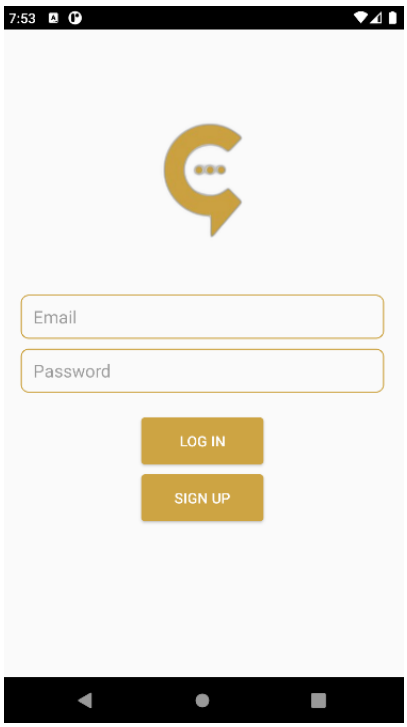
CHATTING APPLICATION

Object: creation of a basic Chatting Application which allows you to chat with other users.

Programming Language: Kotlin
Tools & Libraries: Firebase
IDE: Android Studio

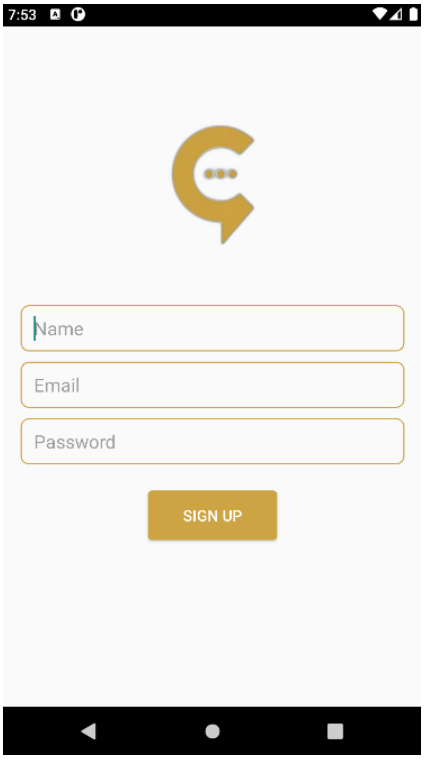
- ## res/layout/**activity_login.xml**

  In this part I defined all the aesthetic characteristics of the login page, adding custom colors (`<color name="gold">#CDA443</color>` res/values/**colors.xml**) and custom backgrounds (res/drawable/**edt_background.xml**).



- ## res/layout/**activity_sign_up.xml**

  In this part I defined all the aesthetic characteristics of the sign up page, adding custom colors and custom backgrounds.



- **java/com.example.chattingapplication/Login.kt** & **java/com.example.chattingapplication/SignUp.kt**

In this part, the login page is connected to the sign up one.
When in the login page the SIGN UP button is clicked the page changes.

```
btnSignUp.setOnClickListener {
    val intent = Intent( packageContext: this, SignUp::class.java)
    startActivity(intent)
}
```

We will use Firebase to authenticate each user. Firebase gives us a lot of methods to do it, I have chosen email and password.

To use authentication in my app:

- Initializing authentication

```
mAuth = FirebaseAuth.getInstance()
```

- Logging in a user
    - Get the email and the password from the user

```
btnLogin.setOnClickListener { it: View!
    val email = edtEmail.text.toString()
    val password = edtPassword.text.toString()

    login(email, password)
}
}
```

Create a method which enables the logging in of a user.
If it is successful we jump to the chat page (java/com.example.chattingapplication/**MainActivity.kt**), if it is not, a message appears to notify the error.

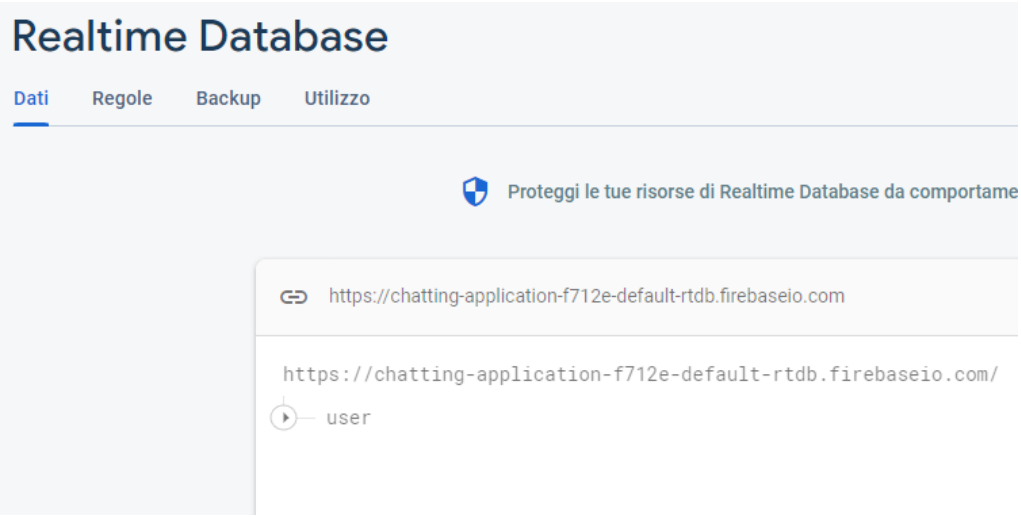
```
private fun login(email: String, password: String) {
    mAuth.signInWithEmailAndPassword(email, password)
        .addOnCompleteListener(this) { task ->
            if (task.isSuccessful) {
                val intent = Intent( packageContext: this@Login, MainActivity::class.java)
                finish()
                startActivity(intent)
            } else {
                Toast.makeText( context: this@Login, text: "User does not exists",
                    Toast.LENGTH_SHORT).show()
            }
    }
}
```

- For the sign up, the procedure is almost the same.
  Unlike login, I collect an extra value (the name).

```
btnSignUp.setOnClickListener{ it: View!
    val name = edtName.text.toString()
    val email = edtEmail.text.toString()
    val password = edtPassword.text.toString()

    signUp(name, email, password)
}
}
```

I also need to add the new user in the database creating a new node in it.

```
private fun addUserToDatabase(name: String, email: String, uid: String) {
    mDbRef = FirebaseDatabase.getInstance().getReference()

    mDbRef.child( pathString: "user").child(uid).setValue(User(name, email, uid))
}
}
```



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

  This class is used to store the values of the users.
  Each user on Firebase has 3 values: name, email and uid (unique id).

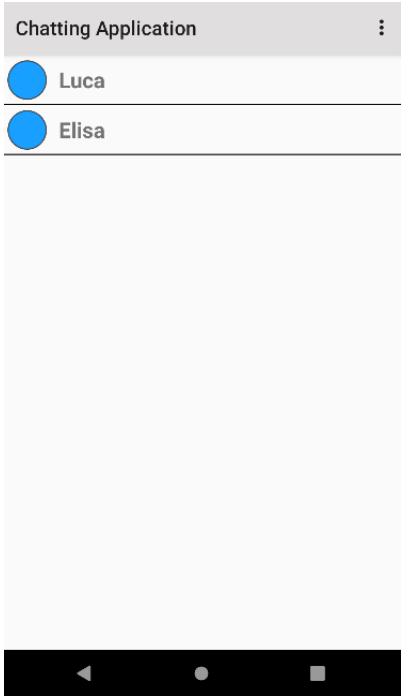
```kotlin
class User {
    var name: String? = null
    var email: String? = null
    var uid: String? = null

    constructor() {}

    constructor(name: String?, email: String?, uid: String?) {
        this.name = name
        this.email = email
        this.uid = uid
    }
}
```

- **res/layout/user_layout.xml**

  In this part is defined how a single user looks like in the chat page.

- **res/layout/activity_main.xml**

  In this part all the users are shown.



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

The UserAdapter class extends RecyclerView. RecyclerView is a flexible view for providing a limited window into a large data set.

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

`RecyclerView.Adapter` provides a binding from an app-specific data set to views that are displayed within a RecyclerView.

I created the class UserViewHolder where will be initialized all the views already created in res/layout/**user_layout.xml**.

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

`onCreateViewHolder` function is called when RecyclerView needs a new RecyclerView.ViewHolder of the given type to represent an item.
This new ViewHolder should be constructed with a new View that can represent the items of the given type.

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

`onBindViewHolder` function is called by RecyclerView to display the data at the specified position. This method should update the contents of the RecyclerView.ViewHolder.itemView to reflect the item at the given position.
In particular it shows the name of the current user.

```kotlin
    override fun onBindViewHolder(holder: UserViewHolder, position: Int) {
        val currentUser = userList[position]

        holder.textName.text = currentUser.name

        holder.itemView.setOnClickListener { it: View!
            val intent = Intent(context, ChatActivity::class.java)

            intent.putExtra( name: "name", currentUser.name)
            intent.putExtra( name: "uid", currentUser.uid)

            context.startActivity(intent)
        }
    }
```

`getItemCount` function returns the total number of items in the data set held by the adapter. So, in this case, it returns the number of users.

```kotlin
    override fun getItemCount(): Int {
        return userList.size
    }
```

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

One of the functionalities of this page is the logout. When the three dots at the top right are pressed, the possibility of logout appears.

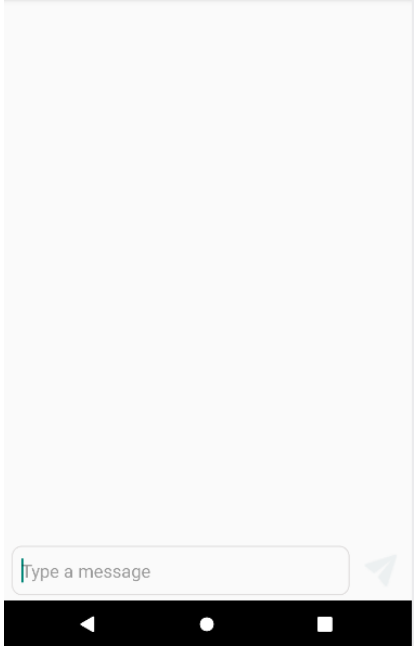
```kotlin
    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        if (item.itemId == R.id.logout) {
            mAuth.signOut()
            val intent = Intent( packageContext: this@MainActivity, Login::class.java)
            finish()
            startActivity(intent)
            return true
        }
        return true
    }
}
```

To read the value in our database and show all the people I can chat with, obviously omitting the name I log in with.

```
mDbRef.child( pathString: "user").addValueEventListener(object: ValueEventListener{
    override fun onDataChange(snapshot: DataSnapshot) {
        userList.clear()
        for (postSnapshot in snapshot.children) {
            val currentUser = postSnapshot.getValue(User::class.java)

            if (mAuth.currentUser?.uid != currentUser?.uid) {
                userList.add(currentUser!!)
            }
        }
        adapter.notifyDataSetChanged()
    }
```

- **res/layout/activity_chat.xml**

What the chat looks like.



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

Clicking on the send button, what is written in the message box is added to the database. Next, the message box is cleared.
A new node for the chat is created. It is divided into: who has sent the message and who has received the message. Both have to be updated when a new message is sent.

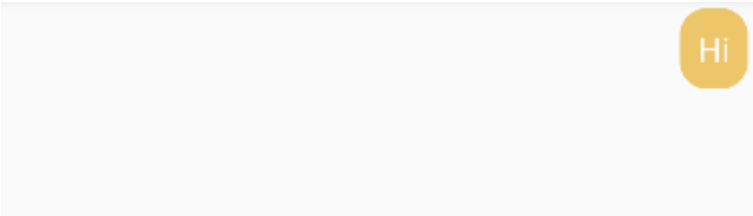
```
//adding the message to the database
sendButton.setOnClickListener {
    val message = messageBox.text.toString()
    val messageObject = Message(message, senderUid)

    mDbRef.child("chats").child(senderRoom!!).child("messages").push()
        .setValue(messageObject).addOnSuccessListener {
            mDbRef.child("chats").child(receiverRoom!!).child("messages").push()
                .setValue(messageObject)
        }
    messageBox.setText("")
}
```

Show data from the database in the RecyclerView.
When a message is sent the `onDataChange` function is called and the message is added to the message list.

```
mDbRef.child("chats").child(senderRoom!!).child("messages")
    .addValueEventListener(object: ValueEventListener {
        override fun onDataChange(snapshot: DataSnapshot) {
            messageList.clear()
            for (postSnapshot in snapshot.children) {
                val message = postSnapshot.getValue(Message::class.java)
                 messageList.add(message!!)
            }
            messageAdapter.notifyDataSetChanged()
        }

        override fun onCancelled(error: DatabaseError) {

        }
    })
```

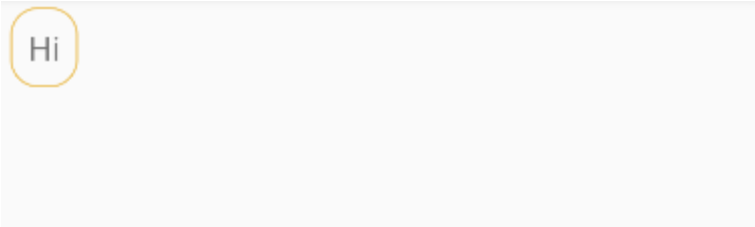
- **res/layout/sent.xml**

What a sent message looks like.



- **res/layout/received.xml**

  What a received message looks like.



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

  This is a model for the message.

```kotlin
    constructor(message: String?, senderId: String?) {
        this.message = message
        this.senderId = senderId
    }
```

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

  This time we need two ViewHolder: one to receive the message and one to send it.

```kotlin
class SentViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
    val sentMessage = itemView.findViewById<TextView>(R.id.txt_sent_message)
}

class ReceiveViewHolder(itemView: View): RecyclerView.ViewHolder(itemView) {
    val receivedMessage = itemView.findViewById<TextView>(R.id.txt_received_message)
}
```

`getItemViewType` function returns an integer depending on the view type.

```kotlin
    val ITEM_RECEIVED = 1
    val ITEM_SENT = 2
```

```
    override fun getItemViewType(position: Int): Int {
        val currentMessage = messageList[position]

        return if (FirebaseAuth.getInstance().currentUser?.uid.equals(currentMessage.senderId)) {
            ITEM_SENT
        } else {
            ITEM_RECEIVED
        }
    }
```

According to the number the function returned, we decide the corresponding layout.

```
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): RecyclerView.ViewHolder {
        return if (viewType == 1) {
            val view: View = LayoutInflater.from(context).inflate(R.layout.received, root: null, attachToRoot: false)
            ReceiveViewHolder(view)
        } else {
            val view: View = LayoutInflater.from(context).inflate(R.layout.sent, root: null, attachToRoot: false)
            SentViewHolder(view)
        }
    }
```