

TripPic

Share your best memories with your friends

Progetto per Programmazione Dispositivi Mobili

Chiara Andreato

Giacomo Grandi

Federico Peiretti

1. Introduction

Project Idea

TripPic è un'applicazione per dispositivi Android.

Si tratta di un social network dedicato a chi ama viaggiare e condividere le proprie esperienze. A differenza dei social più noti però, non si tratta solo di una raccolta multimediale, vengono infatti messe in primo piano le località visitate.

Ogni utente potrà caricare i propri scatti direttamente sulla mappa di Google.

In questo modo condividere la propria esperienza di viaggio è più semplice e divertente.

L'utente ha quindi modo di:

- Creare un **diario** di viaggio personale con foto e descrizione dei luoghi visitati
- Visualizzare sulla **mappa** del mondo i pin che rappresentano i propri contenuti e quelli caricati dagli amici
- Salvare un elenco di post **preferiti** per lasciarsi ispirare e scoprire nuove località da visitare

Ciò che TripPic vuole evitare è tutto ciò che riguarda la competizione e la ricerca della notorietà nel mondo dei social network.

Non è progettato per mettere in risalto alcuni utenti rispetto ad altri, non tiene conto del numero di amici degli utenti, non tiene conto dei like.

Gli utenti possono salvare i contenuti dei propri amici per visualizzarli in una lista separata, ma non possono sapere quante persone hanno salvato un loro contenuto.

Ogni utente per visualizzare i contenuti di altri utenti deve prima aggiungerli come amici.

TripPic non permette di commentare i contenuti altrui.

I contenuti caricati su TripPic non devono essere ragione di liti e "flame" come spesso accade nei social più noti.

App Functionalities

L'utente una volta scaricata TripPic deve registrarsi per poter accedere alle funzionalità offerte da questa applicazione.

TripPic si basa sulla gestione di **contenuti**, detti anche post, e di **utenti**.

I contenuti sono composti da un titolo, un autore (persona che ha effettuato il caricamento del post), una descrizione, una localizzazione geografica e ovviamente un'immagine, una fotografia. TripPic si basa sul fatto che questi contenuti siano fotografie di viaggio, infatti in fase di caricamento è richiesto all'utente di associare la localizzazione geografica alla fotografia che intende caricare.

Gli utenti compaiono sulla piattaforma TripPic caratterizzati da uno username univoco, un'immagine di profilo, la loro città di residenza e una breve descrizione di se stessi.

Completato il login l'utente può navigare attraverso l'applicazione in modo facile e intuitivo utilizzando la barra in basso: i 5 pulsanti gli permettono di accedere alle 5 schermate principali:

Home (Map): presenta la mappa di Google con i pin a cui corrispondono i post caricati dagli utenti. Selezionando uno di questi pin è possibile espanderlo per visionare il contenuto interno. L'utente può navigare sulla mappa utilizzando la barra di ricerca, e trovare più facilmente una località lasciandosi suggerire dal completamento automatico. L'utente può inoltre trovare il luogo in cui si trova sulla mappa concedendo l'accesso alla posizione del dispositivo.

Search: schermata che fornisce all'utente la possibilità di ricercare gli utenti e contenuti dell'applicazione più facilmente. Usando due bottoni di navigazione molto intuitivi può scegliere se cercare un post, ovvero un contenuto, basandosi sul suo titolo, oppure se cercare un altro utente digitando il suo nome. Se l'utente desidera aggiungere alla sua lista di amici uno degli utenti presenti può farlo mediante l'apposito pulsante direttamente dalla lista.

Add: questa schermata permette all'utente di creare un nuovo contenuto caricando un'immagine dalla galleria del proprio smartphone oppure scattando una fotografia sul momento. Per completare il caricamento deve quindi aggiungere il titolo del post, una descrizione e la localizzazione geografica.

Favourites: l'utente visualizzando i contenuti di TripPic può salvare quelli che ritiene più interessanti e visualizzarli in questa schermata. Selezionando uno dei post che compaiono

in questa sezione può espanderne il contenuto e se lo desidera rimuoverlo dalla lista dei contenuti salvati.

Profile: questa schermata consente all'utente di visualizzare il proprio profilo. Può vedere l'immagine di profilo caricata, il proprio username, la propria descrizione e città. Dei pulsanti di navigazione intuitivi gli permettono di visualizzare l'elenco dei propri post e l'elenco dei propri amici. In entrambi gli elenchi i contenuti possono essere espansi: si può vedere meglio uno dei post caricati oppure vedere il profilo di un proprio amico.

Design choices and alternatives

Le scelte progettuali sono dovute alla necessità di rispettare i goal di progettazione che ci siamo posti ideando TripPic. Dato che la nostra idea era quella di creare un'applicazione di viaggi basata sul caricamento dei contenuti degli utenti su una cartina virtuale abbiamo dovuto affidarci a servizi già esistenti. Costruire daccapo un servizio di gestione di mappe virtuali sarebbe stato al di fuori delle nostre capacità, quindi abbiamo deciso di sfruttare le API messe a disposizione da Google.

I vantaggi principali derivanti dalla scelta di utilizzare le mappe di Google anziché altre sono soprattutto legati alla notorietà di queste. Trovare tutorial su come integrare le Google Maps API nella nostra applicazione è stato molto facile, dal momento che erano molto abbondanti. Inoltre abbiamo utilizzato la documentazione ufficiale messa a disposizione da Google.

Uno dei principali svantaggi di utilizzare questo servizio è il fatto che esso non è gratuito. Per lo sviluppo della nostra applicazione abbiamo utilizzato una versione ridotta delle API fornite. Fare ciò ci ha permesso di sviluppare la demo della nostra applicazione, utile per essere presentata all'esame, ma assolutamente improponibile ad un'utenza su larga scala.

Le scelte progettuali riguardo all'implementazione del frontend, ovvero la scelta di utilizzare Android Jetpack Compose, ci ha permesso di velocizzare molto la costruzione dell'interfaccia di TripPic. Inoltre ci ha consentito di renderla adatta a una vasta gamma di smartphones e tablet, senza aver dovuto creare elementi diversi per tipi di schermi a dimensione e risoluzione differente.

Il principale svantaggio dell'adottare questa tecnologia è stato dover imparare a conoscerla e ad utilizzarla. L'abbondanza di documentazione in proposito ha reso per noi possibile apprendere il funzionamento, ma comprendere come utilizzare le librerie di Android Jetpack Compose ha richiesto tempo e un notevole sforzo da parte di noi sviluppatori.

Infine, a proposito di scelte progettuali riguardanti il backend, abbiamo deciso di integrare in TriPic i servizi di Amazon, per avere una struttura completa facile a costruire e utilizzare. AWS è la piattaforma di Cloud computing offerta da Amazon. Con essa è stato per noi possibile collegare le parti di frontend già sviluppate con un database e il servizio di storage.

Dal momento che per soddisfare i requisiti TripPic era necessario salvare delle immagini, il solo utilizzo di un database relazionale non era possibile. AWS fornisce oltre ad un database NoSQL, che è una tipologia di database ottima per un social network, anche un servizio di storage mirato a salvare ogni tipo di dato, nel nostro caso le immagini degli utenti. I vantaggi e gli svantaggi di AWS si avvicinano molto a quelli delle mappe di Google: la notorietà di tali strumenti e la abbondanza di documentazione ci ha reso possibile comprendere e utilizzare tale tecnologia. Inoltre i servizi di Amazon sono davvero veloci e sicuri, sarebbero perfetti se in un futuro decidessimo di ampliare la diffusione di TripPic. Come elemento contrario hanno però il non essere gratuiti: per lo sviluppo dell'applicazione per l'esame abbiamo utilizzato una versione di prova gratuita, che però è limitata in termini di disponibilità temporale, di spazio disponibile e numero di query.

Material Design

In TripPic abbiamo fatto uso di Material Design. Si basa infatti su un tema comune a tutti i widget dell'interfaccia.

In tutta l'applicazione è condiviso lo stesso stile di testo, colori e icone.

La scelta di far uso del Material Design si lega in primis alla necessità di avere una barra di navigazione. Ogni destinazione della barra è rappresentata da un'icona e da un'etichetta di testo facoltativa. Quando viene toccata un'icona di navigazione inferiore, l'utente viene indirizzato alla destinazione di navigazione di primo livello associata a tale icona. Nel nostro caso è stata scelta una barra a 5 destinazioni.

Grazie al Material Design abbiamo potuto gestire anche le icone, e differenziare tra quelle che sono state selezionate e quelle che non lo sono.

Competitors

I tre principali competitors sono:

- **Instagram:** nota applicazione per la condivisione di immagini con gli amici
- **Places Been:** applicazione che permette di contrassegnare su una mappa personale le città visitate
- **Journi Blog:** applicazione che permette di creare un blog pubblico o privato per caricare le proprie foto di viaggio



Project Timeline

Lo sviluppo del progetto ha seguito i seguenti passi principali, che verranno illustrati più approfonditamente più avanti:

- Definizione di obiettivi e caratteristiche dell'applicazione da sviluppare, stesura della scheda del progetto
- Implementazione della main activity e integrazione delle Google API per la mappa
- Utilizzo di Jetpack Compose per la creazione delle interfacce
- Sviluppo della UI e delle funzionalità principali
- Sviluppo del backend con AWS
- Revisione del progetto e test su dispositivi differenti

Main Changes

Rispetto alla proposta iniziale dell'app sono state apportate alcune modifiche, anche se non particolarmente rilevanti: la struttura iniziale è stata mantenuta e i goal rispettati.

Abbiamo mantenuto l'autenticazione di un utente e reso possibile anche la registrazione tramite Facebook.

Come stabilito già in origine l'utente è caratterizzato da un proprio profilo in cui non compaiono però solo foto, username e descrizione ma anche la città in cui risiede.

Il caricamento e la visualizzazione dei contenuti sono stati rispettati. In aggiunta però, il contenuto non è solo caratterizzato da immagine, descrizione e localizzazione ma anche da un titolo.

La possibilità di fare una ricerca di un contenuto o di una persona su TripPic sono state rispettate.

Abbiamo deciso di rimuovere la possibilità di "taggare" altri utenti, ovvero di associare il nome in fase di caricamento di un proprio contenuto.

Al momento non sono state gestite le notifiche, l'utente non riceve avvisi se altri utenti caricano nuovi contenuti o lo inseriscono nella propria lista di amici. Se aggiungere tali notifiche è tuttora oggetto di dibattito nel gruppo di sviluppo.

Abbiamo inoltre deciso di non implementare alcuna sezione di statistiche, come si era pensato in principio. Avendo scelto di dare un taglio più etico alla nostra applicazione, abbiamo deciso di non creare la possibilità che nascano ragioni di competizione tra gli utenti.

Per quanto riguarda invece le aggiunte, rispetto all'idea di progetto originale, abbiamo fornito agli utenti la possibilità di condividere le immagini presenti sulla piattaforma su altre applicazioni (Telegram, Whatsapp...)

Abbiamo inoltre fornito la possibilità di cercare un luogo sulla mappa di Google mediante apposita barra di ricerca e completamento automatico.

Future Developments

Il progetto da noi realizzato è un prototipo con il fine di illustrare quello che potrebbe rappresentare l'applicazione TripPic per gli utenti una volta pronta per la distribuzione. Ci sono comunque già degli aspetti noti che dovranno essere sviluppati in futuro:

- **Contenuti video:** fornire all'utente la possibilità di caricare anche contenuti audiovisivi
- **Partnership:** targetizzazione utenti e partnership con Booking, Airbnb, Trivago per offrire spazio pubblicitario
- **Larga distribuzione:** rendere TripPic fruibile ad un gran numero di utenti garantendo sicurezza ed efficienza

Required Permissions

Per usufruire delle funzionalità dell'app TripPic all'utente saranno richiesti alcuni permessi:

- **Gallery Permission:** per creare contenuti l'utente può caricare le proprie fotografie dalla sua galleria
- **Camera Permission:** l'utente può inoltre creare nuovi contenuti direttamente scattando immagini con la telecamera del proprio smartphone
- **Position Permission:** l'utente può caricare contenuti associando direttamente la propria posizione, perciò deve consentire all'app l'accesso alla propria posizione
- **Internet Permission:** sarà ovviamente richiesta la connessione a internet per accedere all'app

Dopo aver acconsentito ai precedenti permessi l'utente potrà condividere le proprie immagini, o quelle degli amici, anche su altre piattaforme, come ad esempio Instagram, Telegram e Whatsapp.

Development Methodology

La metodologia di sviluppo adottata è **Test Driven Development**. In fase di sviluppo abbiamo infatti testato le porzioni di codice prima di integrarle effettivamente nel progetto. I test sono stati fatti sia per quanto riguarda l'integrazione delle parti di backend, sia per quanto riguarda il frontend. Nel caso dell'implementazione di interfacce utente i test sono stati effettuati in primis su dispositivo virtuale mediante il simulatore di smathpone, e in secundis sono state testate su dispositivi fisici quali cellulari e tablet.

Ogni fase di sviluppo è stata composta di due momenti: un primo di **sperimentazione e test**, che ha preceduto l'implementazione di ogni singolo servizio dell'applicazione. Successivamente vi è stato un momento di **refactoring** in cui il codice sviluppato è stato rivisto e migliorato. In questa seconda fase abbiamo provveduto a pulire il codice di ridondanze e semplificato la sua struttura.

Fasi di test e refactoring hanno inoltre accompagnato la fine dello sviluppo di ogni macro funzionalità di TripPic.

Lo sviluppo di ogni macro componente, ad esempio di una nuova vista, o l'utilizzo di una nuova tecnologia, come Jetpack Compose o ASW Services, è stato preceduto da una fase di sperimentazione in cui sono stati effettuati test su applicazioni "giocattolo". Solo una volta che la nuova porzione si è rivelata funzionante è stata integrata nel progetto.

TripPic è stata sviluppata attraverso release successive. Fare ciò ci ha permesso di lavorare meglio in gruppo e di mantenere un backup del progetto ad ogni step.

2. Front-end Architecture

TripPic Frontend Architecture

Il frontend di TripPic è strutturato a partire da una **Activity principale** e sei **Fragment** sottostanti, cinque per il layout delle schermate che offrono i contenuti e uno per la barra di navigazione tra le cinque schermate.

La scelta di questa struttura si basa sulla necessità di avere la barra di navigazione in basso che permetta all'utente di spostarsi attraverso le varie pagine dell'applicazione in modo semplice, intuitivo e rapido, e senza che la barra di navigazione venga ricaricata ogni volta. Per cui il codice risulta strutturato come segue:

- una classe kotlin **MainActivity** e un file XML relativo, usati per definire il layout e il funzionamento della **bottom navigation bar**, per gestire lo switch sulle varie pagine dell'app e per permettere all'utente di registrarsi la prima volta che viene aperta l'app (funzionalità definita nella classe kotlin **CompleteProfileFragment**)
- per gestire il fragment relativo alla pagina principale della mappa si usa la classe kotlin **MapFragmentCompose**
- per il fragment relativo alla pagina di ricerca la classe kotlin **SearchFragment**
- per il fragment relativo alla pagina per aggiungere nuovi contenuti la classe kotlin **AddFragment**
- per il fragment relativo alla schermata che mostra l'elenco dei post preferiti la classe kotlin **FavoritesFragment**
- per il fragment relativo alla pagina del profilo la classe kotlin **ProfileFragment**

La UI di tutti i fragment è stata realizzata con **Jetpack Compose**, illustrata in seguito.

User Registration


L'utente può accedere a TripPic solo previa registrazione e creazione di un profilo. Mediante questa schermata l'utente ha la possibilità di registrarsi e di accedere al mondo di TripPic: deve inserire una foto profilo, fornire una descrizione personale o un pensiero, indicare la città di provenienza e un ID che sarà univoco e identificativo all'interno dell'applicazione.

L'utente può effettuare l'iscrizione più rapidamente mediante il suo account di Facebook.

20:58

3d6f3-dev.auth.eu-central-1.amazoncognito.com

Sign In with your social account

 Continue with Facebook

We won't post to any of your accounts without asking first

or

Sign in with your username and password

Username

Username

Password

Password

[Forgot your password?](#)

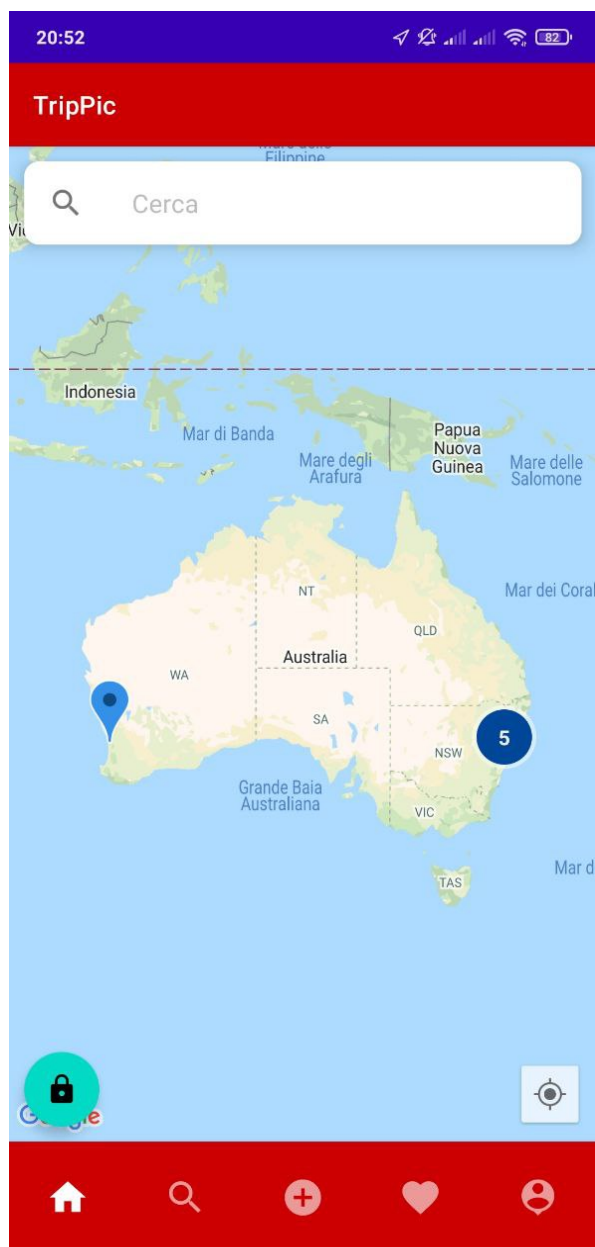
Sign in

Need an account? [Sign up](#)

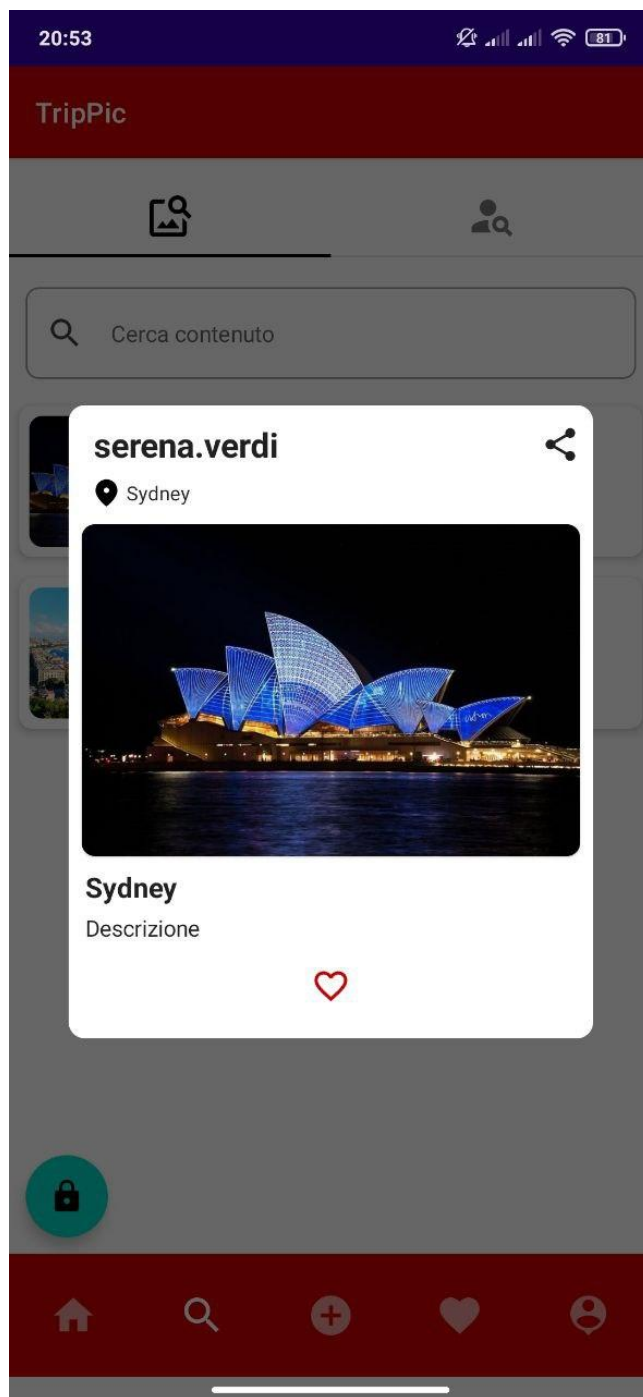
inglese italiano

Home Fragment

La pagina principale dell'app mostra la mappa fornita da Google.

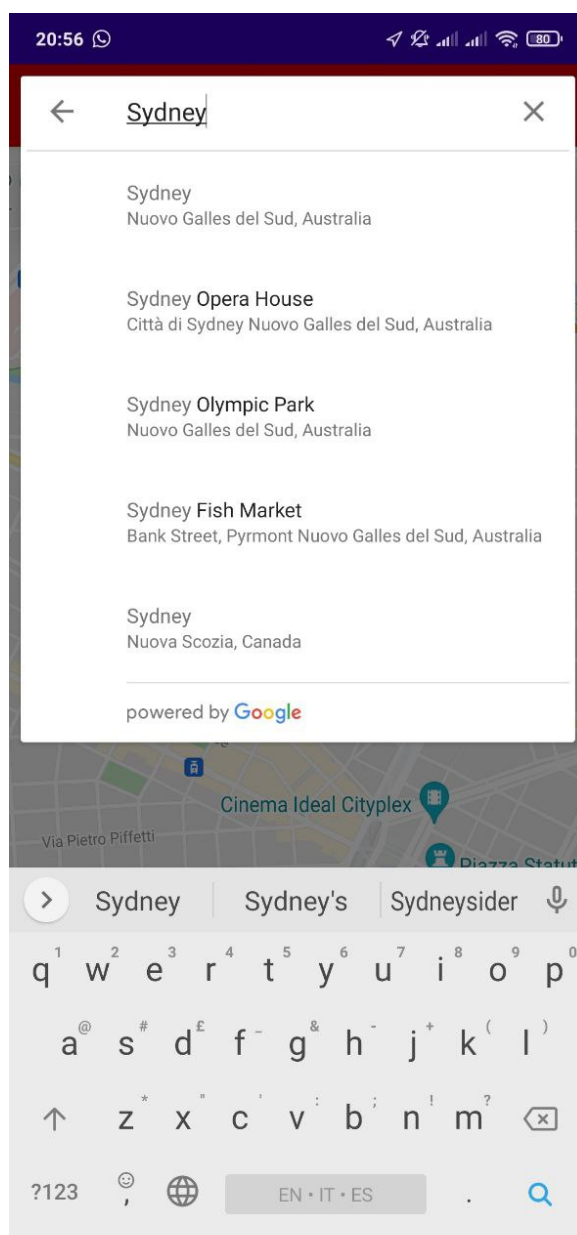


In questa schermata l'utente può cliccare sui pin presenti nella mappa per poter visualizzare il post relativo: verrà così mostrato il popup in figura sotto, da cui sarà possibile aggiungere il post ai preferiti o condividerlo mediante altre applicazioni.



Per realizzare questa schermata abbiamo utilizzato le Google Maps Android API, installando Google Play Services SDK in Android Studio, per integrare i servizi di Google Maps nella nostra applicazione. Abbiamo quindi dovuto creare ed abilitare una API key per l'uso all'interno dell'applicazione. Quindi abbiamo inserito la API key in Android Studio nel file **google_maps_api.xml**.

Il **Manifest** della nostra applicazione (AndroidManifest.xml) contiene i permessi necessari all'utilizzo delle Google Maps API.



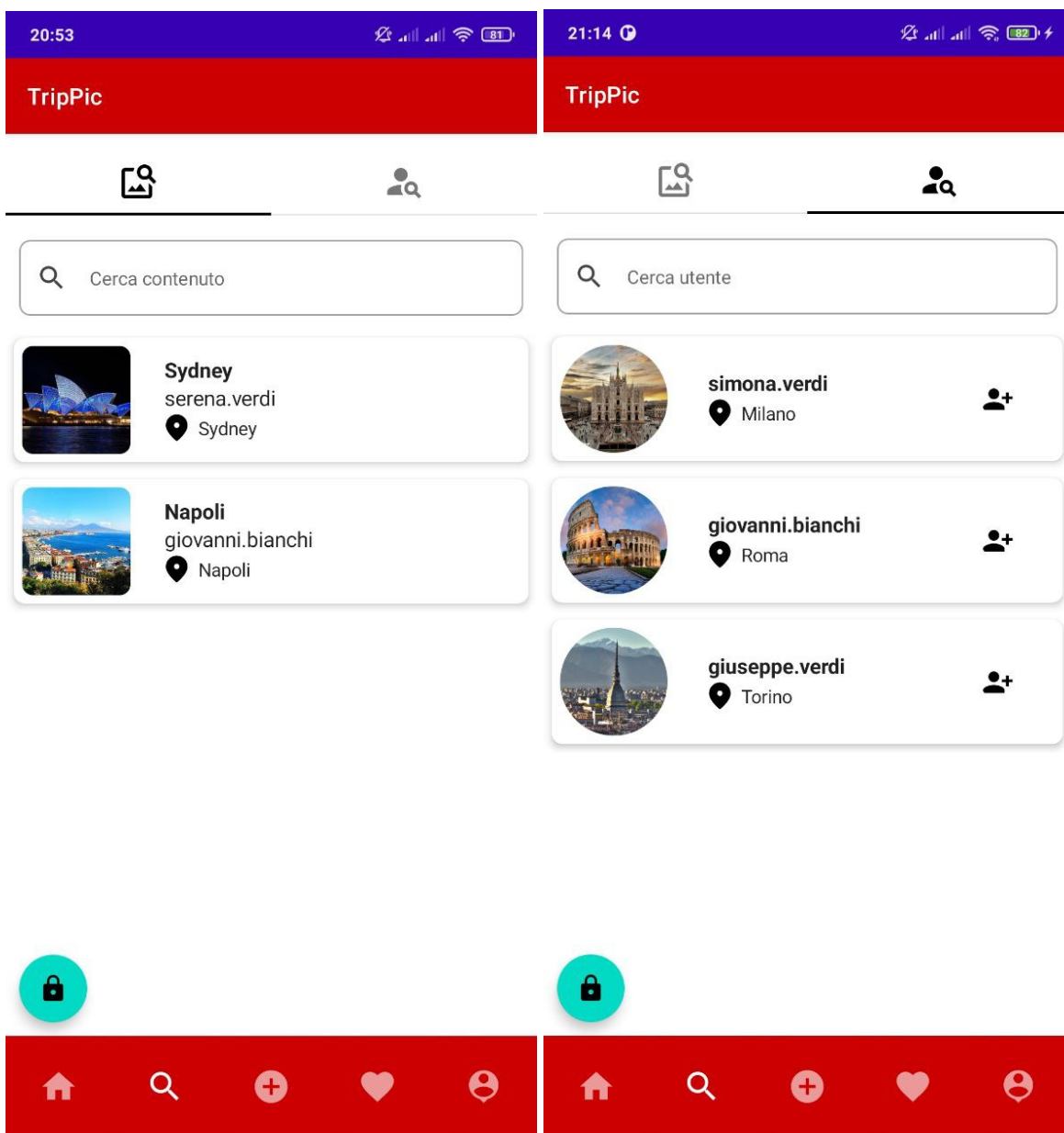
Abbiamo inoltre utilizzato le API di Autocomplete e Places di Google per implementare la barra di ricerca della localizzazione. Lo scopo è dare all'utente la possibilità di cercare più facilmente un luogo, digitando il nome di una città o un indirizzo, e poterne ricavare automaticamente la localizzazione sulla mappa di Google e derivarne latitudine e longitudine.

La localizzazione infatti è gestita tramite il salvataggio di due variabili: latitudine e longitudine. Utilizzando questi due parametri, mediante le Google Maps API, è stato possibile avere accesso alla posizione dell'utente e visualizzare i contenuti creati direttamente sulla mappa.

Per poter integrare la barra di ricerca abbiamo fatto uso del **View Binding**, la barra infatti, al contrario delle viste, è gestita lato frontend mediante un apposito file XML.

Search Fragment

In questa schermata l'utente ha la possibilità di cercare dei post specifici tramite il titolo oppure utenti tramite lo username, in modo da poter visualizzare il loro profilo e i loro post e aggiungerli ai propri amici



Add Fragment

In questa pagina l'utente può caricare un nuovo contenuto sul proprio profilo. Per farlo dovrà occuparsi di:

- Selezionare una **foto** dalla galleria immagini del proprio dispositivo
- Scegliere un **titolo** per il nuovo post
- Scrivere una **descrizione** relativa alla foto
- Scegliere una **posizione** per permettere di visualizzare il pin relativo al post sulla mappa; anche in questo caso la ricerca della posizione è stata implementata servendosi delle API di Autocomplete e Places di Google

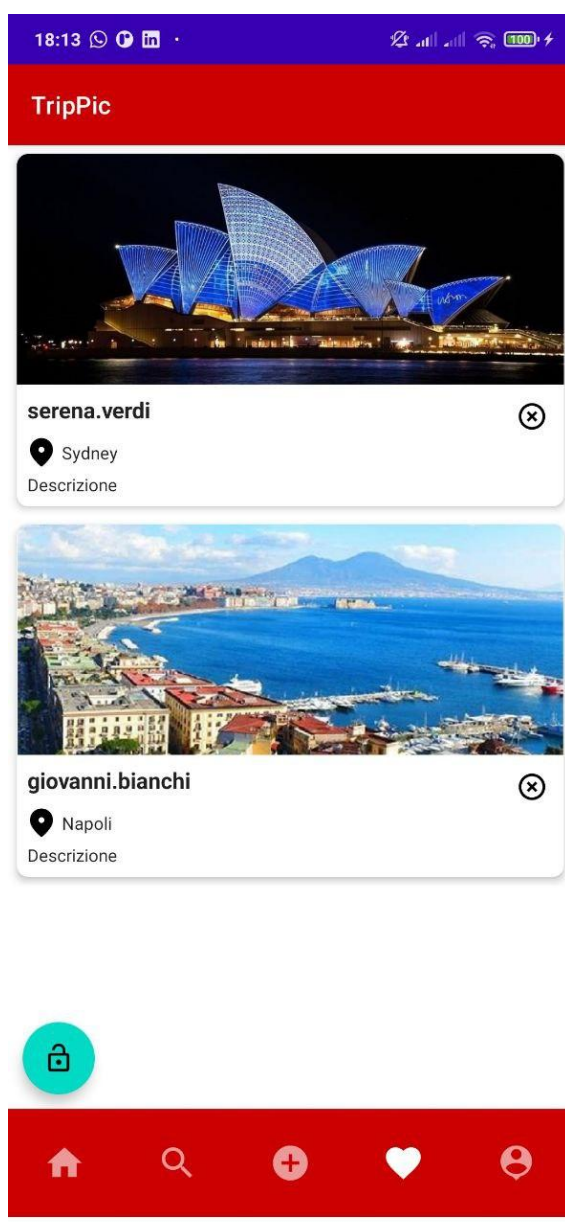
Per poter caricare contenuti dalla galleria e dalla fotocamera e dalla galleria abbiamo utilizzato gli **Intents**. Allo stesso modo è stato necessario utilizzarli per la condivisione delle immagini su altre piattaforme (come Telegram, Whatsapp...).

The screenshot shows the 'TripPic' app interface. At the top, the status bar shows the time 20:55 and various icons. Below the app title 'TripPic', there is a location input field with a pin icon, a text field containing 'Torino', and a close button (X). Below this is a photo of a snowy street scene in Torino, featuring a statue and buildings. Under the photo is a button with a plus icon and the text 'Carica immagine'. Below that are two text input fields: 'Titolo' with the text 'Torino sotto la neve' and 'Descrizione' with the text 'Il monumento del Frejus di Piazza Statuto innevato'. At the bottom of the form is a red button labeled 'CREA POST'. To the left of this button is a circular profile picture placeholder with a lock icon. The bottom of the screen features a red navigation bar with five icons: a home icon, a search icon, a plus icon, a heart icon, and a user profile icon.

Favorite Fragment

In questa schermata viene mostrato un **elenco** di tutti i post che l'utente ha inserito tra i **preferiti**.

Cliccando su uno qualsiasi dei post sull'elenco viene mostrato lo stesso popup della home per visualizzare la foto, il titolo, la descrizione e lo username dell'autore del contenuto. L'utente può inoltre rimuovere dall'elenco un post cliccando sulla apposita icona di rimozione.

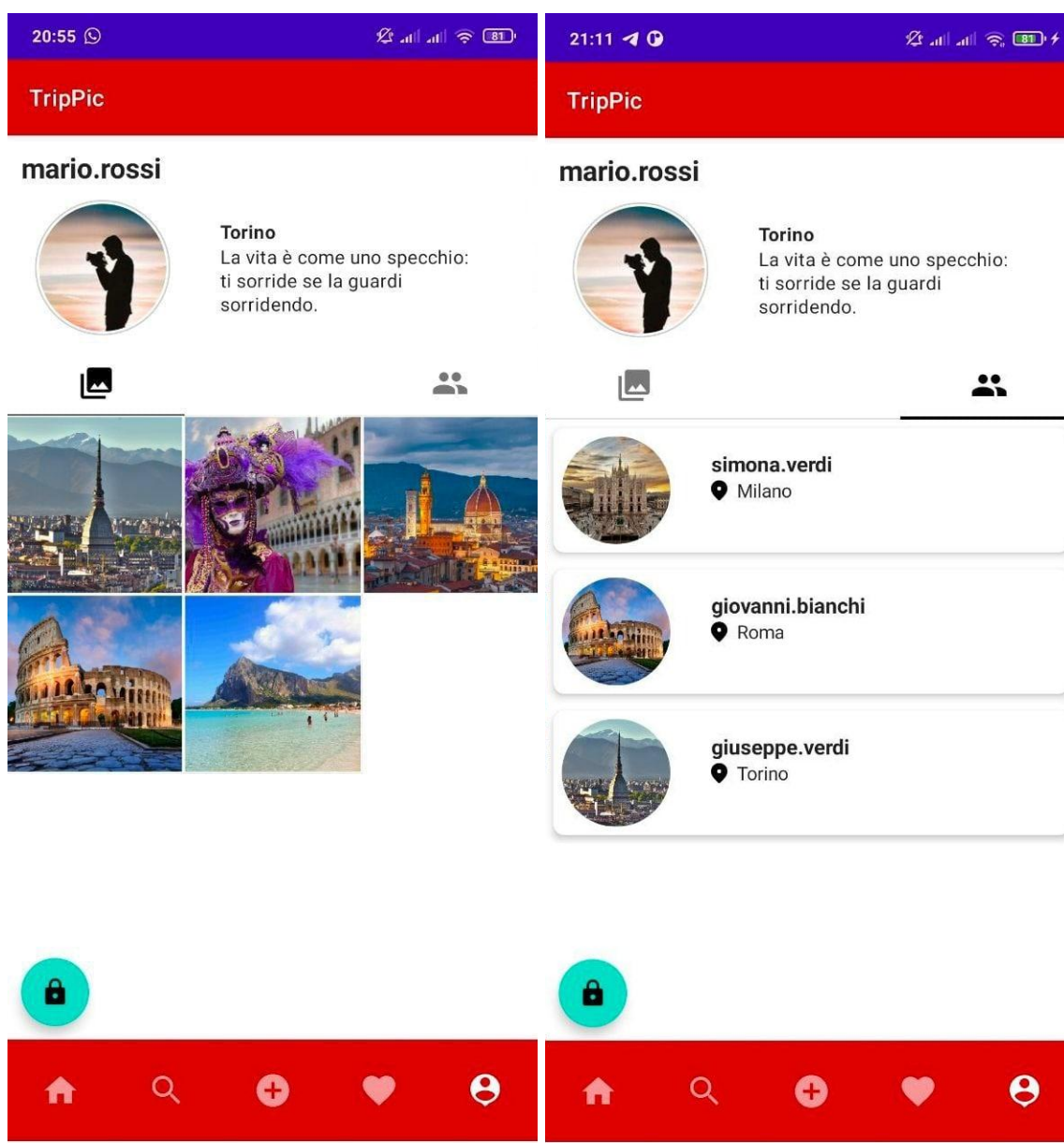


Profile Fragment

Questo fragment gestisce la schermata del profilo. Vengono mostrate: la **foto** profilo, lo **username**, la **città** di provenienza e una breve **biografia**.

L'utente inoltre può scegliere se visualizzare una griglia dei post caricati oppure l'elenco dei propri amici.

Cliccando su uno qualsiasi dei post sulla griglia viene mostrato il solito popup per visualizzarne meglio il contenuto.



Android Jetpack Compose

Come già accennato in precedenza, la UI dei precedenti fragment è stata realizzata con **Jetpack Compose**, ovvero una suite di librerie offerte da Android per la realizzazione di interfacce grafiche:

- **ONLY KOTLIN**: Jetpack Compose permette di non scrivere codice in XML, si utilizza solo Kotlin. Permette di creare interfacce componibili che sfruttano tutte le potenzialità del linguaggio Kotlin
- **DECLARATIVE API**: Jetpack Compose utilizza Declarative API, che permettono di descrivere la funzione dei vari elementi dell'interfaccia. Le componenti non sono legate ad uno specifico fragment, ciò permette di riutilizzarle facilmente
- **LESS CODE**: Jetpack Compose permette di creare UI in modo rapido e scrivendo molto meno codice rispetto ai modi tradizionali di sviluppo in Android. Permette inoltre di ridurre notevolmente bug ed errori e codice da testare
- **COMPATIBLE WITH EXISTING CODE**: non è necessario iniziare a sviluppare l'applicazione in Compose, o nel caso si abbiano già delle funzionalità dell'app non è obbligatorio riscriverle. Si può migrare a Jetpack Compose in modo incrementale. Noi lo abbiamo fatto dopo più di un mese di sviluppo.

External Libraries

Nello sviluppo della parte frontend di TripPic abbiamo fatto uso delle librerie offerte da Jetpack Compose.

Le principali librerie utilizzate sono:

- Librerie di **android** per la gestione delle viste:
 - android.content.Context* per la gestione del Context dell'applicazione
 - android.os.Bundle* per la gestione del passaggio dei dati nell'applicazione
 - androidx.fragment.app.Fragment* per la gestione dei Fragment
 - android.view.View*
 - android.view.ViewGroup*
 - android.Manifest* per la gestione dei riferimenti al Manifest
 - android.content.ContentValues*
 - android.location.Location* per la gestione della localizzazione

com.google.android.libraries.places.api.Places
com.google.android.libraries.places.api.model.Place

e

- Librerie di **androidx.compose.foundation** (Jetpack compose):
*androidx.compose.foundation.layout.** per la gestione dei layout
androidx.compose.foundation.shape.CircleShape per le immagini di profilo
androidx.compose.ui.Modifier collezione di elementi che decorano e gestiscono il comportamento degli elementi dell'interfaccia utente
androidx.compose.ui.draw.clip, *import androidx.compose.ui.graphics.painter.Painter* e *androidx.compose.ui.draw.scale* per la gestione delle immagini
androidx.compose.ui.unit.dp e *androidx.compose.ui.unit.sp* per la gestione della dimensione degli elementi grafici
androidx.compose.foundation.clickable per la gestione degli elementi selezionabili
androidx.compose.foundation.ExperimentalFoundationApi
*androidx.compose.foundation.lazy.** per la gestione di liste e griglie
androidx.compose.material.icons.Icons per la gestione delle icone
androidx.compose.ui.window.Dialog per la gestione delle finestre "Dialog" usate per mostrare il contenuto dei post
- Librerie per l'integrazione delle **Google Maps API**:
com.google.android.gms.location.FusedLocationProviderClient
com.google.android.gms.location.LocationServices
com.google.android.gms.maps.CameraUpdateFactory
com.google.android.gms.maps.GoogleMap
com.google.android.gms.maps.MapView
com.google.android.libraries.places.api.net.PlacesClient
com.google.maps.android.ktx.awaitMap
com.google.android.libraries.places.widget.listener.PlaceSelectionListener
com.google.maps.android.clustering.Cluster
com.google.maps.android.clustering.ClusterManager
- Librerie interne all'applicazione:
educ.unito.myapplication.MainActivity per il riferimento alla Activity principale
educ.unito.myapplication.PostMarkerItem per i pin della mappa
educ.unito.myapplication.PostData per i riferimenti alla dataclass con i post
educ.unito.myapplication.databinding.FragmentMapComposeBinding
- Librerie per il collegamento al **backend**:
com.amplifyframework.core.Amplify per il collegamento ai servizi AWS
androidx.lifecycle.Observer per la gestione degli elementi osservabili

Alternative Frontend Architectures

Inizialmente si era considerato di utilizzare differenti activities, una per ciascuna schermata di TripPic. Questa scelta non avrebbe però permesso di mantenere la barra di navigazione per spostarsi rapidamente all'interno dell'applicazione.

Inoltre, in un primo momento TripPic era stata sviluppata con la combinazione di files XML e Kotlin per costruire le varie viste. La migrazione ad Android Jetpack Compose, avvenuta dopo aver già iniziato a sviluppare alcune viste, ci ha permesso di modificare l'architettura della applicazione da quella più tradizionale a quella di Jetpack Compose in cui è utilizzato solo codice Kotlin.

3. MVVM Architecture

Model View ViewModel Architectural Pattern

Il pattern architetturale **Model View ViewModel** permette di disaccoppiare i compiti fra le componenti software dell'applicazione:

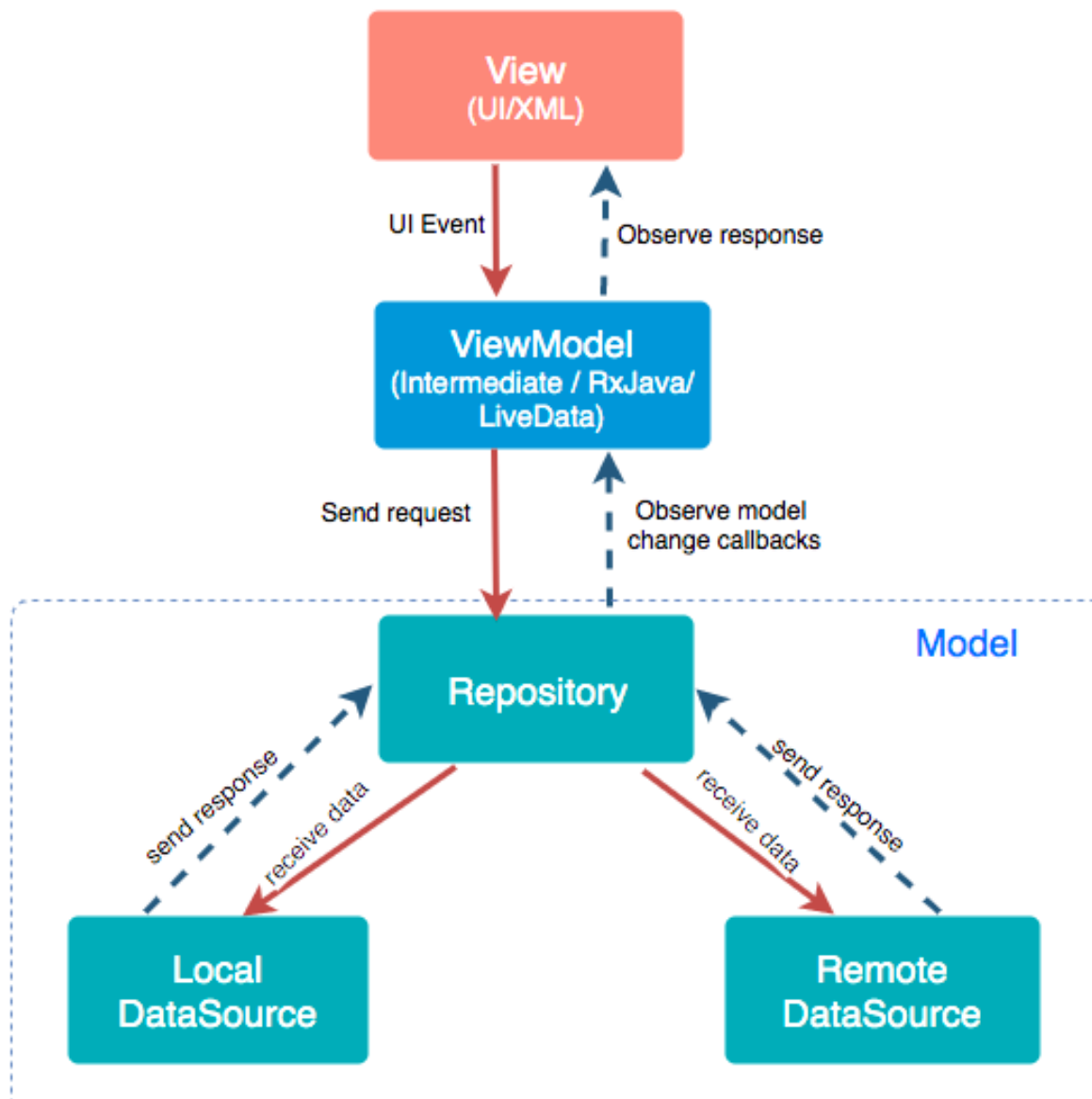
- **Model**: fornisce i metodi per accedere e gestire i dati dell'applicazione, ovvero il Back-end dell'applicazione
- **View**: gestisce la User Interface dell'applicazione, ovvero tutte le classi relative al Front-end dell'app
- **ViewModel**: serve da intermediario tra **View** e **Model**, si occupa perciò della connessione di Front-end e Back-end: esegue il wrapping del **Model** ed espone tutti gli attributi rilevanti per la **View**

In particolare il pattern **MVVM** permette di minimizzare rispetto al pattern **MVC** (Model View Controller) il legame tra **View** e **Model**.

La **View** comunica col **ViewModel** mediante databinding. Il **Model** invece è formato da classi Live Data. Si tratta di wrapper osservabili per ogni tipo di dato, l'observer è notificato quando i dati subiscono variazioni. Quando si attacca un observer ai LiveData esso è associato ad un LifecycleOwner, di solito un Activity o Fragment.

Il Repository, quando **ViewModel** richiede i dati che verranno utilizzati per la visualizzazione, li fornirà scegliendo i dati appropriati localmente o sulla rete. Al **ViewModel** non importa come i dati richiesti sono stati recuperati.

AWS Amplify è stato utilizzato per implementare il Back-end e gestire la connessione con il database. Il **Model** infatti contiene tutte le classi che ha generato AWS con il comando "amplify codegen models".



4. Back-end Architecture

AWS Integration

Abbiamo scelto di fare affidamento ai servizi offerti da Amazon per la gestione della parte Back-end.

Amazon Web Services (AWS) è tra le piattaforme che forniscono Cloud services più utilizzate al mondo. Il cloud computing consiste nella distribuzione on-demand delle risorse IT tramite Internet, con una tariffazione basata sul consumo. Piuttosto che acquistare, possedere e mantenere i data center e i server fisici, è possibile accedere a servizi tecnologici, quali capacità di calcolo, archiviazione e database, sulla base delle proprie necessità affidandosi a un fornitore cloud quale **Amazon Web Services** (AWS).

L'applicazione TripPic permette all'utente di autenticarsi e accedere ai contenuti della piattaforma, mediante il proprio smartphone o tablet, grazie ad **AWS SDKs**, che fornisce le API per collegarsi ai servizi di Back-end.

Amazon SDK è un kit di sviluppo software costituito da vari strumenti che consentono di creare un'applicazione per determinati software, framework, piattaforme hardware, sistemi informatici o qualsiasi altra piattaforma di sviluppo.

AWS SDK fornisce API Java e viene utilizzato per Amazon S3, Amazon EC2, DynamoDB e altro ancora.

AWS AppSync è un servizio completamente gestito che facilita lo sviluppo di API GraphQL gestendo le attività impegnative derivanti dalla connessione sicura a origini dati come AWS DynamoDB, Lambda e altro.



Una volta implementato, **AWS AppSync** ricalibra automaticamente il motore di esecuzione dell'API GraphQL per soddisfare i volumi di richieste API.

Amazon Cognito fornisce un archivio identità sicuro in grado di dimensionare le risorse per milioni di utenti.

I bacini d'utenza di **Cognito** possono essere configurati più facilmente senza effettuare il provisioning di alcuna infrastruttura, e tutti i membri del bacino d'utenza hanno un profilo di directory che è possibile gestire attraverso un Software Development Kit (SDK).

Cognito supporta l'accesso degli utenti tramite l'uso di provider di identità social quali Facebook, Google e Amazon.

Ci siamo serviti inoltre di **AWS Amplify**, ovvero una raccolta di strumenti e servizi per creare web app scalabili con pochissimo codice. Permette la gestione di Back-end cloud serverless da riga di comando.

Abbiamo configurato il progetto per l'utilizzo delle librerie Amplify, che vengono inizializzate a runtime.

Amplify Command Line (CLI) di Amplify è una toolchain unificata che abbiamo utilizzato per creare, integrare e gestire i servizi cloud AWS da terminale per la nostra app.

Amplify CLI supporta più ambienti. Quando si inizializza un progetto con l'interfaccia della riga di comando, si crea un ambiente **Back-end Amplify**.

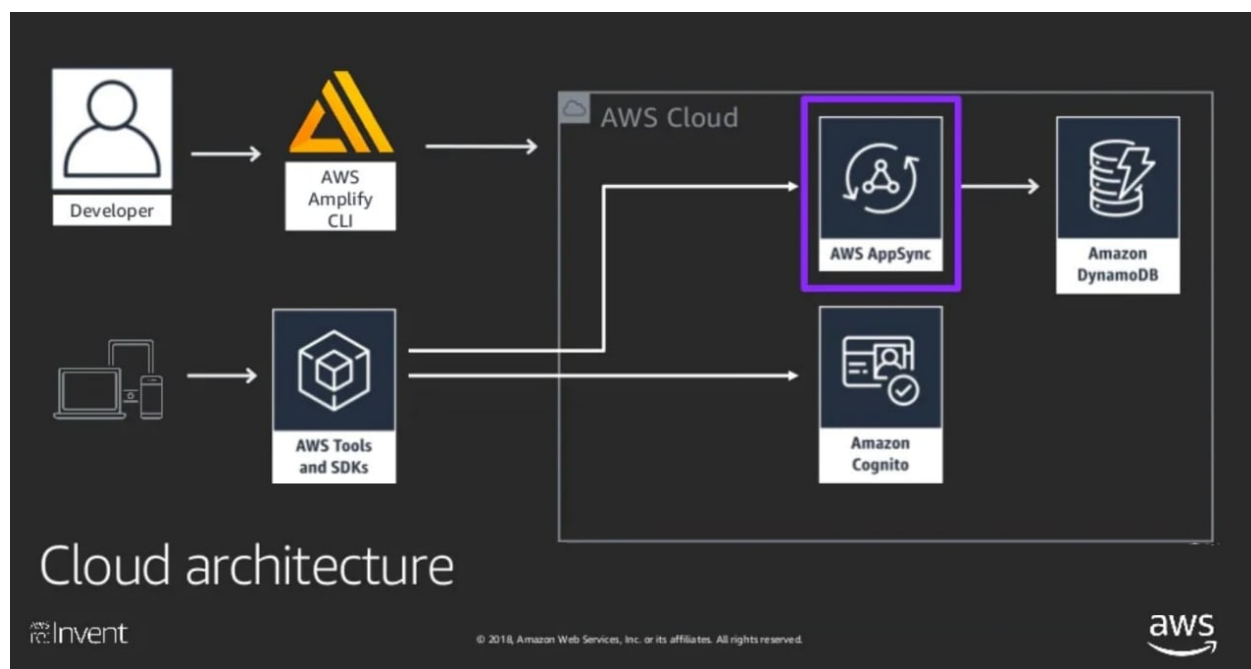
Abbiamo poi creato una classe kotlin **Backend.kt** per raggruppare il codice dell'interazione app-AWS. In tale classe è contenuto il codice per l'inizializzazione di Amplify.

All'avvio dell'app un solo oggetto Back-end viene inizializzato.

Infine per avere un accesso al database strutturato abbiamo fatto uso di **GraphQL**, che è un linguaggio di query e un'API implementation lato server.

Sulla base della definizione del modello di dati GraphQL appena creato, Amplify genera un codice lato client (ossia codice Kotlin) per rappresentare i dati nell'applicazione.

In figura di seguito uno schema riassuntivo dello schema di architettura del Cloud di AWS.



Database and Storage

Il servizio di storage che abbiamo utilizzato è **Amazon S3**, servizio fornito da AWS per il salvataggio di ogni tipo di dato, tra cui immagini. Permette di creare Bucket specifici per ogni tipologia di oggetto inserito.

Il servizio di storage viene collegato con la classe **Backend.kt** creata per la gestione del Back-end.

Per quanto riguarda il database abbiamo utilizzato **DynamoDB** con **GraphQL**.

Amazon DynamoDB si tratta di un database NoSQL. Si tratta di un DB serverless automaticamente gestito che si espande e restringe autonomamente adattandosi ai dati.

Per la gestione del database e degli elementi al suo interno ci siamo serviti delle **GraphQL API**.

In figura è rappresentato lo schema generale del DB.

```

type User
  @model
  @auth (rules: [ { allow: public } ])
  {
    id: ID!
    user: String!
    location: Location @hasOne
    description: String
    image: String
  }

type Location
  @model
  @auth (rules: [ { allow: public } ])
  {
    id: ID!
    lat: Float!
    lng: Float!
    name: String!
  }

type Post
  @model
  @auth (rules: [ { allow: public } ])
  {
    id: ID!
    owner: User! @hasOne
    location: Location! @hasOne
    title: String
    description: String
    image: String!
    date: AWSDateTime!
  }

type Friend
  @model
  @auth (rules: [ { allow: public } ])
  {
    id: ID!
    user: String!
    friends: [User]
  }

type Favorites
  @model
  @auth (rules: [ { allow: public } ])
  {
    id: ID!
    user: User! @hasOne
    post: [Post] @hasMany
  }

```

Back-end Structure

Le classi del **DB Schema** vengono create all'inizio. Tutte quante implementano la classe **Model** e contengono al loro interno i metodi delle API per la gestione dei dati.

Le **Data Classes** sono classi Kotlin, da noi create, per rappresentare gli oggetti contenenti tutte le informazioni necessarie, corrispondono infatti alle classi del **DB Schema**.

Sono quelle che vengono richiamate nel **ViewModel** per interagire con l'interfaccia.

Infine la classe **Backend.kt** gestisce il collegamento col database richiamando le API di AWS Amplify e passandogli gli oggetti creati.

5. Compatibility Test

TripPic Compatibility

TripPic è stata testata su vari dispositivi, in particolare:

- **Virtual Device** (emulatore fornito da Android Studio)
- **Android Smartphone**
- **Android Tablet**

Technological Constraints

L'app è compatibile con tutti i device Android dalla versione 6.0 **Marshmallow** (2015) in poi.

Supportando da questa versione in poi, vengono coperti l'84.9% dei dispositivi.

ANDROID PLATFORM VERSION	API LEVEL	CUMULATIVE DISTRIBUTION	Marshmallow	
4.0 Ice Cream Sandwich	15		Security Fingerprint Authentication Confirm Credential System App Linking Adoptable Storage Devices Multimedia 4K Display Mode Support for MIDI Create digital audio capture and playback objects APIs to associate audio and input devices List of all audio devices Updated video processing APIs Flashlight API Reprocessing Camera2 API Updated ImageWriter objects and ImageReader class User Input Voice Interactions Assist API Bluetooth Stylus Support	User Interface Themeable ColorStateLists Wireless & Connectivity Hotspot 2.0 Improved Bluetooth Low Energy Scanning Android for Work Controls for Corporate-Owned, Single-Use devices Silent install and uninstall of apps by Device Owner Silent enterprise certificate access Auto-acceptance of system updates Delegated certificate installation Data usage tracking Runtime permission management Work status notification
4.1 Jelly Bean	16	99,8%		
4.2 Jelly Bean	17	99,2%		
4.3 Jelly Bean	18	98,4%		
4.4 KitKat	19	98,1%		
5.0 Lollipop	21	94,1%		
5.1 Lollipop	22	92,3%		
6.0 Marshmallow	23	84,9%		
7.0 Nougat	24	73,7%		
7.1 Nougat	25	66,2%		
8.0 Oreo	26	60,8%		
8.1 Oreo	27	53,5%		
9.0 Pie	28	39,5%		
10. Android 10	29	8,2%		

<https://developer.android.com/about/versions/marshmallow/android-6.0.html>

Supported Languages

Attualmente TripPic è disponibile solo in lingua italiana. Dal momento che si tratta di un'applicazione demo, non ancora pronta per la grande distribuzione, abbiamo ritenuto non necessario tradurla in altre lingue.

Tuttavia TripPic non contiene nella sua struttura molte componenti che andrebbero eventualmente tradotte. Le istruzioni che compaiono in italiano sono infatti davvero poche: si tratta di quelle presenti nei campi in cui inserire testo, ovvero nelle barre di ricerca e nel fragment Add per l'inserimento dei contenuti.

I bottoni di navigazione di TripPic sono contrassegnati da immagini (icone) che permettono all'utente di intuirne rapidamente l'uso.

Abbiamo ritenuto che la comunicazione mediante icone, fosse molto più immediata rispetto alle parole.

La scelta delle parole giuste per indicare le funzionalità di un bottone non è affatto semplice ed è indispensabile farlo in poco spazio. Per questo motivo un'icona opportunamente selezionata si rivela molto più efficace.

Le applicazioni più note fanno largo uso di icone, e queste hanno ormai assunto un significato ben preciso, abbiamo quindi deciso di utilizzare tali icone nella nostra applicazione.