ovvero se si verifica una modifica nei valori nel database, tale modifica verrà riflessa a tutti gli utenti solo in quell'istante e non ci sarà alcuna necessità di sincronizzare i dati.
- **Ampia accessibilità**: è possibile accedere al database Firebase Realtime da varie piattaforme come Android, iOS, Web. Quindi, non è necessario scrivere più volte lo stesso codice per piattaforme diverse.
- **Modalità offline**: questo è il miglior vantaggio dell'utilizzo di Firebase Realtime Database. Se non sei connesso a Internet e hai modificato qualcosa nella tua applicazione, tale modifica si rifletterà solo nella tua applicazione in quel momento, ma nel database Firebase, la modifica verrà aggiornata una volta che sarai online, ovvero il tuo dispositivo è connesso al Internet. Quindi, anche se non c'è Internet, l'utente ha la sensazione di utilizzare i servizi come quando c'è Internet.
- **Nessun server delle applicazioni**: qui non è necessario un server delle applicazioni perché si accede direttamente ai dati dal dispositivo mobile.
- **Controllo dell'accesso ai dati**: per impostazione predefinita, nessuno è autorizzato a modificare i dati nel database Firebase Realtime, ma puoi controllare l'accesso ai dati, ovvero puoi impostare quale utente può accedere ai dati.

JSON structrured data

- I dati archiviati nel database Firebase Realtime sono strutturati in JSON, ovvero l'intero database sarà un albero JSON con più nodi.
- Quindi, a differenza del database SQL, non abbiamo tabelle o record nell'albero JSON.
- Ogni volta che aggiungi alcuni dati all'albero JSON, diventa un nodo nella struttura JSON esistente con una chiave associata.
- Pertanto, tutti i dati del database Firebase Realtime vengono archiviati come oggetti JSON. Di seguito è riportato un esempio di dati strutturati JSON:

```
{
        "company": {
                "name": "MindOrks",
                "address": "Via Roma, 10"
        }
}
```

Database configuration rules

- I dati presenti nel database sono molto importanti e non dovresti dare accesso a tutti per utilizzare i dati presenti nel tuo database.
- Quindi, per fare ciò, Firebase Realtime Database ha alcune regole di configurazione del database che possono essere utilizzate per fornire accessi diversi a utenti diversi.
- Di seguito sono riportate le regole di configurazione del database…

Default

- Default: per impostazione predefinita, l'accesso in lettura e scrittura al database è disabilitato e nessuno può leggere o scrivere dati dal database in questa modalità.
- Ma ecco come puoi accedere ai dati del database solo dalla console Firebase.

```
// These rules don't allow anyone read or write
access to your database
{
        "rules": {
                ".read": false,
                ".write": false
        }
}
```

Public

- Public: utilizzando le regole pubbliche, chiunque può modificare i dati presenti nel database.
- Questa regola viene generalmente utilizzata durante il test dell'applicazione e dopo aver testato l'applicazione è possibile impostare la regola su Solo utente.

```
// These rules give anyone, even people who are not users of your app,
// read and write access to your database
{
        "rules": {
                ".read": true,
                ".write": true
        }
}
```

User

- User: in questa regola, l'utente della tua applicazione può leggere e scrivere nel tuo database.
- Puoi autenticare il tuo utente utilizzando Firebase Login and Authentication e, successivamente, il tuo utente avrà accesso al tuo database.

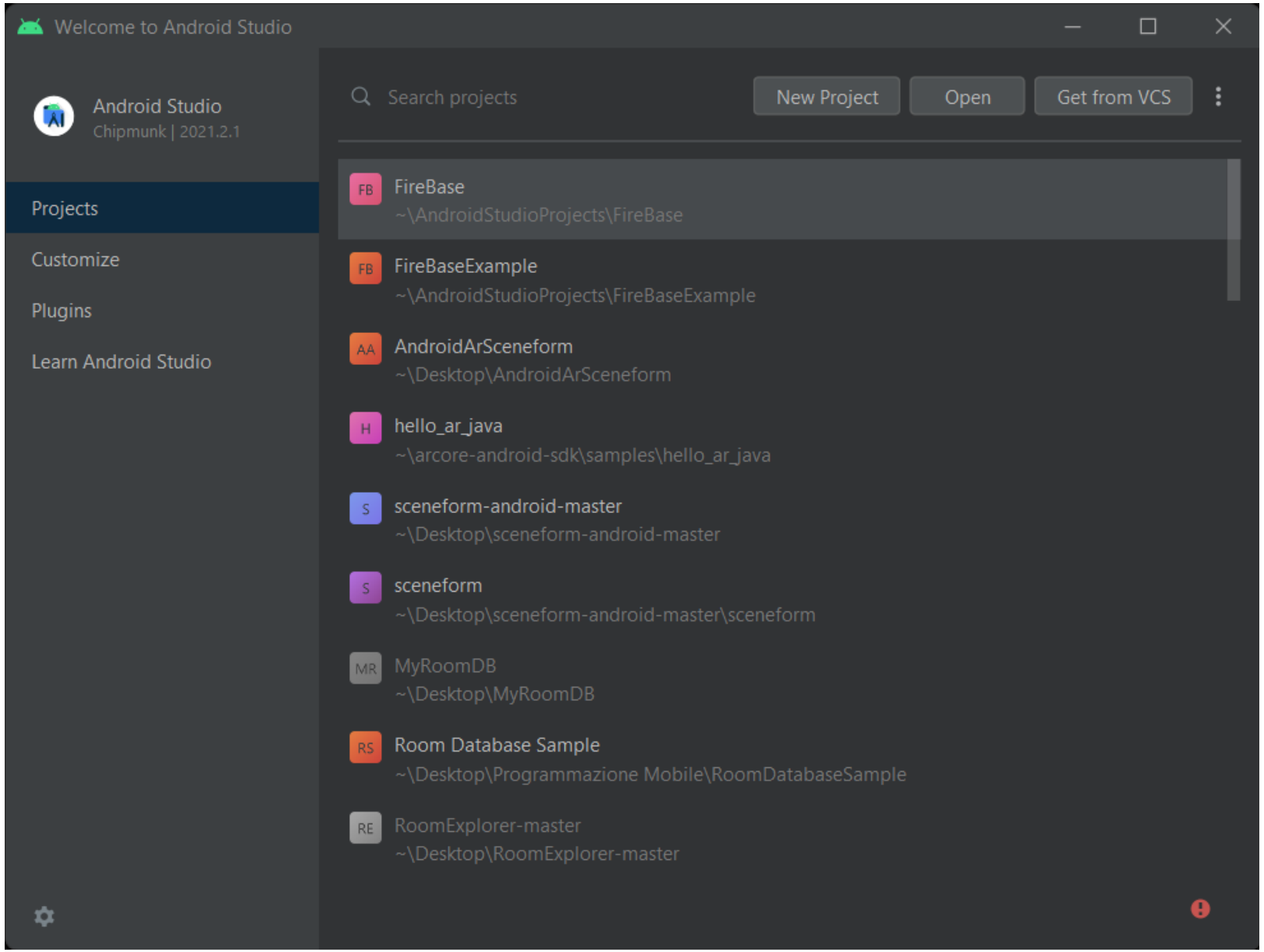
```
// These rules grant access to a node matching the authenticated
// user's ID from the Firebase auth token
{
        "rules": {
                "users": {
                        "$uid": {
                                ".read": "$uid === auth.uid",
                                ".write": "$uid === auth.uid"
                        }
                }
        }
}
```

Esempio

- In questa sezione, impareremo il concetto di Firebase Realtime Database con l'aiuto di un esempio.
- Nel nostro esempio, vedremo come creare un app android e configurare android studio per interfacciarsi al database Firebase.

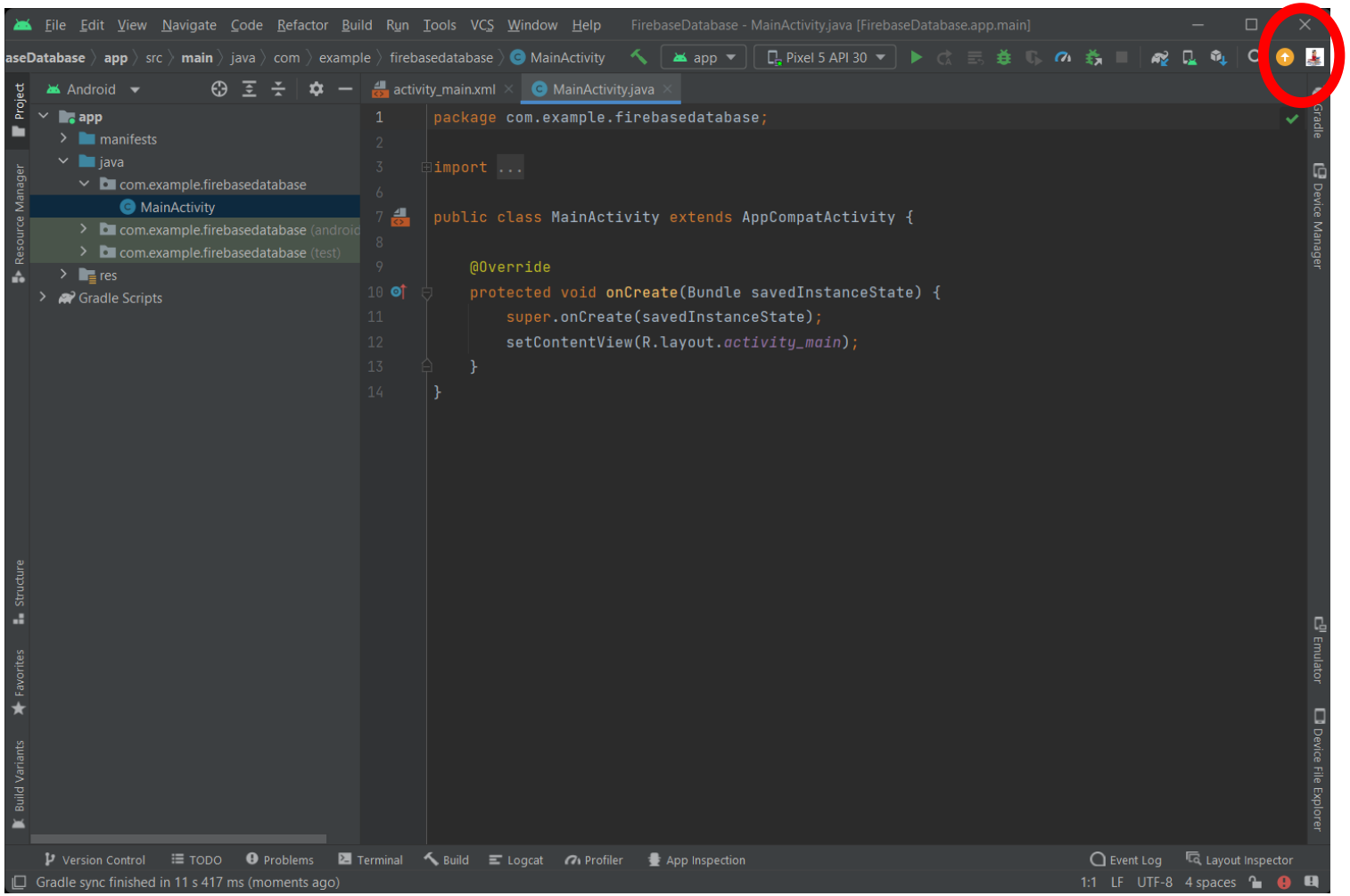
Step 1

- Apri Android Studio e crea un nuovo progetto o apri un progetto esistente.



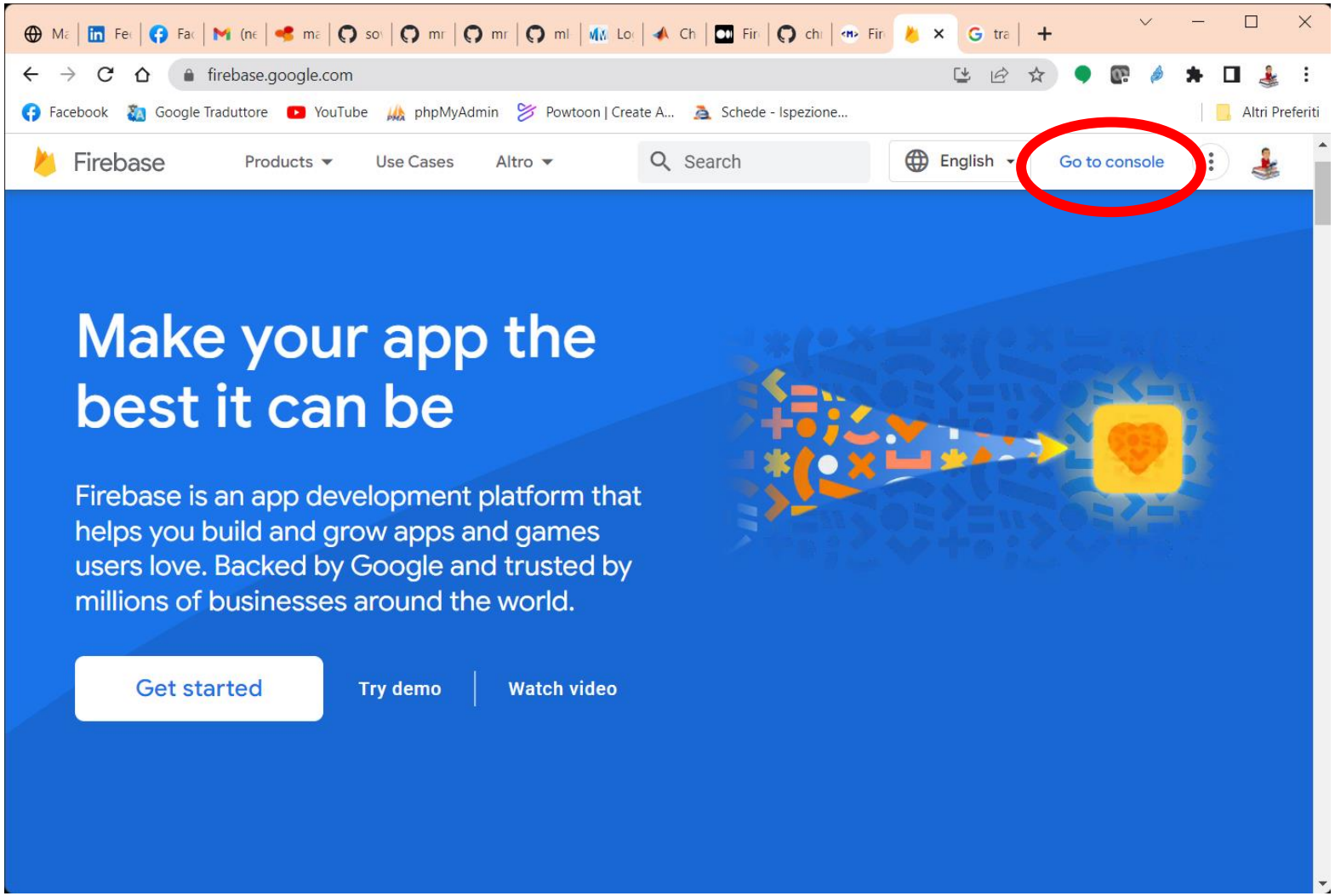
Step 2.

- In Android Studio, accedi con la tua email. Puoi trovare il pulsante di accesso nell'angolo in alto a destra di Android Studio.



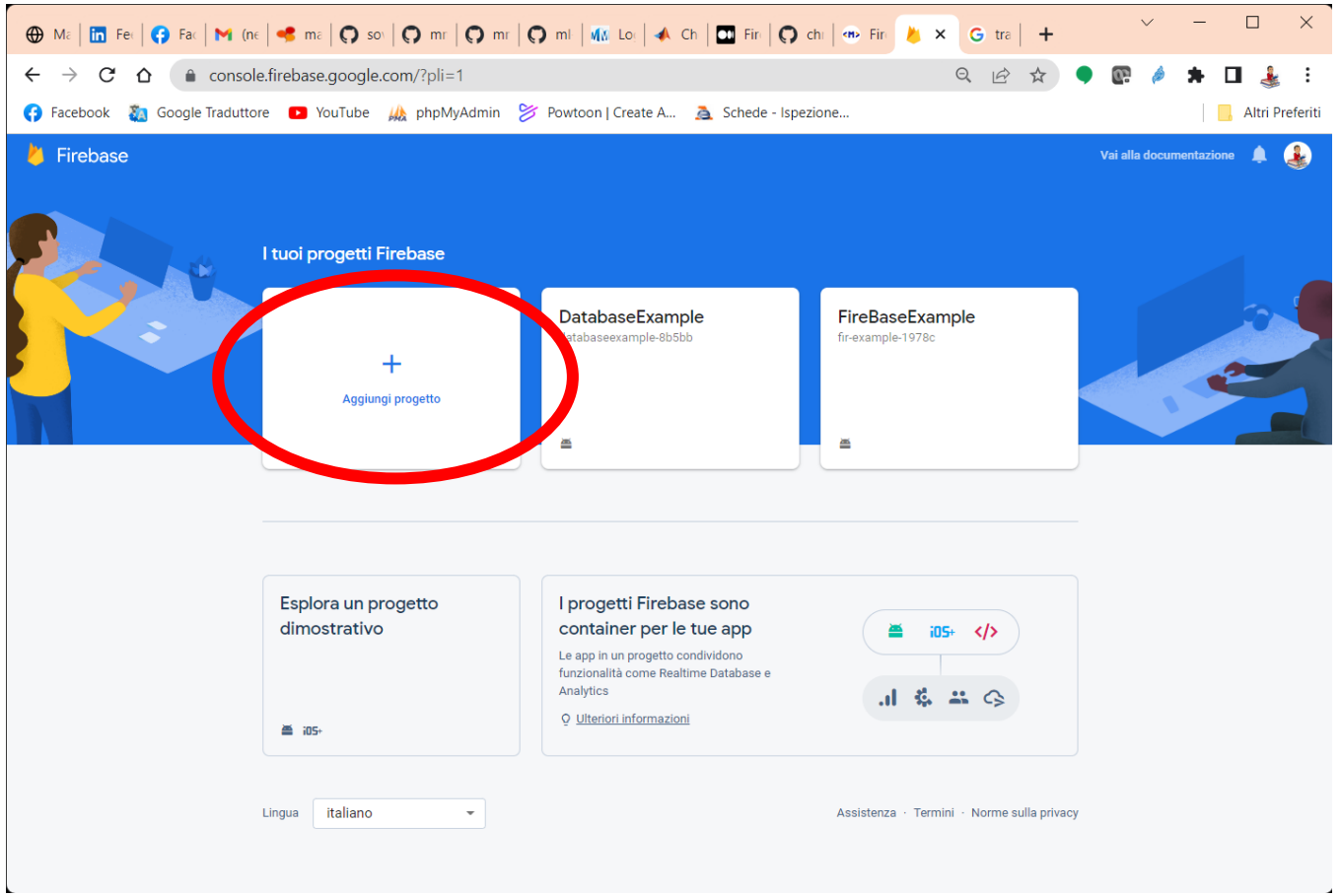
Step3:

- Apri il sito Web di Firebase e accedi ad esso. (utilizza lo stesso ID e-mail utilizzato in Android Studio per l'accesso)
- Dopo il login, clicca sul pulsante «Get started» oppure «Go to console» presente nella parte in alto a destra del sito.



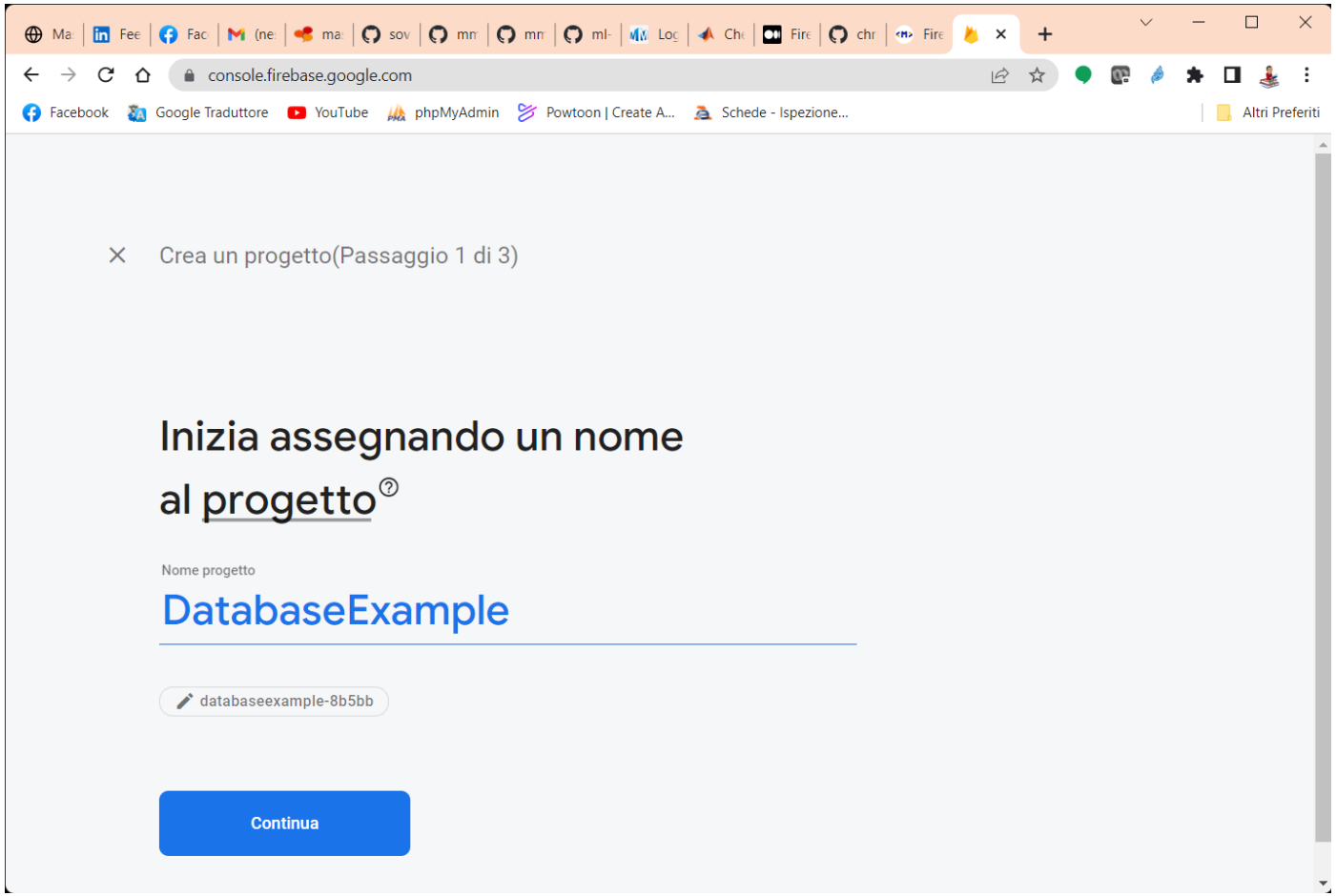
Step 4:

- Click su "Add Project"



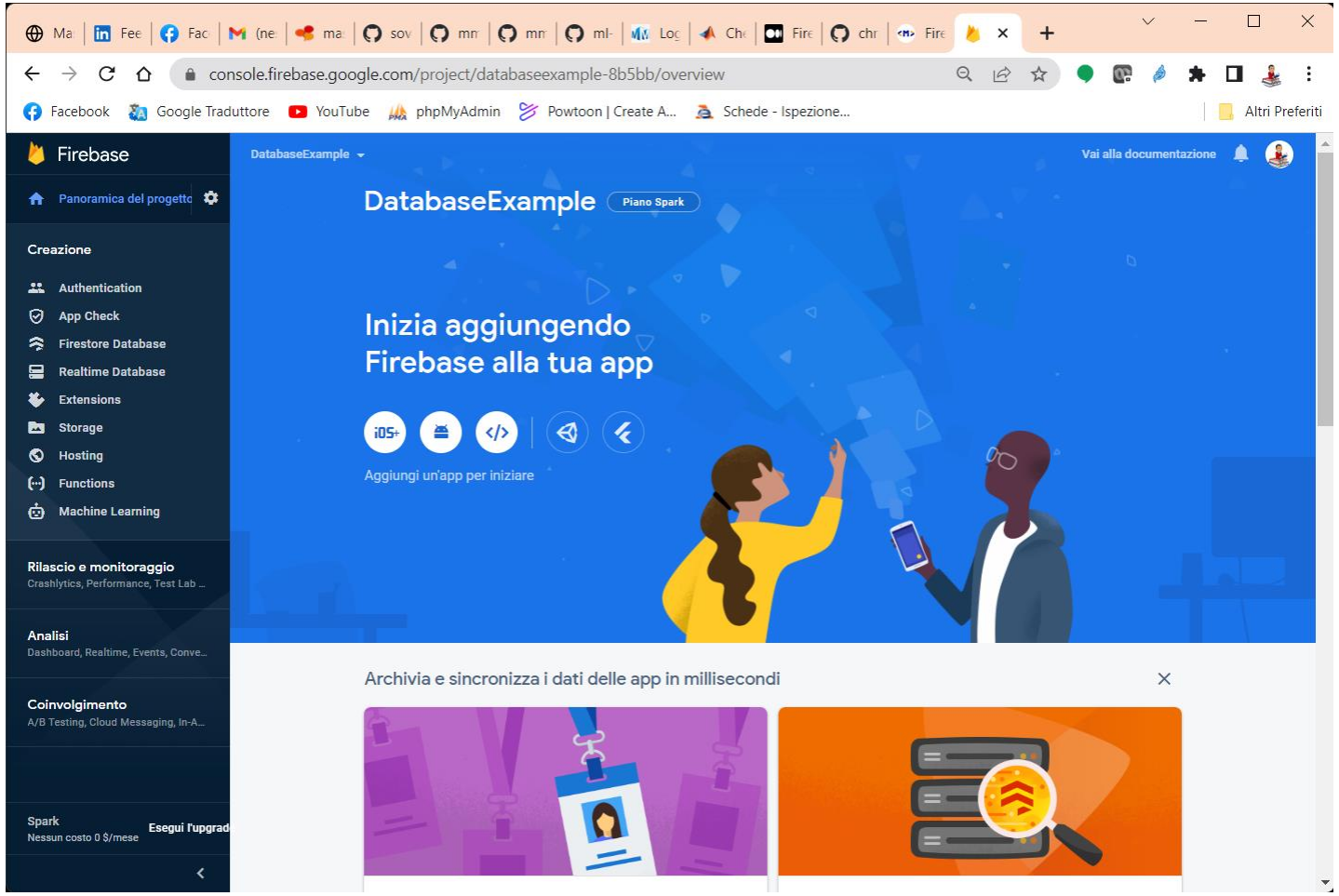
Step 5:

- Inserisci i dettagli richiesti del progetto e clicca su continua.



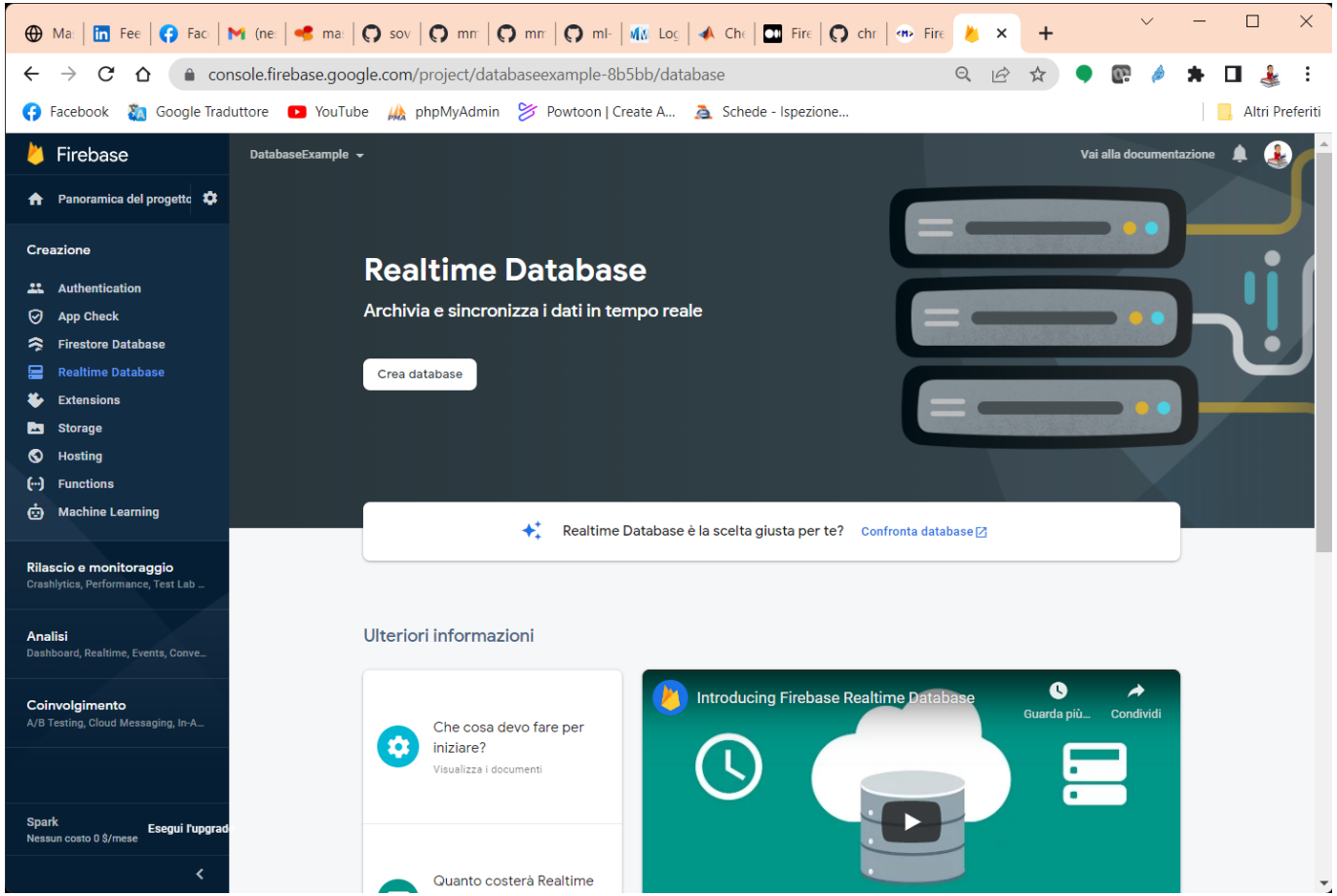
Step 6:

- Dopo aver creato un progetto, vedrai l'immagine sottostante della dashboard del tuo progetto.
- Qui vengono mostrati tutti i servizi di Firebase e puoi usarne uno qualsiasi.



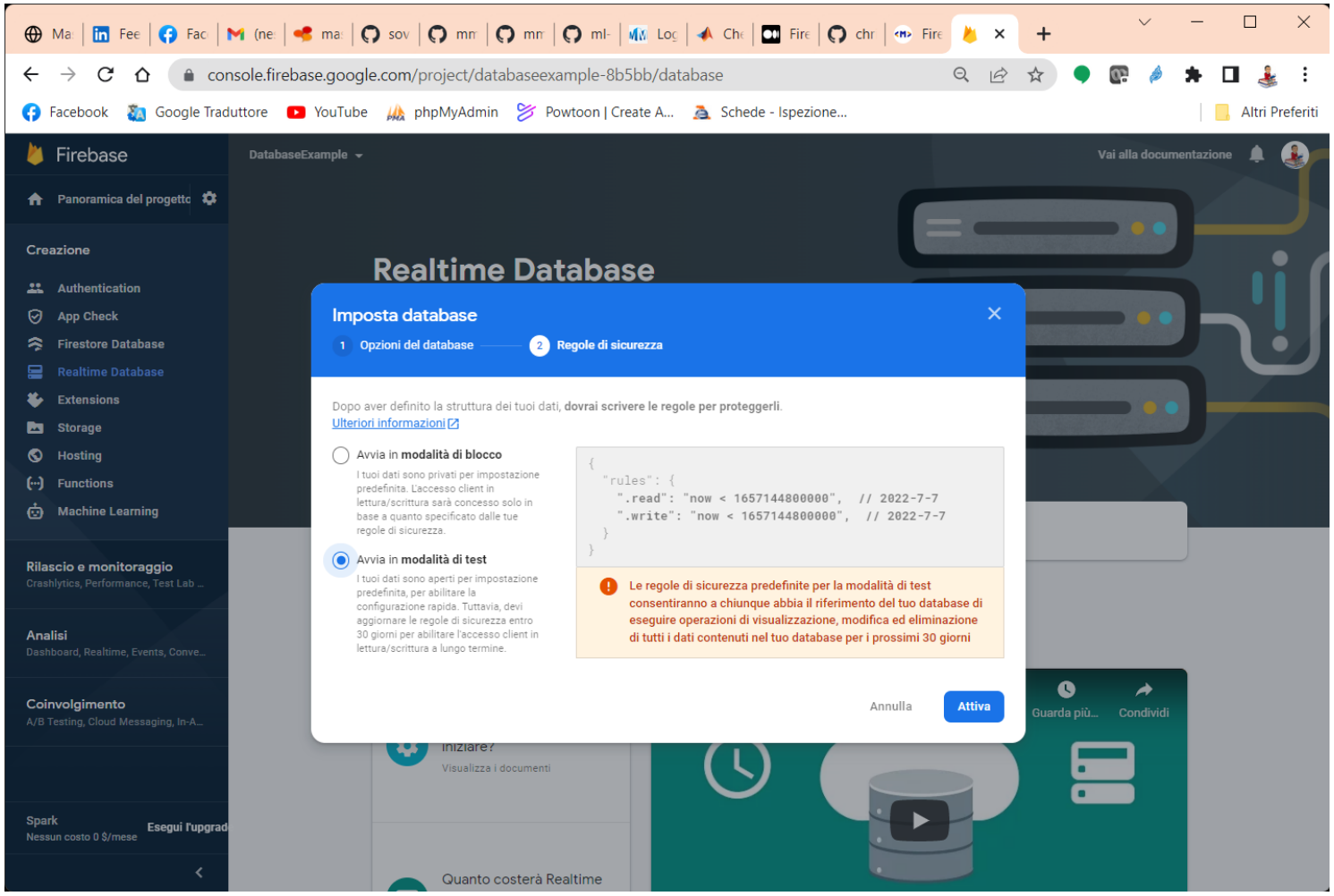
Step 7:

- Fare clic su "Database" e quindi nella sezione Database in tempo reale, fare clic su "Crea database".



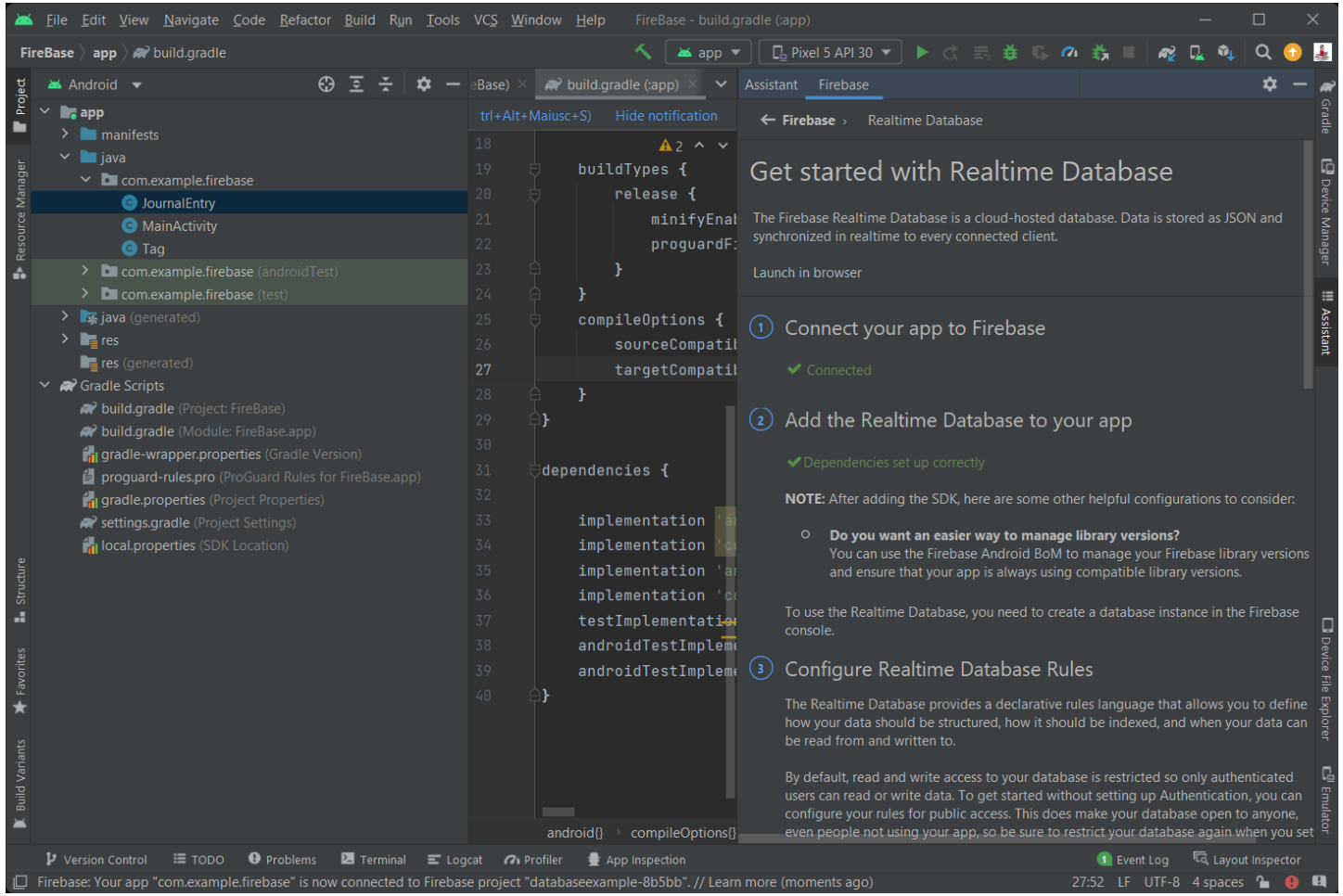
Step 8:

- Dal momento che stiamo solo usando il database per i nostri scopi pratici. Quindi, seleziona l'opzione "Avvia in modalità test" e fai clic su Attiva.



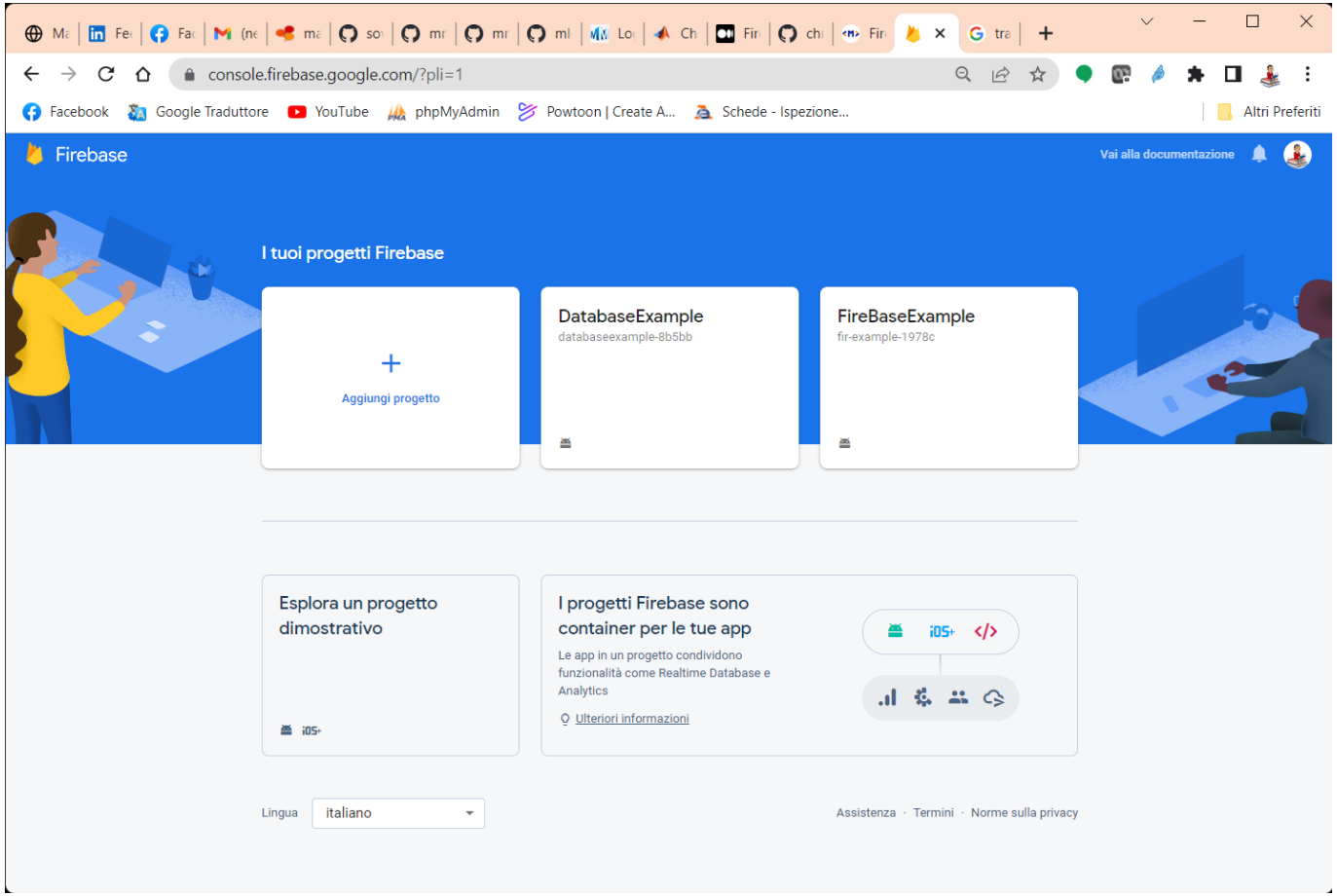
Step 9:

- Ora torna al tuo progetto Android Studio. Dobbiamo collegare il nostro progetto Firebase con il progetto Android Studio. Quindi, fai clic su Tools > Firebase > Realtime Database > Get started.



Step 10:

- Successivamente, fai clic su "Connetti a Firebase".
- Ti verrà mostrato un elenco di progetti.
- Seleziona il progetto che hai creato sul sito Web di Firebase e fai clic su "Connetti a Firebase".

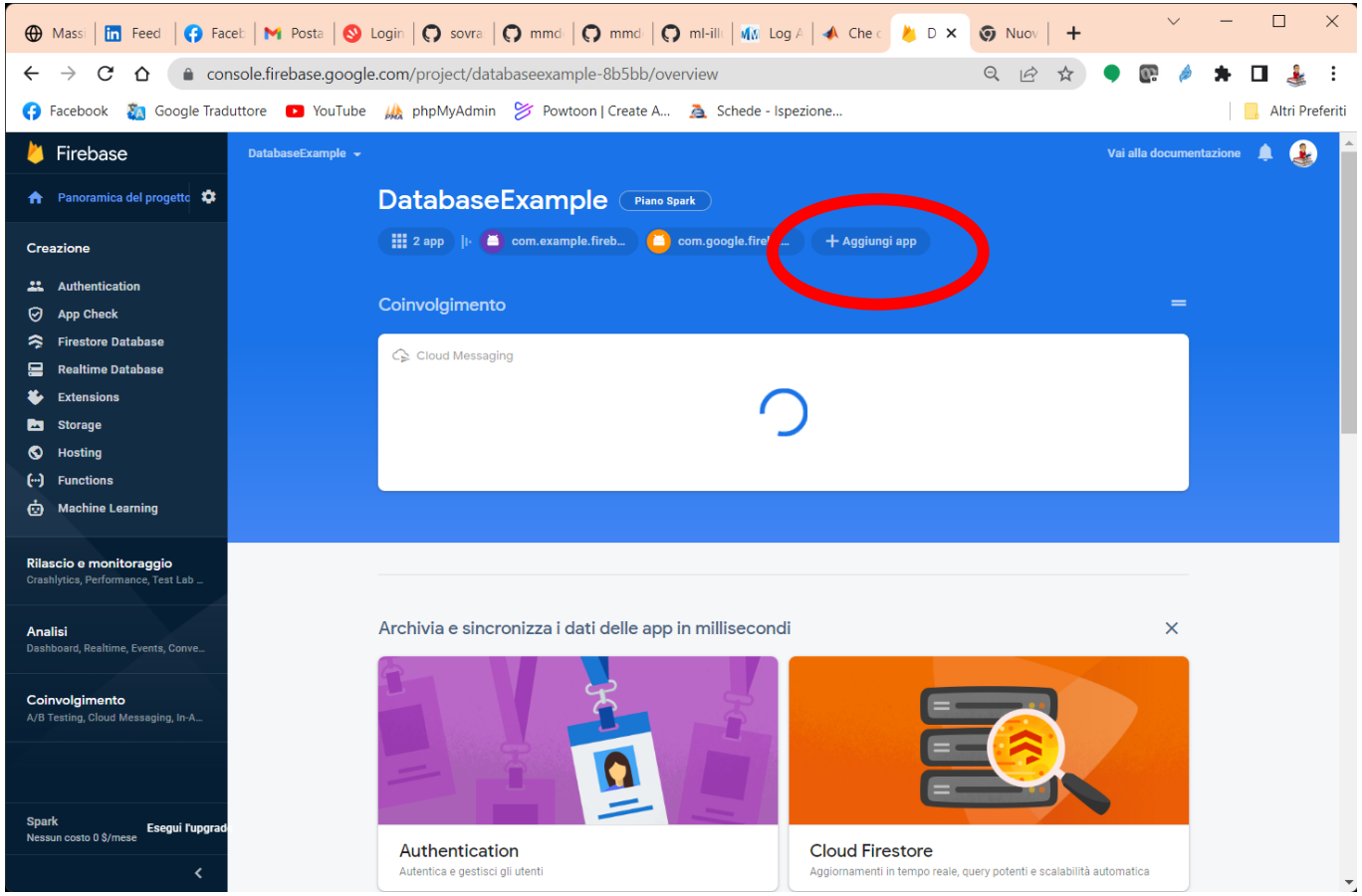


Step 11:

- Infine, devi aggiungere la dipendenza di Firebase Realtime Database nel tuo progetto facendo clic sul pulsante "Aggiungi Firebase Realtime Database alla tua app" e quindi su "Accetta modifiche".

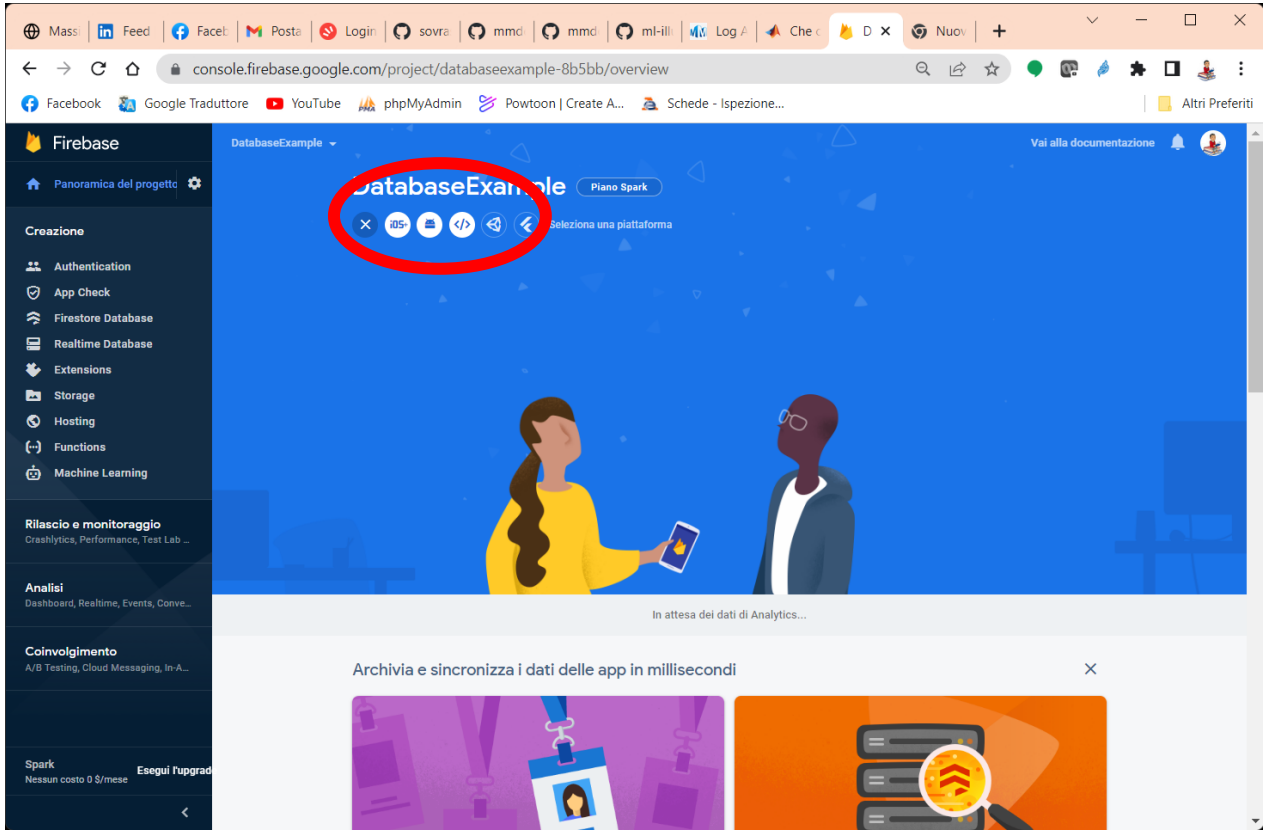
Step 12:

- Infine, andate sulla vostra console di Firebase e selezionate il progetto di database con il quale volete integrare la vostra app.
- Cliccate sul bottone in alto a destra aggiungi app



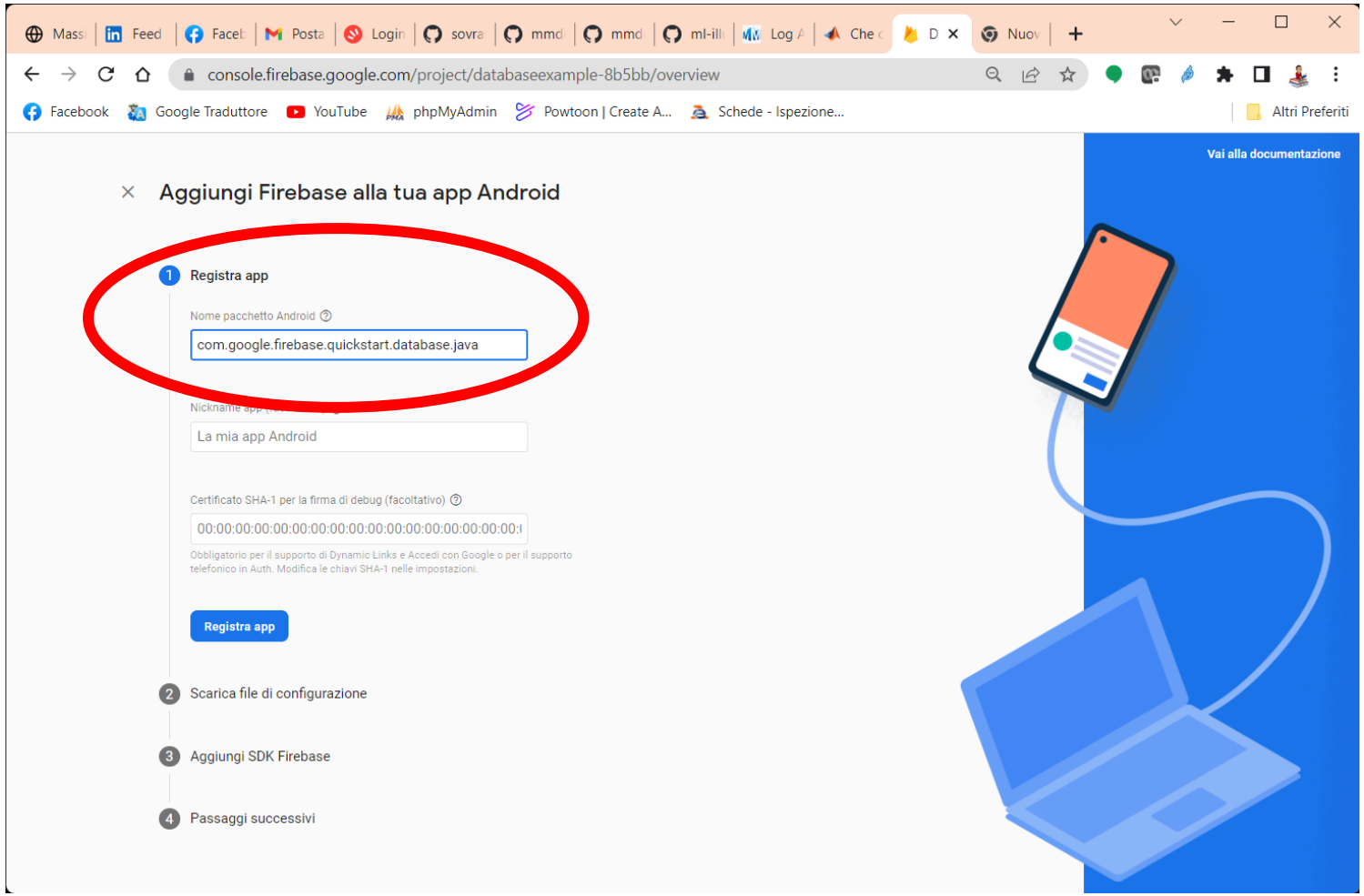
Step 12:

- Cliccate sul logo di android, in alto a sinistra.



Step 12:

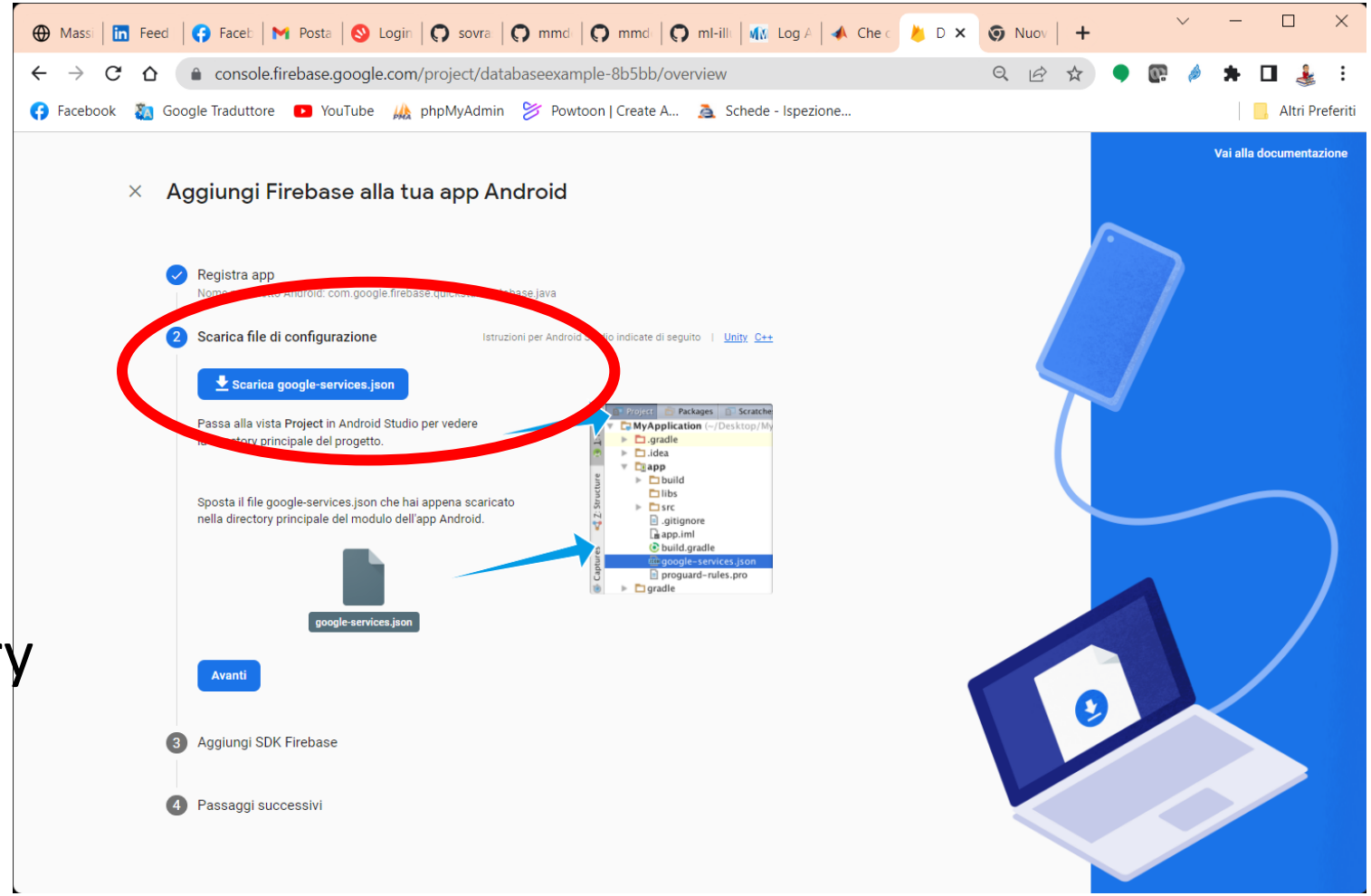
- Inserite il nome del vostro package usato nella text area in alto a sinistra su «registra app».
- Cliccate su registra app:



Step 12:

- Cliccate su scarica file di configurazione:

  google-services.json
- Cliccate su avanti
- Una volta scaricato il file con il drag and drop, aggiungetelo alla directory app della vostra applicazione android.



Ottieni un riferimento al database

- Per leggere o scrivere dati dal database, è necessaria un'istanza di DatabaseReference :

```
private DatabaseReference mDatabase;
// ...
mDatabase = FirebaseDatabase.getInstance().getReference();
```

Operazioni di scrittura di base

- Per le operazioni di scrittura di base, puoi utilizzare **setValue()** per salvare i dati in un riferimento specificato, sostituendo tutti i dati esistenti in quel percorso. Puoi usare questo metodo per i tipi che corrispondono ai tipi JSON disponibili come segue: String, Long, Double, Boolean, Map<String, Object>, List<Object>
- Se utilizzi un oggetto Java, il contenuto del tuo oggetto viene automaticamente mappato alle posizioni figlio in modo nidificato. L'utilizzo di un oggetto Java in genere rende anche il codice più leggibile e più facile da mantenere.

Operazioni di scrittura di base

Ad esempio, se disponi di un'app con un profilo utente di base, il tuo oggetto User potrebbe avere il seguente aspetto:

```
@IgnoreExtraProperties
public class User {

    public String username;
    public String email;

    public User() {
        // Default constructor required for calls to DataSnapshot.getValue(User.class)
    }

    public User(String username, String email) {
        this.username = username;
        this.email = email;
    }

}
```

Operazioni di scrittura di base

- Puoi aggiungere un utente con setValue() come segue:

```
public void writeNewUser(String userId, String name, String email) {
    User user = new User(name, email);

    mDatabase.child("users").child(userId).setValue(user);
}
```

Operazioni di scrittura di base

- L'uso setValue() in questo modo sovrascrive i dati nella posizione specificata, inclusi eventuali nodi figlio. Tuttavia, puoi comunque aggiornare un figlio senza riscrivere l'intero oggetto. Se vuoi consentire agli utenti di aggiornare i loro profili, puoi aggiornare il nome utente come segue:

```
mDatabase.child("users").child(userId).child("username").setValue(name);
```

Leggere i dati

- Per leggere i dati in un percorso e ascoltare le modifiche, usa il metodo **addValueEventListener()** per aggiungere un **ValueEventListener** a un **DatabaseReference**.
- È possibile utilizzare il metodo **onDataChange()** per leggere un'istantanea statica dei contenuti in un determinato percorso, poiché esistevano al momento dell'evento.
- Questo metodo viene attivato una volta quando l'ascoltatore è collegato e di nuovo ogni volta che i dati, inclusi i child, cambiano. Al callback dell'evento viene passato uno snapshot contenente tutti i dati in quella posizione, inclusi i dati figlio. Se non ci sono dati, lo snapshot restituirà false quando chiami **exists()** e null quando chiami **getValue()** su di esso.

Leggere i dati

- L'esempio seguente mostra un'applicazione di social blogging che recupera i dettagli di un post dal database:

```
ValueEventListener postListener = new ValueEventListener() {
    @Override
    public void onDataChange(DataSnapshot dataSnapshot) {
        // Get Post object and use the values to update the UI
        Post post = dataSnapshot.getValue(Post.class);
        // ..
    }

    @Override
    public void onCancelled(DatabaseError databaseError) {
        // Getting Post failed, log a message
        Log.w(TAG, "loadPost:onCancelled", databaseError.toException());
    }
};
mPostReference.addValueEventListener(postListener);
```

Aggiornamento o cancellazione dei dati

- Per scrivere simultaneamente su figli specifici di un nodo senza sovrascrivere altri nodi figli, utilizzare il metodo updateChildren() .

- Quando si chiama updateChildren() , è possibile aggiornare i valori figlio di livello inferiore specificando un percorso per la chiave. Se i dati vengono archiviati in più posizioni per una migliore scalabilità, puoi aggiornare tutte le istanze di tali dati utilizzando il fan-out dei dati .

Aggiornamento o cancellazione dei dati

- Ad esempio, un'app di social blogging potrebbe avere una classe Post come questa:

```
@IgnoreExtraProperties
public class Post {

    public String uid;
    public String author;
    public String title;
    public String body;
    public int starCount = 0;
    public Map<String, Boolean> stars = new HashMap<>();

    public Post() {
        // Default constructor required for calls to DataSnapshot.getValue(Post.class)
    }

    public Post(String uid, String author, String title, String body) {
        this.uid = uid;
        this.author = author;
        this.title = title;
        this.body = body;
    }

    @Exclude
    public Map<String, Object> toMap() {
        HashMap<String, Object> result = new HashMap<>();
        result.put("uid", uid);
        result.put("author", author);
        result.put("title", title);
        result.put("body", body);
        result.put("starCount", starCount);
        result.put("stars", stars);

        return result;
    }
}
```

Aggiornamento o cancellazione dei dati

- Per creare un post e aggiornarlo contemporaneamente al feed delle attività recenti e al feed delle attività dell'utente che pubblica, l'applicazione di blog utilizza un codice come questo:

```
private void writeNewPost(String userId, String username, String title,
String body) {
    // Create new post at /user-posts/$userid/$postid and at
    // /posts/$postid simultaneously
    String key = mDatabase.child("posts").push().getKey();
    Post post = new Post(userId, username, title, body);
    Map<String, Object> postValues = post.toMap();

    Map<String, Object> childUpdates = new HashMap<>();
    childUpdates.put("/posts/" + key, postValues);
    childUpdates.put("/user-posts/" + userId + "/" + key, postValues);

    mDatabase.updateChildren(childUpdates);
}
```

Aggiornamento o cancellazione dei dati

- Questo esempio utilizza push() per creare un post nel nodo contenente post per tutti gli utenti in /posts/$postid e recuperare contemporaneamente la chiave con getKey() . La chiave può quindi essere utilizzata per creare una seconda voce nei post dell'utente in /user-posts/$userid/$postid .

- Utilizzando questi percorsi, puoi eseguire aggiornamenti simultanei in più posizioni nell'albero JSON con una singola chiamata a updateChildren() , ad esempio come questo esempio crea il nuovo post in entrambe le posizioni. Gli aggiornamenti simultanei effettuati in questo modo sono atomici: tutti gli aggiornamenti riescono o tutti gli aggiornamenti falliscono.

Elimina i dati

- Il modo più semplice per eliminare i dati consiste nel chiamare removeValue() su un riferimento alla posizione di quei dati.

- Puoi anche eliminare specificando null come valore per un'altra operazione di scrittura come setValue() o updateChildren() . Puoi utilizzare questa tecnica con updateChildren() per eliminare più elementi figlio in una singola chiamata API.

Distacca gli ascoltatori

- I callback vengono rimossi chiamando il metodo **removeEventListener()** sul riferimento del database Firebase.

- Se un listener è stato aggiunto più volte a una posizione dati, viene chiamato più volte per ogni evento ed è necessario scollegarlo lo stesso numero di volte per rimuoverlo completamente.

- La chiamata a **removeEventListener()** su un listener padre non rimuove automaticamente i listener registrati sui suoi nodi figlio; **removeEventListener()** deve essere chiamato anche su qualsiasi listener figlio per rimuovere il callback.