

EVENT FINDER - Applicazione di Ricerca Eventi



Introduzione	2
Scopo e Motivazioni	2
ARCHITETTURA DEL SISTEMA	3
Materiali Utilizzati	3
Wireframe APP/SITO WEB	6
STORYBOARD	7
SCHERMATA HOME	7
SCHERMATA INFO EVENTI	7
SCHERMATA LOGIN	8
SCHERMATA REGISTRAZIONE	8
SCHERMATA HOME DOPO IL LOGIN	9
SCHERMATA PROFILO	9
SCHERMATA AMICI	10
SCHERMATA PREFERITI	10
SCHERMATA BIGLIETTI	11
Linguaggio e tecnologie utilizzati	12
JAVA	12
Gestione delle Activity in Java	12
PYTHON	13
Flask e Retrofit	13
HTML CSS	15
RISULTATI OTTENUTI	15
CONCLUSIONE	15

Introduzione

Durante il corso IFTS Developer, abbiamo avuto l'opportunità di imparare e sperimentare vari linguaggi di programmazione, ognuno con funzioni e obiettivi specifici. Abbiamo iniziato con il modulo di **Database**, dove abbiamo studiato come utilizzare MySQL per gestire dati e creare tabelle, migliorando le nostre capacità di analisi e gestione dei dati. Successivamente, il modulo di **Java** ci ha permesso di apprendere le basi della programmazione ad oggetti (OOP) e le tecniche di ordinamento degli algoritmi. Nel modulo di **Android**, abbiamo esplorato la creazione di app, imparando a costruire layout e utilizzare Java per gestire le classi e le interazioni. Il modulo di **Backend**, utilizzando strumenti come **Flask**, **Retrofit** e **API REST**, ci ha insegnato a gestire la comunicazione tra server e client. . Il corso si è concluso con la creazione di un'app, collegata a un database e supportata da servizi web e un sito, per mettere in pratica le conoscenze acquisite lavorando in gruppo.

Il gruppo che ha creato l'applicazione "**EVENT FINDER**" è composto da Carolina Leon Banushan Sivakumar, Luigi Concha, Davide Sanzanni, Matteo Zappa,

Scopo e Motivazioni

Il progetto del modulo di **Project Work** ci ha dato come compito l'ideazione di un'app, scegliendo il suo tema e utilizzandola come base per lavorare in gruppo. Questo ci permette di acquisire esperienza nella progettazione e nello sviluppo di applicazioni Android, preparandoci così ad affrontare un eventuale stage.

Nel nostro caso, abbiamo avuto modo di programmare un'applicazione che riguarda la ricerca degli eventi nelle vicinanze o anche in altre città. **Event Finder** è un'app pensata per semplificare questa ricerca e rendere la partecipazione agli eventi più facile e personalizzata. Abbiamo pensato a questo progetto perché ci siamo resi conto che molte persone faticano a tenere traccia degli eventi che potrebbero interessargli, soprattutto quando le informazioni sono sparse su diversi siti o app.

L'obiettivo dell'app è quello di aiutare gli utenti a trovare facilmente eventi locali. Che si tratti di concerti, mostre, eventi sportivi o conferenze, gli utenti possono scoprire cosa accade vicino a loro o in altre città.

"Event Finder" è pensata per tutti, da chi cerca eventi di svago come concerti o sport, a chi vuole pianificare il proprio tempo libero in base a eventi specifici. Inoltre, l'app permette agli organizzatori di eventi di caricare informazioni sui loro eventi e gestire le prenotazioni direttamente. In questo modo, gli utenti e gli organizzatori possono interagire facilmente.

Il progetto nasce con l'idea di semplificare l'esperienza di partecipazione agli eventi, creando una piattaforma che raccoglie tutte le informazioni in un'unica app facile da usare. Così

facendo, gli utenti avranno un accesso più rapido agli eventi che vogliono seguire, mentre gli organizzatori potranno promuovere le loro attività e facilitare la vendita dei biglietti.

ARCHITETTURA DEL SISTEMA

Per lo sviluppo dell'app **Event Finder**, abbiamo utilizzato il modello architetturale **MVC** (Model-View-Controller) per garantire una separazione chiara dei compiti e facilitare la gestione dell'applicazione.

Nel **Model**, abbiamo gestito la logica dei dati, come la gestione delle informazioni sugli eventi e sugli utenti, mantenendo la logica dell'applicazione separata dalla parte visiva. Il **View** è stato sviluppato utilizzando gli strumenti di Android Studio per creare un'interfaccia utente semplice e intuitiva, tenendo conto delle esigenze degli utenti finali. Per il **Controller**, abbiamo gestito le interazioni dell'utente, che aggiornavano il Model e il View in modo dinamico, creando un'esperienza utente fluida e reattiva.

Abbiamo utilizzato **SharedPreferences** per memorizzare piccole quantità di dati in modo persistente, come le preferenze dell'utente o lo stato dell'app. Questo strumento è stato utile per mantenere una continuità dell'esperienza utente, anche dopo che l'app è stata chiusa e riaperta.

Nella parte **utente** dell'app, gli utenti possono cercare eventi, aggiungere eventi alla loro lista **preferiti** .. Possono anche prenotare biglietti per gli eventi, consultare i dettagli come orari, prezzi e location.

Dal lato **organizzatori di eventi**, è stato previsto un sistema web dove gli organizzatori (admin) possono creare e gestire il loro profilo, aggiungere dettagli sugli eventi (orari, indirizzi, descrizioni).

Per la gestione del lavoro in team, abbiamo utilizzato **GitHub** e il sistema di versionamento **GitHub Desktop**. Questo ci ha permesso di lavorare su una cartella condivisa, scambiandoci file in modo efficiente. Abbiamo imparato a gestire il nostro repository usando comandi come **fetch**, **pull**, **push** e **commit**, garantendo un controllo completo sulle versioni dell'applicazione e sui vari componenti del progetto.

Materiali Utilizzati

-android studio

-python

-java

-visualcode

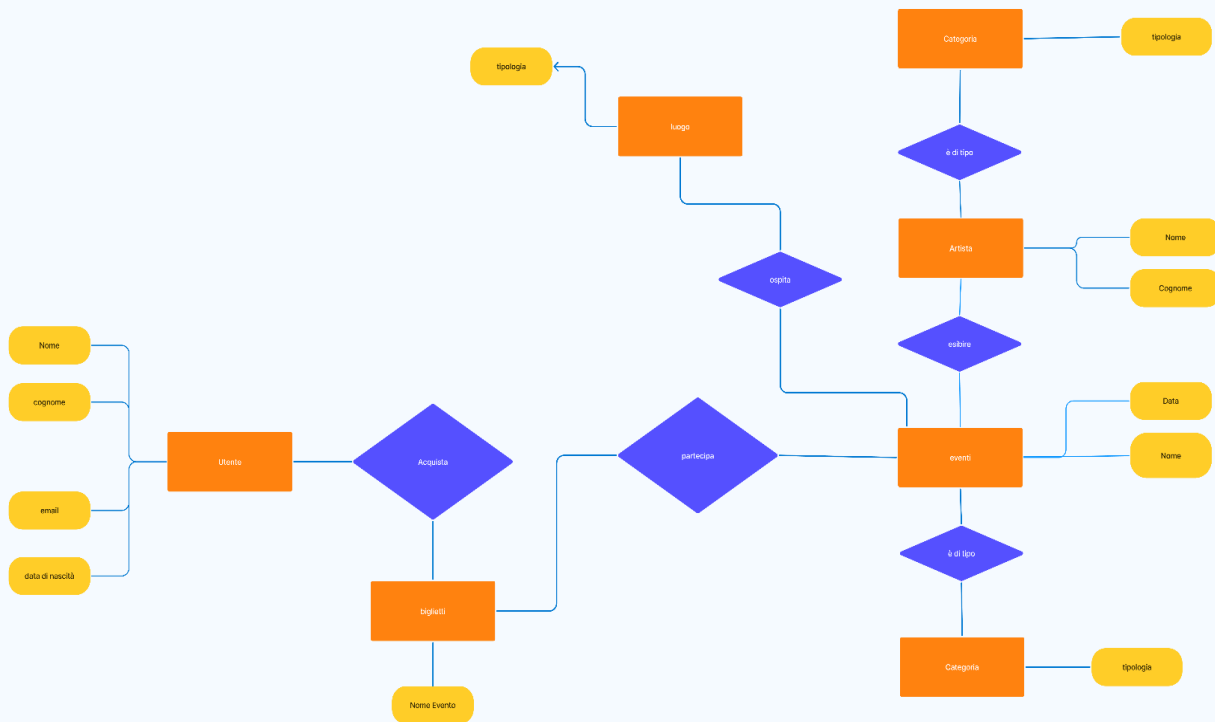
-github

-github desktop

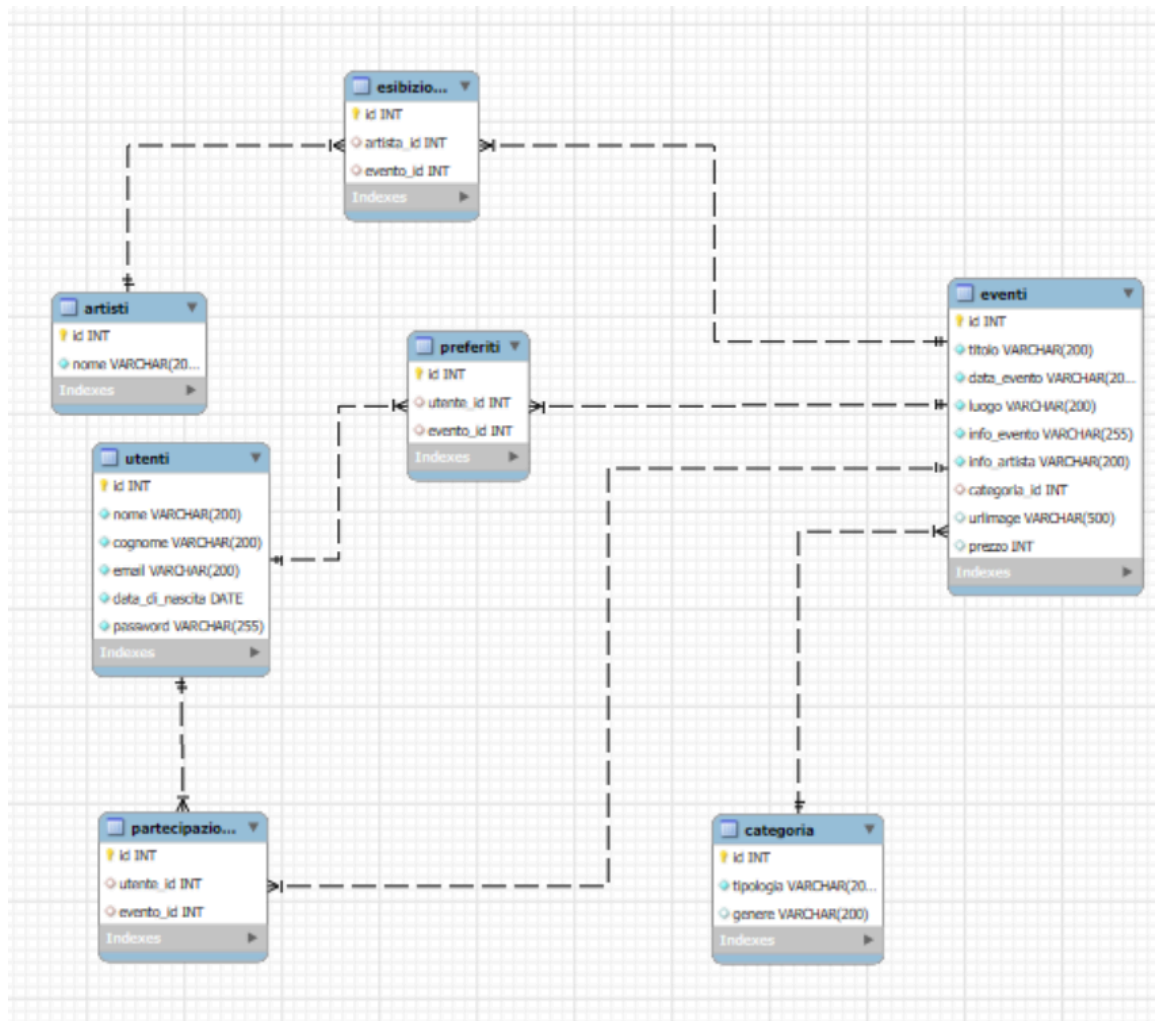
-figma

-retrofit

Diagramma relazioni

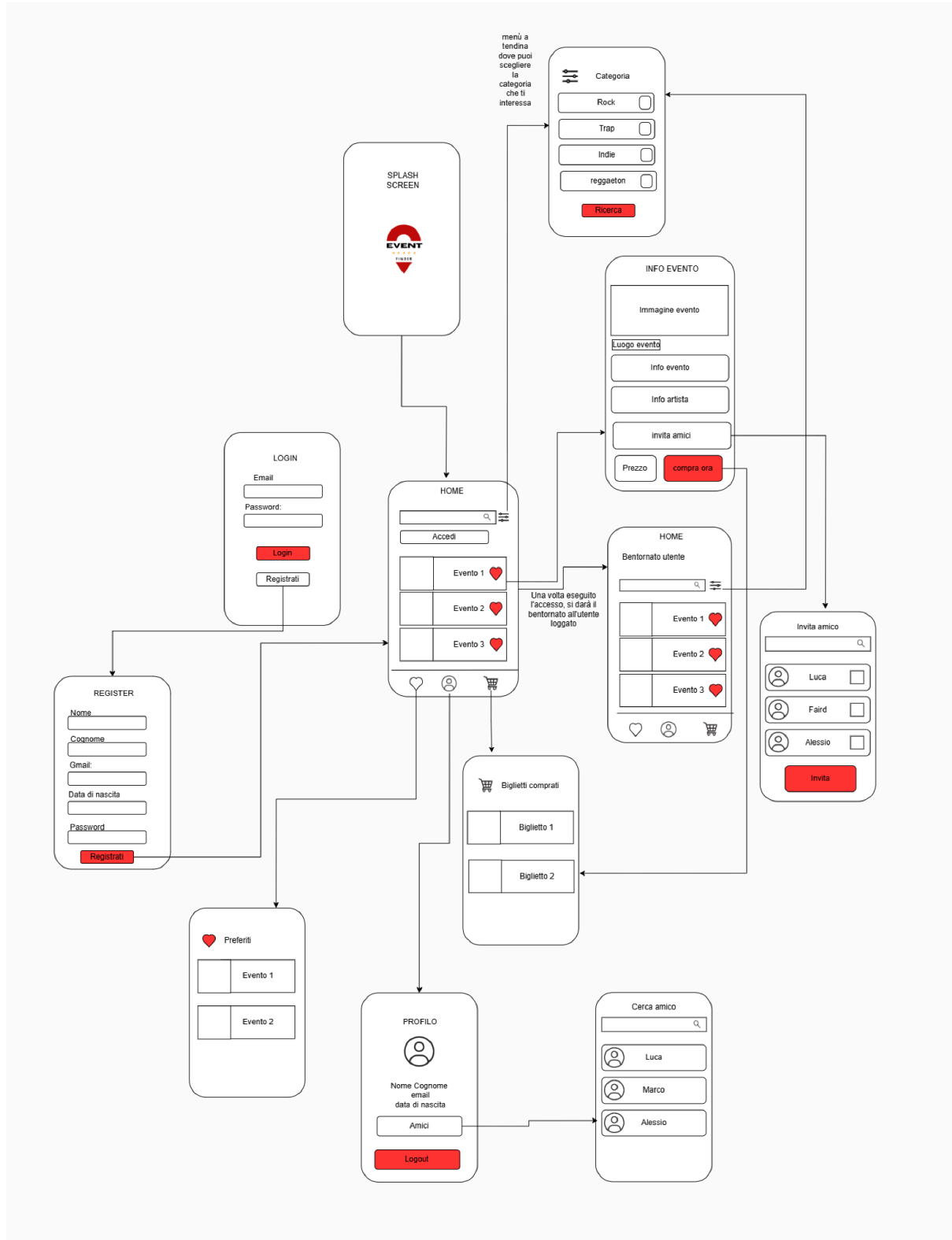


STRUTTURA DATABASE



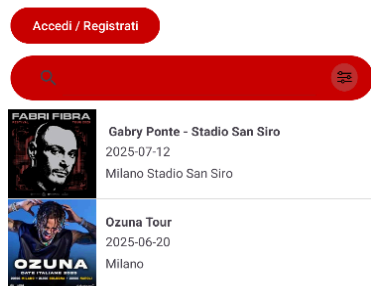
Wireframe APP/SITO WEB

Con Figma abbiamo dato forma al wireframe concettuale ed estetico del progetto .



STORYBOARD

Le schermate principali dell'app in breve saranno le seguenti:



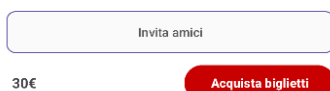
SCHERMATA HOME

Nella schermata home puoi visualizzare gli eventi disponibili e le informazioni relative a ciascun evento. Tuttavia, alcune funzionalità avanzate, come l'aggiunta degli eventi ai preferiti, la visualizzazione del profilo, l'acquisto dei biglietti, l'invito di amici e l'aggiunta di amici, richiedono il login. Senza effettuare il login, queste opzioni non sono accessibili.



SCHERMATA INFO EVENTI

Appena l'utente seleziona un evento, verrà indirizzato alla pagina con le informazioni dettagliate dell'evento scelto, dove potrà vedere il titolo, l'immagine, la data, il luogo, le informazioni sul cantante e il prezzo. Tuttavia, senza aver effettuato il login, non sarà possibile aggiungere l'evento ai preferiti, invitare amici o acquistare il biglietto.





LOGIN

login

non sei registrato? registrati

SCHERMATA LOGIN

Nella schermata di login, l'utente potrà accedere all'app utilizzando le proprie credenziali, inserendo l'email e la password. Se l'utente non è ancora registrato, avrà la possibilità di creare un account tramite il pulsante **"Registrati"** posto sotto il form di login. Questo permetterà all'utente di registrarsi facilmente e accedere successivamente con le proprie credenziali.



REGISTRAZIONE

Nome

Cognome

Email

Data di nascita

Password

Registrati

SCHERMATA REGISTRAZIONE

Nella schermata di registrazione, l'utente potrà inserire i suoi dati personali, come **nome**, **cognome**, **email**, **data di nascita** e **password**. Una volta inseriti, questi dati verranno salvati nel database, creando un nuovo account per l'utente. Successivamente, l'utente potrà utilizzare le sue credenziali (email e password) per effettuare il **login** e accedere all'home.



Benvenuto Banushan
QUA TROVERAI TUTTI GLI EVENTI DEL MOMENTO



Gabry Ponte - Stadio San Siro
2025-07-12
Milano Stadio San Siro



Ozuna Tour
2025-06-20
Milano

SCHERMATA HOME DOPO IL LOGIN

Una volta che l'utente si sarà **loggato** con successo, verrà reindirizzato alla **home page** dell'app. In questa schermata, vedrà una **TextView di benvenuto** che mostrerà il suo nome

Da questa pagina, l'utente avrà accesso a diverse **activity (pagine)** che prima erano bloccate visto che non era loggato, che gli permetteranno di interagire con le varie funzionalità dell'app: tipo amici, profilo, preferiti, biglietti.



Profilo



Nome:

Banushan

Cognome:

Sivakumar

Data di nascita:

12-07-2005

Email:

banu@gmail.com

SCHERMATA PROFILO

Nella schermata **Profilo**, l'utente troverà i suoi dati personali (come nome, email, etc.) che sono stati salvati tramite **SharedPreferences**. Inoltre, ci sarà un **button di logout** che permetterà all'utente di uscire dal suo account, eliminando le informazioni salvate e indirizzandolo alla schermata di **Home**.

Logout





AMICI



banu siva



concha luigi



Davide senza

SCHERMATA AMICI

l'utente può cercare tutte le persone che si sono loggate.



Preferiti

SCHERMATA PREFERITI

l'utente può vedere gli eventi che ha aggiunto nei preferiti tramite il button nella schermata info eventi.



Eventi in programma

Eventi passati


SCHERMATA BIGLIETTI

qua l'utente può vedere tutti i biglietti che hai acquistato sia degli eventi passati che quelli in programma.



Il sito

il sito consente alle aziende o ai privati di aggiungere eventi .però prima si devono loggare per accedere a questa funzionalità



HomeEventiContattiCiao Banushan

AGGIUNGI EVENTO

Link immagine

Titolo

Lungo

€ /m/aaa

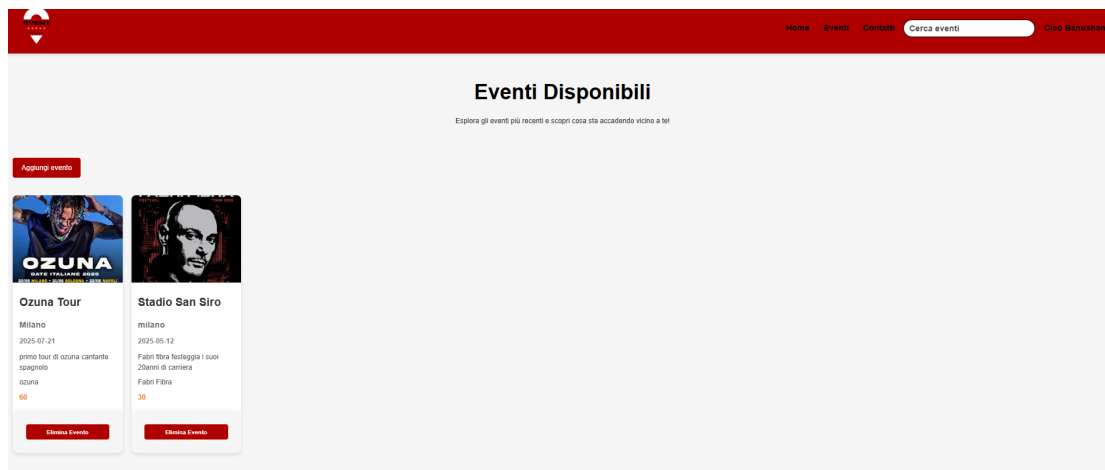
Informazioni sull'evento

Nome artista

Prezzo

Registra evento

appena si aggiunge l'evento viene visualizzato con tutti gli altri eventi.



Linguaggio e tecnologie utilizzati

JAVA

Nel progetto, abbiamo utilizzato **Java** per la gestione del codice all'interno delle **Activity** e per definire le classi modello necessarie per gestire e far interagire gli oggetti all'interno dell'applicazione. Ogni classe svolge una funzione specifica, come la gestione dell'interfaccia utente, la connessione col backend, e la gestione dei dati dell'utente.

Gestione delle Activity in Java

Le **Activity** rappresentano le schermate principali dell'applicazione, come la schermata di login, la home di eventi, il profilo dell'utente, e così via. Ogni Activity è associata a un layout definito in **XML**. In Java, queste Activity gestiscono l'interazione dell'utente con l'interfaccia utente e con le altre componenti dell'app.

```
SearchView cerca = findViewById(R.id.searchView);
Button loginButtonHome = findViewById(R.id.loginHomeBtn);
ImageButton btnFiltro = findViewById(R.id.btnFiltro);
ListView listView = findViewById(R.id.eventiListView);
ImageButton btnBiglietto = findViewById(R.id.btnBiglietti);
ImageButton btnPreferiti = findViewById(R.id.btnPrefe);
ImageButton btnAmici = findViewById(R.id.btnHomeAmici);
ImageButton btnProfilo = findViewById(R.id.btnProfilo);
```

All'interno di ogni **Activity** abbiamo utilizzato il metodo **findViewById()** per collegare i vari componenti dell'interfaccia utente (come bottoni, text views, e list views) definiti nel layout XML con gli oggetti Java, permettendo loro di interagire fra loro.

PYTHON

Python è un linguaggio di programmazione ad alto livello, molto leggibile e semplice da imparare. È ampiamente usato in diversi ambiti, tra cui lo sviluppo web, l'analisi dei dati, l'automazione, l'intelligenza artificiale e molto altro, grazie alla sua versatilità e alla ricca disponibilità di librerie.

Per lo sviluppo del back-end della nostra applicazione abbiamo scelto di utilizzare proprio Python, sfruttando il micro-framework Flask. Questo strumento leggero ma potente ci ha permesso di gestire in modo semplice ed efficace le richieste provenienti sia dall'app Android che dal sito web. Flask ci ha facilitato nella connessione al database, nell'elaborazione dei dati e nel ritorno delle risposte corrette ai client.

Flask e Retrofit

Flask è un **framework**, ovvero un insieme di strumenti e librerie che forniscono una base solida per lo sviluppo di applicazioni web. Scritto in Python, è particolarmente indicato per progetti agili e leggeri come il nostro. Il suo scopo principale nel nostro progetto è stato quello di **mettere a disposizione delle API REST**, ovvero delle interfacce che permettono all'applicazione Android (il client) di comunicare con il server dove vengono salvati i dati degli utenti.

Per interagire con queste API dal lato Android, abbiamo utilizzato **Retrofit**, una libreria che permette di semplificare le chiamate HTTP, come ad esempio inviare dati al server o riceverli in formato JSON.

```
private void registerUser() {  
    String n = nome.getText().toString();  
    String c = cognome.getText().toString();  
    String e = email.getText().toString();  
    String d = data_di_nascita.getText().toString();  
    String p = password.getText().toString();
```

i dati inseriti dall'utente vengono salvati

```
Utente nuovoUtente = new Utente(n, c, e, d, p);
```

viene creato un **oggetto Utente** usando i dati dell'utente inseriti dall'utente .
Questo oggetto verrà poi **inviato al server tramite Retrofit**.

```
Call<Void> call = apiService.registerUser(nuovoUtente);
```

Call raccoglie tutte le informazioni che l'utente ha scritto (nome, email, ecc.) e le mette pronte per essere inviate al server.

```
call.enqueue(new Callback<Void>() {
```

spedisce i dati raccolti tramite la Call al server

```
public interface ApiService {
    1 usage
    @POST("/mobile_register")
    Call<Void> registerUser(@Body Utente user);
}
```

Attraverso una classe chiamata API Service, i dati vengono inviati in formato JSON a un endpoint, cioè un indirizzo URL di Flask (il nome della Call e l'endpoint devono combaciare).

```
@app.route('/mobile_register', methods=['POST'])
def mobile_register():
    data = request.get_json()
    nome = data.get('nome')
    cognome = data.get('cognome')
    email = data.get('email')
    data_di_nascita = data.get('data_di_nascita')
    password = data.get('password')

    cursor = connection.cursor()
    query = "INSERT INTO utenti (nome, cognome, email, data_di_nascita, password) VALUES (%s, %s, %s, %s, %s)"
    cursor.execute(query, (nome, cognome, email, data_di_nascita, password))
    connection.commit()
    cursor.close()
    return jsonify({"message": "Utente registrato correttamente!"}), 201
```

l'endpoint di Flask prende i dati e con essi genera una richiesta(query) al server. Se un utente corrisponde alle credenziali, l'esito della richiesta sarà positivo.

sul **database** viene controllata la presenza dell'utente.

```
--
16 • select * from utenti;
17
```

Result Grid Filter Rows: Edit: Export/Import:						
	id	nome	cognome	email	data_di_nascita	password
▶	1	banu	siva	banu@gmail.com	2005-07-12	banu
	2	concha	luigi	concha@gmail.com	2004-05-15	banu
	3	Davide	sanza	davide@gmail.com	2003-02-12	banu
★	NULL	NULL	NULL	NULL	NULL	NULL

Le informazioni inserite dall'utente partono dall'app Android e vengono inviate al server tramite **Flask**, che agisce come intermediario. Flask riceve la richiesta, la elabora e comunica con il **database** dove sono salvati i dati (ad esempio utenti, eventi, preferiti, ecc.). Una volta ottenuta la risposta dal database, Flask la restituisce all'app, che la mostra all'utente.

HTML CSS

HTML e CSS ci hanno permesso di realizzare il nostro sito web, occupandoci sia della sua struttura (con HTML) che del design e dello stile grafico (con CSS). Grazie a questi linguaggi, abbiamo potuto creare pagine web ordinate, visivamente curate e facilmente navigabili dagli utenti.

RISULTATI OTTENUTI

In generale, **EventFinder** ha raggiunto i suoi obiettivi principali:

- Creare una piattaforma facile da usare per la visualizzazione e la gestione di eventi.
- Offrire agli utenti un sistema di ricerca avanzato e personalizzato.
- Implementare una solida architettura backend con **Flask** per la gestione dei dati in tempo reale.
- Fornire un'esperienza utente coerente e fluida su app mobile e sito web.

CONCLUSIONE

L'app Event Finder rappresenta una soluzione completa e dinamica per la scoperta e la gestione degli eventi, progettata per soddisfare le esigenze di utenti in cerca di esperienze uniche e personalizzate. Grazie alla sua interfaccia intuitiva, sviluppata utilizzando Android Studio con Java e la gestione efficace dei dati tramite SharedPreferences e Retrofit, l'app offre una navigazione fluida e accesso rapido alle informazioni sugli eventi.

Il back-end è stato sviluppato con Python e il framework Flask, garantendo un'architettura robusta e scalabile per gestire le richieste provenienti sia dall'app che dal sito web. Utilizzando API REST per lo scambio di dati in formato JSON, l'app si integra perfettamente con il server, permettendo agli utenti di registrarsi, filtrare eventi e acquistare biglietti in modo semplice e sicuro.

L'integrazione con il database consente una gestione efficiente delle informazioni sugli utenti e sugli eventi, mentre il sistema di filtri avanzati offre un'esperienza personalizzata, consentendo agli utenti di trovare eventi specifici in base ai propri interessi. Inoltre, l'uso di GitHub per il versionamento del codice e di Figma per la progettazione dell'interfaccia ha reso possibile una collaborazione efficiente e una progettazione grafica curata.

In conclusione, Event Finder offre una piattaforma versatile, stabile che migliora l'esperienza di chi cerca eventi da non perdere, con una solida architettura.