UNIVERSITÀ DI PARMA

Dipartimento di Ingegneria e Architettura
Corso di Laurea in Ingegneria Informatica, Elettronica e Telecomunicazioni

Titolo Italiano

English Title

Relatore:
Prof. Relatore Michele Amoretti

Correlatore:
Ing. Nome Correlatore 1
Ing. Nome Correlatore 2 (opzionale)

Tesi di Laurea di:
Alex Spagni

ANNO ACCADEMICO 2021-2022

Ringraziamenti

Indice

Introduzione

Capitolo 1

Stato dell'arte

Il mondo mobile fa sempre più parte della nostra vita e ogni giorno vengono inventate sempre nuove applicazioni che permettono di semplificarla. Esistono app per poter ascoltare la musica, altre per poter eseguire delle transazioni bancarie ed altre ancora destinate alle domotica, per fare alcuni esempi.

Dato che siamo influenzati così tanto dal mondo mobile, viene spontaneo chiedersi quali sono le possibili strade da seguire per poter sviluppare un'app. Innanzitutto è necessario fare una distinzione tra app nativa e app multipiattaforma. Le prime sono delle applicazioni software che sono state sviluppate per funzionare su uno specifico tipo di dispositivo o piattaforma. Invece la seconda è un applicazione che può essere eseguita anche su sistemi operativi differenti ma usando sempre lo stesso codice. In particolare un app nativa sviluppata per un dispositivo Android non funzionarà su dispositivi iOS e viceversa un app sviluppata per un dispositivo iOS non funzionerà su dispositivi Android [1] [2].

1.1 App Multipiattaforma

React native è un Framework per lo sviluppo di app multipiattaforma che permette di sviluppare applicazioni sia per Android che iOS. Per poter utiliz-

zare questo Framework esistono due principali "strumenti": Expo[1] e React Native CLI[1]. Entrambi mettono a disposizione una serie di servizi per aiutare nello sviluppo. In particolare Expo è un Framework che estende React Native

1.1.1 React Native

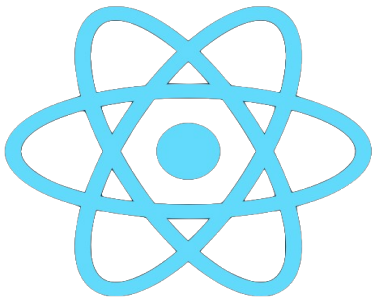

Figura 1.1

In React Native, proprio come in React, vengono costruiti dei componenti JSX, i quali combinano markup e il JavaScript, che lo controlla, in un unico file. A differenza del web non si ha una separazione tra grafica, presentazione e controllo in file diversi, in quanto JSX privilegia la separazione delle "concern" rispetto alla separazione delle tecnologie.

Il ciclo di aggiornamento è lo stesso di React: quando le "props" o gli "state" cambiano, React Native esegue un nuovo rendering delle viste.
La differenza principale tra React Native e React nel browser è che React Native sfrutta le librerie dell'interfaccia utente, della sua piattaforma ospitante, anzichè utilizzare il markup HTML e CSS.

React Native CLI permette di utilizzare React Native per sviluppare diversi tipi di applicazioni, tra cui:

- Prototipi
- Applicazioni multipiattaforma
- Applicazioni che non fanno largo uso di API native

- Applicazioni con una complessa User Interface (UI)
- Applicazioni per un determinato sistema operativo
- Applicazioni che non fanno uso intensivo di animazioni

Tra i vari vantaggi derivanti dall'utilizzo di questo "strumento", bisogna sottolineare che:

* Permette di includere moduli nativi scritti in Java, Kotlin e Object-C
* Consente di ridurre i tempi di sviluppo, in quanto permette di riutilizzare gran parte del codice scritto per le applicazioni web
* Mette a disposizione un gran numero di componenti pre-costruiti
* Fornisce un interfaccia utente semplice
* Permette di utilizzare plugin di terze parti
* Consente di realizzare una migliore User Interface rispetto ad Expo
* Presenta un architettura modulare la quale consente di semplificare lo sviluppo, il test e la manutenzione di programmi di grosse dimensioni

Mentre tra i vari svantaggi, si ha che:

* Per poter effettuare i test è necessario collegare il dispositivo al Pc
* Richiede Android Studio e XCode per l'esecuzione dei progetti
* Per poter condividere la propria app è necessario condividere l'intero file .apk
* La preparazione di un progetto richiedere tempo e non è così semplice come con Expo
* Richiede una conoscenza preliminare della strutturazione delle cartelle in Android e iOS

Una volta installata un'applicazione sviluppata in React Native sul proprio dispositivo, questa non eseguirà il render nel browser DOM, anzi andrà ad invocare le Object-C APIs, per eseguire il render su dispositivi IOS, e andrà ad invocare le Java APIs, per eseguire il render su dispositivi Android. L'utilizzo di tali API è possibile grazie al "bridge", il quale fornisce a React un'interfaccia con gli elementi della UI nativa della piattaforma ospitante.
In particolare i "React component" restituiscono il markup della loro funzione di rendering, il quale descrive il loro aspetto. Quando si utilizza React, per lo sviluppo web, questo si traduce direttamente nel DOM del broswer. Per React Native, il markup di un componente viene tradotto per adattarsi alla piattaforma host. Quindi una <View> potrebbe diventare una UIView specifica per iOS [3] [1].

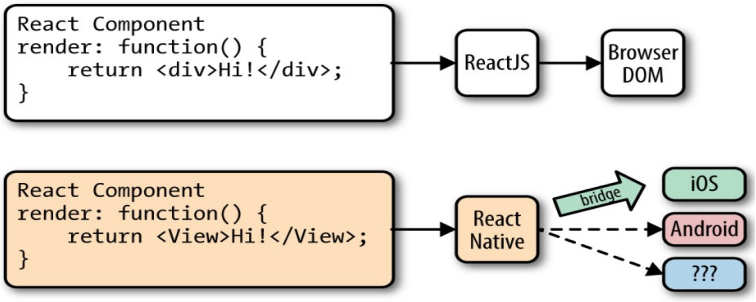

Figura 1.2

1.1.2 Expo

Expo è simile a React Native CLI, ma con uno svantaggio considerevole. In alcune applicazioni è necessario accedere alle API native che non sono disponibili su Javascript, per esempio le API per accedere ad Apple e Google play. In altri casi all'interno di un'applicazione si potrebbe voler riutilizzare le librerie scritte in Java, Object-C,Swift,Kotling o C++ che non sono state implementate in Javascript.

I moduli nativi permettono di esporre delle istanze di classi Java, Object-C e C++ come oggetti JavaScript. In questo modo è possibile eseguire codice

nativo in JavaScript.
Expo non permette di utilizzare i moduli nativi, quindi a meno che una certa libreria non sia già stata implementata in JavaScript, questa non potrà essere utilizzata in un'applicazione sviluppata tramite Expo. Questo non è l'unico svantaggio derivante dall'utilizzo di questo strumento. Alcune device API non sono supportate, come ad esempio il bluetooth.
A parte questo Expo presenta diversi vantaggi, tra cui [4] [1]:

* Una migliore gestione dei link
* Mette a disposizione una maggiore quantità di librerie standart, come ad esempio una libreria per le "Push Notification"
* Offre una migliore "Development Experience". Per poter testare un'applicazione, sviluppata tramite Expo, non è necessario avere un emulatore del dispositivo, su cui la si vuole testare, o collegare il proprio dispositivo al pc Expo permette di testare le applicazioni tramite Wi-fi, collegando il dispositivo e il pc alla stessa rete LAN
* Le applicazioni vengono aggiornate più velocemente rispetto a React Native CLI
* Supporta "l'Over to air update", quindi un applicazione si aggiorna nel momento in cui quest'ultima si sta aprendo sul dispositivo su cui è installata
* Non è necessario effettuare la build dell'applicazione per poterla testare
* È possibile "espellere" il progetto in ExpoKit e integrare del codice nativo, continuando ad utilizzare alcune funzionalità di Expo, ma non tutte

1.1.3 React Navigation

Sia che si utilizzi Expo o React Native CLI, aspetti molto importanti nello sviluppo di un'app sono la presentazione e la navigazione tra i vari "Screen".

Quest'ultimi rappresentano la parte di User Interface con la quale l'utente interagirà. La navigazione tra i vari screen viene gestita da un "Navigator". Il principale Navigator utilizzato sia da Expo che da React Native si chiama "React Navigation". Questa è una standalone library che permette di gestire lo spostamento tra i vari screen di cui l'applicazione è composta.

React Navigation fornisce principalmente quattro Navigator[5], ognuno dei quali consente ad un utente di passare da uno screen ad un altro:

- Native Stack Navigator: Ogni volta che un utente effettua una transizione il nuovo screen viene messo in cima allo stack. Il Native Stack Navigator utilizza le API native UINavigationController per le applicazione sviluppate per iOS e le API native Fragment per le applicazioni sviluppate per Android.
  In questo modo la navigazione tra i vari screen avrà lo stesso comportamento e le stesse prestazioni di applicazioni che usano la navigazione nativa. Nonostante le prestazioni fornite da questo Navigator siano ottime, non è particolarmente "customizzabile" come lo Stack Navigator.

- Stack Navigator: A differenza del Native Stack Navigator questo è incredilmente "customizzabile", poichè implementato in Javascript. Nonostante ciò non usando le primitive di navigazione native presenta prestazioni peggiori. Va comunque detto che per la maggior parte delle applicazioni questa differenza di performance tra il Native Stack Navigator e lo Stack Navigator non è percebile.

- Drawer Navigator: La differenza principale tra questo Navigator e gli altri è che quest'ultimo viene rappresentato con un icona formato da tre linee orizzontali, posta in alto a sinistra della visualizzazione. Attraverso quest'ultima è possibile vedere tutti gli screen su cui si può navigare.

- Tab Navigator: Questo Navigator viene rappresentato tramite una TabBar posta nella parte bassa dello schermo. Essa mostra gli screen verso

cui si può navigare. Questo Navigator è uno dei più utilizzati nelle applicazioni basate su Tab-Bar.

1.1.4 Redux

Al giorno d'oggi le single-page application devono gestire sempre più stati, ed è per questo che nella maggior parte delle app si utilizza Redux. Questa è una libreria molto leggera che aiuta a sviluppare applicazioni che si comportano in modo coerente, che vengono eseguite in ambienti diversi (client e server) e che sono facili da testare. Si dice che Redux permetta di realizzare un "contenitore a stato" prevedibile, per le applicazioni JavasScript.

Questa libreria è completamente compatibile con React Native, attraverso l'utilizzo di React Redux, il quale rappresenta il livello di binding ufficiale di React UI per Redux. Questo livello consente ai "React component" di leggere i dati da un archivio e di inviare azioni a quest'ultimo per aggiornarne lo stato.
In particolare, Redux:

- Permette di realizzare uno store che contiene lo stato globale
- Consente di emettere azioni che aggiornano lo stato dello store. Queste vengono notificate all'archivio nel momento in cui avviene un evento nell'applicazione
- Presenta delle funzioni riduttrici pure che esaminano le azioni e restituiscono uno stato immutabile aggiornato

Quando si parla di "stati" da gestire all'interno di un'app, si fa riferimento, ad esempio: alle risposte di un server, a dati salvati nella cache o creati locamente e che non sono memorizzati in modo persistente. Redux è in grado di gestire tutti questi stati e permette di rendere la mutazione dello stato globale prevedibile, attraverso l'imposizione di alcune restrizioni, su quando e come lo stato può essere modificato.

Queste restrizioni possono essere riassunte nei tre principi cardine di Redux[6]:

1. Lo stato globale dell'applicazione è memorizzato in un albero di oggetti all'interno di un singolo store. In questo modo è possibile realizzare app universali, permettere un miglior debugging dell'applicazione e rendere lo stato globale persistente in modo da avere un ciclo di sviluppo più rapido.

2. Lo stato è Read Only. Questo assicura che né le "viste" né le callback di rete scrivano direttamente sullo stato. Al contrario, esprimono l'intenzione di trasformare lo stato.
   Dato che tutte le modifiche sono centralizzate e avvengono una alla volta in un ordine rigoroso, non vi sono condizioni di concorrenza a cui dover prestare attenzione. Le azioni per aggiornare lo stato sono semplici oggetti JavaScript, quindi possono essere registrate, serializzate, memorizzate e successivamente riprodotte per scopi di debug o di test.

3. Lo stato viene aggiornato da funzioni pure, le quali sono chiamate riduttori. Queste prendono lo stato precendete, un'azione e restituiscono lo stato successivo. In particolare i riduttori devono ritornare nuovi oggetti di stato e non modificare lo stato precendente.

1.2 App Native

Come già menzionato all'inizio di questo capitolo, le applicazioni native sono sviluppate per uno specifico dispositivo o piattaforma ed è quindi necessario un linguaggio di programmazione differente da un sistema operativo all'altro. iOS fa uso dei linguaggi Object-C e Swift, mentre Android sviluppa app native in Java e Kotlin.

1.2.1 Kotlin

Kotlin è un linguaggio sviluppato da JetBrains a partire dal 2010 e successivamente reso open source nel 2012. È definito come general purpose, free, open source e pragmatico. Pragmatico in quanto cerca di distinguersi dalla struttura chiusa e ben definita di Java al fine di permettere uno sviluppo più snello e veloce, grazie soprattutto al fatto che fornisce diversi "shortcut".

Koltin nel 2019 è diventato, secondo Google, il linguaggio più utilizzato per lo sviluppo di applicazioni Android, sostituendo Java [7]. Nonostante lo si utilizzi per lo sviluppo di applicazioni Android native, quest'ultimo permette anche la realizzazione di applicazioni per iOS e di applicazioni web, in quanto è possibile transpilare il codice scritto in Kotlin in codice JavaScript [8].

Kotlin consente di realizzare applicazioni seguendo il paradigma Object Oriented (OOP) o il paradima della programmazione funzionale. Nel primo caso si fa uso delle classi, dell'ereditarietà e del polimorfismo, proprio come in Java. Nel secondo caso il programma si basa sulla valutazione di funzioni matematiche. In questo modo è possibile ottenere il meglio dei due mondi[9].

Un ambiente di esecuzione per le applicazioni scritte tramite questo linguaggio, è la JVM (Java Virtual Machine). Questo ambiente di esecuzione consente di utilizzare Kotlin in qualsiasi contesto in cui viene utilizzato Java. Inoltre, Kotlin è al 100% interoperabile con quest'ultimo. Il vantaggio di questo alto livello di interoperabiltà è che si possono riutilizzare le librerie Java esistenti[10].

Kotlin e Java sono molto simili tra loro, ed entrambi possono essere utilizzati per lo sviluppo di applicazioni per Android. Nonostante ciò presentano alcune differenze[11]:

- Il codice scritto in Kotlin risulta essere più conciso di quello scritto in Java, in quanto riduce la quantità di "BoilerPlate".
- Un'applicazione sviluppata in Kotlin risulta essere più facile da leggere e presenta meno errori dovuti allo sviluppatore.

- Java consente di assegnare ad una variabile il valore "Null", il quale può generare un errore "NullPointerException", nel caso in cui la variabile venisse acceduta.
  Invece Kotlin applica la cosiddetta "Null Safety", cioè non permette di assegnare ad una variabile il valore "Null". In questo modo il codice risulta essere più stabile.

- Kotlin non è un linguaggio fortemente tipizzato come invece lo è Java. Infatti vengono utilizzate solo due parole chiave per la definizione delle variabili. Inoltre, solo nel momento in cui ad una variabile viene assegnato un valore, questa viene identificata come una stringa, un numero o un booleano. Lo stesso meccanismo ricorre anche in JavaScript.

- Una differenza fondamentale tra i due è che Java supporta solo la programmazione ad oggetti, mentre invece Kotlin oltre a quest'ultima supporta anche la programmazione funzionale.

- Java presenta un tempo di compilazione inferiore rispetto a Kotlin. Nel caso di piccole applicazioni si stima che Java sia il 15-20% più veloce rispetto al tempo necessario per compilare la stessa applicazione scritta in Kotlin. Se però consideriamo app di dimensioni maggiori, allora il tempo di compilazione è circa lo stesso.

Secondo alcuni studi solo i neofiti dello sviluppo Android continuano a sviluppare applicazioni in Java, poiché la maggior parte della documentazione e degli esempi sono in Java.
Altri studi dimostrano che il tempo medio impiegato da uno sviluppatore Java per imparare Kotlin è di poche ore. Dopo questa transizione si stima una riduzione del 40% del numero di linee di codice da Java a Kotlin[9].

1.2.2 Swift

Swift è un potente linguaggio di programmazione che permette di creare applicazioni sia per dispotivi mobili come cellulari, IPad e Apple Watch, ma anche per dispositivi come Mac e Apple Tv. Essendo un linguaggio general-purpose, proprio come Kotlin, può essere usato anche per sviluppare applicazioni web e web service. Inoltre, è possibile realizzare applicazioni server che necessitano di "runtime security" e ingombro di memoria ridotto.

Swift è stato realizzzato per essere un linguaggio veloce. Utilizzando la tecnologia di compilazione LLVM il codice Swift viene trasformato in codice macchina ottimizzato che sfrutta al meglio l'hardware moderno.

Questo linguaggio di programmazione è il successore dei linguaggi C e Object-C, in quanto include sia primitive di basso livello come tipi, controllo di flusso e operatori ma fornisce anche funzionalità orientate agli oggetti come classi, protocolli e i tipi generici. Swift ha eliminato intere classi di codice "non sicuro". Le variabili devono sempre essere inizializzate prima di essere utilizzate, gli indici degli array e gli interi vengono sempre controllati per verificare se vi sono errori di overflow [12].

Questo linguaggio presenta diverse caratteristiche, tra cui[13]:

- Supporta i tipi generici
- Le funzioni possono ritornare valori multipli e tuple
- Presenta un iterazione rapida e concisa su un intervallo o un insieme
- Presenta modelli di programmazione funzionale, ad esempio "map" e "filter"
- Presenta una gestione degli errori integrata.

Tramite Swift la memoria viene gestita automaticamente utilizzando l'ARC (Automatic Reference Counting). Esso permette di mantenere l'utilizzo della memoria al minimo e senza l'onere del Garbage Collection [14]. Si ricorda che in Java, il Garbage Collection si occupa di tenere traccia delle allocazioni

di memoria utilizzate e le libera solo quando non sono più impiegate. Esso è un vero e proprio thread in esecuzione parallelamente al programma e anche se la sua priorità è minima, quindi viene avviato solo quando non ci sono altri thread attivi, è un processo da considerare in termini di tempo di CPU.

Swift è open source e cross-platform infatti può essere usato sia su piattaforme Apple che Linux. Presenta anche un package manager, il quale è un unico strumento multipiattaforma per costruire, eseguire, testare e pacchetizzare le librerie e gli eseguibili Swift. Swift Package Manager stesso è costruito con Swift e incluso nel progetto Open Source Swift come pacchetto.

Tramite Questo linguaggio è possibile creare una nuova applicazione oppure utilizzare il codice Swift per implementare nuove caratteristiche e funzionalità nelle proprie applicazioni. Il codice Swift coesiste con i file Object-C esistenti nello stesso progetto, con pieno accesso alle API Object-C[12].
Anche se Swift è interoperabile con Object-C, esso presenta alcuni vantaggi rispetto a quest'ultimo[15]:

* Secondo Apple Swift può essere 2.6 volte più veloce di Object-C
* In Swift possono essere create delle variabili senza doverne definire prima il tipo
* Non è necessario inserire i punti e virgola alla fine di ogni riga di codice scritta in Swift
* Il codice scritto in Swift viene scirtto più velocemente rispetto al codice scritto in Object-C, poichè Swift è stato progettato per essere "Developer-Friendly"

1.2.3 Swift vs Kotlin

Sia Kotlin che Swift sono linguaggi di programmazione che vengono usati per lo sviluppo di app native, ed entrambi sono multipiattaforma. Nonostante ciò presentano alcune differenze, di seguito elencate[16]:

- Lo sviluppo in Kotlin non si limita ad un particolare IDE o OS. Infatti è possibile scrivere il proprio codice Kotlin in qualsiasi IDE o OS come VS Code, Atom, Windows, Linux e Mac.
  Al contrario lo sviluppo in Swift è possibile solo all'interno di XCode perché solo attraverso di esso si è in grado di compilare il codice Swift Questo IDE è disponibile solo su Mac, quindi non sarà possibile sviluppare app per IOS senza averne uno

- Kotlin e Swift approcciano la gestione della memoria in modo differente. In particolare Kotlin, proprio come Java, affronta la gestione di quest'ultima dal punto di vista del Garbage Collection.
  All'contrario Swift fa uso dell'ARC (Automatic Reference Counting). Attraverso di esso le applicazioni sviluppate tramite Swift tendono ad essere più efficienti e prive di bug

- Entrambi, all'interno di un progetto possono coesistere con i rispettivi predecessori. Swift è al 100% interoperabile con Object-C, mentre Kotlin è al 100% interoperabile con Java

Una caratteristica comune ad entrambi è che permettono di ridurre la quantità di "BoilerPlate", necessaria sia nei progetti Java che in quelli Object-C.

1.3 App Native vs App Multipiattaforma

Quando si decide di sviluppare un'applicazione per un dispositivo mobile, come può essere un telefono, è necessario capire se si vuole sviluppare un'app nativa o un'app multipiattaforma. La scelta può essere fatta sulla base di cinque fattori: prestazioni, tempo di sviluppo, sicurezza, User Experience/Interface e stabilità[17].

Per fornire le massime prestazioni agli utenti, la scelta più saggia che si può fare è scegliere un approccio nativo, poiché le app native sono sviluppate tenendo conto dei requisiti specifici della piattaforma di riferimento. Esse

sono compilate per una specifica serie di dispositivi ed eseguite per una precisa architettura. Ciò che consente alle app native di essere più efficienti è l'accesso ad API e componenti esclusivi, ottimizzati per diverse dimensioni di schermo e versioni di sistema. Questa tipologia di app può presentare dimensioni minori rispetto alle app multipiattaforma, consentendo di occupare meno memoria sul dispositivo.

Per quanto riguarda i tempi di sviluppo, le app multipiattaforma presentano uno produzione più rapida. Grazie alla riusabilità del codice tra le piattaforme, un gruppo di sviluppatori non necessita di implementare progetti separati per sistemi operativi diversi.

Dal punto di vista della sicurezza, le app native sono più sicure rispetto alle app multipiattaforma, poiché sono dotate di molteplici funzioni di sicurezza incorporate. Per gli sviluppatori di applicazioni native è solitamente più facile implementare la crittografia dei file, il rilevamento inteligente delle frodi e altre funzioni di sicurezza attraverso le opportune librerie e risorse di ogni piattaforma.
L'aggiornamento delle misure di sicurezza nelle app native richiede meno tempo rispetto alle app multipiattaforma. In quest'ultime è più difficile prevedere quando i framework multipiattaforma saranno aggiornati.

Se l'obiettivo è quello di realizzare la miglior User Experience o User Interface per la propria applicazione, allora è necessario privilegiare un approccio nativo. Le applicazioni native offrono migliori funzionalità dell'interfaccia utente, poiché dispongono di librerie e componenti preimpostati e personalizzabili.

Se l'intento è quello di sviluppare un applicazione stabile nel lungo tempo, allora bisogna adottare un approccio nativo. Dato che Android e IOS provengono rispettivamente da Google e Apple, è certo che queste aziende continueranno a supportare e migliorare i loro sistemi operativi mobili. Questo significa che le app native beneficeranno di stabilità in termini di manutenzione e aggiornamenti.
D'altra parte, dato che i framework multipiattaforma sono creati da azien-

de, organizzatori e comunità di sviluppatori di terze parti, c'è il rischio che l'aggiornamento o lo sviluppo di questi framework possa essere incoerente o interrotto.

Capitolo 2

Architettura Funzionale del Sistema Realizzato

2.1 Obiettivi dell'applicativo

Lo scopo principale dell'applicativo sviluppato è quello di fornire un servizio per la visualizzazione delle immagini scattate dai diversi Rover inviati su Marte nel corso degli ultimi anni.
Queste immagini possono essere visualizzate sul proprio dispositivo mobile con sistema operativo Android o iOS, tramite una applicazione che può essere scaricata da un sito web online.

L'applicazione dovrà fornire una funzione di ricerca, tramite la quale si potranno ricercare immagini inserendo il nome di uno dei Rover oppure digitando la data solare in cui sono state scattate le immagini che si vogliono osservare. In particolare, i nomi dei Rover per i quali si potrà effettuare una ricerca sono: Curiosity,Spirit e Opportunity. Tra una ricerca e l'altra verrà mostrata una schermata di "loading" in modo da indicare all'utente che si sta eseguendo il "fetching" delle immagini.

Oltre a poter ricercare le immagini secondo diversi parametri, l'applicazione dovrà anche fornire la possibilità di nascondere alcune di queste in modo che non vengano visualizzate dall'utente in una ricerca futura. Nonostante

le immagini nascoste non vengano presentate all'utente quando effettua una ricerca, gli verrà sempre concessa la possibilità di ripristinarle.

L'utente potrebbe non essere interessato ad un'intera categoria di immagini, quindi gli verrà fornita la possibilità di nascondere tutte le foto relative ad una sua ricerca.

Ogni volta che verrà chiusa l'applicazione, le ultime immagini visualizzate dovranno essere memorizzata in modo persistente così che possano essere mostrate all'ultilizzatore quando riaprirà l'app.

Oltre a poter visualizzare le immagini in una lunga lista nell'Index Screen dell'applicazione, quest'ultima dovrà fornire anche la possibilità di visualizzare i dettagli di ognuna di esse.

Al primo avvio l'utente dovrà essere ridirezionato verso una schermata di login o di registrazione a seconda che sia in possesso o meno delle credenziali di accesso.

2.2 Requisiti dell'applicativo

2.2.1 Requisiti necessari per il funzionamento dell'applicativo

L'applicazione necessita di una connessione ad Internet in modo da collegarsi tramite il web ai server della Nasa e recuperare le immagini ricercate da ogni utente.

Le credenziali di ogni utente sono memorizzate in un database distribuito che memorizza i dati in documenti flessibili JSON-like. In particolare, per assicurare la sicurezza di ogni utilizzatore, le password di ognuno di essi vengono memorizzate solo dopo essere state cifrate. In questo modo anche se qualcuno dovesse penetrare nel database non riuscirebbe ad accedere ai vari account.

Il database viene consultato attraverso un server locale che espone delle API all'applicativo per far sì che ogni utente possa eseguire la registrazione

e il login. Affinché l'applicazione possa comunicare con il server locale, deve essere attiva un'applicazione Multipiattaforma chiamata Ngrok[1].

Per garantire l'efficienza dell'applicazione il numero di immagini recuperate, ad ogni richiesta GET ai server della Nasa, deve essere pari a 25. In questo modo si evita di sovraccaricare il client che deve elaborare le immagini e presentarle a schermo.

Di seguito si può osservare una rappresentazione grafica dell'interazione tra i vari sistemi:

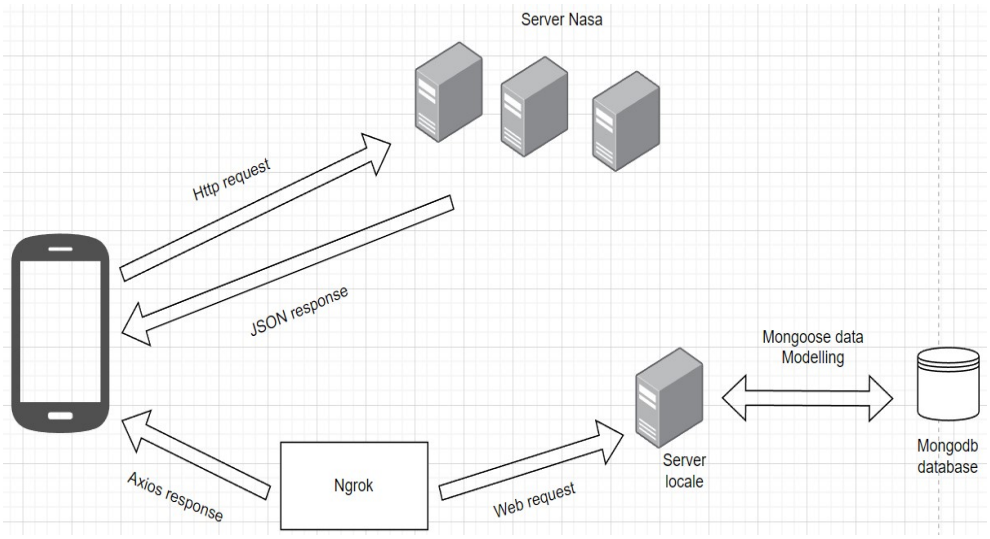

Figura 2.1

2.2.2 Requisiti funzionali

Ad ogni requisito viene associato un nome mnemonico, una breve descrizione e un codice di priorità in modo tale da indicare quali requisiti devono essere implementati per primi e quindi definire una gerarchia di implementazione.

[1]Ngrok consente di aprire una connessione diretta verso le API sviluppate in Express e fornisce un URL tramite il quale il dispositivo mobile può usufruirne. In particolare, espone le porte su cui i server locali sono in ascolto ad Internet.

Nome	**Descrizione**	**Priorità**
Sign in	L'utente deve poter accedere all'applicazione tramite le sue credenziali	Primario
Sign up	L'utente deve potersi registrare tramite la propria e-mail e password in modo da poter accedere all'applicazione	Primario
Search by name	L'utente deve poter ricercare le immagini tramite il nome di uno dei tre possibili Rover: Spirit, Opportunity e Curiosity	Primario
Search by date	L'utente deve poter ricercare le immagini che sono state scattate in una precisa data solare	Primario
Hide image	L'utente deve poter nascondere le immagini che gli vengono presentate	Primario
Restore images	L'utente deve poter ripristinare le immagini che ha nascosto	Secondario
Logout	L'utente deve poter uscire dal proprio profilo	Secondario
Hide all	L'utente deve poter nascondere nascondere tutte le ommagini relative ad una sua ricerca	Secondario
Privacy and Policy	L'utente deve poter conoscere quali condizioni e termini ha accettato scaricando l'applicazione	Secondario

2.2.3 Requisiti non funzionali

In questa sezione verranno descritti i principali requisiti non funzionali necessari per il buon funzionamento dell'applicazione.

Nome	**Descrizione**
Prestazioni	L'applicativo deve essere in grado di presentare le immagini entro 6 secondi dall'avvio della ricerca
Portabilità	L'applicativo deve poter essere scaricabile sia su dispositivi iOS che Android
Manutenibilità	L'applicativo deve essere strutturato seguendo un architettura modulare in modo da facilitare la fase di test e di modifica

2.3 Casi d'uso

In questa sezione verranno illustrati i pricipali casi d'uso.

Caso d'uso U1: Sign in

- **Use case overview:** l'applicazione deve consentire agli utenti la possibilità di poter accedere al proprio account.
- **Actors:** end user, server locale.
- **Triggers:** si preme sul pulsante "Sign in".
- **Pre-Condition:** l'utente non deve aver già effettuato l'accesso all'applicazione oppure si deve trovare nell'Index Screen dove può effettuare il logout.

- **Post-condition:** si verrà indirizzati nell'Index Screen dell'applicazione.
- **Main Flow:**
  1. L'utente ha appena terminato di scaricare l'applicazione e la apre, ritrovandosi come prima schermata lo "Splash screen". Una volta premuto il pulsante "Explore" verrà reindirizzato sullo screen destinato al log in.
  2. Arrivato allo screen di log in l'utente inserirà la propria e-mail e password e premerà sul pulsante "Sign in".
  3. Verrà inviata una richiesta al server locale che andrà a confrontare le credenziali inserite in precedenza con quelle presenti sul database per verificare che coincidano. In caso affermativo l'utente verrà indirizzato sull'Index Screen.
- **Alternative Flow:**
  1. L'utente si trova nell'Index Screen e sta scorrendo tutta la lista di immagini a lui presentate. Una volta arrivato alla fine della lista premerà sul pulsante di "logout".
  2. L'utente si ritroverà nello screen riservato al log in dove inserirà la propria mail e password e premerà sul pulsante "Sign in".
  3. Verrà inviata una richiesta al server locale che andrà a confrontare le credenziali inserite in precedenza con quelle presenti sul database per verificare che coincidano. In caso affermativo l'utente verrà indirizzato sull'Index Screen.
- **Exception Flow:**
  1. Una volta inserite le proprie credenziali verrà inviata una richiesta al server locale che andrà a confrontare quest'ultime con quelle presenti sul database.

2. Le credenziali inserite dall'utente non sono corrette e gli viene presentato un messaggio di errore.

Caso d'uso U2: Sign up

- **Use case overview:** l'applicazione deve consentire agli utenti la possibilità di potersi registrare tramite la propria e-mail e password.
- **Actors:** end user, server locale.
- **Triggers:** si preme sul pulsante "Sign up".
- **Pre-Condition:** l'utente non deve aver già effettuato l'accesso all'applicazione.
- **Post-condition:**si verrà indirizzati nell'Index Screen dell'applicazione.
- **Main Flow:**
  1. L'utente ha appena terminato di scaricare l'applicazione e la apre, ritrovandosi come prima schermata lo "Splash screen". Una volta premuto il pulsante "Explore" verrà reindirizzato sullo screen destinato al log in. Non avendo alcune credenziale valida l'utente dovrà premere sul pulsante "Sign up" in modo da essere reindirizzato nello schermo dedicato alla registrazione.
  2. Arrivato allo screen di registrazione l'utente inserirà la propria e-mail e password e premerà sul pulsante "Sign up".
  3. Verrà inviata una richiesta al server locale che andrà a confrontare che l'e-mail inserita dall'utente non sia già presente nel database. In caso affermativo l'utente verrà indirizzato sull'Index Screen.
- **Exception Flow:**
  1. Una volta inserite le proprie credenziali verrà inviata una richiesta al server locale che andrà a confrontare che l'e-mail inserita dall'utente non sia già presente nel database.

2. Esiste già un utente registrato con quella e-mail, quindi il sistema restituisce un messaggio di errore.

Caso d'uso U3: Ricerca immagini tramite nome Rover

- **Use case overview:** l'applicazione deve consentire agli utenti la possibilità di ricercare le immagini in base al nome del Rover.
- **Actors:** End user, server Nasa.
- **Triggers:** si digita il nome di uno dei tre Rover nella Search Bar e si preme "fatto" sulla propria tastiera.
- **Pre-Condition:** l'utente deve aver effettuato l'accesso oppure essere in possesso di un JSON Web Token.
- **Post-condition:** verranno presentate delle nuove immagini o una modale che informa l'utente che nessuna immagine è stata trovata.
- **Main Flow:**
  1. L'utente si trova nella schermata principale e digita il nome del Rover che ha scattato le immagini richieste. Per avviare la ricerca si dovrè premere sul pulsante "All".
  2. Viene effettuata una richiesta HTTP verso i server della Nasa i quali rispondono con un oggetto JSON che contiene tutte le foto che sono state trovate.
  3. Le immagini ottenute vengono inserite all'interno di una lista e vengono visualizzate a schermo.
- **Alternative Flow:**
  1. L'utente si trova nella schermata principale e digita il nome del Rover che ha scattato le immagini le immagini richieste. Per avviare la ricerca si dovrè premere sul pulsante "All".

2. Viene effettuata una richiesta HTTP verso i server della Nasa i quali rispondono con un oggetto JSON che contiene tutte le foto che sono state trovate.
3. Il nome del Rover inserito non è corretto. L'utente viene informato che non è stata trovata alcuna foto e gli viene presentata una lista di immagini vuota.

Caso d'uso U4: Ricerca immagini per data solare

- **Use case overview:** l'applicazione deve consentire agli utenti la possibilità di ricercare le immagini in base a una certa data solare.
- **Actors:** end user, server Nasa.
- **Triggers:** si preme sul pulsante "Search by date".
- **Pre-Condition:** l'utente deve aver effettuato l'accesso oppure essere in possesso di un JSON Web Token.
- **Post-condition:** verranno presentate delle nuove immagini o una modale che informa l'utente che nessuna immagine è stata trovata.
- **Main Flow:**
  1. L'utente si trova nell'Index Screen e preme il pulsante "Data" in modo da spostarsi nello schermo dedicato alla ricerca per data solare. Dovrà quindi inserire giorno, mese e anno delle foto che vuole ricercare.
  2. Viene effettuata una richiesta HTTP verso i server della Nasa i quali rispondono con un oggetto JSON che contiene tutte le foto che sono state trovate.
  3. Le immagini ottenute vengono inserite all'interno di una lista e vengono visualizzate a schermo.
- **Alternative Flow:**

1. L'utente si trova nell'Index Screen e preme il pulsante "Data" in modo da spostarsi nello schermo dedicato alla ricerca per data solare. Dovrà quindi inserire giorno, mese e anno delle foto che vuole ricercare.
2. Viene effettuata una richiesta HTTP verso i server della Nasa i quali rispondono con un oggetto JSON che contiene tutte le foto che sono state trovate.
3. Nella data solare inserita dall'utente non è stata scattata alcuna foto. Si viene quindi informati che non sono state trovate immagini e quindi non viene presentata alcuna foto.

Caso d'uso U5: Nascondere un immagine

- **Use case overview:** l'applicazione deve consentire agli utenti la possibilità di nascondere le immagini mostrate in modo che non vengano visualizzate in futuro.
- **Actors:** end user.
- **Triggers:** si preme sul pulsante "Hide this image".
- **Pre-Condition:** l'utente deve aver effettuato l'accesso oppure essere in possesso di un JSON Web Token.
- **Post-condition:** l'immagine selezionata verrà nascosta.
- **Main Flow:**
  1. L'utente si trova nell'Index Screen e premendo su un'immagine accede ai dettagli di quest'ultima.
  2. Premendo sul tasto "Hide this image" verrà nascosta l'immagine in questione e verrà notificato all'utente un messaggio di conferma. A questo punto si verrà reindirizzati sull'Index Screen.

- **Alternative Flow:**

    1. L'utente si trova nell'Index Screen e premendo su un'immagine accede ai dettagli di quest'ultima.
    2. L'utente decide di non nascondere l'immagine in questione e ritorna sull'Index Screen premendo l'icona con la "X".

Caso d'uso U6: Ripristinare le immagini nascoste precedentemente

- **Use case overview:** l'applicazione deve consentire agli utenti la possibilità di vedere anche le immagini nascoste in precedenza.
- **Actors:** end user.
- **Triggers:** si preme sul pulsante "Photos".
- **Pre-Condition:** l'utente deve aver effettuato l'accesso oppure essere in possesso di un JSON Web Token.
- **Post-condition:** in base ai parametri di ricerca impostati dall'utente verranno mostrate tutte le immagini, anche quelle nascoste.
- **Main Flow:**

    1. L'utente si trova nell'Index Screen e preme il pulsante "Photos" il quale passa dal colore grigio ad azzurro in modo da indicare che è stato abilitato.
    2. Oltre alle immagini che l'utente stava già visualizzando, vengono mostrate anche quelle che aveva nascosto in precedenza. In particolare vengono ripristinate solo le foto nascoste che soddisfano i criteri di ricerca dell'utente.

- **Alternative Flow:**

  1. L'utente si trova nell'Index Screen e preme il pulsante "Photos" il quale passa dal colore azzurro a grigio in modo da indicare che è stato disabilitato.
  2. Tutte le immagini che l'utente aveva nascosto in precedenza vengono nuovamente occultate. Ora l'utente vedrà solo le foto che soddisfano i suoi parametri di ricerca e che non sono state nascoste.

Capitolo 3

Implementazione

Capitolo 4

Risultati

Conclusioni

Bibliografia

[1] fulcrum. React native cli vs expo. https://fulcrum.rocks/blog/react-native-init-vs-expo#:~:text=to%20get%20started.-,What%20is%20Expo,testing%20of%20React%20Native%20app, 2022.

[2] Corael srl. Differenza tra app nativa e app ibrida. https://corael.it/qual-e-la-differenza-tra-app-nativa-e-app-ibrida/?gclid=Cj0KCQjwgO2XBhCaARIsANrW2X2GyVHY_r_m3okI4e8Kdy-KShS7Ftj0sZf30GWkEAD5f4R7_KFGfL4aAlIiEALw_wcB, 2021.

[3] Bonnie Eisenman. *Learning React Native*, chapter Working with React Native. O'Reilly Media, 2016.

[4] Meta. React native documentation. https://reactnative.dev/docs/native-modules-intro, 2022.

[5] Expo, Software Mansion, and Callstack. React navigation documentation. https://reactnavigation.org/docs/getting-started/, https://reactnavigation.org/docs/stack-navigator, 2022.

[6] Dan Abramov and the Redux documentation authors. Redux and react redux documentation. https://redux.js.org/introduction/getting-started, https://redux.js.org/understanding/thinking-in-redux/three-principles, https://redux.js.org/understanding/thinking-in-redux/motivation,

https://react-redux.js.org/introduction/getting-started, 2022.

[7] Giuneco S.r.l. Applicazioni native android e ios. https://tech.giuneco.it/applicazioni-native-android-e-ios-parte-1/, 2021.

[8] Jet Brains. Kotlin documentation. https://kotlinlang.org/docs/android-overview.html, 2022.

[9] Martin Heller. What is kotlin? the java alternative explained, 2020. https://www.infoworld.com/article/3224868/what-is-kotlin-the-java-alternative-explained.html#:~:text=Kotlin%20is%20a%20general%20purpose,%2C%20clarity%2C%20and%20tooling%20support.

[10] Dawn griffiths. *Head First Kotlin*, chapter A quik Dip. O'Reilly Media, 2019.

[11] Anshul Bansal. Java vs. kotlin. https://www.baeldung.com/kotlin/java-vs-kotlin#:~:text=Java%20is%20a%20strictly%20typed,type%20of%20the%20assignment%20value, 2021.

[12] Apple. Swift overview. https://developer.apple.com/swift/#open-source, 2022.

[13] Apple. Swift about. https://www.swift.org/about/, 2022.

[14] Massimo Carli. Il garbage collector di java. http://www.di-srv.unisa.it/~ads/corso-security/www/CORSO-9900/java/mirror/mokabyte/garbagecollector.htm, 1998.

[15] Code Academy. Kotlin vs swift. https://www.codecademy.com/resources/blog/kotlin-vs-swift/, 2021.

[16] Lerma Gray. Kotlin vs swift. https://devcount.com/kotlin-vs-swift/, 2022.

[17] Satinder Singh. Natie app e multiplatform app. https://www.netguru.com/blog/cross-platform-vs-native-app-development, 2021.