

## CHATTING APPLICATION

Oggetto: creazione di un'applicazione base di chatting che consente di chattare con altri utenti.

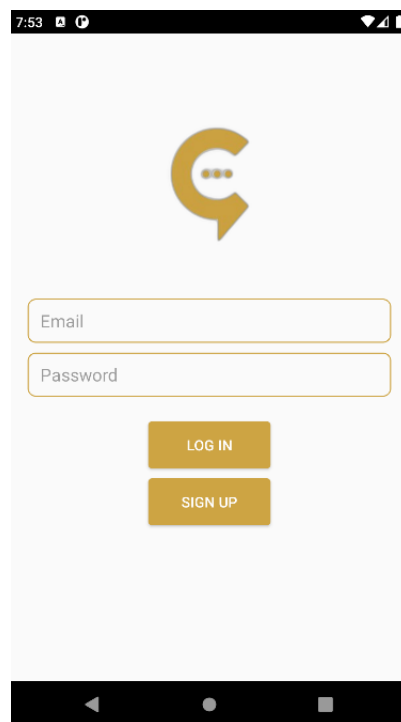
Linguaggio di programmazione: Kotlin

Strumenti e librerie: Firebase

IDE: Android Studio

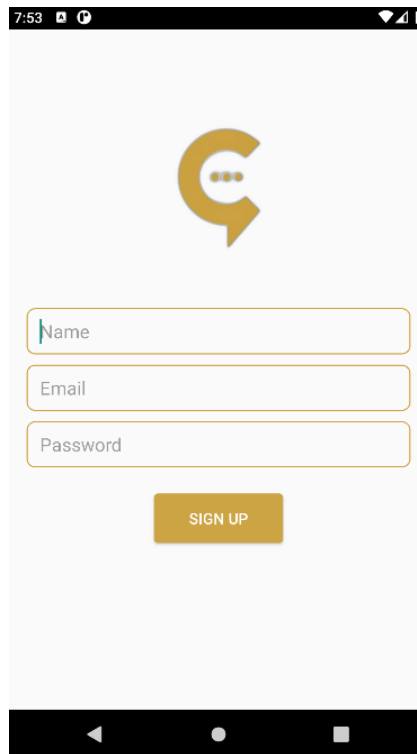
- **res/layout/activity\_login.xml**

In questa parte sono definite tutte le caratteristiche estetiche della pagina di login, aggiungendo colori personalizzati (`<color name="gold">#CDA443</color>` res/values/colors.xml) e sfondi personalizzati (res/drawable/edt\_background.xml).



- **res/layout/activity\_sign\_up.xml**

In questa parte sono state definite tutte le caratteristiche estetiche della pagina di iscrizione, aggiungendo colori e sfondi personalizzati.



- [java/com.example.chattingapplication/Login.kt](#) & [java/com.example.chattingapplication/SignUp.kt](#)

In questa parte la pagina di login è collegata a quella di registrazione.  
Quando nella pagina di accesso viene cliccato il pulsante SIGN UP la pagina cambia.

```

33 btnSignUp.setOnClickListener { it: View!
34     val intent = Intent( packageContext: this, SignUp::class.java)
35     startActivity(intent)
36 }

```

Useremo Firebase per autenticare ogni utente. Firebase ci offre molti metodi per farlo, in questo caso useremo e-mail e password.

Per utilizzare l'autenticazione nella mia app:

- Inizializzazione dell'autenticazione

```

26 mAuth = FirebaseAuth.getInstance()

```

- Log in un utente
  - Ottenimento dell'e-mail e della password dall'utente

```

38         btnLogin.setOnClickListener { it: View!
39             val email = edtEmail.text.toString()
40             val password = edtPassword.text.toString()
41
42             login(email, password)
43         }
44     }

```

Creare un metodo che consenta l'accesso di un utente.

Se l'accesso dell'utente ha successo, si passa alla pagina della chat ([java/com.example.chattingapplication/MainActivity.kt](#)), se non lo ha, appare un messaggio per notificare l'errore.

```

46     private fun login(email: String, password: String) {
47         mAuth.signInWithEmailAndPassword(email, password)
48             .addOnCompleteListener(this) { task ->
49             if (task.isSuccessful) {
50                 val intent = Intent(packageContext, this@Login, MainActivity::class.java)
51                 finish()
52                 startActivity(intent)
53             } else {
54                 Toast.makeText(context, this@Login, text: "User does not exists",
55                     Toast.LENGTH_SHORT).show()
56             }
57         }
58     }

```

Per la registrazione, la procedura è quasi la stessa.

A differenza del login, si ottiene un valore extra dall'utente: il nome.

```

36         btnSignUp.setOnClickListener{ it: View!
37             val name = edtName.text.toString()
38             val email = edtEmail.text.toString()
39             val password = edtPassword.text.toString()
40
41             signUp(name, email, password)
42         }
43     }

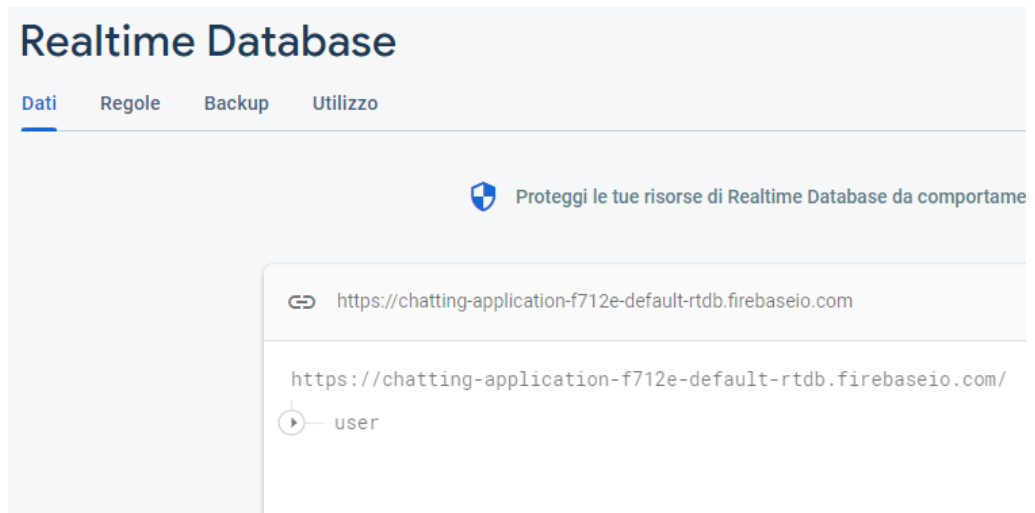
```

Il nuovo utente deve essere aggiunto nel database creando un nuovo nodo al suo interno.

```

59     private fun addUserToDatabase(name: String, email: String, uid: String) {
60         mDbRef = FirebaseDatabase.getInstance().getReference()
61
62         mDbRef.child(pathString: "User").child(uid).setValue(User(name, email, uid))
63     }
64 }

```



- **java/com.example.chattingapplication/User.kt**

Questa classe viene utilizzata per memorizzare i valori degli utenti.  
Ogni utente su Firebase ha 3 valori: nome, email e uid (id univoco).

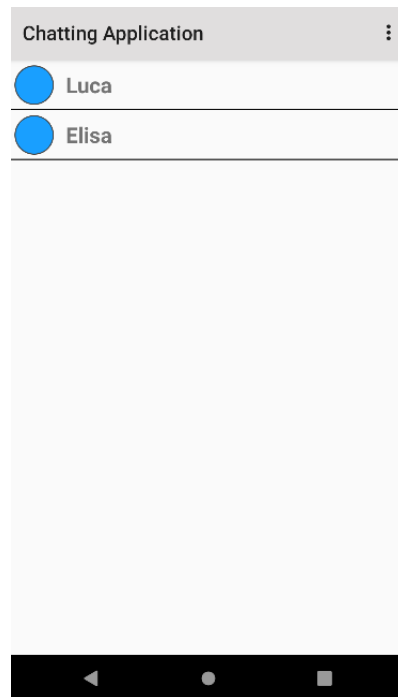
```
3 class User {
4     var name: String? = null
5     var email: String? = null
6     var uid: String? = null
7
8     constructor() {}
9
10    constructor(name: String?, email: String?, uid: String?) {
11        this.name = name
12        this.email = email
13        this.uid = uid
14    }
15 }
```

- **res/layout/user\_layout.xml**

In questa parte viene definito come appare un singolo utente nella pagina della chat.

- **res/layout/activity\_main.xml**

In questa parte vengono mostrati tutti gli utenti.



- **java/com.example.chattingapplication/UserAdapter.kt**

La classe UserAdapter estende RecyclerView. RecyclerView è una vista flessibile per fornire una finestra limitata in un set di dati di grandi dimensioni.

```
12 class UserAdapter(val context: Context, val userList: ArrayList<User>):
13     RecyclerView.Adapter<UserAdapter.UserViewHolder>() {
```

`RecyclerView.Adapter` fornisce un'associazione da un set di dati specifico dell'app alle viste visualizzate all'interno di RecyclerView.

Creazione della classe UserViewHolder in cui verranno inizializzate tutte le viste già create res/layout/user\_layout.xml.

```
39 class UserViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
40     val txtName = itemView.findViewById<TextView>(R.id.txt_name)
41 }
```

`onCreateViewHolder` la funzione viene chiamata quando RecyclerView necessita di un nuovo RecyclerView.ViewHolder del tipo specificato per rappresentare un elemento.

Questo nuovo ViewHolder dovrebbe essere costruito con una nuova vista che può rappresentare gli elementi del tipo specificato.

```
15 override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): UserViewHolder {
16     val view: View = LayoutInflater.from(context)
17         .inflate(R.layout.user_layout, parent, attachToRoot: false)
18     return UserViewHolder(view)
19 }
```

`onBindViewHolder` la funzione viene chiamata da RecyclerView per visualizzare i dati nella posizione specificata. Questo metodo dovrebbe aggiornare il contenuto di `RecyclerView.ViewHolder.itemView` per riflettere l'elemento nella posizione specificata. In particolare mostra il nome dell'utente corrente.

```
21 override fun onBindViewHolder(holder: ViewHolder, position: Int) {
22     val currentUser = userList[position]
23
24     holder.textName.text = currentUser.name
25
26     holder.itemView.setOnClickListener { it: View!
27         val intent = Intent(context, ChatActivity::class.java)
28
29         intent.putExtra(name: "name", currentUser.name)
30         intent.putExtra(name: "uid", currentUser.uid)
31
32         context.startActivity(intent)
33     }
34 }
```

`getItemCount` la funzione restituisce il numero totale di elementi nel set di dati detenuti dall'adattatore. Quindi restituisce il numero di utenti.

```
36 override fun getItemCount(): Int {
37     return userList.size
38 }
```

- **[java/com.example.chattingapplication/MainActivity.kt](#)**

Una delle funzionalità di questa pagina è il logout. Quando vengono premuti i tre punti in alto a destra, compare la possibilità di logout.

```
61 override fun onOptionsItemSelected(item: MenuItem): Boolean {
62     if (item.itemId == R.id.logout) {
63         mAuth.signOut()
64         val intent = Intent(packageContext: this@MainActivity, Login::class.java)
65         finish()
66         startActivity(intent)
67         return true
68     }
69     return true
70 }
71 }
```

Chatting Application

Log Out

Per leggere il valore nel nostro database e mostrare tutte le persone con cui posso chattare, ovviamente omettendo il nome con cui accedo:

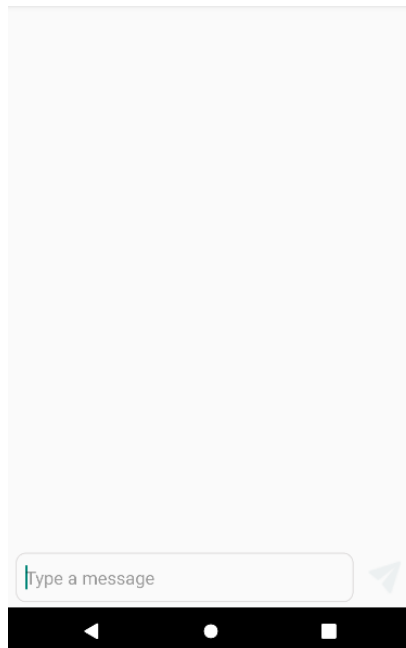
```

37      mDbRef.child( pathString: "user").addValueEventListener(object: ValueEventListener{
38  override fun onDataChange(snapshot: DataSnapshot) {
39      userList.clear()
40      for (postSnapshot in snapshot.children) {
41          val currentUser = postSnapshot.getValue(User::class.java)
42
43          if (mAuth.currentUser?.uid != currentUser?.uid) {
44              userList.add(currentUser!!)
45          }
46      }
47      adapter.notifyDataSetChanged()
48  }

```

- **res/layout/activity\_chat.xml**

Come appare la chat.



- **java/com.example.chattingapplication/ChatActivity.kt**

Cliccando sul pulsante “invia”, quanto scritto nella finestra di messaggio viene aggiunto al database. Successivamente, la finestra di messaggio viene svuotata.

Viene creato un nuovo nodo per la chat, che si divide in: chi ha inviato il messaggio e chi ha ricevuto il messaggio. Entrambi i membri devono essere aggiornati quando viene inviato un nuovo messaggio.


```

66 //adding the message to the database
67 sendButton.setOnClickListener { it: View!
68     val message = messageBox.text.toString()
69     val messageObject = Message(message, senderUid)
70
71     mDbRef.child( pathString: "chats").child(senderRoom!!).child( pathString: "messages").push()
72     .setValue(messageObject).addOnSuccessListener { it: Void!
73         mDbRef.child( pathString: "chats").child(receiverRoom!!).child( pathString: "messages").push()
74         .setValue(messageObject)
75     }
76     messageBox.setText("")
77 }

```

## Realtime Database

[Dati](#) [Regole](#) [Backup](#) [Utilizzo](#)

 Proteggi le tue risorse di Realtime Database da comportamenti non autorizzati

<https://chatting-application-f712e-default-rtdb.firebaseio.com>

<https://chatting-application-f712e-default-rtdb.firebaseio.com/>

 chats  
 user

Mostra i dati dal database in RecyclerView.

Quando un messaggio viene inviato viene chiamata la funzione `onDataChange` e il messaggio viene aggiunto all'elenco dei messaggi.

```

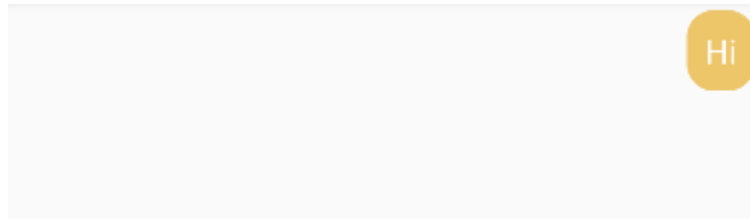
50 mDbRef.child( pathString: "chats").child(senderRoom!!).child( pathString: "messages")
51 .addValueEventListener(object: ValueEventListener {
52     override fun onDataChange(snapshot: DataSnapshot) {
53         messageList.clear()
54         for (postSnapshot in snapshot.children) {
55             val message = postSnapshot.getValue(Message::class.java)
56             messageList.add(message!!)
57         }
58         messageAdapter.notifyDataSetChanged()
59     }
60
61     override fun onCancelled(error: DatabaseError) {
62     }
63 }
64 })

```

- **res/layout/sent.xml**

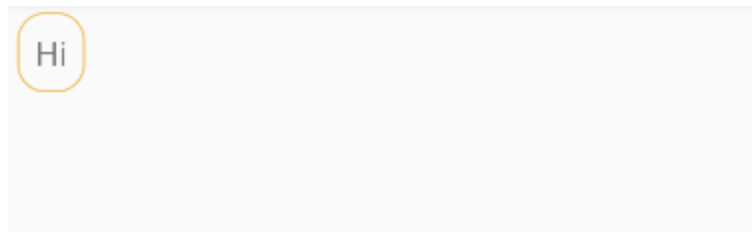
Come appare un messaggio inviato.





- res/layout/received.xml

Come appare un messaggio ricevuto.



- java/com.example.chattingapplication/Message.kt

Questo è il modello per il messaggio.

```

9      constructor(message: String?, senderId: String?) {
10         this.message = message
11         this.senderId = senderId
12     }

```

- java/com.example.chattingapplication/MessageAdapter.kt

Questa volta abbiamo bisogno di due ViewHolder: uno per ricevere il messaggio e uno per inviarlo.

```

53     class SentViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
54         val sentMessage = itemView.findViewById<TextView>(R.id.txt_sent_message)
55     }
56
57     class ReceiveViewHolder(itemView: View): RecyclerView.ViewHolder(itemView) {
58         val receivedMessage = itemView.findViewById<TextView>(R.id.txt_received_message)
59     }

```

`getItemViewType` la funzione restituisce un numero intero a seconda del tipo di vista.

```

14         val ITEM_RECEIVED = 1
15         val ITEM_SENT = 2

```

```

39 override fun getItemViewType(position: Int): Int {
40     val currentMessage = messageList[position]
41
42     return if (FirebaseAuth.getInstance().currentUser?.uid.equals(currentMessage.senderId)) {
43         ITEM_SENT
44     } else {
45         ITEM_RECEIVED
46     }
47 }

```

In base al numero restituito dalla funzione, decidiamo il layout corrispondente.

```

17 override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): RecyclerView.ViewHolder {
18     return if (viewType == 1) {
19         val view: View = LayoutInflater.from(context).inflate(R.layout.received, root: null, attachToRoot: false)
20         ReceiveViewHolder(view)
21     } else {
22         val view: View = LayoutInflater.from(context).inflate(R.layout.sent, root: null, attachToRoot: false)
23         SentViewHolder(view)
24     }
25 }

```