# RELAZIONE PROGETTO PROGRAMMAZIONE MOBILE 2023 – 2024

## VIBORA (APP PRENOTAZIONE CAMPI DA PADEL)

### INTRODUZIONE

Autore: Alessandro Privitera

Nome App: Vibora

Vibora è un applicazione Android per la gestione di un circolo di campi da Padel.

Essa permette la creazione di utenti di due tipologie: utente base e amministratore.

L' applicazione fornisce all'utente la possibilità di prenotare i campi da gioco del circolo, rendendosi disponibile per giocare o riservando definitivamente un campo insieme ad altri 3 giocatori.
L'utente è caratterizzato, oltre dalle sue informazioni personali, anche da uno skill rating derivante dal numero di vittorie e di sconfitte.
L'utente ha inoltre la possibilità di prenotare delle lezioni con un maestro.
Ancora, egli può visualizzare gli altri utenti e segnalarli nel caso di comportamenti non corretti.

L'amministratore oltre a disporre delle funzionalità dell'utente base può modificare i campi da gioco presenti nel circolo, organizzare le lezioni e revisionare le segnalazioni ricevute dagli utenti per poi prendere provvedimenti quali l'allontanamento dal circolo stesso o il reset dello skill rating.

### FUNZIONAMENTO GENERALE DELL'APPLICAZIONE

**Login/Registrazione**

L'utente dopo una Splash Screen di benvenuto ha la possibilita di effettuare il login o la registrazione all'app.

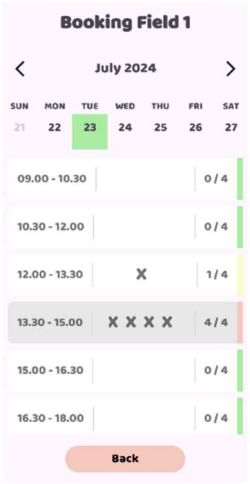


Firebase viene utilizzato per mantenere le informazioni di Login (Email e Password) e i dati dell'utente.

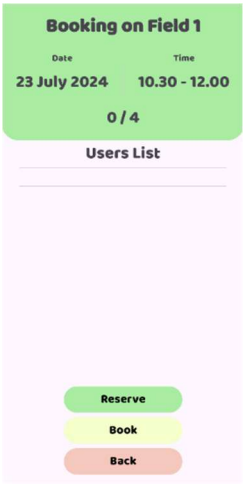
Una volta eseguito l'accesso si presenta la schermata Home, ovvero quella che consente la prenotazione di campi da gioco.

## Prenotazione di un campo

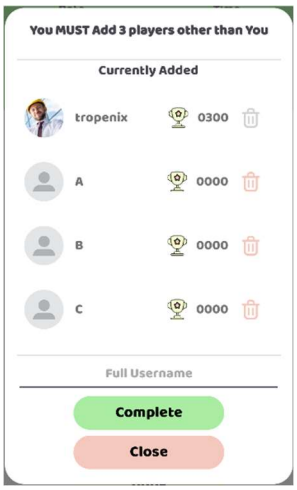

Tappando su uno dei campi è possibile accedere alla schermata che consente di selezionare uno slot (da 90 minuti) e un giorno per cui prenotare il campo.



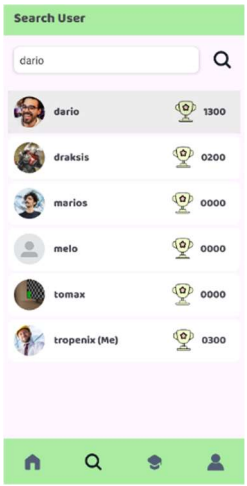

Tappando su uno dei timeslot disponibili è possibile procedere con la prenotazione.



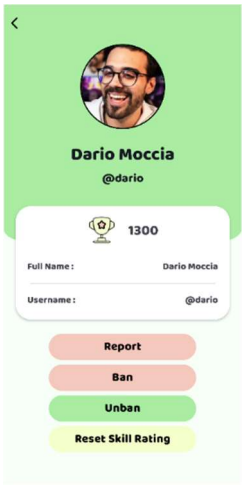

E' possibile sia prenotarsi da solo (Book) sia prenotare definitivamente il campo per 4 giocatori (Reserve).




## Ricerca di un utente

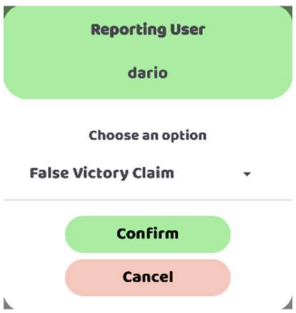



Tappando sull'icona apposita è possibile accedere alla schermata per la ricerca di altri utenti.




Cliccando su uno dei risultati è possibile visualizzarne il profilo.

I tasti *Ban, Unban, Reset Skill Rating* sono disponibili soltanto per gli account amministratore.

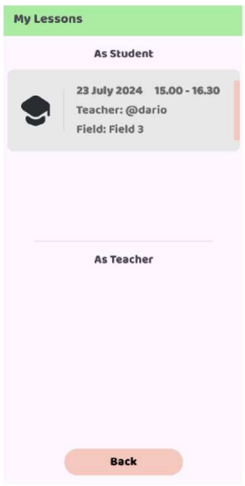



Tappando su *Report* è possibile inviare una segnalazione per il giocatore.

## Prenotare una lezione


Tappando sull'icona apposita è possibile accedere alla schermata per la prenotazione delle lezioni. Viene mostrata una lista di lezioni disponibili.
Il tasto Set Available Lessons è disponibile solo per gli amministratori.

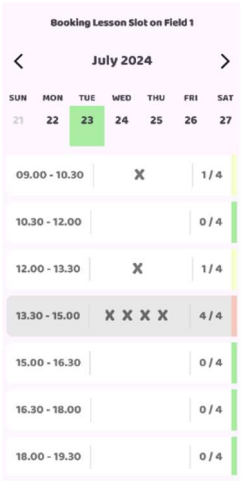

Tappando su una lezione disponibile si accede ad una schermata di conferma della prenotazione.



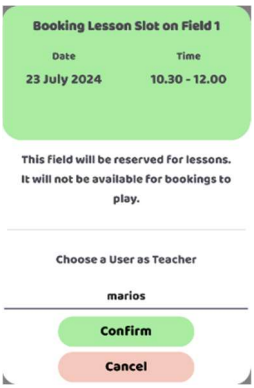

Tappando invece su My Lessons si accede alla lista di lezioni prenotate, che sia da studente o da maestro (solo se qualcuno ha prenotato la lezione)

## Organizzare una lezione


Tappando su Set Available Lessons viene mostrata la schermata per scegliere il campo su cui si terrà la lezione.



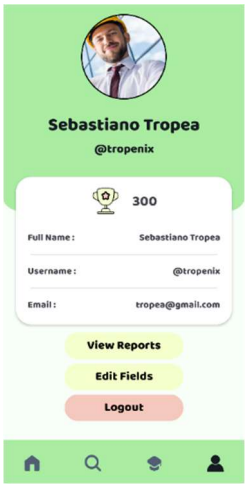

Tappando su uno dei campi viene mostrata la lista degli slot disponibili (condivisi con le prenotazioni normali)




Tappando su uno degli slot viene mostrata una schermata di conferma di organizzazione della lezione dove bisogna indicare il maestro.

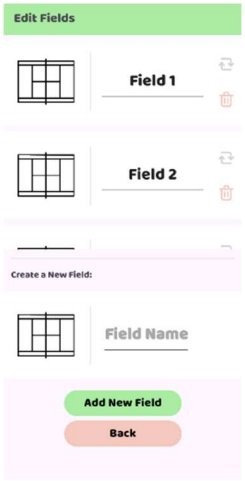
La lezione sarà dunque mostrata nella schermata principale.

## Schermata Profilo




Tappando sull'apposita icona è possibile accedere alla schermata del proprio profilo. I bottoni View Reports e Edit Fields sono esclusivi degli amministratori.

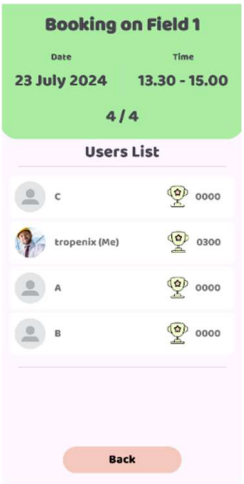

Tappando su View Reports è possibile visualizzare una lista delle segnalazioni ricevute da tutti gli utenti.
Tappando su una segnalazione si accede al profilo dell'utente.



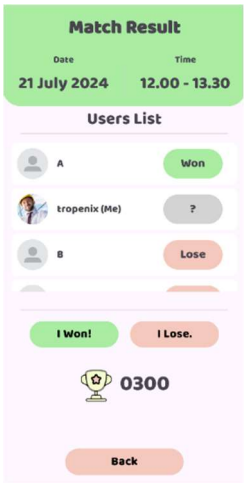

Tappando invece su Edit Fields si accede ad una schermata che permette di modificare i campi presenti nel circolo.

## Definire l'esito di un match


Tappando su My Bookings (nella Home) è possibile visualizzare tutte le prenotazioni effettutate dall'utente.




Tappando su una prenotazione per un match non ancora terminato viene mostrata una schermata di riepilogo della prenotazione.




Tappando invece su una prenotazione per un match terminato viene visualizzata una schermata che permette di autodichiarare se si è vinto o si è perso in quel match.
Inoltre, vengono visualizzate le dichiarazioni degli altri giocatori, in modo da poterli segnalare qualora mentissero.

Se si dichiara una vittoria si ricevono 200 punti. Se si dichiara una sconfitta si perdono 200 punti.
E' cura dell'amministratore gestire le segnalazioni che riguardano le dichiarazioni di vittoria e prendere provvedimenti di conseguenza.

**PORZIONI DI CODICE DA EVIDENZIARE**

### HomeFragment

```
public class HomeFragment extends Fragment {
    Button mybookings_btn;
    RecyclerView recyclerView;
    FieldRecyclerAdapter adapter;
    FirestoreRecyclerOptions<FieldModel> options;
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        this.getActivity().getWindow().setStatusBarColor(this.getResources().getColor(R.color.main_green));

        View view = inflater.inflate(R.layout.fragment_home, container, false);

        initViews(view);

        recyclerView.setLayoutManager(new LinearLayoutManager(requireContext()));
        recyclerView.setAdapter(adapter);
        setupFieldRecyclerView();

        mybookings_btn.setOnClickListener(v -> {
            startActivity(new Intent(getContext(), MyBookings.class));
        });

        return view;
    }
```

`HomeFragment` gestisce la visualizzazione di un elenco di campi da gioco recuperati da Firestore in un RecyclerView. Inizializza le viste, imposta il layout manager e l'adapter per il RecyclerView, e gestisce le azioni dell'utente come cliccare sul pulsante delle prenotazioni. Ogni volta che il frammento riprende, assicura che il RecyclerView sia aggiornato con i dati più recenti.

```
private void setupFieldRecyclerView(){
        Query query = FirebaseUtils.allFieldsCollectionReference();

        // Costruisci nuove opzioni per l'adapter
        FirestoreRecyclerOptions<FieldModel> newOptions = new FirestoreRecyclerOptions.Builder<FieldModel>()
                .setQuery(query, FieldModel.class).build();

        // Se l'adapter non è stato inizializzato o la query è cambiata, aggiorna l'adapter
        if (adapter == null || !newOptions.equals(options)) {
            options = newOptions; // Aggiorna le opzioni memorizzate
            adapter = new FieldRecyclerAdapter(options, requireContext());
            recyclerView.setLayoutManager(new LinearLayoutManager(requireContext()));
            recyclerView.setAdapter(adapter);
            adapter.startListening();
        }
    }
```

L'adapter utilizzato in questo caso è `FieldRecyclerAdapter`, ovvero un adapter che estende la classe `FirestoreRecyclerAdapter`.

### FirestoreRecyclerAdapter

```
public class FieldRecyclerAdapter extends FirestoreRecyclerAdapter<FieldModel, FieldRecyclerAdapter.FieldModelViewHolder> {
    Context context;

    public FieldRecyclerAdapter(@NonNull FirestoreRecyclerOptions<FieldModel> options, Context context) {
        super(options);
        this.context = context;
    }
```

Qui viene definita la classe `FieldRecyclerAdapter` che estende `FirestoreRecyclerAdapter`. Il costruttore accetta le opzioni per Firestore e il contesto dell'applicazione.

```
@Override
protected void onBindViewHolder(@NonNull FieldModelViewHolder viewHolder, int i, @NonNull FieldModel fieldModel) {
    viewHolder.fieldName.setText(fieldModel.getName());
    viewHolder.status_indicator.setBackgroundColor(ContextCompat.getColor(context, R.color.main_green));

    FirebaseUtils.dailyBookingsCollectionReference(fieldModel.getFieldId(), CalendarUtils.formattedDate(LocalDate.now())).get()
            .addOnCompleteListener(task -> {
                if (task.isSuccessful()) {
                    int occupied_slot_counter = 0;
                    List<TimeSlotModel> available_slots = new ArrayList<>();
                    for(int s = 0; s <= 7; s++) available_slots.add(new TimeSlotModel(s, 0));
                    QuerySnapshot snapshots = task.getResult();
                    for (DocumentSnapshot doc : snapshots) {
                        BookingModel bookingModel = doc.toObject(BookingModel.class);
                        int s;
                        for(s = 0; s <= available_slots.size(); s++){
                            if(available_slots.get(s).getIndex() == bookingModel.getTimeSlot()){
                                available_slots.get(s).setReserved_spots(bookingModel.getUserIdList().size());
                                break;
                            }
                        }
                        if(bookingModel.isReserved() || isPast(bookingModel)){
                            occupied_slot_counter += 1;
                            available_slots.remove(s);
                            Log.d("AVSLOTS", "Removing " + bookingModel.getTimeSlot());
                        }
                    }

                    viewHolder.occupied_slots.setText(occupied_slot_counter + " / 8");
                    if(occupied_slot_counter >= 7) viewHolder.status_indicator.setBackgroundColor(context.getResources().getColor(R.color.red));
                    else if(occupied_slot_counter >= 3) viewHolder.status_indicator.setBackgroundColor(context.getResources().getColor(R.color.yellow));
                    int minTimeSlot = 7;

                    removePastTimeSlots(available_slots);

                    if(available_slots.size() == 0) return;

                    int minIndex = 0;
                    for(int x = 0; x < available_slots.size(); x++){
                        TimeSlotModel ts = available_slots.get(x);
                        if(ts.getIndex() < minTimeSlot){
                            minTimeSlot = ts.getIndex();
                            minIndex = x;
                        }
                    }

                    viewHolder.first_available_slot.setText("1° Available Slot:\n" + CalendarUtils.mapIndexToTimeSlot(minTimeSlot));
                    FirebaseUtils.getFieldImageStorageRef("field" + available_slots.get(minIndex).getReserved_spots() + ".png").getDownloadUrl()
                            .addOnCompleteListener(task2 -> {
                                if(task2.isSuccessful()){
                                    Uri uri = task2.getResult();
                                    Activity activity = (Activity) context;
                                    if(!activity.isFinishing() && !activity.isDestroyed()) Glide.with(context).load(uri).into(viewHolder.imageView);
                                }
                            });
```

Questo metodo associa i dati del modello `FieldModel` alle viste del ViewHolder. Imposta il nome del campo, il colore dell'indicatore di stato e recupera le prenotazioni giornaliere dal database Firestore. Conta i posti occupati e aggiorna l'indicatore di stato e il testo relativo ai posti disponibili. Carica l'immagine del campo e gestisce il clic sull'elemento della lista per aprire un'attività dettagliata.

```
public FieldModelViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
    View view = LayoutInflater.from(context).inflate(R.layout.field_list_item, parent, false);
    return new FieldModelViewHolder(view);
}
```

Questo metodo crea e restituisce un `FieldModelViewHolder`, inflando il layout per l'elemento della lista.

```
public class FieldModelViewHolder extends RecyclerView.ViewHolder{
    ImageView imageView;
    TextView fieldName, occupied_slots, first_available_slot;
    View status_indicator;
    public FieldModelViewHolder(@NonNull View itemView) {
        super(itemView);
        imageView = itemView.findViewById(R.id.field_image);
        fieldName = itemView.findViewById(R.id.field_name);
        occupied_slots = itemView.findViewById(R.id.occupied_slots_number);
        first_available_slot = itemView.findViewById(R.id.next_available_slot);
        status_indicator = itemView.findViewById(R.id.field_status_indicator);
    }
}
```

Questa classe contiene i riferimenti alle view all'interno dell'elemento del RecyclerView, inclusa l'immagine del campo, il nome del campo, il numero di posti occupati, il primo slot disponibile e l'indicatore di stato.

**Booking**

```
public class Booking extends AppCompatActivity{
    static String FIELD_NAME, DATE, TIMESLOT;
    int RESERVED_SPOTS;
    TextView field_text, date_text, timeslot_text, reservedspots_text;
    RecyclerView userlist_recycler;
    SearchUserRecyclerAdapter adapter;
    FirestoreRecyclerOptions<UserModel> options;
    Button book_btn, back_btn, reserve_btn;
    boolean recyclerInitialized = false;

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        EdgeToEdge.enable(this);
        getWindow().setStatusBarColor(this.getResources().getColor(R.color.main_green));
        setContentView(R.layout.activity_booking);

        FIELD_NAME = getIntent().getStringExtra("FIELD_NAME");
        DATE = getIntent().getStringExtra("DATE");
        TIMESLOT = getIntent().getStringExtra("TIMESLOT");
        RESERVED_SPOTS = getIntent().getIntExtra("RESERVED_SPOTS", 0);

        initViews();
        setupUserRecyclerView();

        back_btn.setOnClickListener(v -> {
            finish();
        });

        book_btn.setOnClickListener(v -> {
            book();
        });

        reserve_btn.setOnClickListener(v -> {
            Intent intent = new Intent(getApplicationContext(), ReserveDialog.class);
            intent.putExtra("FIELD_NAME", FIELD_NAME);
            intent.putExtra("TIMESLOT", TIMESLOT);
            startActivity(intent);
        });
    }
```

Il metodo `onCreate` imposta l'attività quando viene creata. Ottiene i dati passati dall'intent, inizializza le viste, imposta il RecyclerView per mostrare gli utenti e definisce i listener per i pulsanti. La RecyclerView viene impostata con un adapter simile a quello visto prima.

```
private static Task<Boolean> addToExistingBooking(int _timeSlot) {
    TaskCompletionSource<Boolean> taskCompletionSource = new TaskCompletionSource<>();

    FirebaseUtils.dailyBookingsCollectionReference(FIELD_NAME, CalendarUtils.formattedDate(CalendarUtils.selectedDate)).get()
            .addOnCompleteListener(task -> {
                if (task.isSuccessful()) {
                    QuerySnapshot snapshots = task.getResult();
                    boolean found = false;
                    for (DocumentSnapshot doc : snapshots) {
                        BookingModel bookingModel = doc.toObject(BookingModel.class);
                        int timeSlot = bookingModel.getTimeSlot();
                        if (timeSlot == _timeSlot) {
                            if(bookingModel.getUserIdList().size() + 1 >= 4)
                                bookingModel.setReserved(true);
                            ArrayList<String> users = bookingModel.getUserIdList();
                            users.add(FirebaseUtils.currentUserId());
                            ArrayList<PlayerResult> matchResults = bookingModel.getMatchResults();
                            matchResults.add(new PlayerResult(FirebaseUtils.currentUserId(), "?"));
                            doc.getReference().update(
                                        new HashMap<String, Object>() {{
                                            put("userIdList", users);
                                            put("matchResults", matchResults);
                                            put("reserved", bookingModel.isReserved());
                                        }}
                                    )
                                    .addOnSuccessListener(aVoid -> taskCompletionSource.setResult(true))
                                    .addOnFailureListener(taskCompletionSource::setException);
                            found = true;
                            break;  // Exit loop once the time slot is found and updated
                        }
                    }
                    if (!found) {
                        taskCompletionSource.setResult(false);
                    }
                } else {
                    taskCompletionSource.setException(task.getException());
                }
            });
    return taskCompletionSource.getTask();
}
```

Questo metodo aggiunge l'utente corrente a una prenotazione esistente per un determinato time slot. Se la prenotazione esiste già, aggiorna il documento con il nuovo utente; altrimenti, restituisce `false`. A rendere necessario il tipo di ritorno `Task<Boolean>` è l'asincronicità delle operazioni su Firestore, così facendo il metodo `book` (che segue) attende che `addToExistingBooking` abbia completato, ovvero attende per sapere se l'utente si sta aggiungendo ad una prenotazione già presente nel database o se è necessario crearne una nuova.

```
void book(){
    addToExistingBooking(CalendarUtils.mapTimeSlot(TIMESLOT)).addOnSuccessListener(isFound -> {
        if (isFound) {
            // Operazione di aggiornamento riuscita
        } else {
            // Nessun documento trovato con il time slot specificato
            BookingModel bookingModel= new BookingModel(FIELD_NAME, CalendarUtils.selectedDate, CalendarUtils.mapTimeSlot(TIMESLOT), FirebaseUtils.currentUserId());
            String userList = "";
            for(String user : bookingModel.getUserIdList())
                userList += user + " ";

            FirebaseUtils.dailyBookingsCollectionReference(FIELD_NAME, CalendarUtils.formattedDate(CalendarUtils.selectedDate)).add(bookingModel)
                    .addOnSuccessListener(documentReference -> {
                        Log.d("Firestore", "DocumentSnapshot added with ID: " + documentReference.getId());
                    })
                    .addOnFailureListener(e -> {
                        Log.w("Firestore", "Error adding document", e);
                    });
        }
        AndroidUtils.showToast(this, "Field Booked");
        finish();
    });
}
```

Questo metodo viene chiamato quando l'utente preme il pulsante per prenotare. Prova ad aggiungere l'utente a una prenotazione esistente; se non ne trova una, crea una nuova prenotazione. Alla fine mostra un messaggio di conferma e chiude l'attività.

**ProfileFragment**

```
public class ProfileFragment extends Fragment {
    TextView full_name, full_name2, username, username2, email, ranking_score;
    Button logoutBtn;
    CircleImageView profile_image;
    ProgressBar progressBar;

    UserModel currentUser;
    ActivityResultLauncher<Intent> imagePickLauncher;

    @Override
    public void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        final Uri[] selectedImageUri = new Uri[1];
        imagePickLauncher = registerForActivityResult(new ActivityResultContracts.StartActivityForResult(), result ->{
                    if(result.getResultCode() == Activity.RESULT_OK) {
                        Intent data = result.getData();
                        if (data != null && data.getData() != null){
                            selectedImageUri[0] = data.getData();
                            currentUser.setProfilePic(selectedImageUri[0].toString());
                            AndroidUtils.setProfilePic(getContext(), selectedImageUri[0], profile_image);

                            if(selectedImageUri[0] != null) {
                                progressBar.setVisibility(View.VISIBLE);
                                logoutBtn.setEnabled(false);
                                disableBottomNavigationButtons();
                                FirebaseUtils.getCurrentProfilePicStorageRef().putFile(selectedImageUri[0])
                                        .addOnCompleteListener(task -> {
                                            if(task.isSuccessful()){
                                                AndroidUtils.showToast(getContext(), "Profile Pic Updated Successfully");
                                            }
                                            else{
                                                AndroidUtils.showToast(getContext(), "Profile Pic update failed");
                                            }
                                            progressBar.setVisibility(View.GONE);
                                            logoutBtn.setEnabled(true);
                                            enableBottomNavigationButtons();
                                        });
                            }
                        }
                    }
                });
    }
```

Il metodo `onCreate` viene chiamato quando il frammento viene creato. Registra un launcher per la selezione delle immagini utilizzando la libreria `ImagePicker`. Quando l'utente seleziona un'immagine, questa viene caricata in Firebase Storage e aggiorna l'immagine del profilo.

```
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.fragment_profile, container, false);
    initViews(view);

    logoutBtn.setOnClickListener(v -> {
        FirebaseUtils.signOut();
        Intent intent = new Intent(getActivity(), Login.class);
        intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK);
        startActivity(intent);
        if (getActivity() != null) {
            getActivity().finish();
        }
    });

    profile_image.setOnClickListener(v -> {
        ImagePicker.with(ProfileFragment.this).cropSquare().compress(512).maxResultSize(512,512)
                .createIntent(new Function1<Intent, Unit>() {
                    @Override
                    public Unit invoke(Intent intent) {
                        imagePickLauncher.launch(intent);
                        return null;
                    }
                });
    });

    return view;
}
```

Questo metodo viene chiamato per creare la vista gerarchica del frammento. Inizializza le viste e definisce i listener per il pulsante di logout e l'immagine del profilo. Quando l'utente clicca sull'immagine del profilo, si avvia l'ImagePicker per permettere la selezione di una nuova immagine.

```
void getUserData(){
    currentUser = FirebaseUtils.currentUserModel;

    if(currentUser.getProfilePic() != null) AndroidUtils.setProfilePic(getContext(), Uri.parse(currentUser.getProfilePic()), profile_image);
    else {
        FirebaseUtils.getCurrentProfilePicStorageRef().getDownloadUrl()
                .addOnCompleteListener(task -> {
                    if(task.isSuccessful()){
                        Uri uri = task.getResult();
                        if(isAdded() && getActivity() != null){
                            AndroidUtils.setProfilePic(getContext(), uri, profile_image);
                            currentUser.setProfilePic(uri.toString());
                        }
                    }
                });
    }

    full_name.setText(currentUser.getFull_name());
    full_name2.setText(currentUser.getFull_name());
    String _username = "@" + currentUser.getUsername();
    username.setText(_username);
    username2.setText(_username);
    email.setText(currentUser.getEmail());
    ranking_score.setText(String.format("%d", currentUser.getSkill_rating()));
}
```

Questo metodo recupera i dati dell'utente dal modello corrente e aggiorna le viste del profilo. Se l'utente ha un'immagine del profilo, la imposta; altrimenti, tenta di scaricarla da Firebase Storage.

**ELENCO DEI FILE JAVA**

ACTIVITY / FRAGMENT

| Nome (A – Z) | Descrizione |
|---|---|
| `Booking` | Activity per la prenotazione di un campo per uno specifico giorno e orario |
| `BookingWeek` | Activity per mostrare e scegliere gli slot disponibili giorno per giorno |
| `BookLessonDialog` | Activity per prenotare una lezione con un maestro |
| `EditFields` | Activity per modificare i campi disponibili nel circolo |
| `HomeFragment` | Fragment che visualizza tutti i campi disponibili e consente la prenotazione al click di uno di questi |
| `LearnBookingDialog` | Activity disponibile solo per l'amministratore per prenotare un campo per una lezione ad uno specifico giorno ed orario |
| `LearnBookingWeek` | Activity per mostrare e scegliere gli slot disponibili giorno per giorno per le lezioni |
| `LearnFragment` | Activity per mostrare le lezioni con maestro disponibili |
| `Login` | Activity per effettuare il login |
| `MainActivity` | Activity principale che consente di navigare fra i fragment attraverso la bottombar |
| `MatchResults` | Activity per stabilire l'esito di un match disputato |
| `MyBookings` | Activity che mostra le prenotazioni effettuate da un utente |
| `MyLessons` | Activity che mostra le lezioni prenotate da un utente |
| `OtherProfile` | Activity che mostra il profilo di un altro utente |
| `ProfileFragment` | Activity per mostrare il proprio profilo |
| `ProfileFragmentAdmin` | Activity per mostrare il proprio profilo se si è admin |
| `Register` | Activity per effettuare la registrazione |
| `ReportDialog` | Activity per segnalare un giocatore |
| `ReserveDialog` | Activity per prenotare un campo definitivamente per 4 persone |
| `SearchFragment` | Fragment per cercare altri utenti |
| `SetLessons` | Activity per mostrare i campi disponibili su cui rendere disponibile una lezione con maestro |
| `SplashScreen` | Activity di introduzione con logo |
| `ViewAllReports` | Activity per l'amministratore che consente di mostrare tutte le segnalazioni |

UTILS

| Nome (A – Z) | Descrizione |
|---|---|
| `AndroidUtils` | Funzioni utili generiche |
| `CalendarUtils` | Funzioni che riguardano calendario e date |
| `FirebaseUtils` | Funzioni utili per gestire l'accesso a Firebase |

MODEL

| Nome (A – Z) | Descrizione |
|---|---|
| `BookingModel` | Modello di oggetto per le prenotazioni |
| `FieldModel` | Modello di oggetto per i campi |
| `LessonModel` | Modello di oggetto per le lezioni |
| `PlayerResult` | Modello di oggetto per il risultato di un giocatore |
| `ReportModel` | Modello di oggetto per una segnalazione |
| `TimeSlotModel` | Modello di oggetto per un timeslot |
| `UserMatchResultModel` | Modello di oggetto per il risultato di un match di un giocatore |
| `UserModel` | Modello di oggetto per un utente |

ADAPTER

| Nome (A – Z) | Descrizione |
|---|---|
| `AdminFieldAdapter` | Adapter per visualizzare la lista di campi nell'Activity `EditFields` |
| `BookingAdapter` | Adapter per visualizzare la lista di slot nelle Activity `BookingWeek` e `LearnBookingWeek` |
| `CalendarAdapter` | Adapter per mostrare i giorni della settimana nelle Activity `BookingWeek` e `LearnBookingWeek` |
| `FieldRecyclerAdapter` | Adapter per mostrare la lista di campi nel Fragment `HomeFragment` e nella Activity `SetLessons` |
| `LessonsAdapter` | Adapter per mostrare la lista di lezioni nel Fragment `LearnFragment` e nella Activity `MyLessons` |
| `MyBookingsAdapter` | Adapter per mostrare la lista di prenotazioni dei campi nella Activity `MyBookings` |
| `ReportsAdapter` | Adapter per mostrare la lista di segnalazioni nella Activty `ViewAllReports` |
| `SearchUserRecyclerAdapter` | Adapter per mostrare la lista di utenti nel Fragment `SearchFragment` e nella Activity `Booking` |
| `UserMatchResultsAdapter` | Adapter per mostrare la lista di giocatori nella Activity `MatchResult` |
| `UserReserveRowAdapter` | Adapter per mostrare la lista di giocatori nella Activity `ReserveDialog` |