UNIVERSITÀ POLITECNICA DELLE MARCHE
FACOLTÀ DI INGEGNERIA

Corso di Laurea in Ingegneria Informatica e dell'Automazione

Sviluppo di un'app per il monitoraggio dell'inquinamento della costa marchigiana – Interfacciamento lato server e presentazione dei dati lato utente.

Development of an app for monitoring the pollution of the Marche coast – Server-side interfacing and user-side data presentation.

Relatore:
Prof. Emanuele Storti

Tesi di Laurea di:
Cristian Di Silvestre

A.A. 2020/2021


AQUA

“Gioca, non stare a guardare.”

Robert Baden-Powell

Sommario

Introduzione

Robert Baden-Powell, fondatore del movimento scout mondiale, scrisse:

"Nel vostro passaggio in questo mondo, che ve ne accorgiate o no, chiunque voi siate e dovunque andiate, state lasciando dietro di voi una traccia.
[...]
e quindi, volgendo i propri passi nella giusta direzione, potete indirizzare bene anche coloro che vi seguono."[1]

Soffermandosi un momento a riflettere sulle tracce che stiamo lasciando alle generazioni future, risulterebbe difficile non notare come, oltre alle innovazioni tecnologiche, frutto del progresso scientifico, ci sia l'eredità di mondo ridotto ormai allo stremo.
Ciò è dovuto principalmente all'agire dell'uomo, che, per perseguire l'idea di benessere diffuso nella società odierna, sembra essersi dimenticato della necessità di rispettare e salvaguardare il pianeta.
Tra gli effetti più evidenti di tale noncuranza, ci sono sicuramente l'inquinamento e il surriscaldamento globale.
Tali problematiche ancora oggi sono fortemente sottovalutate dalla maggior parte della popolazione mondiale e, sebbene stiano sorgendo ogni giorno nuove iniziative di sensibilizzazione, monitoraggio e prevenzione di tali effetti del cambiamento climatico, esse sono tutt'ora insufficienti.
Uno dei fenomeni di maggior rilievo, che affligge anche le coste italiane, è certamente quello relativo all'aumento di temperatura media dell'acqua. Tale cambiamento favorisce la prolificazione dell'alga tossica *Ostreopsis* cf. *Ovata*.
Oltre a questa problematica, vi è un secondo fattore che favorisce lo sviluppo di tale alga nel fondale marino: la presenza nell'ecosistema acquatico di dosi troppo elevate di sostanze nutritive contenenti azoto, zolfo e fosforo provenienti da fonti naturali ed artificiali, come fertilizzanti ed alcuni tipi di detersivi.

1 "*Blazing the Trail: Being Wise Saws and Modern instances from the Works of the Chief Scout*", Raccolta di articoli di Baden-Powel curato da Laura Holt, Londra, C. Arthur Pearson Ltd., 1923, pp. 63.

La presenza dell'Ostreopsis cf. Ovata nel substrato sabbioso o roccioso marino provoca la morte di tutti gli organismi che vi vivono. Infatti, prolificando, esaurisce l'ossigeno a disposizione di animali come stelle marine e ricci di mare dei quali è possibile constare una brusca diminuzione.
Gli effetti della fioritura dell'alga tossica sono riscontrabili anche nell'essere umano; infatti, l'inalazione di aereosol marino (microparticelle acquose in sospensione) contenente l'alga porta a sintomi di diversa natura tutti legati all'irritazione delle mucose respiratorie e congiuntivali: rinorrea (raffreddore), difficoltà respiratorie e febbre.
La costa marchigiana non è esente da tale problematica: dal settembre del 2008 si verifica annualmente la presenza di *Ostreopsis* cf. *Ovata* lungo tale tratto costiero, tale evento causa ogni estate l'accesso di numerose persone presso le strutture ospedaliere. [1]
L' Agenzia regionale per la protezione ambientale delle Marche (ARPAM) monitora da anni questa situazione. Valutando però le modalità di presentazione dei dati raccolti nelle rilevazioni settimanali, è possibile notare come tali informazioni, riguardanti la qualità dell'acqua che bagna le varie spiagge marchigiane, hanno una bassa visibilità.
Infatti, la consultazione dei dati è limitata al sito web di ARPAM e al suo interno essi sono presentati con una modalità che rende la loro visione poco confortevole, soprattutto se effettuata attraverso uno smartphone.
Nato dall'idea dei ricercatori del Dipartimento di Scienze della Vita e dell'Ambiente (DISVA) dell'Università Politecnica delle Marche, nello specifico nella figura del Prof. Accoroni Stefano, "AQUA" è un progetto che si pone come scopo quello di sopperire a tale problematica.
In seguito ad un colloquio con il professor Storti, relatore di tale progetto di tesi, insieme al mio collega Angelini Giordano, ho deciso di intraprendere tale percorso con il fine di realizzare un'applicazione per dispositivi mobili che permettesse una comoda consultazione delle informazioni relative all'inquinamento delle acque della costa marchigiana.

Vengono esplicitate, di seguito, le modalità di suddivisione del lavoro di implementazione del sistema "AQUA".
È possibile suddividere l'intero processo realizzativo in due macro aree:

- Sviluppo dell'applicazione in una prima versione statica;
- Implementazione del sistema "AQUA" completo.

La prima di esse vede al centro la realizzazione di un'app nativa Android. Quest'ultima si interfaccerà con un file di tipo JSON [2] contenente dei dati statici che simulano le condizioni di inquinamento delle acque della costa marchigiana. Tale applicativo verrà implementato interamente in collaborazione con Giordano Angelini.

La seconda fase di lavoro, invece, porterà alla realizzazione di un'app di tipo cross-platform che costituirà la parte front-end del sistema e di un servizio back-end, realizzato dal collega appena citato. L'applicazione, attraverso delle richieste effettuate mediante delle rotte[2] inoltrate al server che ospiterà il servizio back-end, otterrà i dati da mostrare al suo interno. Tali informazioni, quindi, indicheranno all'utente la situazione corrente dello stato di inquinamento delle acque che bagnano le spiagge marchigiane.

L'elaborato di tesi è strutturato come specificato di seguito:

- Nel *Capitolo 1* viene descritta ed affrontata la fase di analisi dei requisiti riportandone il documento risultante. Vengono inoltre analizzate le differenze fra le varie tipologie di applicazioni.
- Nel *Capitolo 2* vengono presentati i principali strumenti con cui è stato realizzato l'intero progetto di tesi, spiegandone le caratteristiche e le motivazioni che hanno portato a tali scelte.
- Nel *Capitolo 3* vengono mostrati i mockup dell'applicativo realizzati in fase di progettazione.
- Nel *Capitolo 4* viene riportato il resoconto della fase di progettazione logica dell'applicativo.
- Nel *Capitolo 5* viene presentata l'applicazione Android, mostrando il suo aspetto grafico e definendo i concetti fondamentali su cui essa si basa.
- Nel *Capitolo 6* viene riportata la fase di implementazione dell'applicazione mostrando la struttura del progetto, l'architettura dell'applicativo e i tratti di codice salienti.
- Nel *Capitolo 7* viene presentata l'applicazione cross-platform facendo un punto sulle modifiche apportate rispetto alla versione precedente.
- Nel *Capitolo 8* viene mostrata un sunto della fase di realizzazione dell'app mostrando la struttura del progetto, l'architettura dell'app e i frammenti di codice da porre all'attenzione del lettore.

2 http://193.205.129.120:63434/

Capitolo 1 – Analisi dei requisiti

In ingegneria del software, l'analisi dei requisiti è la prima fase del processo di sviluppo ed è un'attività fondamentale per la buona riuscita del prodotto. Lo scopo principale è quello di definire, solitamente insieme al cliente che commissiona l'applicazione, tutte le funzionalità che verranno offerte dall'applicativo. L'analisi dei requisiti si conclude con la stesura di una dettagliata documentazione, chiamata *specifica dei requisiti*, che descrive le funzionalità del nuovo software nella loro interezza.
In questo capitolo vengono riportati i prodotti delle due fasi principali di tale analisi: la raccolta delle informazioni e la stesura della specifica dei requisiti. Inoltre, verranno approfondite le differenze che ci sono tra app native e ibride andando così a valutare come garantire il soddisfacimento dei requisiti durante la realizzazione di entrambe le applicazioni.

1.1 Raccolta delle informazioni

Prima della definizione dei requisiti del sistema, occorre recuperare quante più informazioni utili al fine di avere un quadro completo del contesto in cui si sta operando. Generalmente tali informazioni vengono raccolte attraverso delle interviste al committente e agli utenti dell'eventuale tecnologia che si sta sostituendo. Inoltre, qualora quest'ultima esista, è opportuno analizzarne i punti di forza e di debolezza per capire quali aspetti mantenere nel nuovo progetto, quali migliorare e quali aggiungere.

1.1.1 Intervista al ricercatore

Per comprendere meglio i requisiti espressi di seguito è innanzitutto necessario comprendere quale sia l'ambiente nel quale tale progetto di tesi si troverà ad operare. Per questo stesso motivo, all'inizio dei lavori di progettazione e sviluppo, è stato necessario un incontro con il ricercatore, identificabile come il committente dell'applicazione al centro di questo elaborato. Durante tale incontro, è stato presentato il contesto nel quale ci si sarebbe trovati a lavorare da lì in avanti.
Di seguito vengono riportati gli estratti più importanti dell'intervista di cui sopra.
L'Agenzia regionale per la protezione ambientale delle Marche (ARPAM) ha come scopo principale garantire il giusto supporto tecnico-scientifico alla Regione e agli Enti locali nelle materie d'interesse ambientale.
I vari ambiti comprendono: la bonifica dei suoli inquinati, la tutela dell'acqua e dell'aria, il ciclo dei rifiuti, la difesa dalle nuove forme d'inquinamento (acustico, luminoso, elettromagnetico), impegnando l'agenzia sotto i profili della redazione di pareri, del monitoraggio sullo stato dell'ambiente, della prevenzione, della vigilanza e del controllo.
Nello specifico, per quanto concerne la tutela delle acque marchigiane, gli inquinanti che vengono monitorati sono principalmente tre:

- Enterococchi;
- Escherichia coli;
- *Ostreopsis* cf. *Ovata*.

L'intera costa ed alcune aree interne, come laghi o fiumi, sono caratterizzate da una presenza capillare di stazioni di rilevamento.
Le prime due sostanze inquinanti dell'elenco di cui sopra sono rilevate in ogni stazione, la situazione l'alga *Ostreopsis* cf. *Ovata*, invece, viene monitorata solo in alcune delle oltre 250 stazioni. Questa scelta è dovuta a due principali motivi: la rilevazione fatta in un determinato punto della costa ha validità per il tratto costiero compreso in un cerchio di raggio 6km centrato nel punto in cui è stata effettuata la suddetta rilevazione; le zone costiere in cui tale parametro non viene rilevato, non sono affette dalla presenza dell'alga. Inoltre, tali misurazioni avvengono nel periodo di prolificazione dell'alga tossica che generalmente si estende da aprile ad ottobre.

1.1.2 Analisi della tecnologia già in uso

Il Committente ha espresso la volontà di rinnovare il modo di presentare i dati contenuti nel proprio sistema informativo in modo da poter rendere quest'ultimi fruibili con una maggior facilità.
Difatti l'unico modo per godere delle informazioni di cui sopra, prima dello sviluppo dell'applicativo in analisi in questo elaborato di tesi, è il sito internet dell'ARPAM[3]. Al suo interno i dati vengono mostrati attraverso due diverse schermate.
La prima riporta i dati di balneabilità in merito a escherichia coli ed enterococchi attraverso una mappa con dei poligoni che distinguono le diverse aree di rilevazione ed una tabella informativa (*Figura 1.1*).

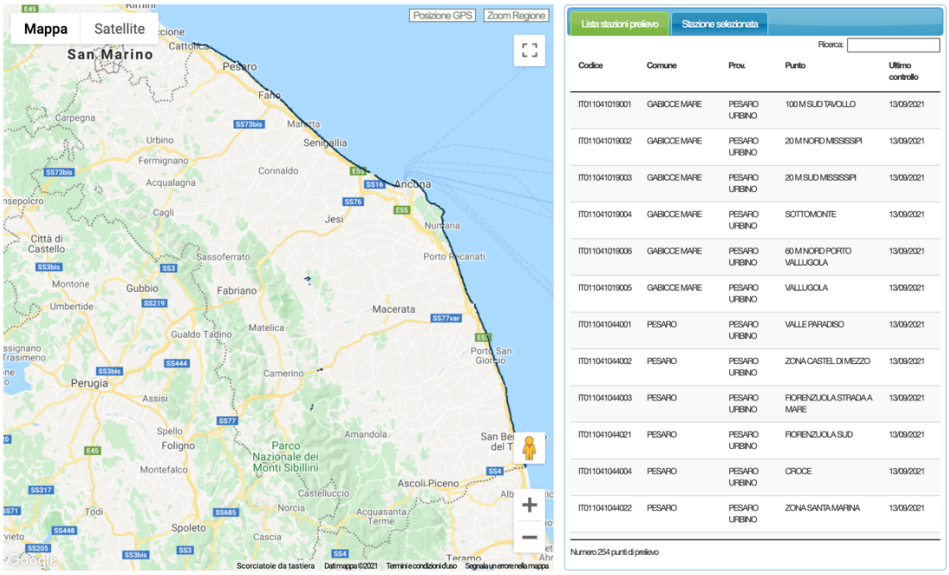

Figura 1.1 – Situazione generale balneabilità, sito ARPAM

Una delle problematiche che si evincono da questa schermata è che solo selezionando una delle stazioni se ne riesce a capire la situazione riguardante la balneabilità (*Figura 1.2*) e, di conseguenza, non si ha una visione di insieme della condizione di inquinamento dell'intera costa marchigiana.

3 https://www.arpa.marche.it/

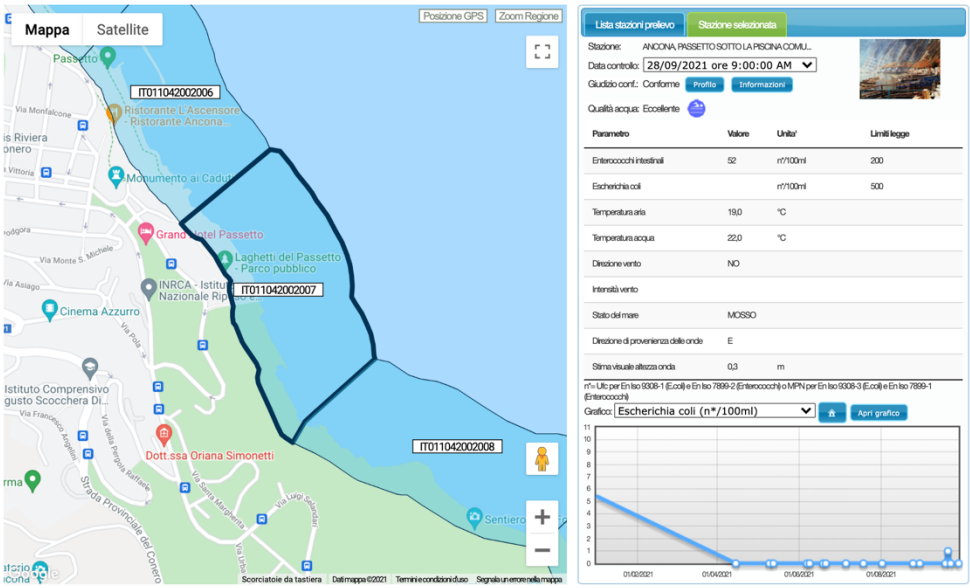

Figura 1.2 – Dettaglio balneabilità, sito ARPAM

La situazione dell'alga tossica *Ostreopsis* cf. *Ovata* viene presentata all'interno di una diversa sezione del sito attraverso una tabella (Figura 1.3).

< 10.000 Cell/L	ROUTINE	10.000/30.000 Cell/L	ALLERTA	> 30.000 Cell/L	EMERGENZA

CODICE	NOME PUNTO	LUGLIO (Cell/L)		AGOSTO (Cell/L)		SETTEMBRE (Cell/L)					OTTOBRE (Cell/L)	
0Y21	VIALE VACCAI (PS)	14/07	28/07	11/08	30/08			14/09	22/09			
		<120	<120	<120	<120			<120	<120			
0Y06	PASSETTO ASCENSORE (AN)	07/07	27/07	09/08	17/08	06/09	09/09	15/09	23/09	29/09		
		<120	<120	<120	<120	17680	21360	1239719	9920	2880		
0Y16	PIETRALACROCE (CASA DIROCCATA) (*1) (AN)						09/09					
							1720					
0Y03	PORTONOVO (AN)	07/07	27/07	09/08	17/08	06/09	09/09	15/09	23/09	29/09		
		<120	<120	<120	<120	920	28800	17520	5000	1120		
0Y07	MOLO SPIAGGIA URBANI (AN)	07/07	27/07	09/08	17/08	06/09	09/09	15/09	23/09	29/09		
		<120	<120	<120	<120	7000	3120	1400	6840	2480		
0Y08	NUMANA ALTA (*1) (AN)						09/09					
							3760					
0Y04	200 M NORD SCARICO FIUMARELLA (MC)	12/07		09/08	23/08	06/09			27/09			
		<120		<120	<120	<120			5360			
0Y05	200 M SUD SCARICO CENTRALE ENEL (AP)	21/07		10/08	24/08	06/09			21/09			
		<120		<120	<120	<120			<120			
0Y12	DAVANTI SCOGLIO SAN NICOLA (AP)	20/07	28/07	10/08	24/08	06/09			21/09			
		<120	<120	<120	<120	<120			<120			

Figura 1.3 – Situazione generale Osreopsis cf. Ovata, sito ARPAM

In generale, un'ulteriore criticità è costituita dalla mancanza di una visione di insieme dei tre inquinanti rilevati, infatti, non consultando entrambe le pagine non si ha l'idea della situazione di una determinata spiaggia. Ciò rende sicuramente i dati poco godibili da parte di un utente generico.

1.2 Specifica dei requisiti

La specifica dei requisiti ha lo scopo di tradurre formalmente tutte le informazioni raccolte durante l'incontro con il committente, di solito espresse attraverso un linguaggio naturale. Esso documento fondamentale e guida tutte le fasi successive dello sviluppo del software.
Di seguito viene riportato tale documento presentando, quindi, l'analisi dei requisiti funzionali e non funzionali relativi all'applicazione.
Con requisiti funzionali si intende la totalità delle caratteristiche che l'applicazione dovrà implementare, quelli non funzionali indicano l'insieme dei vincoli realizzativi, di tipo tecnico, che il sistema sviluppato dovrà necessariamente rispettare.

1.2.1 Requisiti funzionali

Le funzionalità che l'applicazione dovrà garantire sono state pensate per due diversi target di obiettivi: gli utenti abituali, ossia per persone che frequentano con regolarità le spiagge della costa marchigiana; i turisti che, trovandosi momentaneamente in regione, desiderano scegliere una spiaggia che rispetti determinate condizioni di inquinamento.
Di seguito verranno valutati nel dettaglio i vari requisiti funzionali del sistema.
Mappa: tale funzionalità mostrerà una mappa della regione Marche, attraverso la quale verrà resa l'idea della situazione della qualità dell'acqua dell'intera costa marchigiana. L'utente dovrà avere la possibilità di aumentare e diminuire il livello di zoom della mappa e da essa dovrà poter accedere direttamente alle informazioni generali sulla situazione attuale di una spiaggia.

Ricerca: questa funzione consentirà di ricercare, attraverso l'inserimento del nome, la spiaggia di cui l'utente ha intenzione di visualizzare le informazioni.

Preferiti: l'applicazione dovrà contenere una sezione in cui verranno mostrate le spiagge che verranno indicate come preferite dall'utente. Le spiagge potranno essere aggiunte o rimosse da tale lista ogni volta che l'utente lo riterrà opportuno.

Classifica: per mezzo di tale funzionalità l'utente potrà visualizzare la classifica delle spiagge sulla base della presenza di uno specifico inquinante. Dovrà essere presente una classifica per ciascun inquinante preso in considerazione per la realizzazione di tale progetto di tesi.

Notifiche: l'applicazione dovrà prevedere due diversi tipi di notifiche:

- *Notifiche basate sui preferiti*: l'applicazione avvertirà l'utente qualora i valori degli inquinanti, delle spiagge indicate come preferite, saranno al di sopra dei valori di soglia.

- *Notifiche basate sulla posizione*: se l'utente acconsentirà al rilevamento della posizione del dispositivo da parte del sistema, l'applicazione inoltrerà delle notifiche laddove, nei tratti di costa nei pressi della posizione acquisita, i livelli di soglia verranno superati.

L'utente dovrà avere la possibilità di abilitare e disabilitare entrambi i tipi di notifiche ogni qual volta lo reputi necessario.

Informazioni: l'applicazione dovrà mostrare all'utente delle informazioni al fine di esplicitare all'utente cosa sono gli inquinanti, le motivazioni per cui se i valori di soglia vengono superati la balneazione diventa vietata e le conseguenze a cui si incorre se tale divieto non viene rispettato.

Situazione corrente: l'utente, attraverso tale funzionalità, potrà visualizzare le informazioni riguardanti la situazione corrente della spiaggia selezionata. Tra le informazioni mostrate dovranno essere presenti: la qualità dell'acqua, la situazione dell'alga *Ostreopsis* cf. *Ovata*, il valore di ogni inquinante corrispondente all'ultima rilevazione.

Storico dei valori rilevati: questa funzione dovrà presentare all'utente uno storico delle rilevazioni dei vari inquinanti. Oltre ad una lista di valori, dovranno essere mostrate anche una serie di statistiche, tra le quali le medie mensili dei valori rilevati per ogni inquinante.

1.2.2 Requisiti non funzionali

Per quanto riguarda i requisiti non funzionali, l'applicativo dovrà essere sviluppato con un design intuitivo, in modo tale da rendere l'esperienza dell'utente il più fluida e semplice possibile.
Le varie spiagge dovranno avere un nome che le identifichi univocamente e che indichi, in modo chiaro, la loro posizione all'utente.
Di seguito vengono riportati nel dettaglio i vari requisiti non funzionali dell'applicazione.

Usabilità: l'applicazione dovrà presentare in maniera fruibile e funzionale i dati che indicano i livelli di inquinamento delle acque marchigiane, i quali possono risultare poco leggibili a prima vista.
La posizione dei tasti dovrà essere facilmente raggiungibile sullo schermo e dovranno essere di una grandezza sufficientemente grande per essere cliccata, i testi non dovranno essere di dimensioni ridotte e, infine, i colori presenti nell'applicazione dovranno essere scelti in modo tale da rendere tutto ciò che verrà mostrato godibile senza alcuno sforzo visivo.

Reattività: generalmente, gli utenti mobile sono più distratti e meno pazienti di quelli seduti ad una scrivania. È importante, quindi, che un'app risponda alle richieste dell'utente in tempi brevi. Il caricamento dei dati, quindi, dovrà essere altamente efficiente, rendendo così la *user experience* il più veloce possibile.

Persistenza dei dati: sarà necessario implementare un sistema di persistenza dei dati per tenere traccia di diverse informazioni:

- Valori delle sostanze inquinanti rilevati;
- Spiagge indicate come preferite;
- Notifiche abilitate.

In questo modo si eviterà di dover riconfigurare ad ogni avvio le preferenze impostate dall'utente.

Tecnologia open source: L'intero applicativo sarà sviluppato in un ambiente open source, grazie anche alle dispense che lo stesso Google Developer [3] mette a disposizione dei programmatori.

1.3 Applicazioni native, ibride e web app

Con il termine mobile application si intende un'applicazione software dedicata a dispositivi mobile. Essa è progettata secondo specifici criteri legati alle caratteristiche hardware dei dispositivi (che spesso possono rappresentare un limite) ed alle modalità di fruizione. Quest'ultime, a causa dei contesti più disparati in cui diventa necessario interagire con un'app, sono mobili, dinamiche e discontinue. [4]
Volendo classificare le applicazioni, sarebbe possibile dividere quest'ultime in tre macro-blocchi:

- Web app;
- App ibride;
- App native.

Figura 1.4 – Classificazione tipologie di app

Di seguito si andranno ad analizzare nel dettaglio tutte le tipologie di app appena elencate.

1.3.1 Web app

Dovendo definire una delle caratteristiche più importanti delle applicazioni web si dovrebbe necessariamente far rifermento al concetto di distribuzione: le web app, infatti, sono accessibili via web per mezzo di una rete; in definitiva, sono la versione mobile di siti web.
Esse hanno diversi vantaggi come l'essere in grado di funzionare su qualsiasi piattaforma o il non richiedere alcuno spazio di memoria per l'installazione o per il download.
Al contempo, però, presentano anche degli svantaggi: non sono in grado di interagire con l'hardware del dispositivo (come ad esempio il giroscopio), non possono funzionare senza connessione, hanno tipicamente prestazioni inferiori.

1.3.2 App native

Le app native sono applicazioni sviluppate specificamente per un sistema operativo (es: Android, iOS), per mezzo di uno specifico linguaggio di programmazione.
Esse si interfacciano al O.S. mobile nel modo più completo possibile. Essendo sviluppate per una specifica piattaforma possono interfacciarsi del tutto con le API della stessa, andando, così, ad interagire totalmente con l'hardware e il software. Ne consegue che potranno sfruttare tutte le funzionalità accessibili del dispositivo (fotocamera, NFC, geolocalizzazione, giroscopio, etc.), integrandole ed eventualmente potenziandole.
Tali applicazioni, inoltre, funzionano sempre, anche off-line in assenza di connettività e sono molto più veloci ed affidabili rispetto alle altre app, e quindi offrono una User Experience di più alto livello.
Altra caratteristica che viene annoverata tra i vantaggi che sviluppare un'applicazione nativa porta con sé è sicuramente la possibilità di utilizzare degli elementi nativi della piattaforma in termini di user interface. Un *look and feel* comune ad altre applicazioni che l'utente è già abituato ad utilizzare, renderà l'ambiente con cui quest'ultimo andrà ad interfacciarsi più familiare e intuitivo.
Di contro, però, svilupparle è tipicamente più costoso di altre tipologie di app. Infatti, nonostante il risultato finale sia più o meno lo stesso in tutte le piattaforme, l'applicazione dovrà essere scritta in linguaggi differenti. La logica alla base dell'app sarà la stessa ma a cambiare saranno, oltre al linguaggio, l'architettura ed il processo di sviluppo.
Stesso discorso vale per la manutenzione e l'aggiornamento: sviluppare un'app diversa per ogni sistema operativo per cui si intende rendere disponibile l'applicativo, infatti, richiede di manutenere ed aggiornare singolarmente tutte le applicazioni.

1.3.3 App ibride

Le App ibride sono realizzate per adattarsi a dispositivi differenti. Spesso sono sviluppate come app native provviste di interfacce HTML pari alle web app. Esse rappresentano un buon compromesso in termini di funzionalità tra app native e web. Sono più rapide da manutenere delle app native, in quanto il processo di sviluppo è unico, anche se le performance sono inferiori e presentano una stabilità ridotta.

Capitolo 2 – Strumenti

Nel capitolo che segue vengono presentati nel dettaglio tutti gli strumenti utilizzati per la progettazione e la realizzazione dell'intero progetto al centro di questo elaborato di tesi.
Verranno analizzate nel dettaglio le relative caratteristiche e motivazioni che hanno portato a tali scelte. Per completezza, in calce ad ogni pagina, saranno inseriti i riferimenti a documentazioni e siti web di ogni strumento, qualora esistenti.

2.1 IDE

In questa sezione verrà analizzato il concetto degli IDE. Partendo da una panoramica generale su di essi e spiegandone le caratteristiche principali, si procederà ad analizzare i due IDE adottati per la realizzazione dell'intero progetto di tesi e si valuteranno le motivazioni che hanno portato a tali scelte.

2.1.1 Cosa sono gli IDE

Un IDE (Integrated Development Environment), o ambiente di sviluppo integrato, è un software progettato per la realizzazione di applicazioni che aggrega strumenti di sviluppo comuni in un'unica interfaccia utente grafica. Esso assiste, quindi, i programmatori nello sviluppo di programmi e, normalmente, è costituito datali elementi:
un editor di codice sorgente;
un compilatore (interprete);
un linker;
un debugger;
strumenti di utilità.
Nei paragrafi seguenti verranno analizzate nel dettaglio le varie componenti.

2.1.2 Editor di codice sorgente

Si tratta di un editor di testo che agevola la scrittura di codice software, grazie ad utili funzionalità come:
l'evidenziazione della sintassi con suggerimenti visivi;
il completamento automatico specifico del linguaggio;
l'individuazione di bug durante la scrittura.

2.1.3 Compilatore

Un compilatore è un programma che traduce una serie di istruzioni (codice sorgente), scritte in un linguaggio di programmazione (linguaggio sorgente), in istruzioni (codice oggetto) di un altro linguaggio (linguaggio oggetto, o target).

2.1.4 Linker

Prima di trattare tale componente, è fondamentale capire un concetto che vi è alla base: il linking. Il linking (letteralmente "collegamento"), in informatica, è il procedimento di integrazione dei vari moduli a cui un programma fa riferimento (i quali possono essere sottoprogrammi o librerie), per creare una singola unità eseguibile.
Il linker (o link editor), quindi, è un programma che effettua il collegamento tra il codice oggetto, e le librerie del linguaggio necessarie per l'esecuzione del programma.

2.1.5 Debugger

Un debugger è un programma utilizzato per testare altri programmi; esso, infatti, è in grado di visualizzare graficamente la posizione di un bug nel codice originale.
Il compito principale del debugger è quello di mostrare il frammento di codice macchina che genera il problema. Il codice può essere mostrato nella sua forma nativa, tradotto in linguaggio assembly o perfino sotto forma di codice sorgente nel linguaggio di programmazione in cui il programma analizzato è stato scritto.
Poiché la compilazione con debug inserisce nel programma grandi quantità di istruzioni in più, un programma eseguito in modalità debug è tipicamente molto più lento di quando è eseguito direttamente sul processore per cui è stato sviluppato.
Quasi tutti i debugger consentono l'esecuzione del programma analizzato a piccoli passi, mostrando nelle interruzioni il codice relativo all'istruzione sorgente corrente e lo stato attuale della CPU ovvero lo stato o valore delle variabili associate alle rispettive nelle celle di memoria. L'interruzione dell'esecuzione può avvenire passo passo ad ogni singola istruzione, entrando eventualmente in altri sottoprogrammi, oppure in punti esplicitamente assegnati dall'utente (breakpoint).

2.2 Android Studio

Android Studio [5] è l'ambiente di sviluppo integrato ufficiale per lo sviluppo di app Android, basato su IntelliJ IDEA [6]. Oltre al potente editor di codice e agli strumenti per sviluppatori di IntelliJ, Android Studio offre ancora più funzionalità che migliorano l'efficienza durante la creazione di app Android, come:

- Un sistema di build flessibile basato su Gradle[4];
- Un emulatore veloce e ricco di funzionalità;
- Un'integrazione con Github (*paragrafo 2.5*) per facilitare la collaborazione tra sviluppatori e importare codice di esempio;
- Strumenti Lint per rilevare prestazioni, usabilità, compatibilità delle versioni.

Questo tipo di ambiente, nato appositamente per Android, è molto agile e versatile. Il suo marchio di fabbrica è la volontà di rendere più produttivo il lavoro del programmatore offrendo un editor snello, con template di applicazioni già pronti, molti strumenti di supporto, e soprattutto, una totale apertura al mondo Google.
La praticità di utilizzo è stata posta come uno degli obiettivi primari in questo ambiente. Infatti, l'IDE è molto comodo e veloce nell'interazione con l'utente.
Altro aspetto degno di nota è che esiste un efficiente meccanismo di preview dei layout. L'aspetto dell'interfaccia viene velocemente mostrato in anteprima. È, inoltre, possibile visualizzare tale anteprima per diversi modelli di dispositivi così da valutare come l'interfaccia che si sta sviluppando si adatta a schermi di diverse proporzioni e grandezza.

4 https://gradle.org

2.2.1 Perché Android Studio

Nello sviluppo di applicazioni è necessario scrivere molto codice; quindi, l'aspetto semplice ed intuitivo dell'ambiente di sviluppo rende tutto più facile e veloce, il che abbrevia i tempi di realizzazione dell'applicativo.
Un'altra caratteristica fondamentale di questo IDE è la presenza di funzionalità che rendono più agevoli diversi aspetti come la navigazione all'interno del codice o la rilevazione degli errori che vengono evidenziati in tempo reale.

2.2.2 Emulatore Android

Altra caratteristica fondamentale di Android Studio è la presenza di un simulatore Android (Figura. Attraverso l'apposito pannello di configurazione è possibile creare il proprio device virtuale personalizzando ogni aspetto del dispositivo che si sta simulando; RAM, processore, versione del sistema operativo e grandezza dello schermo sono solo alcune delle caratteristiche che è possibile scegliere. È possibile, inoltre, simulare un telefono già presente sul mercato e prodotto da Google (come ad esempio un Pixel).

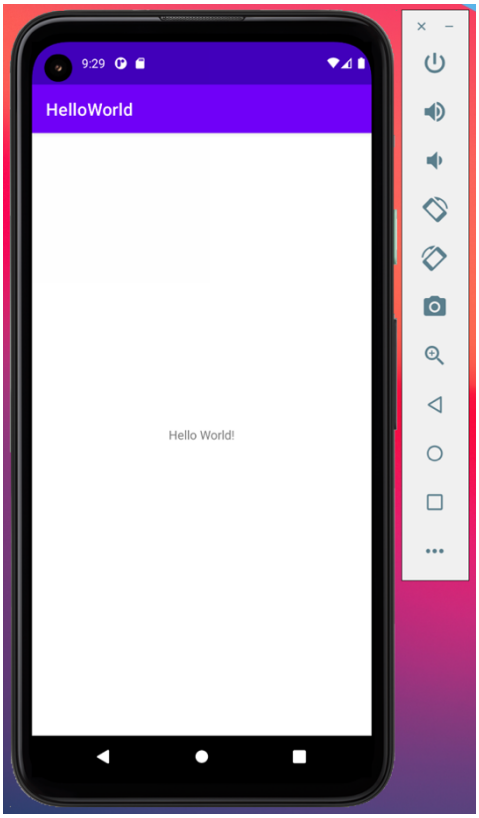

Figura 2.1 – Emulatore Android

Questa funzionalità dell'IDE permette di avere a propria disposizione diversi modelli di telefono su cui testare l'applicazione che si sta sviluppando e, inoltre, rende possibile il testing a chi non avesse un telefono Android fisico su cui lanciare l'app.
In definitiva, attraverso l'emulatore fornito da Android Studio riusciamo a simulare alla perfezione un vero e proprio smatphone (in formato digitale) nell'ambiente di lavoro che si sta utilizzando, avendo, così, la possibilità di testare non solo le proprie applicazioni, ma anche quelle presenti nello store, e quelle preinstallate nel dispositivo generato virtualmente.

2.3 Xcode

Xcode è un ambiente di sviluppo integrato completamente sviluppato e mantenuto da Apple, contenente una suite di strumenti utili allo sviluppo di Software per i sistemi macOs, iOS, iPadOS, watchOS, e tvOS.
È possibile scaricarlo gratuitamente dal Mac App Store ed è noto per supportare la distribuzione in rete del lavoro di compilazione. Xcode è, infatti, in grado di compilare un progetto su più computer riducendo i tempi; supporta, inoltre, la compilazione incrementale, ovvero è in grado di compilare il codice mentre viene scritto, in modo da ridurre il tempo di compilazione. [7]

2.3.1 Perché Xcode

Sebbene lo sviluppo dell'applicazione cross-platform sarà svolto all'interno di Android Studio, per il debugging su un dispositivo iOS (fisico o virtuale che sia) occorrerà esportare il codice in Xcode.
Per lanciare l'applicazione su un dispositivo iOS fisico occorre, innanzitutto, connettere il device al proprio Mac attraverso l'apposito cavo. In seguito, dopo aver scaricato i certificati di sviluppatore e averli associati al proprio id Apple, è possibile, attraverso il tasto "Run" di Xcode, lanciare l'applicativo sul dispositivo associato.

2.3.2 Emulatore iOS

Anche Xcode, come Android studio, permette di testare l'applicazione che si sta sviluppando su uno device virtuale. Si potrà scegliere tra diversi modelli di dispositivi iOS e, una volta aperto l'emulatore, come nel caso precedente, si avrà la possibilità di utilizzare il dispositivo nella sua interezza, quindi, non interagire solamente con le proprie applicazioni.

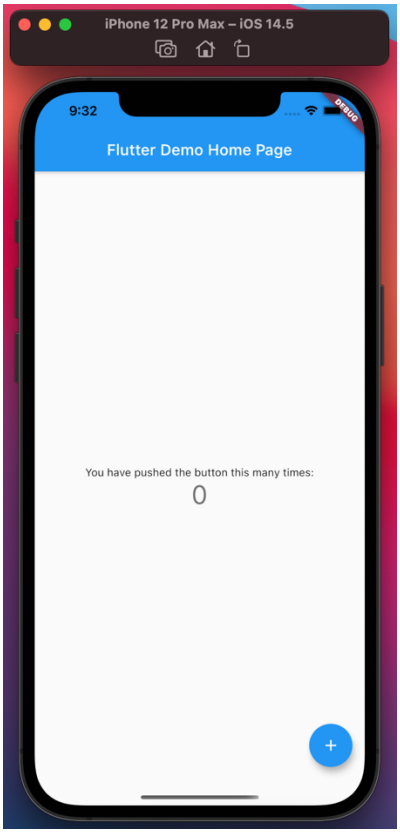

Figura 2.2 – Emulatore iOS

2.4 Framer

Framer[5] è un sito web che mette gratuitamente a disposizione dei suoi utenti una serie di strumenti per la progettazione di interfacce grafiche. Inoltre, offre la possibilità a più utenti, appartenenti allo stesso gruppo di sviluppo, di lavorare contemporaneamente ad un unico progetto.
Per la realizzazione di tutti i layout delle interfacce dell'applicativo soggetto di questo progetto di tesi è stato utilizzato questo servizio.

5 https://www.framer.com/

2.5 GitHub

GitHub[6] è un servizio web e cloud-based che aiuta gli sviluppatori ad archiviare e gestire il loro codice e a tracciare e controllare le modifiche apportate ad esso. Per capire esattamente cos'è GitHub, è necessario introdurre due principi che vi sono alla base:

- Controllo delle versioni;
- Git.

2.5.1 Controllo delle versioni

Il controllo delle versioni aiuta gli sviluppatori a tracciare e gestire le modifiche al codice di un progetto software. Esso permette agli sviluppatori di lavorare in sicurezza attraverso il *branching* (ramificazione) e il *merging* (fusione).
Con il branching, uno sviluppatore duplica parte del codice sorgente (chiamato *repository*); esso può quindi apportare in modo sicuro modifiche a quella parte del codice senza influenzare il resto del progetto.
Infine, una volta che le modifiche funzionano correttamente, la sezione di codice che le contiene può essere fusa nel codice sorgente principale così da renderla ufficiale.
Tutte queste modifiche vengono monitorate e, se necessario, possono essere ripristinate. [8]

2.5.2 Git

Git è un sistema di controllo versioni distribuito, il che significa che l'intero codice base e la cronologia sono disponibili sul computer di ogni sviluppatore; ciò permette di creare facilmente ramificazioni e fusioni.

2.5.3 Perchè GitHub

GitHub offre un servizio di hosting di repository Git basato su cloud. In sostanza, rende molto più facile per individui e team utilizzare Git per il controllo delle versioni e la collaborazione.
Attraverso la sua interfaccia è possibile visualizzare le varie ramificazioni (branch) su cui si sta lavorando, così come i *commit* (in pratica dei "salvataggi") di chiunque si stia adoperando per il progetto.
Una volta apportate alcune modifiche, è possibile inviare il codice ad un branch facendo una *pull request*. Una pull request consiste fondamentalmente nel chiedere all'utente responsabile del branch di includere il proprio codice.

6 https://github.com/

Capitolo 3 – Progettazione grafica

L'interfaccia grafica di un'applicazione (in inglese Graphical User Interface, abbreviata in GUI) è una componente fondamentale per ogni applicativo, grazie alla quale si presenta l'aspetto complessivo dell'app.
Sebbene le funzioni di un'app possano risvegliare l'interesse degli utenti e portarli a scaricarla molte volte, un design poco curato può compromettere l'utilizzo dell'applicazione e far allontanare gli utenti. Di conseguenza, l'aspetto grafico dell'applicativo deve essere curato e progettato sin dalle prime fasi di sviluppo tanto quante la sua parte funzionale.
Il design di un'applicazione non comprende solo la scelta di caratteri e colori, è fondamentale fare un uso ben ponderato di tutti i componenti grafici dell'app. Infatti, a determinare l'usabilità di un applicativo sono soprattutto la posizione, la dimensione e la strutturazione dei contenuti.
In questo capitolo viene riportata l'intera progettazione grafica dell'applicazione.

3.1 Mockup

Lo strumento di progettazione grafica usato per questo progetto di tesi è quello dei mockup.
Il termine mockup, dall'inglese "modello", ha molti significati ed utilizzi a seconda del contesto in cui viene usato. In ambito grafico, si riferisce a quei particolari elementi – file – che servono a mostrare la resa finale di un elaborato grafico.
È, in pratica, una simulazione della realtà che permette di capire come potrebbe essere prodotto, realizzato o stampato un progetto.
Creare un mockup significa dare vita ad un'anteprima convincente di un progetto che mostra la resa finale delle sue applicazioni pratiche.
In ambito informatico, il significato di tale termine non si discosta molto da quanto appena affermato, infatti, un mockup è la rappresentazione grafica, eventualmente pulita e stilizzata, delle interfacce di un sistema informativo.
Solitamente, questo tipo di modello visivo viene sottoposto al cliente prima di passare alla realizzazione del template vero e proprio, per riuscire a far capire la strategia di comunicazione della realtà e per mostrargli la disposizione ed il layout dei principali elementi.
Nel caso di tale progetto di tesi, sono stati realizzati diversi mockup dell'intera applicazione così da valutare la posizione più adatta per le varie funzionalità e capire quale strategia grafica potesse essere più accattivante per l'utente.
Di seguito vengono riportati i mockup definitivi di tutte le schermate dell'applicazione.
Nello specifico:

- Homepage (*Figura 3.1*);
- Impostazioni (*Figura 3.2*);
- Preferiti (*Figura 3.3*);
- Classifica (*Figura 3.4*);
- Info (*Figura 3.5*);
- Info inquinante (*Figura 3.6*);
- Dettagli spiaggia (*Figura 3.7*);
- Dettagli inquinante per una determinata spiaggia (*Figura 3.8*).

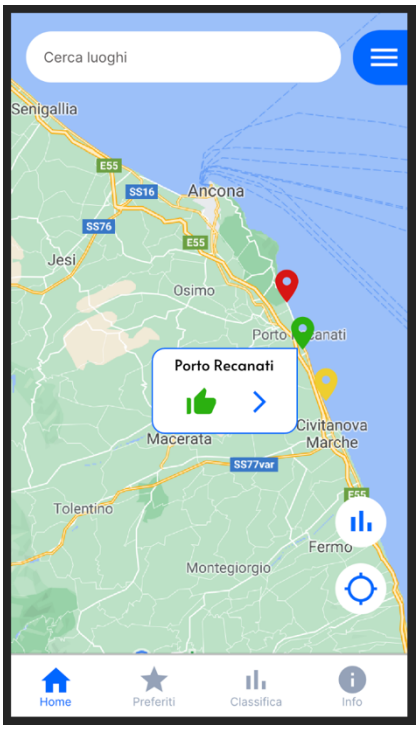

Figura 3.1 – Mockup Homepage

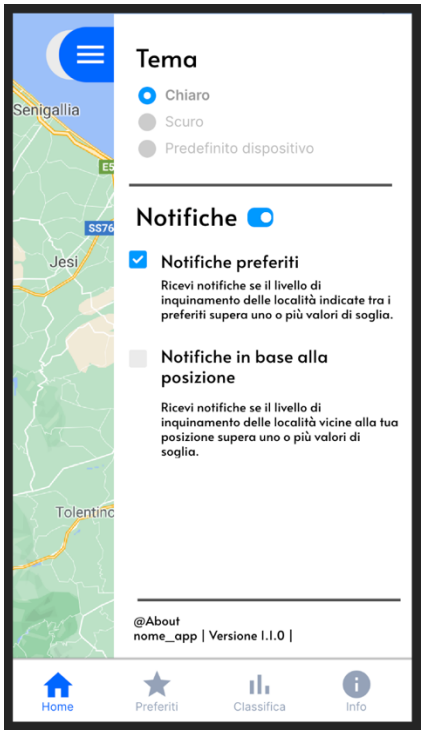

Figura 3.2 – Mockup Impostazioni

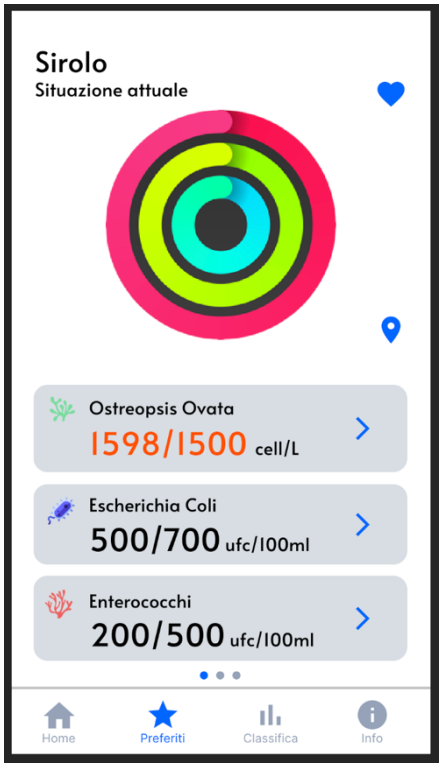

Figura 3.3 – Mockup Preferiti

Figura 3.4 – Mockup Classifica

Figura 3.5 – Mockup Info

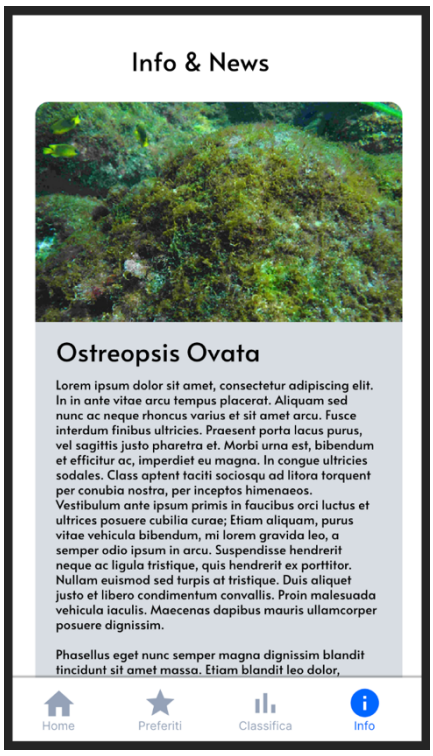

Figura 3.6 – Mockup Info inquinante

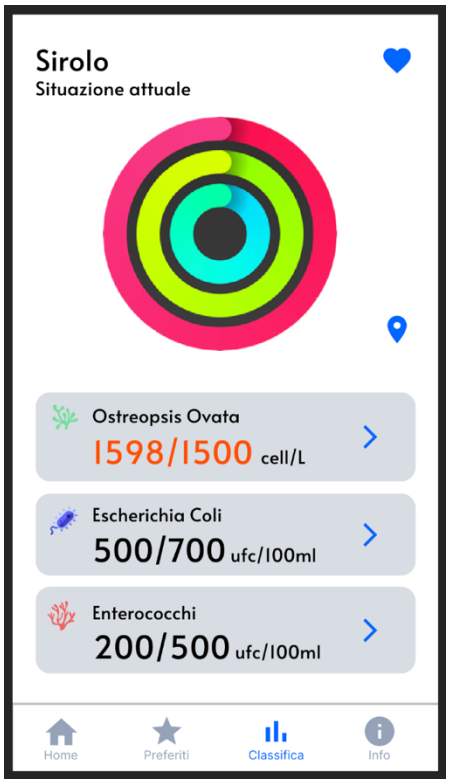

Figura 3.7 – Mockup Dettagli spiaggia

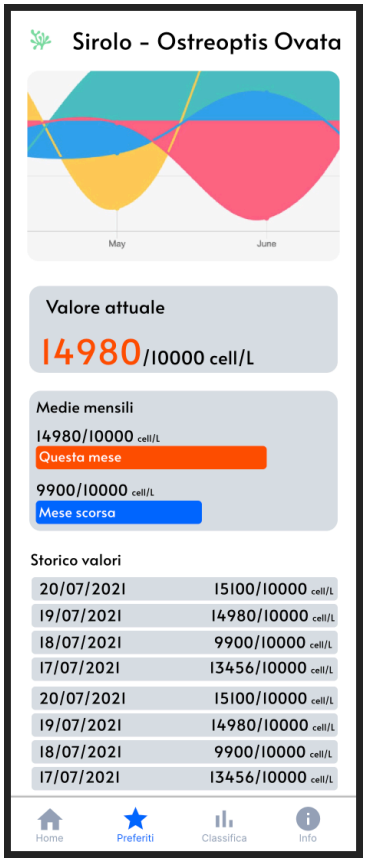

Figura 3.8 – Mockup Dettagli inquinante

Capitolo 4 – Funzionalità dell'app

In questa sezione viene mostrato come, analizzando i requisiti che l'applicazione deve avere (trattati nel *Capitolo 1 – Analisi dei requisiti*), l'applicativo soggetto di questo elaborato di tesi è stato progettato a livello funzionale.

4.1 Struttura dell'applicazione

Prima di spiegare le varie funzionalità occorre fare un accenno alla struttura nella quale le varie schermate saranno inserite. Verrà, quindi, mostrato come sarà possibile navigare tra le varie schermate e solo in seguito come esse sono state pensate per adempiere alle varie funzionalità delineate nell'analisi dei requisiti (*Capitolo 1*).

4.1.1 Albero di navigazione

L'albero di navigazione mostra come è possibile navigare tra le varie schermate dell'app.
Allo schema riportato di seguito sono state apportate alcune semplificazioni al fine di rendere il grafico più leggibile. Di seguito sono riportate le direzioni di navigazione omesse dall'albero:

- Da ogni schermata sarà possibile giungere a quella precedente con il tasto "indietro";
- Da ogni punto si potrà raggiungere una qualsiasi pagine di primo livello attraverso il corrispondente tasto presente nella barra di navigazione che consente di accedere alle schermate di cui sopra.

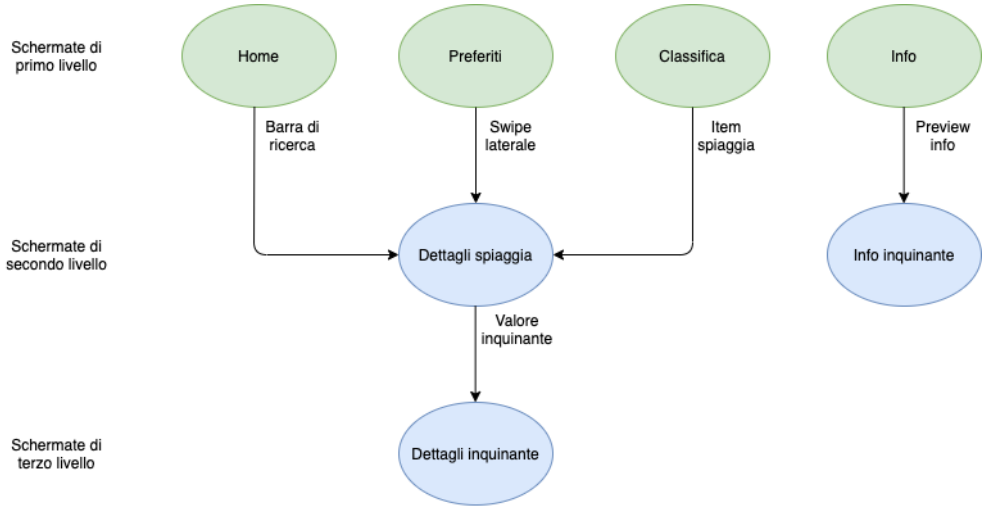

Figura 4.1 – Albero di navigazione

Tutte le schermate che saranno accessibili all'interno dell'applicazione verranno organizzate in quattro sottoalberi di navigazione, ognuno corrispondente ad una delle quattro funzionalità principali dell'app.
La navigazione all'interno di ogni sottoalbero sarà gestita tramite uno stack.
In informatica, con il termine stack (pila) si intende una struttura, le cui modalità d'accesso ai dati in essa contenuti, seguono una modalità LIFO (last in first out), per cui i dati vengono letti in ordine inverso rispetto a quello in cui sono stati inseriti nella pila.
Questo meccanismo farà si che l'utente non perda lo stato della propria navigazione in un determinato sottoalbero se dovesse decidere di usufruire di una funzionalità dell'applicazione presente in un'altra sezione dell'albero di navigazione.

4.1.2 Bottom Navigation Bar

L'applicazione, presenterà, nella parte bassa di ogni schermata, una Bottom Navigation Bar[7] che conterà al suo interno quattro tasti, uno per ogni schermata di primo livello:

- Tasto "Home";
- Tasto "Preferiti";
- Tasto "Classifica";
- Tasto "Info".

Essa è una barra di navigazione posta nella parte inferiore del layout che permette di accedere a diverse aree dell'applicazione, nel caso corrente quattro.
Di seguito verranno analizzate nel dettaglio le schermate principali, i percorsi che si potranno intraprendere a partire da esse e le relative schermate secondarie che verranno raggiunte.

7 https://developer.android.com/reference/com/google/android/material/bottomnavigation/BottomNavigationView

4.2 Home

L'homepage dell'applicazione presenta al suo interno tre delle funzionalità più importanti per l'intero applicativo. Nello specifico, sarà possibile visualizzare la mappa, effettuare la ricerca di una spiaggia attraverso il suo nome, accedere al pannello di gestione delle notifiche.
Da questa pagina si potranno raggiungere (come mostrato nella *Figura 4.2*) la schermata "Dettagli spiaggia" e di conseguenza quella denominata "Dettagli inquinante".

Figura 4.2 – Albero di navigazione home

4.2.1 Mappa

Aprendo l'applicazione la prima cosa visibile nella home sarà la mappa della regione Marche. Su di essa saranno posti dei marcatori che attraverso il loro colore indicheranno la qualità dell'acqua del tratto di costa relativo.
I colori che i markers potranno assumere sono:

- Verde scuro: qualità ottima;
- Verde chiaro: qualità buona;
- Giallo: qualità sufficiente;
- Rosso: qualità scarsa.

Cliccando su un marcatore comparirà sopra di esso una info window che mostrerà il nome della spiaggia e ribadirà la qualità dell'acqua. Cliccando sulla finestra informativa si verrà reindirizzati verso la schermata in cui vengono mostrati i dettagli della spiaggia.
Essa è stata pensata per poter rendere l'idea, con un solo sguardo, di quale fossero le condizioni dell'intero tratto costiero monitorato. Tale schermata è dunque in grado di rispondere in maniera efficiente alle esigenze di un utente "occasionale", che, trovandosi nella regione presa in considerazione per

questo elaborato di tesi, voglia avere informazioni sul tratto di costa analizzato in modo semplice ed intuitivo.
Inoltre, aumentando lo zoom su una specifica area di interesse, la distribuzione dei marcatori diventerebbe più capillare mostrando con maggior dettaglio la relativa situazione di inquinamento.
Sullo schermo saranno presenti due tasti: il primo, se cliccato, centrerà la mappa sulla regione Marche, mentre il secondo la centrerà sulla posizione del dispositivo.

4.2.2 Ricerca

Attraverso la barra di ricerca posizionata nella parte alta dello schermo sarà possibile ricercare attraverso il nome la spiaggia a cui si è interessati e, cliccando su uno dei risultati che appariranno man mano che si il nome viene digitato, si giungerà alla schermata in cui vengono mostrati i dettagli della spiaggia.

4.2.3 Notifiche

Aprendo, con l'apposito tasto, il *navigation drawer* si accederà alla sezione di gestione delle notifiche. Per attivare tale funzione occorrerà cliccare sul *radio buttom*, posto di fianco al titolo della scheda. Una volta fatto ciò appariranno due sezioni contenenti l'una, una spiegazione del tipo di notifica e l'altra, una *check box* che, se cliccata, attiverà quel determinato avviso.

4.3 Preferiti

Mentre la funzionalità "Mappa", già analizzata, è stata pensata per un utente occasionale dell'applicazione, la schermata "Preferiti" è stata studiata per mostrare agli utenti abituali i dati delle spiagge da loro indicate come preferite. Questo rende più veloce e funzionale la navigazione di tale tipologia di utenti, che non dovranno cercare le spiagge che frequentano comunemente ad ogni accesso.
Infatti, cliccando sul corrispondete tasto nella *Bottom Navigation Bar* e giungendo nella sezione "Preferiti", attraverso uno *Slider* orizzontale, si potrà visionare la condizione attuale di balneabilità delle spiagge indicate come preferite dall'utente attraverso la pagina "Dettagli spiaggia" (*Figura 4.3).*

Figura 4.3 – Albero di navigazione preferiti

4.3.1 Slider

Ogni *item* dello *slider* sarà dedicato ad una singola spiaggia, e le informazioni verranno mostrate con una grafica simile a quella presente nella schermata "Dettagli spiaggia" *(analizzata nel paragrafo 4.6)*.
Questa scelta è stata adottata per rendere l'applicazione il più semplice ed intuibile possibile, infatti, ricevere la stessa informazione in modi differenti, per esempio cambiando grafica, potrebbe confondere l'utente rendendo la *user experience* non del tutto fluida.

4.4 Classifica

Questa schermata è composta da due sezioni, l'una presenta un *drop-down menù, nell'altra* viene mostrata una lista delle spiagge.

4.4.1 Drop-down menù

Nel framework Android, una lista drop-down[8] (anche detta *lista a cascata*) è realizzata come Widget, esso che permette all'utente di selezionare un valore da un elenco.
Inizialmente essa può non contenere alcun valore, contenere un valore di default o una scritta che indica la necessità di sceglierne uno. Quando è inattivo, il menù mostra un solo item, mentre, quando è attivato, mostra una lista di valori selezionabili. Una volta che l'utente sceglie un valore dalla lista, il controllo ritorna allo stato inattivo, mostrando il solo valore attualmente selezionato.
Il menù presente in questa schermata conterrà i nomi dei tre inquinanti che sono stati presi in considerazione per tale progetto di tesi, mostrando di default "*Ostreopsis* cf. *Ovata*".

4.4.2 Lista spiagge

In tale sezione verrà visualizzata una lista contenente le spiagge collocate in ordine crescente in base al relativo livello di inquinamento dell'inquinante selezionato nella lista drop-down.
Ogni item della lista presenterà un numero che indica la posizione della spiaggia in classifica, il nome del tratto di costa e un'icona cliccabile che condurrà alla schermata "Dettagli spiaggia" *(analizzata nel paragrafo 4.6)*.
L'albero di navigazione che è possibile percorrere a partire da tale schermata è quello mostrato nella *Figura 4.4*.

8 https://material.io/components/menus

4.5 Info

Nella pagina "Info" sarà presente un elenco di schede informative. Ognuna di esse sarà costituita da un'immagine e un testo riferiti all'argomento a cui si riferisce la scheda. Il testo, nello specifico, fungerà da preview, infatti, cliccando la relativa icona si giungerà ad un'altra schermata *(analizzata nel paragrafo 4.8)* in cui verrà trattato, in maniera esaustiva, il tema della scheda stessa. Tale possibilità di navigazione è mostrata nell'apposito grafico riportato di seguito (*Figura 4.5*).

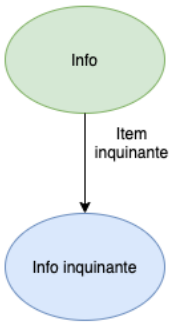

Figura 4.5 – Albero di navigazione info

4.6 Dettagli spiaggia

Questa schermata può essere definita come la più importante dell'intera applicazione, infatti, sarà essa a mostrare le informazioni che saranno valutate più interessanti dagli utenti, in merito alla situazione di balneabilità della spiaggia selezionata.
Come già visto, l'utente potrà raggiungere questa pagina di secondo livello dalla Homepage (*paragrafo 4.2*), dalla schermata dei Preferiti (*paragrafo 4.3*) e dalla pagina della Classifica (*paragrafo 4.4*).
Come è possibile visionare nel capitolo seguente attraverso le immagini di mockup, questa pagina presenta diversi elementi:

- Intestazione: contenente il nome della spiaggia;
- Grafico circolare;
- Due icone;
- Due schede riportanti le condizioni di balneabilità in tempo reale;
- Tre schede con i dettagli di ogni inquinante.

Di seguito vengono analizzate nel dettaglio alcune delle componenti di cui sopra, per le quali occorre una spiegazione più accurata.

4.6.1 Grafico

Il grafico è composto da tre *circular progress chart*, uno per ogni inquinante. Un circular progress chart è costituito da una barra di caricamento circolare che, generalmente, indica lo stato di avanzamento di un processo che si sta monitorando.
Nel caso dell'applicativo posto al centro di questo elaborato di tesi, tali grafici sono stati utilizzati per indicare la presenza degli inquinanti presi in considerazione nel tratto costiero di cui si stanno mostrando le informazioni. Nello specifico, sul chart viene riportata la percentuale del livello di inquinamento rispetto al corrispondente valore di soglia.
Ogni progress bar sarà contraddistinta da un colore diverso, colore che sarà riportato anche nelle schede degli inquinanti (*paragrafo 4.6.5*) per collegare visivamente i due elementi.

4.6.2 Icona dei preferiti

Quest'icona permetterà all'utente di indicare la corrispondente spiaggia come preferita così da poterla visualizzare nell'apposita pagina (*paragrafo 4.3*).
Il layout dell'icona sarà diverso in base alla presenza o meno della spiaggia nell'elenco dei preferiti:

- *Icona del cuore definita dal solo bordo*: spiaggia non presente nell'elenco;
- *Icona del cuore interamente colorata*: spiaggia indicata come preferita dall'utente.

4.6.3 Icona posizione

Rilevato un click su questa icona, l'applicazione reindirizzerà l'utente ad un'app di navigazione presente sul dispositivo passandole le coordinate della stazione in cui vengono effettuati i rilevamenti per quel tratto costiero.
Attraverso questa applicazione l'utente sarà in grado di impostare un percorso di navigazione verso la spiaggia di cui stanno osservando le condizioni di balneabilità.
Questa funzionalità è stata pensata per dare la possibilità agli users, una volta aver visualizzato la situazione degli inquinanti presenti in una spiaggia, di raggiungerla con facilità.

4.6.4 Schede condizioni balneabilità

Tali schede forniscono immediatamente all'utente un'idea sulla presenza dei tre inquinanti senza costringerlo a leggere i valori riportati nelle schede successive (*paragrafo seguente*).
La prima di esse mostra la qualità dell'acqua, tale valore può assumere quattro stati:

- Scarsa;
- Sufficiente;
- Buona;
- Ottima.

La seconda, invece, permette di tenere monitorata la presenza dell'*Ostreopsis* cf. *Ovata*, riportandone la condizione. Gli stati che la situazione dell'alga può assumere sono tre:

- Situazione regolare;
- Situazione di allerta;
- Situazione di emergenza.

4.6.5 Schede inquinanti

Ognuna di queste schede è composta da diversi elementi. Nella parte alta presentano un'intestazione con al suo interno un'icona che raffigura l'inquinante e il nome dello stesso. I componenti di questa prima sezione sono colorati dello stesso colore della barra di caricamento corrispondente nel grafico (*paragrafo 4.6.1*).
Subito sotto l'intestazione viene riportato il valore dell'ultima rilevazione dell'inquinante e la corrispondente unità di misura.
Infine, vi è un'icona che, se cliccata, reindirizzerà l'utente alla pagina "Dettagli inquinante" (*approfondita nel paragrafo seguente*).

4.7 Dettagli inquinante

Questa schermata di terzo livello è raggiungibile cliccando sulle icone presenti nelle schede degli inquinanti, nella pagina "Dettagli spiaggia" (*paragrafo 4.6*).
Lo screen "Dettagli inquinante" mostra una situazione più dettagliata sulla presenza dell'inquinante selezionato nelle acque della spiaggia di cui si stanno visualizzando le informazioni.
Entrando nel dettaglio vengono proposti all'utente:

- Il valore dell'ultima rilevazione indicato come "Valore attuale";
- Un grafico riportante l'andamento delle ultime rilevazioni;
- Le medie mensili delle rilevazioni effettuate negli ultimi tre mesi;
- Un elenco delle ultime rilevazioni.

Di seguito vengono approfonditi gli ultimi due punti dell'elenco sopra riportato.

4.7.1 Medie mensili

Per rendere maggiormente l'idea dell'andamento dei valori rilevati, oltre che mostrandone il valore, le medie mensili verranno espresse tramite una *linear progress bar*. Quest'ultima si riempirà in maniera proporzionale alla percentuale di presenza dell'inquinante prendendo come valore massimo quello di soglia.

4.7.2 Storico valori

I valori verranno mostrati attraverso una *recycler view* verticale. Tale Widget è una comune lista scrollabile, con il vantaggio di creare dinamicamente gli item che stanno per essere visualizzati e distruggere quelli che escono fuori dallo spazio di visualizzazione. In questo modo non si appesantiscono le prestazioni dell'applicazione tenendo in memoria diversi elementi che potrebbero rimanere inutilizzati.
Ogni item è composto da due campi di testo: il primo mostra la data della rilevazione, il secondo presenta il valore rilevato e la corrispondente unità di misura.

4.8 Info inquinante

Tale schermata è raggiungibile attraverso l'icona cliccabile nelle schede di preview presenti nella schermata Info (*paragrafo 4.5*).
In definitiva, essa ha lo scopo di presentare una serie di nozioni che informeranno l'utente sulla natura degli inquinanti e sui rischi ai quali si incorre qualora essi superino i valori di allerta.

4.9 Ulteriori scelte progettuali

Oltre ad aver progettato l'intera interfaccia e parte del funzionamento dell'applicativo, occorre effettuare delle scelte di natura progettuale volte sia a soddisfare alcuni requisiti non funzionali, sia a incrementare le prestazioni dell'app e a garantire che la user experience sia il più fluida possibile.
Di seguito sono riportate alcune tra le più importanti scelte progettuali.

4.9.1 Reattività

Per massimizzare la velocità di caricamento dei dati da parte dell'applicativo verrà adottato un approccio al download dei dati di tipo lazy. Infatti, verranno scaricate solo le informazioni che saranno visualizzate dall'utente. Questo eviterà, all'avvio dell'applicazione, di dover effettuare il download dell'intera mole di dati contenuta nel database, e dunque di dover attendere diversi secondi prima che il sistema diventi operativo.
Ciò, infatti, renderebbe l'applicazione poco attraente agli occhi dell'utente, il quale, soprattutto quando si interfaccia con lo smartphone, è abituato ad ottenere nell'immediato le informazioni che cerca.

4.9.2 Persistenza dei dati

Per poter tener traccia dello storico dei valori di inquinamento misurati nelle varie rilevazioni sarà necessario implementare un database nel quale immagazzinare e tenera traccia di tutti i dati rilevati.
Per quanto riguarda la memorizzazione delle preferenze dell'utente, quali il settaggio delle notifiche e le spiagge indicate come preferite, si dovrà ricorrere al sistema di persistenza messo a disposizione da entrambi i framework che verranno utilizzati per la programmazione dell'applicazione. Sostanzialmente, l'app si interfaccerà con un file dedicato, presente nella memoria del telefono, nel quale leggerà e scriverà i dati di cui intende tenere traccia.
Tali dati verranno registrati attraverso delle *coppie chiave-valore*.
Nel caso delle spiagge preferite, la chiave corrisponderà all'ID della spiaggia ed assumerà un valore di tipo booleano:

- 0 - spiaggia non indicata come preferita;
- 1 - spiaggia preferita.

Anche il settaggio delle notifiche verrà implementato con un sistema simile: la chiave indicherà il tipo di notifica e l'attributo booleano mostrerà se essa è stata abilitata o meno dall'utente.

4.9.3 Tracciamento della posizione

Affinché le notifiche basate sulla posizione e la centratura della mappa sulle coordinate del dispositivo, possano operare correttamente occorrerà che l'utente abiliti l'app al tracciamento della posizione del suo dispositivo.
Ciò potrà accadere in due momenti differenti:

- Quando l'applicazione verrà scaricata dallo store del proprio device mobile;
- Quando l'utente tenterà di interfacciarsi con una delle due funzioni sopra riportate.

Capitolo 5 – Applicazione Android

In questo capitolo verrà analizzata l'applicazione Android, sviluppata per questo progetto di tesi. Dopo aver fatto un breve cenno alle componenti base che costituiscono la maggior parte delle app Android [9], verrà mostrato l'aspetto grafico dell'interfaccia utente dell'applicazione.

5.1 Activity

Citando la documentazione ufficiale[9], un'Activity rappresenta una singola cosa che può essere fatta nell'app, un singolo modo per eseguire un obiettivo dell'utente.
Nello specifico, quindi, un'Activity può essere vista come la rappresentazione di una schermata (screen) avente come scopo la gestione dell'interazione tra l'utente e l'applicazione.
In generale, un'app può essere composta da una o più Activity, ma solo una di esse può essere attiva in un dato istante; occorre, di conseguenza, analizzare come le Activity si interscambino tra di loro nella gestione dell'interazione dello user.
Per ogni applicazione che viene avviata su un dispositivo Android il sistema mantiene una struttura denominata Activity Stack: una pila gestita con la modalità LIFO. L'Activity che si trova in cima allo stack è dunque quella attualmente attiva e quando termina viene espulsa e distrutta.

5.1.1 Ciclo di vita di un'Activity

A partire dal momento in cui l'applicazione viene lanciata fino al momento in cui essa viene posta in background, l'Activity percorre varie fasi; da qui nasce il concetto del ciclo di vita.
Nell'immagine che segue (*Figura 5.1*), le fasi salienti della vita dell'Activity sono rappresentate ed individuate con forme colorate e distinguono i momenti in cui essa viene lanciata, quando è in fase di *running* (cioè quando l'utente la utilizza), fino alle circostanze che generano la sua rimozione.
Un'Activity può trovarsi in uno dei seguenti stati durante la sua esecuzione all'interno di un'applicazione:

- *Active/Running*: L'Activity è in cima all'Activity Stack, è visibile sullo schermo e sta interagendo con l'utente;
- *Paused*: L'Activity è visibile ma non è quella che sta interagendo con l'utente, magari perché è parzialmente oscurata da altre Activity;
- *Stopped*: L'Activity non è visibile all'utente, è totalmente oscurata da altre Activity;
- *Destroyed*: L'Activity è stata terminata e non è più presente nell'Activity Stack.

9 https://developer.android.com/reference/kotlin/android/app/Activity

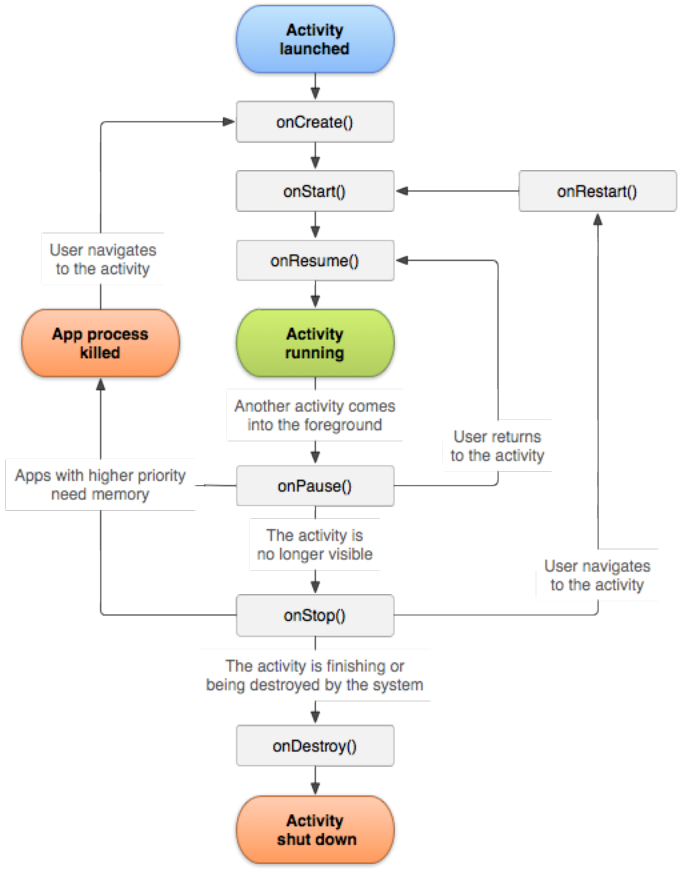

Figura 5.1 – Ciclo di vita di un'Activity

5.2 Fragment

Un Fragment[10] può essere considerato (secondo la definizione ufficiale presente nella documentazione) una porzione dell'interfaccia utente, completa non solo di layout ma anche di tutto il codice necessario a rappresentarne la logica di gestione. Offrono la possibilità di creare interfacce utente composte da porzioni riutilizzabili e alternabili senza la necessità di distruggere l'Activity.
Se lo si ritiene opportuno è possibile combinare più Fragment all'interno della stessa Activity, allo stesso modo è possibile usare nuovamente lo stesso Fragment all'interno di diverse Activity. In definitiva, esso costituisce un "modulo" dell'Activity, una sua sottosezione.
Questa caratteristica lo rende utile al fine di rendere l'interfaccia grafica dell'applicazione più modulare.
Nello specifico, incapsulando le varie sezioni di un'interfaccia in diversi Fragment, è possibile gestire più comodamente la loro collocazione in differenti tipi di schermi con dimensioni, proporzioni e orientamento diversi.

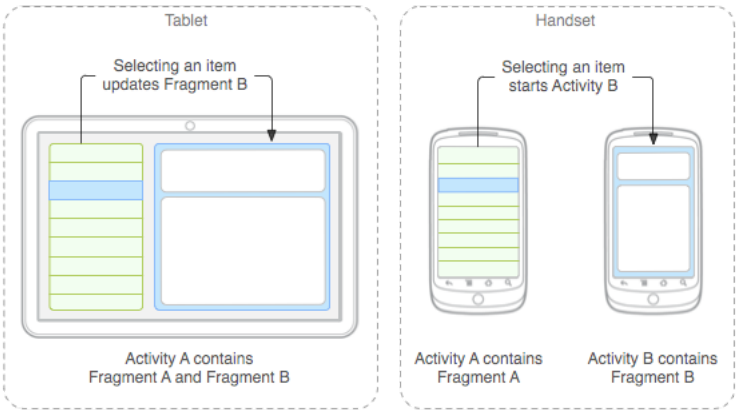

Figura 5.2 – Fragment applicati a diversi device

Un Fragment ha bisogno di un'Activity che gli faccia da contenitore, pertanto, l'uso di questi componenti facilita, da un lato, la gestione del ciclo di vita di un'Activity (visto che questa non cambia ma si alternano solo i Fragment al suo interno) ma dall'altro aggiunge ulteriore complessità considerando che essi hanno un proprio ciclo di vita piuttosto articolato.

5.2.1 Ciclo di vita di un Fragment

Come detto il Fragment ha il suo ciclo di vita fortemente collegato con quello dell'Activity di appartenenza.
La figura riportata di seguito mostra la sequenza di stati che scandiscono la vita del Fragment.

10 https://developer.android.com/guide/fragments

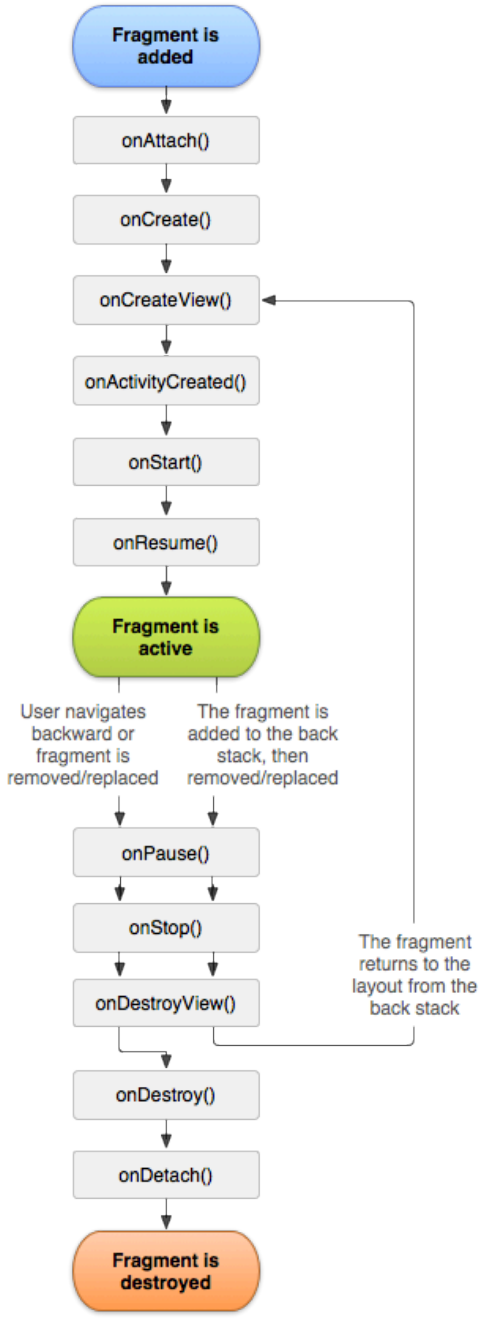

Figura 5.3 – Ciclo di vita di un Fragment

Come è possibile notare sia i metodi che gli stati attraversati dal Fragment ricordano molto quelli dell'Activity.
Eccetto alcuni metodi che rendono più variegata l'inizializzazione del Fragment le altre funzioni di callback del ciclo di vita vengono chiamate in corrispondenza degli omonimi metodi dell'Activity.

5.3 Aspetto User Interface app Android

In questa sezione vengono riportate le immagini che descrivono l'aspetto della prima versione dell'applicazione "AQUA" disponibile solo per Android.
Lo scopo di tale sezione è quello di mostrare come l'aspetto della user interface sia variato rispetto ai mockup iniziali, e di conseguenza, quali migliorie grafiche siano state apportate al fine di migliorare la user experience.

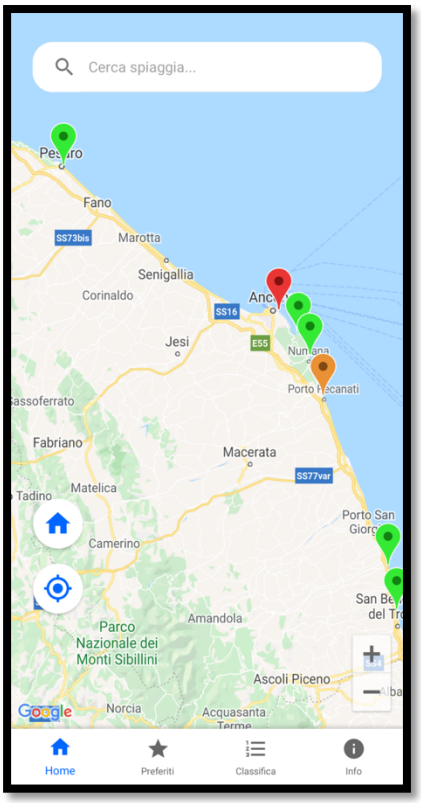

Figura 5.4 – Aspetto Home app Android

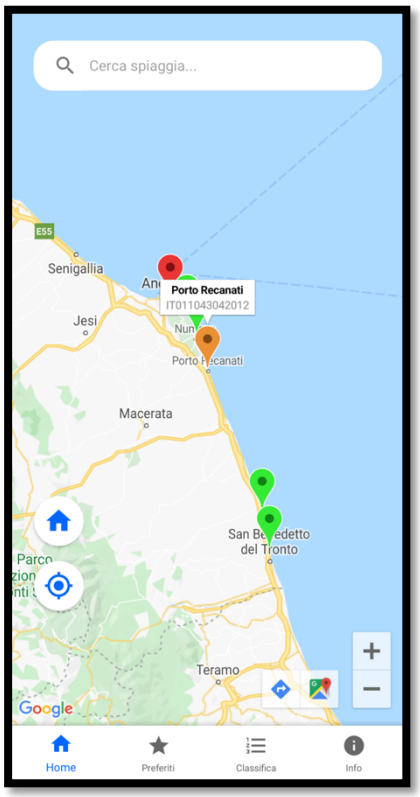

Figura 5.5 – Aspetto Marker home app Android

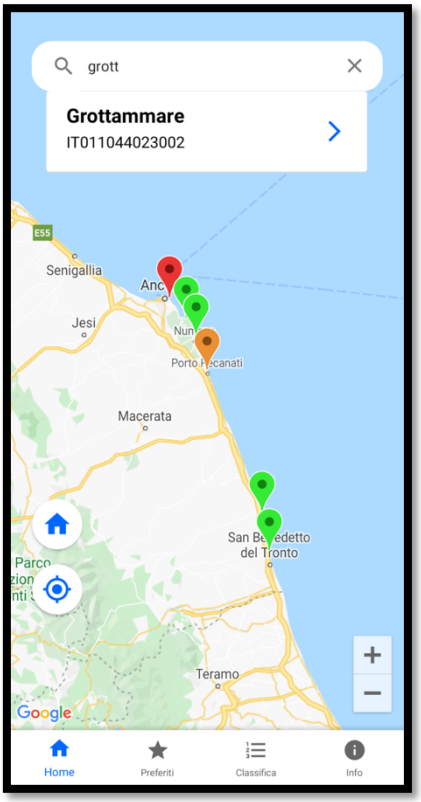

Figura 5.6 – Aspetto Barra di ricercar app Android

Figura 5.7 – Aspetto Preferiti app Android

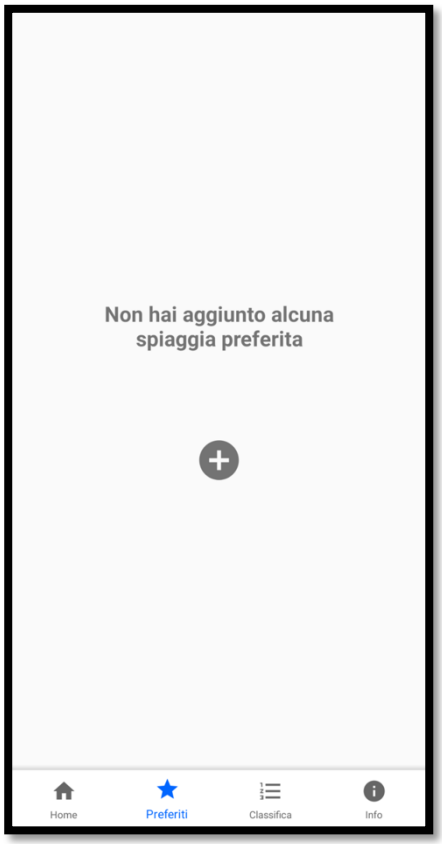

Figura 5.8 – Aspetto Preferiti vuoto app Android

Figura 5.9 – Aspetto Classifica app Android

Figura 5.10 – Aspetto Dettagli situazione spiaggia app Android

Figura 5.11 – Aspetto Dettagli storico Ostreopsis cf. Ovata app Android

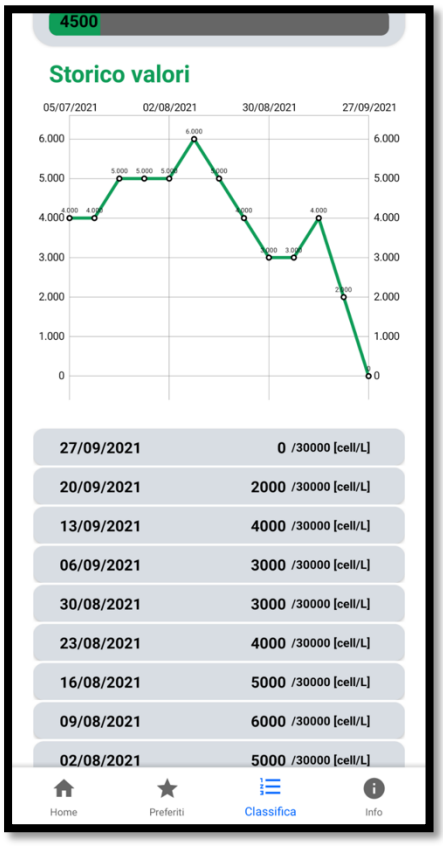

Figura 5.12 – Aspetto Dettagli storico seconda parte app Android

Figura 5.13 – Aspetto Dettagli storico Escherichia coli app Android

Figura 5.14 – Aspetto Dettagli storico Enterococcus app Android

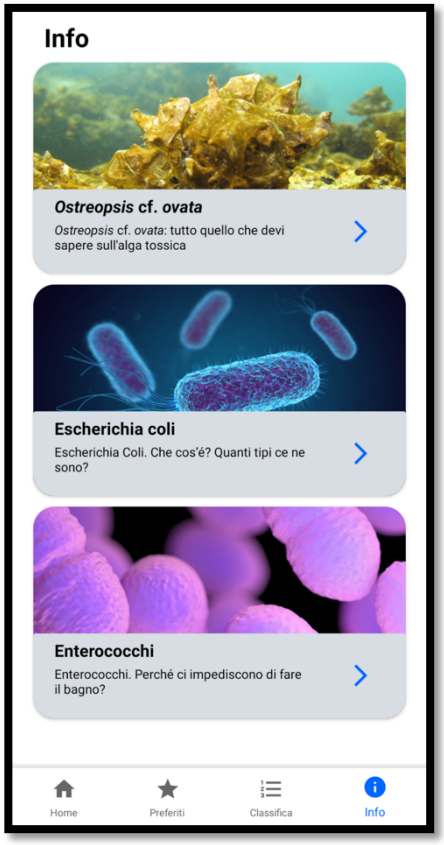

Figura 5.15 – Aspetto Preview Info app Android

Figura 5.16 Aspetto Dettagli Info app Android

Capitolo 6 – Implementazione applicazione Android

In questo capitolo viene trattata nel dettaglio la fase di sviluppo dell'applicativo "AQUA" realizzato per piattaforma Android. Nel dettaglio verranno analizzati la struttura del progetto, l'architettura dell'applicazione e i frammenti di codice salienti, scritti in Kotlin [10].
L'intero progetto è scaricabile dal repository GitHub raggiungibile al seguente link: https://github.com/giordanoangelini/AppTesi.git.

6.1 Struttura del progetto

In Android Studio la struttura di un progetto è divisa in tre parti principali:

- La cartella contenente il codice Java;
- La cartella "res" contenente risorse per lo più realizzate in XML;
- Un file di configurazione denominato AndroidManifest.xml.

La figura seguente mostra la disposizione di tali elementi.

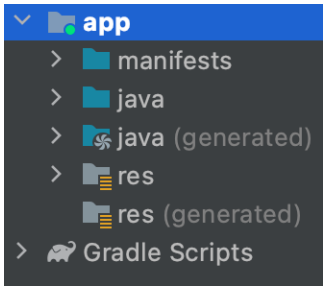

Figura 6.1 – Struttura del progetto

Si noti che il progetto è contenuto in una cartella denominata app. Questo è il modulo di default. L'IDE, infatti, suddivide un progetto in più moduli, ognuno dei quali può svolgere un ruolo diverso (libreria Java, libreria Android, inclusione di un progetto esterno, eccetera…). Il modulo app include i file manifest, il codice Java e le risorse.

6.1.1 File manifest

Ogni applicazione Android dev'essere accompagnata da un file chiamato AndroidManifest.xml[11] nella sua cartella principale. Esso raccoglie informazioni basilari sull'app necessarie al sistema per far girare qualsiasi porzione di codice della stessa. Il Manifest presente in ciascuna applicazione del Play Store si occupa di:

- Descrivere le componenti dell'applicazione (attività, servizi, receiver, provider, ecc.), nomina le classi e pubblica le loro "competenze";
- Determina quali processi ospiteranno componenti dell'applicazione;
- Dichiara i permessi dell'app e quelli necessari alle altre app per interagire con la stessa;
- Dichiara il livello minimo di API Android che l'app richiede;
- Elenca le librerie necessarie all'app.

11 https://developer.android.com/guide/topics/manifest/manifest-intro

6.1.2 Cartella java

In questa cartella vi sono tutti i file contenenti il codice per la gestione della parte logica dell'app. Tali file, nell'applicativo che si sta presentando in questa unità, hanno tutti estensione "kotlin". Essi, come è visibile in figura, sono stati divisi in diversi package per rendere la lettura del progetto più fruibile.

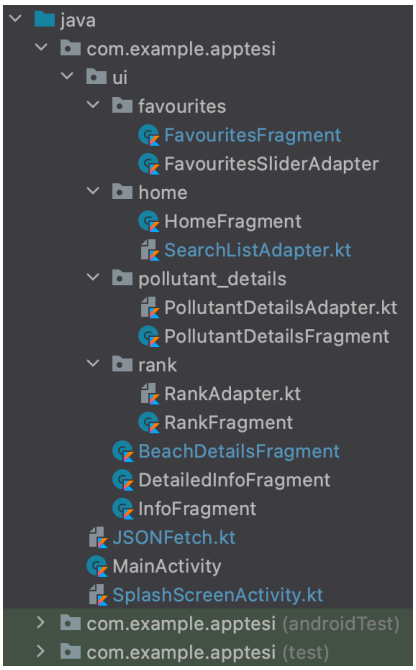

Figura 6.2 – Cartella java

Di seguito vengono analizzati nel dettaglio i vari package evidenziandone lo scopo specifico:

- *"com.example.apptesi"*: racchiude il package "ui" e i file che si occupano della fase di avvio dell'applicazione, nello specifico lo SplashScreen, il file contenente le classi che si occupano di effettuare il fetch del file JSON e la classe che descrive la Main Activity;
- *"ui"*: contiene al suo interno tutti i file responsabili del settaggio delle varie views dinamiche che andranno a comporre la User Interface;
- *"favourites"*: al suo interno sono inclusi tutti i file responsabili della funzionalità "Preferiti" analizzata nella sezione 4.3.
- *"home"*: contiene i file che racchiudono il codice che gestisce le varie funzionalità della schermata home, come la ricerca o la mappa.
- *"pollutant_details"*: include le classi ed i file responsabili dei contenuti e della gestione degli eventi legati alla schermata che mostra lo storico di un dato inquinante per una determinata spiaggia;
- *"rank"*: contiene la classe che descrive il Fragment "Rank" e l'Adapter che di occupa di popolare la lista delle spiagge mostrata in tale schermata.

6.1.3 Risorse

Come si evince dall'immagine, tale cartella è composta da diverse directory:

- anim: contiene i file XML che descrivono le animazioni;
- drawable: contiene tutte le immagini e le icone usate dall'applicazione;
- layout: al suo interno vi sono tutti i file XML che descrivono i Fragment e le Activity che compongono l'app;
- menu: contiene il file in cui vi è descritta la Bottom Navigation Bar;
- mipmap: aprendo questa directory si trova il file dell'icona dell'app;
- navigation: contiene tutti i file XML che identificano i percorsi di navigazione;
- values: al suo interno vi sono diverse sottodirectory contenenti valori costanti che vengono usati nell'app quali stringhe, colori, numeri.

Quest'ultima, in particolare, garantisce un'ottima manutenibilità dell'app in quanto, non avendo tutti questi valori scritti in modalità hard coded nel codice, permetterà di correggerli e/o modificarli più semplicemente.

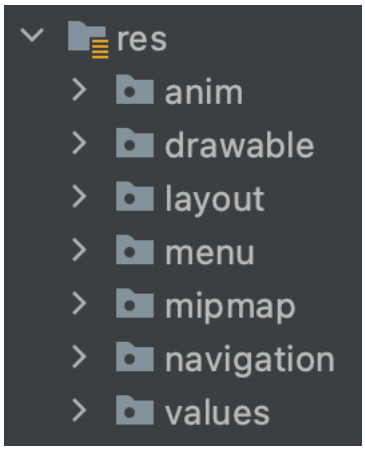

Figura 6.3 – Cartella delle risorse

6.1.4 Gradle Scripts

Al di fuori del modulo app si può notare la sezione Gradle Scripts. Qui ci sono i file di build che userà Gradle per trasformare il progetto in un'applicazione funzionante. In particolare, vi sono due file di build: uno per tutto il progetto ed uno per il solo modulo app.
Quest'ultimo, nel dettaglio, è stato modificato al fine di inserire le dipendenze necessarie affinché si siano potute usare componenti e librerie esterne al progetto.

6.2 Architettura dell'applicazione

Con il termine architettura applicativa si indica la struttura di un sistema in termini di risorse e le interconnessioni che vi sono tra di esse.
Nello specifico l'architettura dell'applicazione in analisi in questa sezione è riassumibile nello schema riportato di seguito.

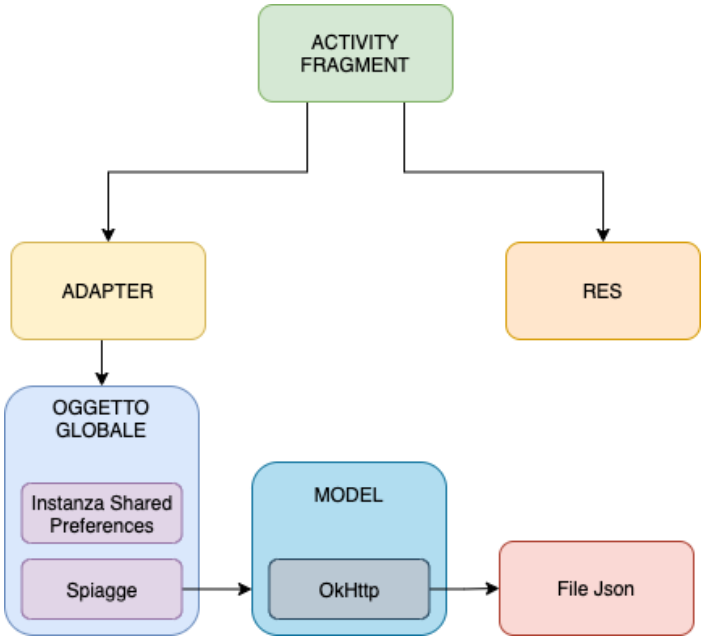

Figura 6.4 – Schema di architettura di alto livello app Android

Dato lo schema di architettura ad alto livello dell'app è possibile scendere nel dettaglio per valutare come le varie componenti interagiscono tra di loro.
Per garantire la maggior leggibilità possibile sono stati tralasciati i riferimenti ai file contenuti nella cartella "res" (analizzata nel *paragrafo 6.1.3*).
Sebbene i grafici siano riportati in maniera separata per favorire una buona fruibilità degli stessi, in realtà, sono da intendersi come uno schema unico. Infatti, tutti i Fragment descritti sono collocati nel NavHost posto all'interno della MainActivity.
Negli schemi che seguiranno il verso delle frecce indica se le interazioni sono unilaterali (chiamata ad un metodo di tipo void – senza alcun ritorno, navigazione verso una nuova Activity, etc.) oppure se le interazioni sono percorribili in entrambi i sensi (chiamata ad un metodo che torna un valore utile all'elemento chiamante, etc.).

6.2.1 Schema di architettura – Fase di avvio

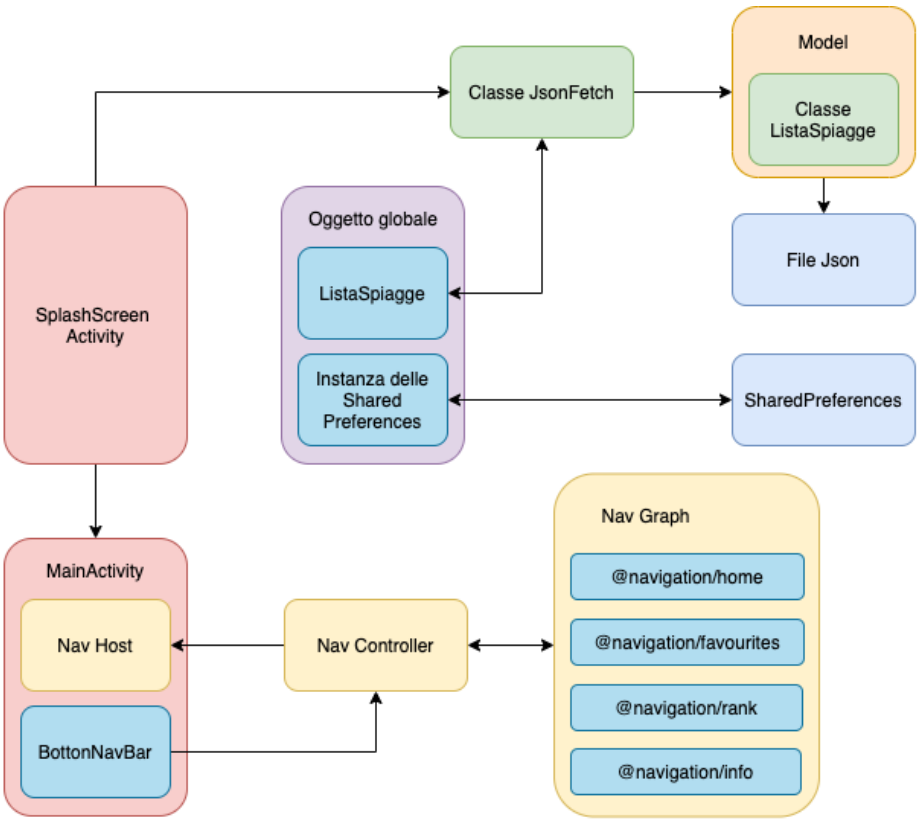

Figura 6.5 – Architettura dei processi protagonisti della fase di avvio dell'app Android

L'applicazione è costituita da due sole Activity. La prima di esse racchiude lo SplashScreen[12]: schermata iniziale, contenente il logo dell'app, che viene visualizzata mentre l'applicativo effettua il download dei dati necessari per il funzionamento dell'applicazione.
Tale processo è gestito da una classe denominata "JsonFetch"; essa, si occupa di mappare il contenuto del file JSON in una classe Kotlin definita nel model e denominata "ListaSpiaggia".
I model sono delle componenti indipendenti dalle View che costituiscono le interfacce, esse permettono di astrarre in classi Kotlin o Java i dati derivati dalla struttura scelta per mantenere le informazioni in maniera persistente.

12 https://developer.android.com/guide/topics/ui/splash-screen

6.2.2 File JSON

Come già esplicitato nella sezione "Introduzione", questa prima versione dell'app permette di visualizzare i dati relati alle rilevazioni di soli sette siti, nel dettaglio le sette stazioni in cui viene rilevata anche la presenza dell'alga tossica *Ostreopsis* cf. *Ovata*.
Inoltre, i valori mostrati, sebbene siano verosimili a quelli reali, in realtà sono dati statici, ossia non vengono aggiornati periodicamente sulla base delle nuove rilevazioni effettuate; tali valori sono contenuti in un file JSON.
L'applicazione ad ogni avvio ne scarica l'intero contenuto creando un oggetto accessibile da tutte le classi del progetto (tale maccanismo verrà approfondito nel paragrafo 6.3.1).
Il file, per l'appunto, contiene al suo interno un vettore di elementi del formato riportato nella figura che segue.

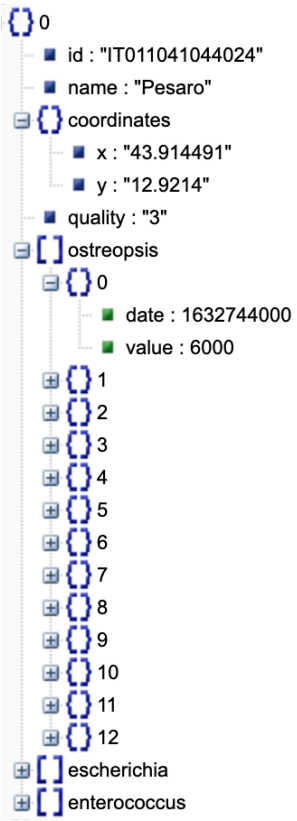

Figura 6.6 – File JSON

Ogni item del vettore contiene una serie di attributi fondamentali per il corretto funzionamento dell'applicazione:

- *"id"*: identificatore univo della spiaggia;
- *"name"*: nome della spiaggia;
- *"coordinates"*: coordinate esatte del punto in cui sono state effettuate le rilevazioni;
- *"quality"*: qualità dell'acqua espressa attraverso un numero
  - *0*: qualità scarsa;
  - *1*: qualità sufficiente;
  - *2*: qualità buona;
  - *3*: qualità ottima;
- *"ostreopsis"*: storico delle rilevazioni dell'alga tossica, è costituito da un vettore di elementi contenenti la data e il valore della rilevazione;
- *"escherichia"*: storico delle rilevazioni di escherichia coli, costituito come il precedente;
- *"enteorcoccus"*: storico delle rilevazioni di enterococchi, costituito come i precedenti due.

6.2.3 Oggetto globale

Il Singleton[13] rappresenta un tipo particolare di classe che garantisce che soltanto un'unica istanza della classe stessa possa essere creata all'interno di un programma. Nello specifico tale oggetto è composto da due attributi: il primo è un elemento di tipo "ListaSpiaggia" e contiene al suo interno i dati letti nel file JSON, il secondo racchiude un'istanza delle SharedPreferences[14]. Quest'ultima tecnologia serve a memorizzare informazioni utili per il funzionamento delle applicazioni sotto forma di coppie nome-valore.
I dati collocati nelle SharedPreferences vengono salvati in un file XML contenuto nello storage interno, precisamente nella cartella *shared_prefs,* e forniscono un'ottima soluzione nel caso in cui si ha bisogno di salvare dati di tipo primitivo in locale, come password, indirizzi IP, numeri, impostazioni, preferiti o altri dati di configurazione piuttosto elementari.
Ciò permette di mantenere delle informazioni, generalmente indentificate come preferenze dell'utente, e riproporle ad ogni avvio dell'app senza che esse debbano essere settate nuovamente dallo user.

13 https://developer.android.com/training/volley/requestqueue
14https://developer.android.com/reference/android/content/SharedPreferences

6.2.4 MainActivity e Navigazione

Una volta scaricati e istanziati i dati necessari al funzionamento dell'applicazione, la SplashScreen Activity attraverso un Intent[15] invocherà la MainActivity.
Quest'ultima mantiene il focus per tutto il resto del tempo in cui l'applicativo rimane in funzione. Infatti, sebbene all'interno di un applicativo possano coesistere diverse Activity, una solo di esse può essere mostrata all'utente in un dato istante, quest'ultima si dice che abbia il focus dell'applicazione.
Il layout della MainActivity contiene due elementi:

- Una Bottom Navigation Bar;
- Un NavHost[16].

Quest'ultimo elemento è uno dei tre componenti che gestiscono la navigazione all'interno dell'app.
Essa, infatti, avviene caricando di volta in volta un Fragment diverso all'interno del NavHost, che gestirà lo swap dei Fragment.
Gli altri due componenti protagonisti della navigazione sono:

- Il grafico di navigazione[17];
- Il NavController[18].

Il primo, di cui si ha un esempio nella figura sottostante, è una risorsa XML che contiene tutte le informazioni relative alla navigazione all'interno dell'app. Nel dettaglio, include tutte le destinazioni (in generale Fragment o Activity) su cui si può navigare e le azioni associate per transitare tra loro.
Inoltre, opzionalmente, contiene l'elenco delle animazioni da visualizzare in entrata o in uscita da un Fragment.

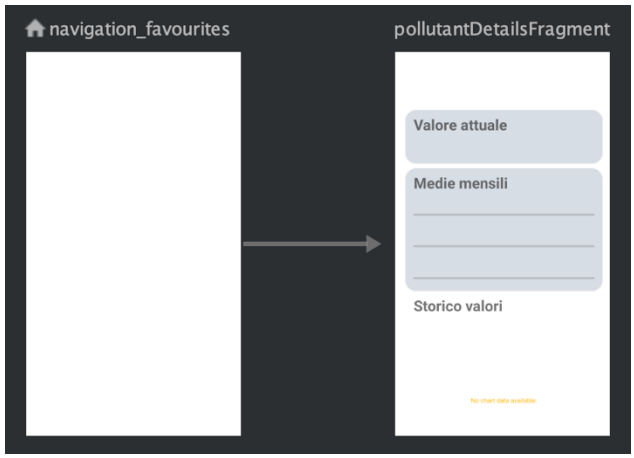

Figura 6.7 – Esempio di Navigation graph

15 https://developer.android.com/reference/android/content/Intent
16 https://developer.android.com/reference/androidx/navigation/NavHost
17 https://developer.android.com/reference/androidx/navigation/NavGraph
18 https://developer.android.com/reference/androidx/navigation/NavController

Il secondo è un oggetto che gestisce la navigazione dell'app all'interno del navigation host. Il NavController orchestra lo scambio del contenuto di destinazione nel "contenitore" mentre gli utenti si spostano all'interno dell'app.
Infatti, specificare un percorso di destinazione come action all'interno del navigation graph non basta ad eseguire la navigazione. Occorre, quindi, usare il NavController passandogli un determinato ID a cui un'azione è associata, per far si che quest'ultima venga eseguita.
Come già discusso nel paragrafo 4.1.1, la navigazione viene gestita attraverso quattro stack, uno per ogni elemento della Bottom Navigation Bar. L'implementazione di tale meccanismo è riportata nella sezione 6.3.2.

6.2.5 Schema di architettura – Ramo "Home"

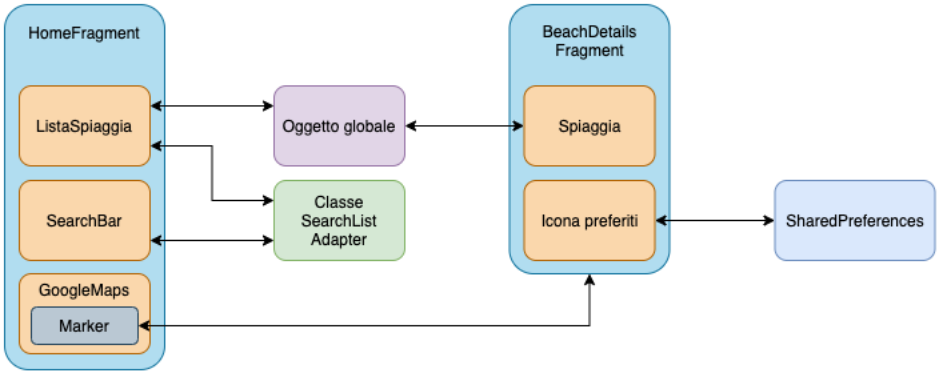

Figura 6.8 – Architettura del ramo "Home"

Il primo Fragment che viene visualizzati all'interno del navigation host della MainActivity è denominato "HomeFragment". Al suo interno, oltre che contenere una mappa, generata attraverso le API di Google Maps, presenta una SearchBar[19].
Il meccanismo implementato per garantire il funzionamento di tale barra di ricerca prevede che il Fragment istanzi al suo interno una variabile contenente la lista delle spiagge invocando l'oggetto Singleton. Fatto ciò, HomeFragment rimane in ascolto di un click sulla SearchBar ed eventualmente confronta il testo inserito nel campo di ricerca con i nomi delle spiagge.
La lista dei risultati viene aggiornata attraverso un Adapter, nello specifico dal SearchListAdapter.

19 https://developer.android.com/guide/topics/search

6.2.6 Adapter

In generale un Adapter è un componente che si occupa della rappresentazione grafica dei dati e dell'interazione con essi, per ogni elemento di una lista in cui i suoi item hanno tutti la medesima struttura.
Nell'applicazione Android in esame in questa sezione, questo elemento, oltre che per lo scopo appena citato, è stato utilizzato per popolare i Fragment dinamici dell'Activity principale, nello specifico le schermate con i dettagli delle spiagge o i dettagli dello storico di un dato inquinante.
Scendendo nel dettaglio, la schermata è stata considerata come l'unico item visibile di una lista ed è stata popolata opportunamente attraverso un apposito Adapter.
Questo meccanismo è stato implementato attraverso il Data Binding (approfondito nel paragrafo seguente).

6.2.7 DataBinding

Il Data Binding[20] è il meccanismo che stabilisce una connessione e/o una sincronizzazione tra la UI e la business logic e, volendo sintetizzare, i Model. Questa connessione permette che il cambiamento della base dati si rifletta sulla UI e nel particolare sulle varie Views senza scrivere ulteriori righe di codice.
Nell'applicazione soggetto di questo elaborato viene utilizzata questa tecnica per popolare tutte gli elementi dinamici delle varie schermate dipendenti dai valori presi letti in un file JSON.

20 https://developer.android.com/topic/libraries/data-binding

6.2.8 Bundle

Cliccando su uno dei marcatori della mappa presente nell' "HomeFragment", è possibile giungere al "BeachDetailsFragment". Al suo interno sono mostrate le informazioni relative alla situazione corrente di inquinamento della spiaggia indicata dall'utente.
Per rendere questa schermata dinamica occorre passare degli argomenti attraverso la chiamata che viene fatta a tale Fragment. Lo scopo è quello di leggere l'id della spiaggia selezionata negli argomenti ricevuti e popolare la schermata con i relativi dati.
In Kotlin per inviare e ricevere valori tra le Activity e/o tra i Fragment è possibile racchiudere una o più informazioni sotto forma di coppie chiave valore in un Bundle[21]. Quest'ultimo è semplicemente una map tra chiavi di tipo String e valori.
A questo punto il "BeachDetailsFragment" non deve far altro che estrarre dal Bundle l'id della spiaggia da visualizzare e, leggendone i relativi dati nell'oggetto globale popolare i propri campi.
Oltre a questo, tale Fragment deve valutare se la spiaggia è indicata come preferita dall'utente e quindi settare di conseguenza la relativa icona. Ciò è possibile andando a leggere le coppie chiave valore contenute nell'istanza delle SharedPreferences all'interno del file globale.

6.2.9 Schema di architettura – Ramo "Preferiti"

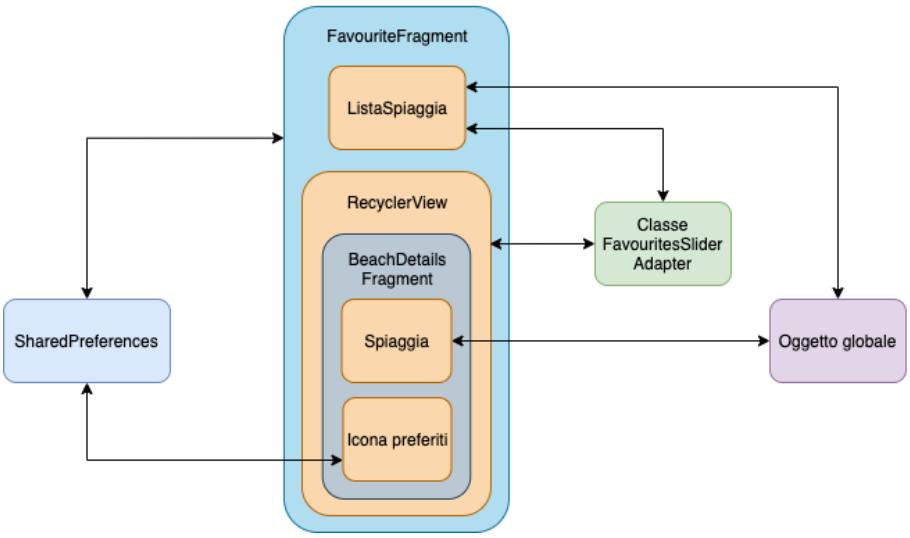

Figura 6.9 – Architettura del ramo "Preferiti"
Il "FavouriteFragment", accessibile dal tasto "Preferiti" della BottomNavigationBar, presenta al suo interno una RecyclerView[22] i cui item sono composti dal Fragment "BeachDetailsFragment".

21 https://developer.android.com/reference/android/os/Bundle
22 https://developer.android.com/guide/topics/ui/layout/recyclerview

6.2.10 RecyclerView

Generalmente, quando ci si trova a dover gestire delle liste, nonostante venga visualizzato solo un sottoinsieme di elementi sullo schermo, in realtà l'applicazione deve creare il layout anche per tutti gli elementi che non sono visibili nella schermata, inoltre, ogni elemento visualizzato deve essere salvato in memoria per poterlo mostrare in un secondo momento all'utente senza doverlo ricreare.
Il RecyclerView utilizza un approccio diverso per la gestione delle liste. Anziché creare tutti gli elementi della lista, durante lo scroll, esso mantiene in una coda, chiamata "*recycler bin*" (cestino per il riciclaggio), alcuni degli elementi precedentemente visualizzati per riutilizzarli in un secondo momento. Infatti, durante lo scorrimento della lista, il RecyclerView recupererà dalla coda un elemento e lo popolerà con le nuove informazioni da mostrare all'utente.
La lista orizzontale dei preferiti, come tutte le altre liste presenti nell'applicazione sono gestite tramite questo meccanismo. Affinché gli elementi possano essere creati a partire dall'oggetto globale contenente i dati delle rilevazioni di tutte le spiagge, occorre affiancare al RecyclerView un Adapter che popoli i vari item: in questo specifico caso esso è denominato "FavouritesSliderAdapter".
È importante segnalare come esso si interfacci anche con le SharedPreferences, in quanto deve reperire da esse la lista delle spiagge da visualizzare come preferite.

6.2.11 Schema di architettura – Ramo "Classifica"

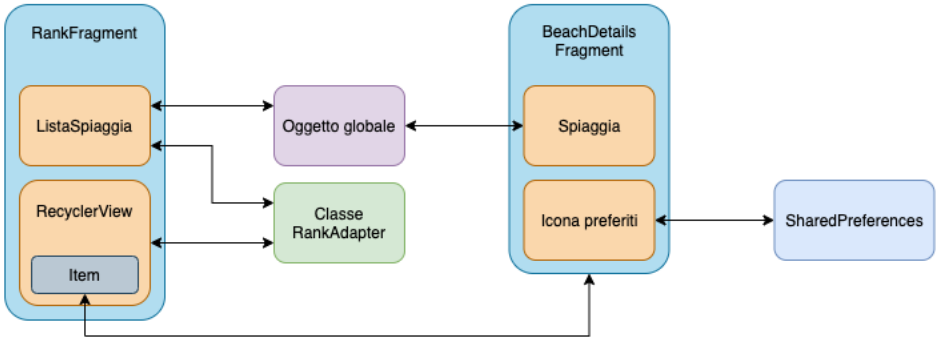

Figura 6.10 – Architettura del ramo "Classifica"

Tale ramo è molto simile a quello definito per la "Home", l'unica differenza è che il "BeachDetailsFragment" è accessibile a partire dagli item della RecyclerView presente nel "RankFragment". Quest'ultimo per popolare i suoi elementi si appoggia alla classe "RankAdapter" con un meccanismo simile a quello appena enunciato per il ramo "Preferiti".

6.2.12 Schema di architettura – "Storico"

Dato che la schermata generata dal Fragment "BeachDetails" è presente in molti dei grafici appena analizzati, per semplicità, gli sviluppi a partire da essa vengono trattati in maniera separata.

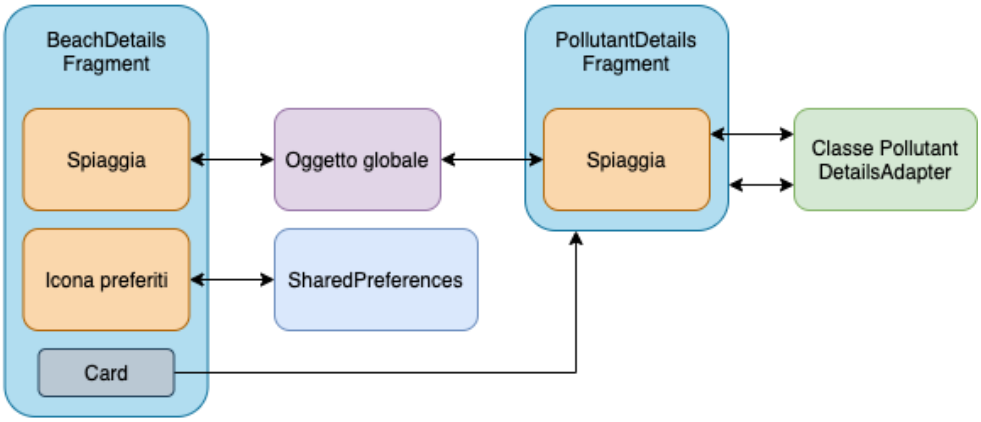

Figura 6.11 – Architettura dello "Storico"

Tale Fragment contiene delle sezioni che mostrano il valore corrente del relativo inquinante. Cliccando su di esse è possibile raggiungere lo storico dell'inquinante selezionato.
Nello specifico, viene popolato un Bundle con il nome dell'inquinante e l'id della spiaggia selezionata dall'utente cosicché il Fragment ricevente, denominato "PollutantDetailsFragment", possa risalire alle informazioni necessarie recuperandole dall'oggetto globale e popolare così la schermata attraverso la classe "PollutantDetailsAdapter".

6.2.13 Schema di architettura – Ramo “Info”

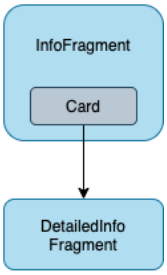

Figura 6.12 – Architettura del ramo “Info”

Quello presentato in figura è sicuramente il ramo meno complesso dell’intera architettura: è composto da soli due Fragment e tra di loro vi è solo una possibile interazione.
L’ “InfoFragment” contiene delle sezioni informative che mostrano una preview dell’articolo completo che verrà visualizzato nel “DetailedInfoFragment”. Anche in questo caso viene utilizzato un Bundle per comunicare al Fragment appena citato quale articolo deve essere mostrato a seconda di quale preview viene selezionata dall’utente.

6.3 Frammenti di codice salienti

Nei paragrafi che seguono verranno descritti i frammenti di codice nei quali sono state adottate delle soluzioni da evidenziare.

6.3.1 Creazione oggetto Singleton

Di seguito vengono riportati i due passaggi fondamentali nella creazione dell'oggetto a singola istanza che conterrà l'insieme dei valori di inquinamento rilevati nei vari tratti costieri e le preferenze dell'utente.
Come già discusso nel paragrafo 6.2.3, tale oggetto contiene due attributi:

- "spiagge": popolato invocando il metodo "setSpiagge" (analizzato di seguito);
- "prefs": inizializzato come vuoto, questo attributo conterrà un'istanza delle shared preferences dell'applicazione.

Nello specifico, nella figura sottostante viene mostrata la sezione di codice in cui viene popolata l'unica istanza dell'oggetto globale.

```
/**
 * Oggetto globale a singola istanza contentente le shared preferences
 * e la lista delle spiagge con i relativi attributi.
 */
object global {
    var spiagge : ListaSpiagge? = null
    @JvmName( name: "setSpiaggeGlobal")
    fun setSpiagge (lista: ListaSpiagge){
        spiagge = lista
    }
    var prefs: SharedPreferences? = null
}
```

Figura 6.13 – Oggetto Singleton

Il metodo "setSpiagge" effettua la fase di fetch, e di conseguenza quella di parse, del file JSON contenente i dati necessari al funzionamento dell'applicazione.
Entrambe le fasi sono state effettuate con l'utilizzo di librerie esterne; nello specifico per la fase ti fetch è stata utilizzata "OkHttp" [11], mentre per quella di parse "Gson" [12].

Di seguito ne viene riportata la parte di codice più importante.

```
/**
 * Classe che effettua il processo di fetch e di parse del file JSON
 * contente i dati relativi all'inquinamento delle spiagge analizzate.
 */
class JSONFetch {

    fun fetchParseJson() {
        val url = "https://pastebin.com/raw/2WAsKh2K"
        val request = Request.Builder().url(url).build()
        val client = OkHttpClient()
        client.newCall(request).enqueue(object : Callback {
            override fun onResponse(call: Call, response: Response) {
                val body = response?.body?.string()
                val gson = GsonBuilder().create()
                val spiagge= gson.fromJson(body, ListaSpiagge::class.java)
                global.setSpiagge(spiagge)
            }

            override fun onFailure(call: Call, e: IOException) {
                println("Failed to execute request")
            }
        })
    }
}
```

Figura 6.14 – Fetch dei dati

Affinchè tale metodo funzioni correttamente occorre che l'intero JSON venga mappato attraverso una classe in grado di contenere tutti gli attributi del file. In questo caso ne è stata creata una, denominata "Spiaggia", che contiene tutte le informazioni relative ad un singolo tratto di costa, ed è stata creata una lista di istanze di questa classe.

6.3.2 Stack di navigazione

In questo paragrafo viene mostrato il frammento di codice XML nel quale vengono settati i quattro stack di navigazione, uno per ogni tasto della Bottom Navigation Bar. Le righe di codice mostrate in figura sono contenute nel file *../res/navigation/navigation.xml.*

```
<navigation xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/navigation"
    app:startDestination="@id/home">

    <include app:graph="@navigation/home"/>
    <include app:graph="@navigation/favourites"/>
    <include app:graph="@navigation/rank"/>
    <include app:graph="@navigation/info"/>

</navigation>
```

Figura 6.15 – Stack di navigazione

Le istruzioni contenenti il tag <*include*> fanno riferimento a quattro file, in essi sono inseriti tutti i percorsi di navigazione possibili. Di seguito ne viene riportato uno, ad esempio, in quanto gli altri sono simili ad esso.

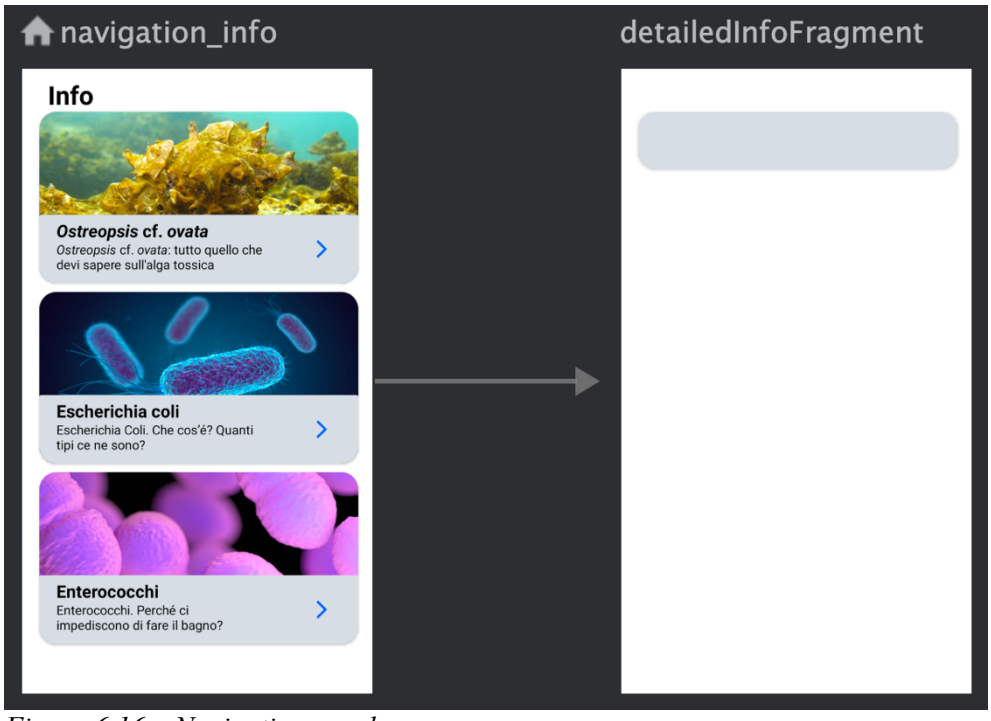

Figura 6.16 – Navigation graph

6.3.3 Mappa

Nella figura seguente viene mostrata la sezione di codice, contenuta nella classe "HomeFragment", in cui vengono aggiunti i marcatori alla mappa presente nella homepage dell'applicazione.
Nelle prime righe di codice si può notare come venga settata la variabile "*color*" in base alla presenza dell'alga *Ostreopsis* cf. *Ovata* nel tratto di costa relativo al marker. Questa è, naturalmente, una versione di prova della funzione finale in quanto poi i colori dei marcatori saranno determinati dalla qualità dell'acqua e non dalla presenza dell'alga. In sostanza, però, il procedimento è pressoché lo stesso. Per questo motivo si ritiene che questo esempio, sebbene riporti del codice di prova, sia comunque valido al fine di mostrare la natura di tale metodo.

```
/**
 * Aggiunta di marcatori (colorati in base alla situazione dell'alga inquinante).
 */
global.spiagge?.forEach { it: Spiaggia
    val color: Float =
        when (it.ostreopsis[0].value) {
            in 0..10000 -> BitmapDescriptorFactory.HUE_GREEN
            in 10000..30000 -> BitmapDescriptorFactory.HUE_ORANGE
            else -> BitmapDescriptorFactory.HUE_RED
        }
    googleMap.addMarker(
        MarkerOptions()
            .position(LatLng(it.coordinates.x.toDouble(), it.coordinates.y.toDouble()))
            .title(it.name)
            .snippet(it.id)
            .icon((BitmapDescriptorFactory.defaultMarker(color)))
    )
}
```

Figura 6.17 - Marcatori

6.3.4 SearchView

In questo paragrafo vengono riportati i passaggi essenziali dell'implementazione della barra di ricerca presente nella homepage, contenuti nella classe "HomeFragment".
Con la figura seguente viene mostrata l'inizializzazione della lista che conterrà i risultati di ricerca e l'associazione dell'oggetto "search" al relativo elemento XML.
Nella settima riga di codice viene invocato il metodo "clear( )" sulla variabile "displayList" per evitare di mostrare l'intera lista delle spiagge non appena l'utente clicchi sulla barra di ricerca.

```
/**
 * Associazione della searchview al corrispettivo in xml e dichiarazione della lista
 * "displayList" ottenuta filtrando la precedente (homeList).
 */
val search: androidx.appcompat.widget.SearchView = requireView().findViewById(R.id.search_bar)
val displayList: ListaSpiagge = homeList.clone() as ListaSpiagge
displayList.clear()

/**
 * Inizializzazione della recyclerview, all'adapter passiamo come parametri (oltre alla
 * lista da visualizzare) la mappa per ottenere l'animazione del movimento di camera al
 * click sugli elementi della recyclerview e la searchiew per cambiare il focus della barra
 * di ricerca una volta che avviene il submit del testo.
 */
recyclerView.adapter = SearchListAdapter(displayList, googleMap, recyclerView, search)
```

Figura 6.18 – Lista dei risultati della ricerca

Altro aspetto fondamentale (implementato con il codice visibile nell'immagine sottostante) è l'override di due metodi essenziali per il funzionamento della barra di ricerca:

- onQueryTextSubmit;
- onQueryTextChange.

Il primo gestisce gli eventi che scaturirebbero cliccando sul tasto "Invio" mentre si digita il nome della spiaggia che si vuole ricercare. Per scelta progettuale, è sostanzialmente un metodo vuoto. Infatti, per avere la sicurezza di essere reindirizzati ad una spiaggia esistente è necessario che l'utente clicchi su uno dei risultati che compariranno al di sotto della barra di ricerca. Il tasto "Invio" non genererà alcuna azione.
Il secondo, invece, aggiorna la lista dei risultati man mano che lo user inserisce del testo nella barra di ricerca. Gli elementi, contenenti nel proprio nome il testo inserito dall'utente, saranno presenti in tale lista.

```
/**
 * Metodi della searchview che intercettano il submit del testo da parte dell'utente
 * la homeList viene filtrata in base alla query e il tutto viene salvato nella displayList
 * che verrà poi visualizzata dalla recyclerview.
 */
search.setOnQueryTextListener(object: SearchView.OnQueryTextListener,
    androidx.appcompat.widget.SearchView.OnQueryTextListener {
    override fun onQueryTextSubmit(newText: String?): Boolean {
        return false
    }
    override fun onQueryTextChange(newText: String?): Boolean {
        /**
         * Ad ogni tasto premuto dalla keyboard dell'utente la lista da visualizzare viene
         * svuotata e riempita con tutti gli elementi che contengono la stringa della query;
         * infine l'adapter è notificato del cambiamento.
         */
        if (newText!!.isNotEmpty()) {
            displayList.clear()
            homeList.forEach {  it: Spiaggia
                if (it.name.toLowerCase().contains(newText.toLowerCase())) displayList.add(it)
            }
        } else displayList.clear()
        recyclerView.adapter!!.notifyDataSetChanged()
        return false
    }
})
```

Figura 6.19 - onQueryTextChange

Nella figura seguente è possibile visualizzare, nella parte superiore, le istruzioni con cui vengono popolati gli item della lista dei risultati e, nella parte inferiore, le righe di codice che si occupano di effettuare le azioni che seguono la selezione della spiaggia.
Quest'ultime, nello specifico, rilevano l'evento attraverso il metodo "setOnCLickListener"[23] e di conseguenza svolgono una serie di azioni:

- Svuotano la lista per mostrare l'animazione;
- Svuotano la barra di ricerca;
- Tolgono il focus alla tastiera così che essa scompaia;
- Effettuano l'animazione portando aumentando lo zoom e centrandolo sul marcatore relativo alla spiaggia cercata dallo user.

Le istruzioni sono contenute nella classe "SearchViewHolder".

23 https://developer.android.com/reference/android/view/View.OnClickListener

```
class SearchViewHolder(val view: View, val map: GoogleMap, val recyclerView: RecyclerView,
                       val list: ListaSpiagge, val searchView: SearchView): RecyclerView.ViewHolder(view) {
    fun bind(beach_item: Spiaggia) {
        /**
         * Riempimento degli elementi grafici.
         */
        view.findViewById<TextView>(R.id.nome_spiaggia).text = beach_item.name
        view.findViewById<TextView>(R.id.id_spiaggia).text = beach_item.id

        /**
         * Una volta che viene cliccato un elemento della recyclerview, la lista da visualizzare
         * viene pulita così da poter visualizzare l'animazione della mappa al di sotto della stessa.
         * La camera viene centrata sulle coordinate della spiaggia selezionata dal click.
         */
        view.findViewById<CardView>(R.id.spiaggia)
            .setOnClickListener {  it: View!
                list.clear()
                recyclerView.adapter!!.notifyDataSetChanged()
                searchView.setQuery( query: "",  submit: false)
                searchView.clearFocus()
                val latLng = LatLng(
                    beach_item.coordinates.x.toDouble(),
                    beach_item.coordinates.y.toDouble()
                )
                map.animateCamera(
                    CameraUpdateFactory.newLatLngZoom(latLng, 17F)
                )
            }
    }
}
```

Figura 6.20 – Click sulla barra di ricerca

6.3.5 Lista preferiti

Il metodo mostrato nell'immagine seguente, appartenente alla classe "FavouritesFragment", è il responsabile della creazione della lista delle spiagge preferite.
Nel dettaglio, tale metodo copia in una variabile locale il valore contenuto nel file delle shared preferences e scorre tutte le coppie chiave valore contenute al suo interno. Se il valore è pari a "1" (true), inserisce la spiaggia con "id" uguale alla chiave nella lista.

```
/**
 * Metodo che si occupa di creare la lista contentente le spiagge indicate come preferite.
 */
fun createListFav(): MutableList<Spiaggia> {
    val map = global.prefs?.all
    val listFav = mutableListOf<Spiaggia>()
    map?.forEach { i ->
        if (i.value == 1) {
            val item = global.spiagge?.find { it.id == i.key }
            item?.let { listFav.add(it) }
        }
    }
    return listFav
}
```

Figura 6.21 – Creazione lista preferiti

6.3.6 Dettagli spiaggia

In questa sezione sono riportate alcuni dei più importanti frammenti di codice, contenuti nella classe "BeachDetailsFragment", che popolano la schermata "Dettagli spiaggia".
Per settare i graph ring [13], nel codice che segue, per ognuno dei tre Widget, viene popolato l'attributo "progress" passando il valore corrispondente letto nell'oggetto globale (*definito nel paragrafo 6.3.1*). Se il valore letto è pari a "0", allora, per evitare che l'anello risulti completamente vuoto, viene passato un valore pari o inferiore all'1% del massimo valore indicabile nel grafico.
Altro attributo ad essere settato è "progessMax" che viene popolato con il valore di soglia del rispettivo inquinante.

```
/**
 * Comportamento degli anelli grafici, vengono definiti il massimo e il valore attuale
 * che viene letto dall'oggetto globale.
 */
val ostreopsis_ring = view.findViewById<CircularProgressBar>(R.id.circular_progress)
ostreopsis_ring.progress =
    if (spiaggia!!.ostreopsis[0].value.toFloat() == 0f) 30f else spiaggia.ostreopsis[0].value.toFloat()
ostreopsis_ring.progressMax = 30000f

val escherichia_ring = view.findViewById<CircularProgressBar>(R.id.circular_progress2)
escherichia_ring.progress =
    if (spiaggia.escherichia[0].value.toFloat() == 0f) 5f else spiaggia.escherichia[0].value.toFloat()
escherichia_ring.progressMax = 500f

val enterococcus_ring = view.findViewById<CircularProgressBar>(R.id.circular_progress3)
enterococcus_ring.progress =
    if (spiaggia.enterococcus[0].value.toFloat() == 0f) 2f else spiaggia.enterococcus[0].value.toFloat()
enterococcus_ring.progressMax = 200f
```

Figura 6.22 – Graph ring

Con le istruzioni presenti nell'immagine sottostante a tale paragrafo, viene settata l'icona che permette di indicare una spiaggia come preferita o di toglierla dalla stessa lista.

Viene fatto un doppio controllo per settare l'icona, uno prima ed uno in seguito al click andando a consultare la variabile contenente il valore letto nelle shared preferences in corrispondenza dell'id della spiaggia.

```
/**
 * In base alla presenza o meno della spiaggia tra i preferiti viene impostato il
 * drawable da visualizzare come tasto preferiti. Il click sul tasto funge da toggle:
 * se la spiaggia è tra i preferiti allora il valore logico nel file delle preferenze è
 * settato a 0, altrimenti la spiaggia viene aggiunta al file.
 */
if (fav == 0) view.findViewById<ImageView>(R.id.fav_star)
    .setImageResource(R.drawable.ic_baseline_star_border_24)
else view.findViewById<ImageView>(R.id.fav_star)
    .setImageResource(R.drawable.ic_baseline_star_24)
view.findViewById<ImageView>(R.id.fav_star).setOnClickListener { it: View!
    fav = global.prefs?.getInt(id, defValue: 0)
    if (fav == 0) {
        global.prefs?.edit()?.putInt(id, 1)?.apply()
        view.findViewById<ImageView>(R.id.fav_star)
            .setImageResource(R.drawable.ic_baseline_star_24)
    } else {
        global.prefs?.edit()?.putInt(id, 0)?.apply()
        view.findViewById<ImageView>(R.id.fav_star)
            .setImageResource(R.drawable.ic_baseline_star_border_24)
    }
}
```

Figura 6.23 – Icona dei preferiti

Nel codice mostrato in figura viene mostrato come, al click dell'icona della posizione della spiaggia, viene lanciato un Intent che apre un'applicazione per la visualizzazione di mappe e percorsi. Se presente sul dispositivo, generalmente, viene aperta Google Maps, ma la scelta è riservata all'utente. La query fa in modo che all'apertura dell'applicazione esterna venga visualizzato il luogo identificato dalle coordinate della spiaggia in questione.

```
view.findViewById<ImageView>(R.id.place_icon).setOnClickListener { it: View!
    val mapIntent = Intent(
        Intent.ACTION_VIEW,
        Uri.parse(
            uriString: "https://www.google.com/maps/search/?api=1&parameters"
                    + "&query=" + spiaggia.coordinates.x + "," + spiaggia.coordinates.y
        )
    )
    startActivity(mapIntent)
}
```

Figura 6.24 – Click sull'icona della posizione

6.3.7 Grafico storico inquinanti

Nel frammento di codice che segue, appartenente alla classe "PollutantDetailsFragment", si può notare come, dopo aver definito il formato della data che si intende utilizzare, nelle istruzioni a seguire, vengono inizializzate due liste di valori vuote, la prima che andrà a popolare l'asse delle x e la seconda quello delle y del grafico relativo allo storico dell'inquinante indicato.
Quest'ultimo è composto da una lista di oggetti contenenti due attributi: la data della rilevazione ed il valore rilevato.
Scorrendo questa lista dall'oggetto rappresentante la rilevazione più recente a quello più indietro nel passato, vengono riempite le due liste di cui sopra: una con i valori e una con le date.
Al termine di tale processo, le liste vengono invertite attraverso il metodo "reverse( )" per permettere all'utente una visualizzazione cronologicamente corretta.

```
/**
 * Settaggio del grafico:
 * - Popolamento delle liste contenti i paramentri delle ascisse e delle ordinate
 */
val format = SimpleDateFormat( pattern: "dd/MM/yyyy")
val xvalues = ArrayList <String>()
val yvalues = ArrayList <Entry> ()
var index = date_value.size -1

date_value.forEach{ it: Values
    xvalues.add(format.format(Date( date: it.date*1000)).toString())
    yvalues.add(Entry(it.value.toFloat(), index))
    index--
}

xvalues.reverse()
yvalues.reverse()
```

Figura 6.25 – Grafico storico inquinanti

Capitolo 7 – App cross-platform

In questo capitolo viene posta l'attenzione sull'applicazione cross-platform "AQUA". Essa costituisce la parte front-end dell'intero progetto, mentre la parte di back-end, ossia la parte che si incentra sull'accesso ai dati, ai servizi e agli altri sistemi esistenti che consentono il funzionamento dell'app, verrà trattata dal collega Angelini Giordano nel suo lavoro di tesi.
Di seguito vengono quindi analizzate i cambiamenti grafici dell'interfaccia utente e le innovazioni di questa versione rispetto alla precedente, illustrata nei capitoli 5 e 6 di tale elaborato. Verranno indicati, inoltre, eventuali sviluppi futuri per l'applicativo.

7.1 Flutter

Flutter [14] è un framework per lo sviluppo di app per diverse piattaforme. Esso offre una vasta serie di librerie di elementi di interfaccia utente (UI) standard, di Android e iOS, ma è adatto anche per lo sviluppo di applicazioni web o desktop tradizionali.
Le applicazioni sviluppate con Flutter hanno l'aspetto delle app tipiche del sistema a cui sono destinate, comportandosi come quest'ultime, senza che sia necessario scrivere un codebase differente per ogni piattaforma.
Tale framework è disponibile per Android Studio ma, per poter eseguire l'applicazione anche su emulatori iOS, occorre installare sulla macchina sulla quale si intende sviluppare l'app anche Xcode, ambiente di sviluppo Apple.
L'intera applicazione è stata sviluppata utilizzando Dart [15], linguaggio di programmazione open source sviluppato da Google.

7.1.2 Widget

La strategia di Flutter "tutto è un Widget"[24] applica la programmazione orientata agli oggetti a tutto, includendo l'interfaccia utente: l'interfaccia di un programma è così composta da vari Widget, che possono essere nidificati gli uni negli altri. Ogni pulsante, grafico, forma, animazione e testo è un Widget contenente diverse caratteristiche che possono essere modificate. Essi possono influenzarsi a vicenda e reagire, tramite metodi integrati, a cambiamenti di stato dall'esterno.
Per tutti gli elementi principali dell'interfaccia utente vengono forniti in dotazione i rispettivi Widget che soddisfano i requisiti di progettazione di Android e iOS e/o delle più comuni applicazioni web. Tutto ciò per garantire all'utente una *user-experience* ottimale, mantenendo un *look and feel* simile a quello delle applicazioni native.
La libreria Material Design [16] fornisce molti altri componenti di base per la u.i., tra cui Cupertino[25], la libreria attraverso la quale flutter riproduce lo stile grafico corrente del sistema operativo iOS.
Se necessario, i Widget possono essere ampliati con funzioni aggiuntive. Altrimenti, è possibile creare Widget personalizzati che possono essere combinati perfettamente con quelli esistenti.

24 https://flutter.dev/docs/development/ui/widgets
25 https://api.flutter.dev/flutter/cupertino/cupertino-library.html

Durante la creazione del layout di un'app ci troviamo a dover classificare i suoi componenti in due categorie:

- layout statico
- layout dinamico

Il primo è quel tipo di layout che non cambia una volta che è stato costruito e nessuna azione o evento generato dall'utente potrà modificare gli elementi che lo compongono. Contrariamente al precedente, il secondo si modifica in base alle attività e agli eventi legati agli utenti.
Per venire incontro all'esigenza di dover modellare queste due tipologie di layout, sono state definite in Flutter due tipi di Widget, Stateful[26] e Stateless[27] Widget, utilizzate rispettivamente per il layout dinamico e statico. Entrambe le classi si differenziano in un unico aspetto: l'abilità di ricaricare il Widget a runtime.

Gli StateLess Widget, una volta creati, non subiranno alcuna variazione e non modificheranno il proprio comportamento nemmeno in base a eventi o azioni dell'utente.
Per creare uno stateless Widget deve essere estesa la classe StatelessWidget che richiede la sovrascrittura del metodo *"build( )"* per definire la porzione di interfaccia utente rappresentata dal Widget in questione.

Contrariamente ai primi, gli stateful Widget sono dinamici: permettono di cambiare il loro contenuto nel tempo in base alle azioni o agli eventi scaturiti dall'utente e non sono relativi a un qualche stato immutabile passato all'inizializzazione dell'oggetto. I Widget Stateful sono pertanto utili quando la porzione dell'interfaccia utente che si sta realizzando può cambiare in modo dinamico.
Allo stesso modo dei precedenti, gli stateful Widget vengono creati estendendo la classe StatefulWidget che permette di rendere dinamico lo stato del Widget. Questa tipologia di Widget è, infatti, caratterizzata dal Widget State che contiene i dati del Widget. Nello specifico, essopuò essere letto in modo sincrono quando il Widget viene creato e potrebbe cambiare durante il ciclo di vita del Widget stesso.
Lo State quindi gestisce le informazioni per interagire con il Widget in termini di comportamento e layout. Pertanto, ogni volta che avverrà un cambiamento nello State ciò forzerà una ricreazione del Widget al fine di apportare le modifiche, rendendole visibile all'utente finale.

26 https://api.flutter.dev/flutter/widgets/StatefulWidget-class.html
27 https://api.flutter.dev/flutter/widgets/StatelessWidget-class.html

7.2 Aspetto User Interface app cross-platform

In questo paragrafo vengono riportate le immagini che descrivono l'aspetto finale dell'applicazione "AQUA". È sicuramente interessante notare come da una versione all'altra dell'app ci sia stato un notevole cambiamento estetico dell'interfaccia utente, non solo dovuto alla modifica di alcune funzionalità, ma anche ad un restyling puramente grafico che ha interessato colori, forme e dimensioni dei vari elementi che compongono la user interface.

Figura 7.1 – SplashScreen app Flutter

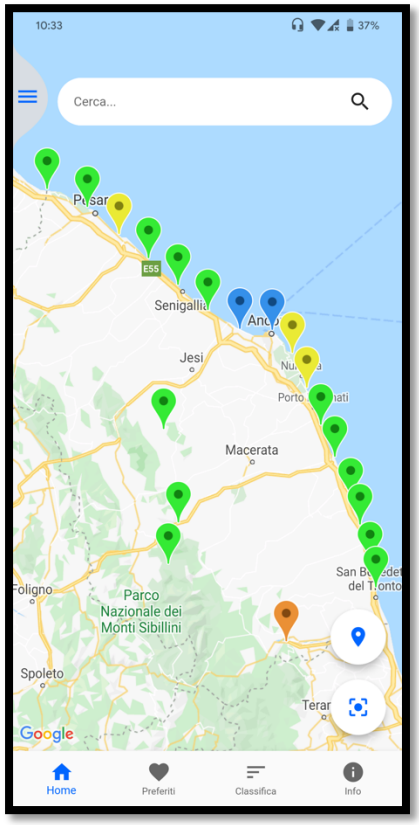

Figura 7.2 – Home app Flutter

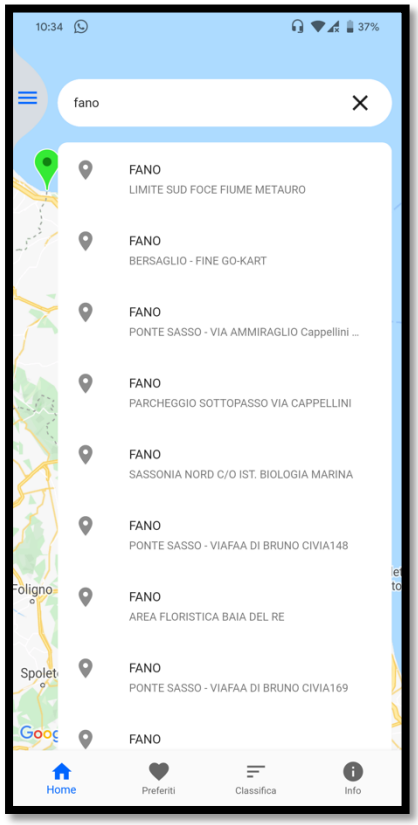

Figura 7.3 – Ricerca app Flutter

Figura 7.4 – Notifiche app Flutter

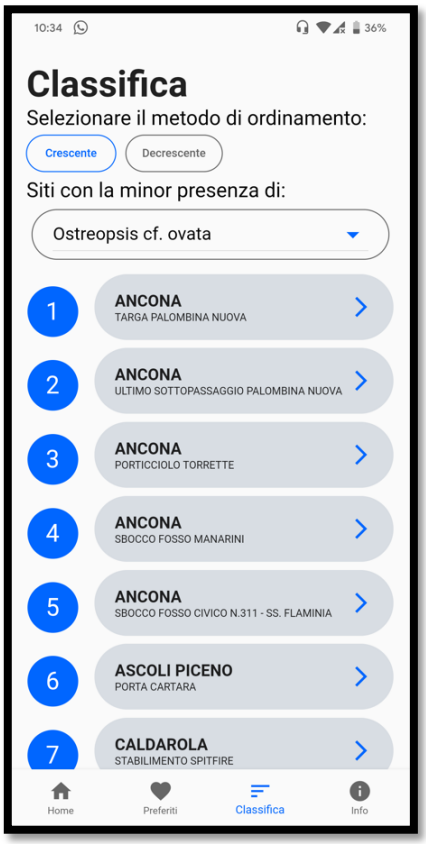

Figura 7.5 – Classifica app Flutter

Figura 7.6 – Preferiti app Flutter

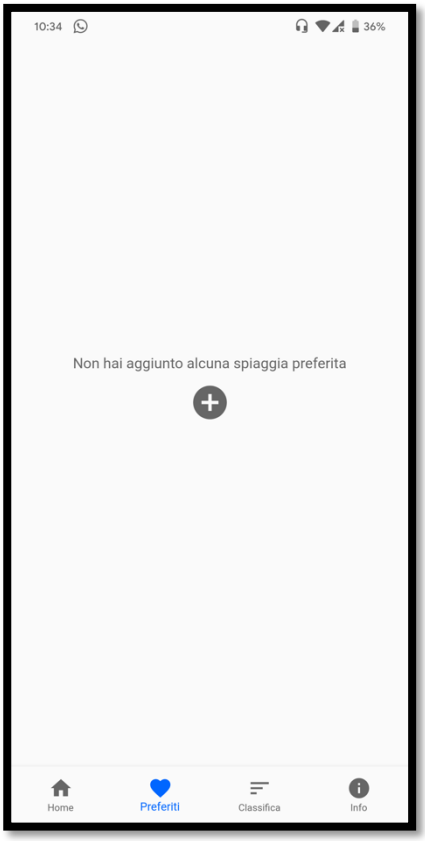

Figura 7.7 – Assenza di preferiti app Flutter

Figura 7.8 – Dettagli spiaggia app Flutter

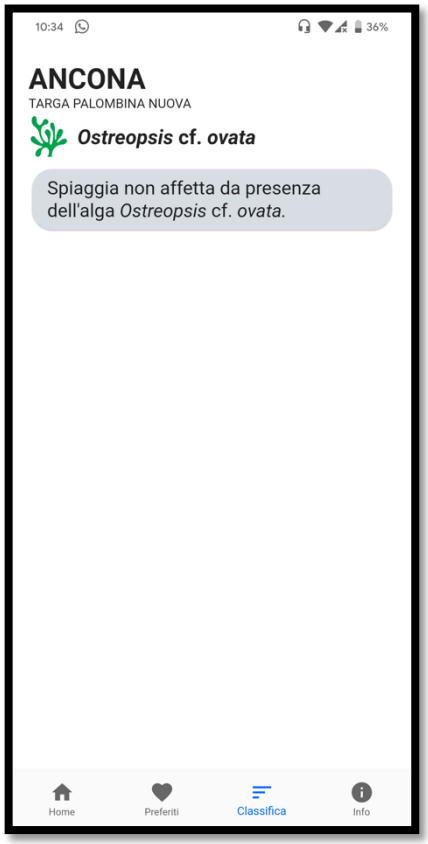

Figura 7.9 – Assenza storico Ostreopsis cf. Ovata app Flutter

Figura 7.10 – Storico Ostreopsis cf. Ovata app Flutter

Figura 7.11 – Storico seconda parte app Flutter

Figura 7.12 – Storico Enterococchi app Flutter

Figura 7.13 – Storico Escherichia coli app Flutter

Figura 7.14 – Info Preview app Flutter

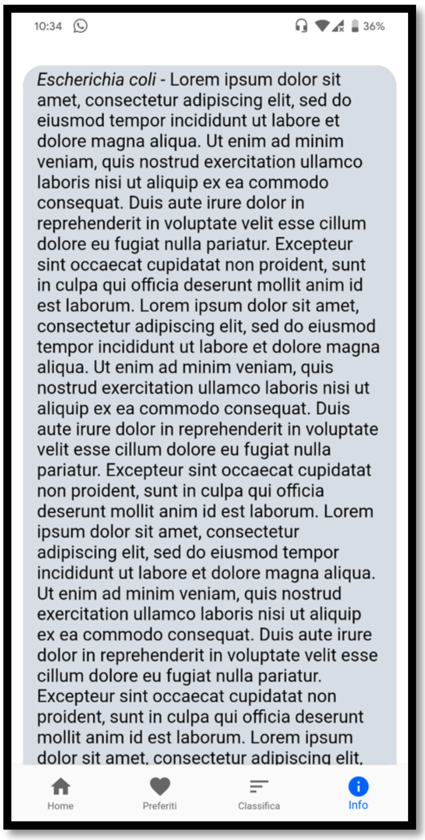

Figure 7.15 – Info Dettagli app Flutter

7.3 Changelog

Il termine changelog è universalmente utilizzato nel campo della progettazione, soprattutto in informatica, relativamente alla progettazione di software; esso è un registro di tutte le modifiche notevoli apportate a un progetto.
Di seguito, quindi, vengono riportate tutte le innovazioni che la versione di "AQUA" analizzata in questa sezione presenta rispetto alla versione disponibile solo su piattaforma Android.

7.3.1 Informazioni mostrate

In questa versione, l'app, utilizzando il sistema back-end del progetto "AQUA" e le relative rotte per acquisire le informazioni necessarie per il suo funzionamento, è in grado di mostrare all'utente non più dei dati rappresentanti una simulazione ma la situazione corrente dell'inquinamento delle coste marchigiane.
Ciò porta l'applicazione a dover gestire situazioni che nella scorsa versione, a causa della staticità dei dati utilizzati, non era tenuta a considerare.
Nello specifico i due principali problemi da risolvere sono:

- Gestire la distribuzione dei marcatori sulla mappa in quanto, essendo aumentato notevolmente il numero delle spiagge monitorate, stampare a schermo tutti i marcatori renderebbe la lettura della mappa impossibile da parte dell'utente;
- Effettuare dei controlli sui valori rilevati dato che il valore delle rilevazioni potrebbe essere pari a *NULL* e, inoltre, non in tutte le spiagge viene rilevata la presenza dell'alga tossica *Ostreopsis* cf. *Ovata*.
- La maggior parte dei siti di rilevazione viene contrassegnata con dei nomi che, estraniati dal comune nel quale si trovano, risultano poco comprensibili.

Di seguito vengono analizzate nel dettaglio le soluzioni adottate per risolvere tali problematiche.

7.3.2 Marcatori

In questa versione il colore dei marcatori verrà indicato sulla base del parametro "Qualità dell'acqua" e non più sul valore corrente dell'alga tossica.
Come già affermato il posizionamento di diverse centinaia di marker su una mappa può portare rapidamente a un sovraccarico visivo o impedirne l'analisi.
La prima soluzione provata per esprimere un significato chiaro, e quindi per semplificare le informazioni, è stata quella di raggruppare i punti in cluster e solo con un notevole aumento di zoom mostrare tutti i marcatori.
Le mappe cluster, infatti, aiutano a rappresentare densi pacchetti di dati puntuali usando un singolo punto.

Il problema sostanziale di tale soluzione è che non risulta possibile assegnare un colore al cluster in base alla qualità dell'acqua in maniera coerente. Infatti, una media dei valori relativi ai marker raggruppati in ogni cluster non sarebbe affatto esplicativa della condizione di inquinamento di una determinata area. Quindi, la soluzione adottata è quella di aumentare o ridurre la capillarità della distribuzione dei marker in maniera concorde al livello di zoom.
Mantenendo il livello di zoom di default della mappa viene visualizzato all'incirca un marker ogni 7,5 km; aumentando lo zoom, ne verrà mostrato uno ogni 15 km. Indicando uno zoom pari o superiore a quello a cui si giunge in seguito ad una ricerca effettuata attraverso la SearchBar invece, l'applicazione mostrerà sulla mappa tutti i marcatori corrispondenti alle stazioni di rilevamento degli inquinanti.

7.3.3 Controllo sui valori

Come già affermato nel paragrafo *7.3.1 – Informazioni mostrate,* il trattare dei dati reali impone che l'applicazione effettui diversi controlli affinché non incappi in errori dovuti a valori nulli. Nello specifico i controlli effettuati per evitare ciò sono numerosi e vengono riportati di seguito raggruppati in base alla schermata a cui essi fanno riferimento.

Classifica: dalla classifica vengono esclusi tutti gli elementi contenenti valori nulli per evitare che in fase di ordinamento venga prodotto un errore e quindi la chiusura dell'app in quanto il valore "NULL" non può essere confrontato con nessuna altro valore.
Anche a livello logico questa scelta progettuale mantiene la sua coerenza in quanto non avrebbe senso inserire in una classifica, il cui criterio di ordinamento si basa su un determinato valore, un elemento che ha tale parametro.

Dettagli spiaggia: in questa schermata vengono visualizzati i valori correnti degli inquinanti in due modalità: attraverso una barra progressiva circolare e mostrando un testo contenente il valore corrente, il valore di soglia relativo e l'unità di misura.
Nel primo caso il grafico viene definito come se l'inquinante avesse valori pari a 0, mentre, per quanto riguarda il testo, al posto della stringa appena citata viene stampato a video: "Non rilevato".
Dato che anche le schede contenenti la descrizione della situazione corrente della balneabilità e dell'alga tossica dipendono dai valori correnti degli inquinanti, se un elemento dovesse essere nullo, la scheda rispettiva mostrerebbe la stringa "Non rilevata".

Home: per lo stesso motivo appena citato in "*Dettagli spiaggia*", dato che il colore dei marcatori dipende dalla qualità dell'acqua, è stato previsto un colore nel caso in cui questo parametro risulti pari a "NULL".

Storico: dalla schermata in cui vengono visualizzati i dettagli dello storico di un determinato inquinante, sono stati eliminate tutte le rilevazioni contenenti un valore nullo in quanto ciò avrebbe portato a problemi di visualizzazione del grafico e nella generazione delle medie mensili.

Se, una volta tolti tutti i valori, nulli non rimanesse alcuna rilevazione da mostrare, l'applicazione mostrerebbe un messaggio contenente la stringa "*Non è presente alcuna rilevazione valida dell'inquinante selezionato per questa spiaggia*".
Oltre alla gestione dei valori nulli, occorre considerare anche il fatto che non in tutte le stazioni viene rilevata la presenza dell'alga tossica *Ostreopsis* cf. *Ovata*. Con il ricercatore, identificabile nella figura del committente, è stata valutato tale approccio di risoluzione del problema:

- Le spiagge entro 6 km dalle stazioni in cui viene effettuata tale rilevazione vengono trattate nello stesso modo delle suddette stazioni mostrando i dati relativi ad esse.
- Nelle spiagge oltre questo limite spaziale non vengono effettuate tali rilevazioni in quanto questi siti non sono affetti dalla presenza dell'alga. Di conseguenza, nella schermata dei dettagli della stazione viene visualizzata la grafica che verrebbe stampata nel caso in cui il valore dell'*Ostreopsis* cf. *Ovata* fosse zero, mentre nello storico viene visualizzata il seguente testo: "*Spiaggia non affetta dalla presenza dell'alga Ostreopsis cf. Ovata.*".

7.3.4 Nomi siti di rilevazione

Come già indicato pocanzi, i nomi delle stazioni sono in realtà più delle descrizioni della posizione in riferimento a luoghi di interesse importanti per il comune di appartenenza del sito. Un esempio potrebbe essere "80 M SUD FOSSO SEJORE".
Naturalmente sarebbe impensabile indicare una spiaggia con solo questa informazione, quindi, per ovviare tale problema nell'applicazione vengono riportate due informazioni per far riferimento ad una spiaggia: comune di appartenenza e nome del sito (che da questo punto dell'elaborato di tesi in poi verrà indicato con il termine "descrizione").
Di conseguenza, oltre ad essere state aggiornate tutte le schermate per renderle concordi con tale modifica, anche la funzione di ricerca è stata modificata in quanto, i risultati che compariranno al disotto della barra di ricerca saranno scelti con un ulteriore criterio: verrà valutato se il testo inserito come indice di ricerca è contenuto nella descrizione della spiaggia, oltre che nel nome del comune a cui essa appartiene.

7.3.5 Classifica

Nella schermata che mostra la classifica delle spiagge in base alla presenza di un determinato inquinante, è stata predisposta la possibilità di far scegliere all'utente se visualizzare un ordinamento crescente o decrescente e, di conseguenza, visionare nelle prime posizioni i siti meno inquinati o con una maggiore presenza dell'inquinante selezionato.

7.4 Implementazioni future

Il codice all'interno dei file che compongono il progetto è stato predisposto affinché in futuro possano essere implementate funzionalità aggiuntive.

Nello specifico, nella home è già stato inserito uno Navigation Drawer[28], ossia un menù laterale, che, se aperto, mostra all'utente il pannello di controllo delle notifiche. Da esso è possibile abilitare i due tipi di notifiche trattati nel paragrafo *1.2.1 Requisiti funzionali*: notifiche basate sulla posizione, notifiche basate sui preferiti.
Sono già stati implementati, inoltre, due elementi fondamentali per lo sviluppo di tale funzionalità:

- il meccanismo che, sfruttando le SharedPreferences, mantiene traccia delle preferenze dell'utente espresse in tale pannello;
- i CroneJob: questo elemento permette l'avvio di processi in background per il controllo dei dati mostrati nell'app e il conseguente inoltro delle notifiche in maniera periodica anche quando l'applicazione è chiusa.

Un ulteriore miglioramento futuro potrebbe essere il perfezionamento del meccanismo di visualizzazione dei marcatori definendo un numero di livelli di zoom più elevato rispetto a quelli ora in funzione.

28 https://material.io/components/navigation-drawer

Capitolo 8 – Implementazione applicazione cross-platform

Questo ultimo capitolo verterà interamente sullo sviluppo dell'applicazione "AQUA" nella sua versione cross-platform.
Verranno analizzati la struttura del progetto, l'architettura dell'app e i frammenti di codice salienti sviluppati in dart che meritano un approfondimento.
L'intero progetto è scaricabile dal repository GitHub raggiungibile al seguente link: https://github.com/DiSilvestreCristian/aqua.git

8.1 Struttura del progetto

Flutter non impone una struttura di progetto che rispetti determinati canoni, di seguito, quindi, viene riportata la struttura delle directory pensata per questo progetto.

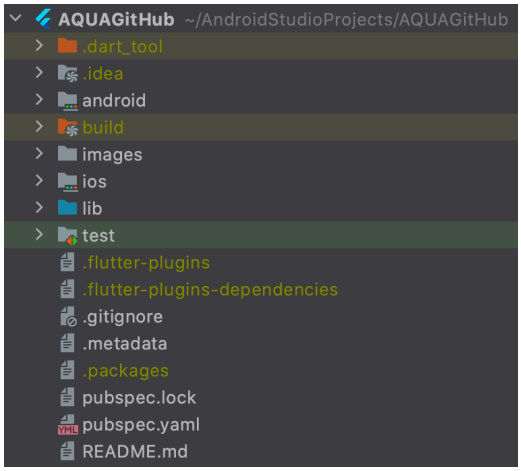

Figura 8.1 – Struttura directory principale progetto Flutter

La directory principale contiene al suo interno diverse cartelle e file, di seguito sono indicati quelli usati e/o creati per la realizzazione dell'applicativo:

- *"android"*: all'interno di tale cartella vi sono i file specifici di un progetto di un'app nativa Andorid, andando a modificarli è possibile fare degli interventi mirati sulla versione Andorid dell'app. Può essere usata per chiedere i permessi di cui l'applicazione ha bisogno per funzionare o settare dei parametri specifici come l'icona dell'applicativo.
- *"ios"*: è il corrispettivo della cartella precedente per il sistema iOS.
- *"build"*: è una cartella autogenerata contenete i file necessari per la build dell'applicazione.
- *"images"*: questa cartella è stata creata appositamente per tale progetto, al suo interno vi sono tutti gli assets esterni che verranno importati all'interno dell'app come immagini o icone.
- *"lib"*: directory approfondita nel paragrafo 7.2.1.
- *"pubspec.yaml"*: file approfondito nel paragrafo 7.3.1.

8.1.1 Directory lib

All'interno della directory "lib" vi sono tuti i file responsabile dell'interfaccia utente, di ciò che essa contiene, della gestione degli eventi legati agli Widget e della logica che c'è dietro.

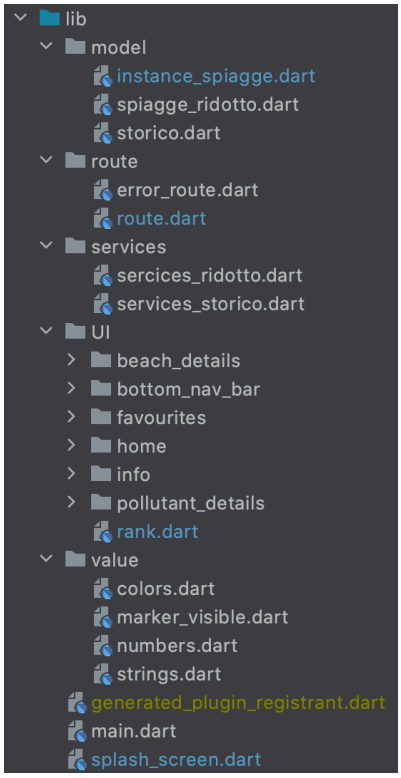

Figura 8.2 – Directory "lib"

Come è possibile notare nella figura 7.2 soprariportata, al suo interno vi sono diverse sottodirectory, ognuna delle quali contiene dei file o ulteriori sottodirectory riguardanti uno stesso ambito del progetto:

- *"model"*: contiene al suo interno i tre file che responsabili della creazione dell'interfaccia verso i dati contenuti nei vari file JSON che l'applicazione scarica per recuperare le informazioni di cui ha bisogno.
- *"route"*: racchiude i file per la gestione della navigazione all'interno dell'app.
- *"services"*: vi sono i file che si interfacciano direttamente con le rotte dei richiesta dati[29] a cui l'applicativo fa riferimento per scaricare i file JSON di cui sopra.
- *"value"*: al suo interno vi sono quattro librerie contenenti i dati statici che vengono usati all'interno dell'applicazione come stringhe, colori, valori numerici.
- *"UI"*: contiene i file che si occupano della gestione dell'interfaccia utente nella sua interezza.

29 http://193.205.129.120:63434/

Nello specifico la directory “UI” ha altre sottodirectory al suo interno su cui vale la pena porre maggiormente l’attenzione.

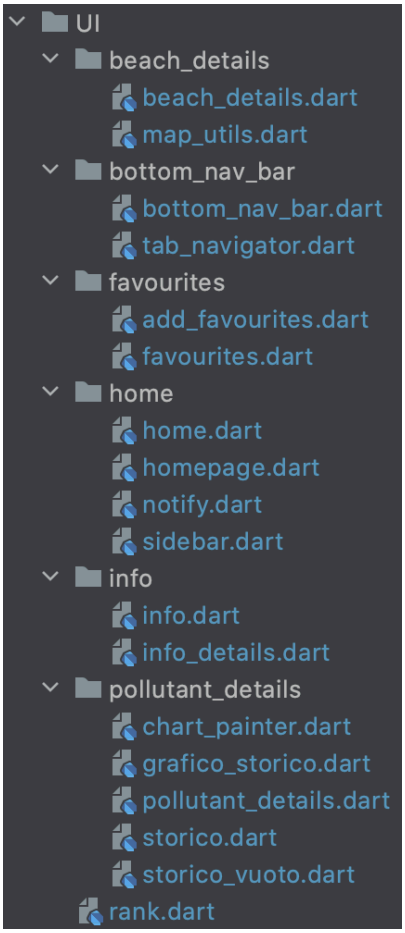

Figura 8.3 – Directory “UI”

Come è visibile nell’immagine 7.3 appena riportata, “UI” è divisa in sette sezioni:

- *“beach_details”*: al suo interno vi sono i file di tipo Dart che si responsabili della schermata che mostra i dettagli della situazione corrente della spiaggia selezionata dall’utente.
- *“bottom_nav_bar”*: contiene il file che definisce le rotte e la logica che c’è dietro i quattro stack per la navigazione all’interno dell’app (meccanismo simile all’app Andorid argomentato nel dettaglio nella sezione x.y) ed il file responsabile dell’aspetto della barra di navigazione.
- *“favourite”*: contiene al duo interno due file, uno responsabile della schermata dei preferiti ed uno che definisce il messaggio da comunicare all’utente nel caso in cui acceda a tale schermata senza aver indicato nessuna spiaggia come preferita.
- *“home”*: racchiude i file che si occupano di gestire tutte le funzionalità e l’interfaccia utente della homepage.

- *"info"*:al suo interno vi sono i file che definiscono la schermata Info e la schermata Dettagli Info, che rispettivamente mostrano le preview dei vari articoli informativi e gli articoli comleti.
- *"pollutant_details"*: contiene al suo interno i file responsabili della creazione e gestione dell'intera schermata che mostra lo storico di un'inquinante selezionato per una determinata spiaggia.
- *"rank"*: è il file che responsabile della schermata Classifica.

8.1.2 File pubspec.yaml

Il pubspec.yaml è un file di configurazione, presente in ogni progetto Flutter, che contiene al suo interno tre sezioni:

- *"Sezione generale (Metadata)"*: in questa sezione vengono definiti dati generali come il nome del pacchetto o il formato per importare librerie nei vari file che comporranno il progetto.
- *"Ambiente"*: questo campo ci consente sia di aggiungere vincoli all'SDK Dart, imponendo che l'applicazione verrà eseguita solo su versioni o SDK comprese in un dato range, sia definire che versione di Flutter si intende utilizzare.
- *"Dipendenze"*: contiene tutti i pacchetti che verranno utilizzati nell'app, indicando generalmente il nome del pacchetto e la versione che si intende utilizzare.

8.2 Architettura dell'app cross-platform

In questo paragrafo verranno analizzate tutte le componenti che compongono l'applicazione in analisi in questa sezione.
Di seguito viene riportato lo schema dell'architettura di alto livello dell'applicazione.

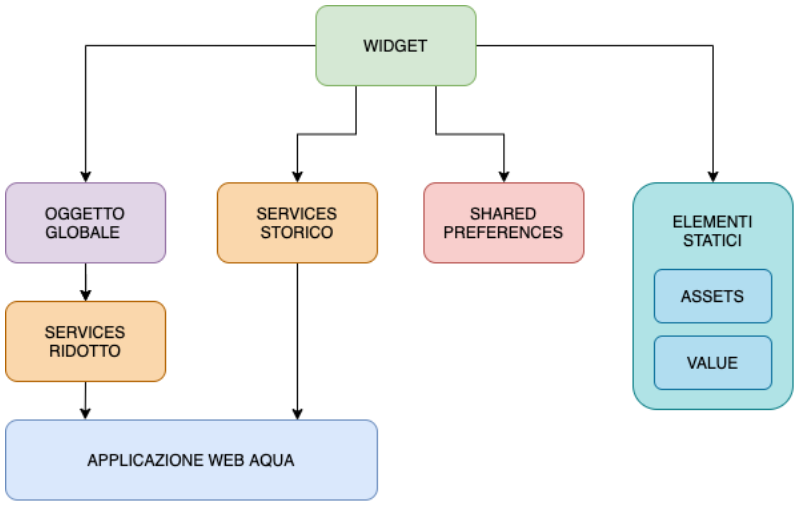

Figura 8.4 - Schema dell'architettura di alto livello dell'app Flutter

È possibile, però, entrare maggiormente nel dettaglio per approfondire come le varie componenti interagiscono fra di loro. Al fine di rendere il grafico che risulterebbe da tale analisi più leggibile esso è stato diviso in 5 parti:

- La fase di avvio dell'applicazione;
- Home;
- Preferiti;
- Rank;
- Info;

Nei paragrafi seguenti sono riportati gli schemi di cui sopra e le relative analisi, ma prima di valutare come le componenti interagiscono fra di loro, è opportuno analizzarle per comprenderne il funzionamento.

8.2.1 Meccanismo di navigazione

La modalità di navigazione che viene utilizzata all'interno dell'app è la Named Route[30].
Le varie rotte all'interno dell'app vengono descritte all'interno di un file apposito e associate ad un identificatore univoco, il nome appunto.
Per navigare da una schermata all'altra, non serve far altro che invocare l'apposito metodo per la navigazione passando come parametro il nome della rotta.

8.2.2 Singleton_instance e SharedPreferences

La classe "singleton_instance" definisce un oggetto di tipo Singleton. Tale pattern è un modello di progettazione che garantisce che una classe abbia una sola istanza e fornisce un punto di accesso globale ad essa.
Tale oggetto contiene al suo interno un attributo che racchiude le rilevazioni correnti di tutte le spiagge monitorate per ognuno dei tre inquinanti presi in analisi.
A differenza dell'omonimo oggetto creato per l'applicazione Android, singleton_instance non ha al suo interno un riferimento all'istanza delle SharedPreferences in quanto Flutter fornisce già un metodo che ritorna sempre la stessa istanza di quest'ultima classe. Il metodo nello specifico è invocabile nel seguente modo:

SharedPreferences prefs = ***await*** *SharedPreferences.getInstance( );*[31]

Di conseguenza, ogni qualvolta che una classe dovrà far riferimento alle SharedPreferences, non verrà invocato l'oggetto Singleton globale ma direttamente il metodo appena riportato.

30 https://flutter.dev/docs/cookbook/navigation/named-routes
31 https://pub.dev/packages/shared_preferences

8.2.3 Applicazione Web AQUA e file JSON

In questa sezione vengono analizzati i servizi dell'applicazione web "AQUA" utilizzati dall'omonima app mobile e, quindi, i risultati di tali interazioni.

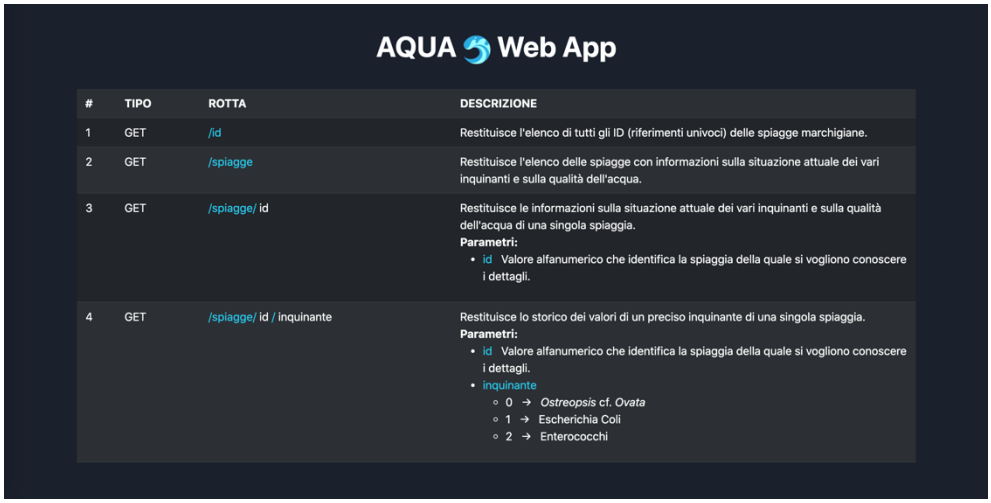

Figura 8.5 – Servizi Web App AQUA

I servizi utilizzati dalla versione Flutter dell'applicativo in analisi in questo elaborato di tesi, facendo riferimento alla *Figura 8.5*, sono il numero #2 e il numero #4.
Infatti, per poter soddisfare uno dei requisiti non funzionali, descritti nel *paragrafo 1.2.2*, ossia mantenere un buon livello di reattività per l'applicazione, è stato adottato un approccio al download di tipo lazy. Infatti, all'avvio dell'app, vengono scaricati solo i dati necessari per visualizzare la situazione corrente di tutte i siti di rilevazione, e solo quando necessario verrà effettuato il download dei dati relativi allo storico di un dato inquinante per la spiaggia selezionata dall'utente.
In questo modo le performance dell'applicazione vengono incrementate notevolmente, in quanto essa non deve effettuare il download di tutte le informazioni durante la fase di avvio.
Entrambe le chiamate ai servizi forniscono come risultato un file JSON contente i dati richiesti. Di seguito viene approfondita la struttura di tali file.

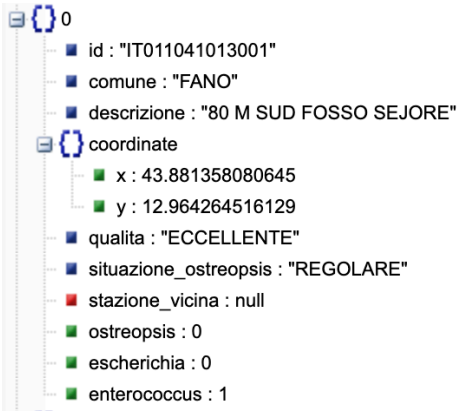

Figura 8.6 – Struttura file JSON Spiagge

L'intero file, contenente le informazioni correnti di tutte le spiagge, è un vettore i cui elementi hanno la stessa struttura di quello riportato nella *Figura 8.6*:

- *"id"*: identificatore univoco della spiaggia;
- *"comune"*: comune di appartenenza del sito;
- *"descrizione"*: descrizione geografica della stazione di rilevazione;
- *"coordinate"*: coordinate della spiaggia;
- *"qualità"*: qualità dell'acqua complessiva rilevata;
- *"situazione_ostreopsis"*: situazione dell'alga *Ostreopsis* cf. *Ovata;*
- *"stazione_vicina"*: indica se c'è una stazione di rilevazione per l'alga tossica entro 6 km ed eventualmente quale;
- *"ostreopsis"*: valore dell'ultima rilevazione dell'*Ostreopsis* cf. *Ovata;*
- *"escherichia"*: valore dell'ultima rilevazione di escherichia coli;
- *"enterococcus"*: valore dell'ultima rilevazione di enterococchi;

Il file JSON contenente gli storici delle rilevazioni, invece, ha una struttura molto più elementare del precedente.

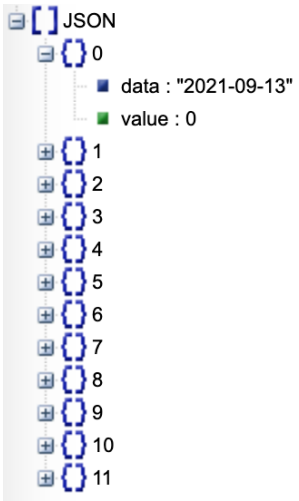

Figura 8.7 Struttura file JSON Storico

Esso è costituito da un vettore di oggetti contenti ognuno due attributi:

- *"data"*: contiene la data in cui è stata effettuata la rilevazione;
- *"value"*: racchiude il valore della rilevazione.

8.2.4 Services e Model

I due elementi descritti in tale sotto paragrafo sono di fondamentale importanza per la fase di acquisizione delle informazioni necessarie al funzionamento dell'applicativo "AQUA".
I model, come già descritto più volte all'interno di questo elaborato, hanno la funzionalità di mappare il contenuto del file JSON in una classe dart così da creare un'interfaccia verso il file stesso.
I service, invece, sono responsabili del download dei dati e, una volta compiuto tale incarico, invocano il metodo di lettura del rispettivo model per attivarne le funzionalità.

8.2.5 Schema di architettura – Fase di avvio

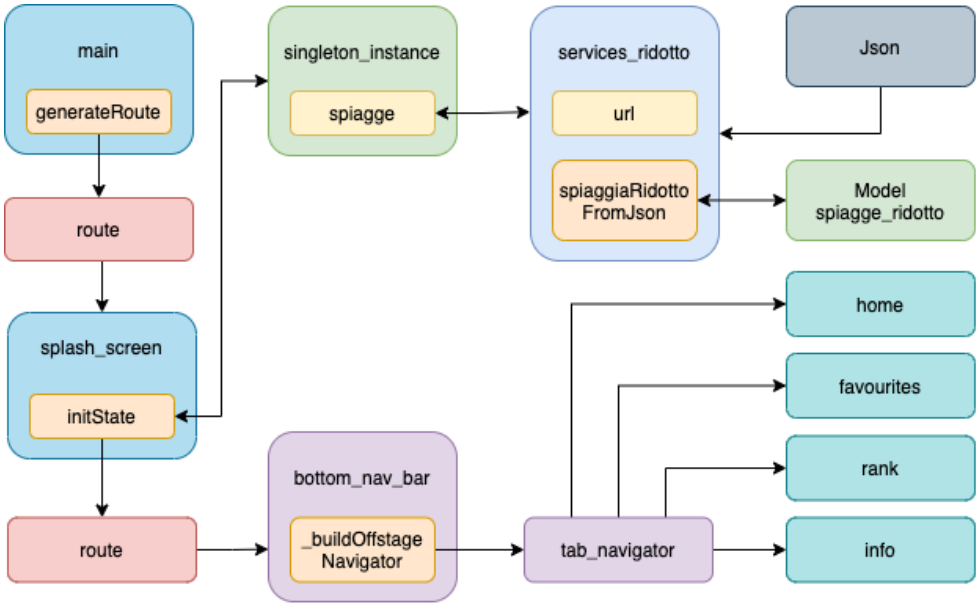

Figura 8.4 – Architettura dei processi protagonisti della fase di avvio dell'app cross-platform

La prima classe eseguita dall'applicazione è il "main" che, attraverso il meccanismo di routing descritto nel paragrafo 8.2.1 invoca la classe "splash_screen". Quest'ultima, all'interno del suo metodo "initState ( )", eseguito una sola volta quando il Widget corrispondente alla classe viene costruito, invoca per la prima volta l'oggetto "singleton_instance" che attraverso il suo costruttore interno popola il suo unico attributo.
Questo meccanismo è realizzabile grazie all'interazione tra services e model già descritta del paragrafo precedente.
Una volta istanziato l'oggetto globale, sempre attraverso lo stesso meccanismo di routing già citato, viene invocata la "bottom_nav_bar", responsabile della navigazione tra i 4 stack corrispondenti alle 4 sezioni dell'applicazione: Home, Preferiti, Classifica, Info.
La barra di navigazione, attraverso la classe "tab_navigator" invoca le quattro classi corrispondenti alle sezioni appena riportate. Dato che ci si trova in fase di avvio la sezione invocata in questo specifico caso è la Home.

8.2.6 Schema di architetura – Ramo "Home"

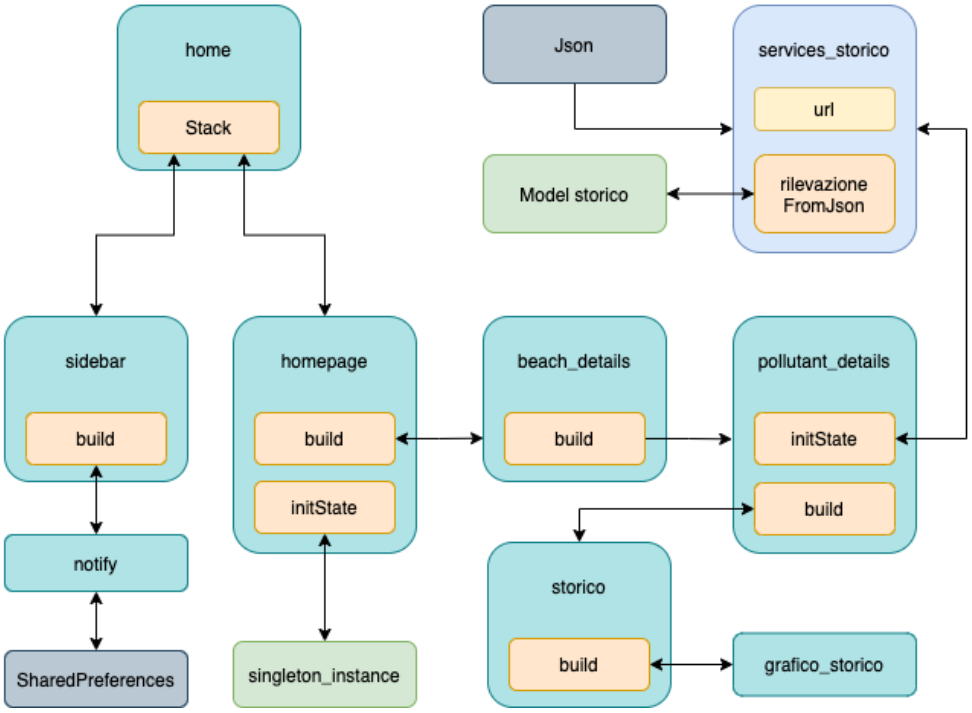

Figura 8.5 - Architettura ramo "Home" app cross-platform

La home contiene al suo interno due sezioni sovrapposte in uno stack: al livello più alto c'è un navigation drawer contenente un pannello di controllo per le notifiche, al secondo livello vi è la mappa con la barra di ricerca.
Il primo, all'interno del suo metodo build( ), invoca l'istanza delle SharedPreferences per settare le impostazioni dello notifiche allo stato in cui l'utente le ha settate durante gli avvii precedenti dell'applicazione.
L'homepage, invece, all'interno del suo metodo initState( ) crea una copia in locale dell'oggetto globale "singleton_instance" così da poter creare correttamente i marcatori e fornire i risultati di una eventuale ricerca.

Cliccando sull'infobox generato da un marcatore si potrà navigare verso la classe "beach_details" responsabile della creazione e gestione della pagina "Dettagli spiaggia".
Al suo interno sarà possibile giungere alla schermata che mostra lo storico di un inquinante cliccando sulla relativa scheda.
Quest'ultima pagina, si occupa innanzitutto di invocare il service "services_storico" per reperire i dati necessari per il proprio funzionamento all'interno del metodo initState( ) ed, in secondo luogo, all'interno del metodo build( ), invoca la classe "storico".
Essa è responsabile di effettuare tutti i controlli descritti nel paragrafo 7.3.3 e di conseguenza di invocare la classe "grafico_storico" per la generazione del grafico contenuto all'interno della pagina.

8.2.7 Schema di architettura – Ramo “Preferiti”

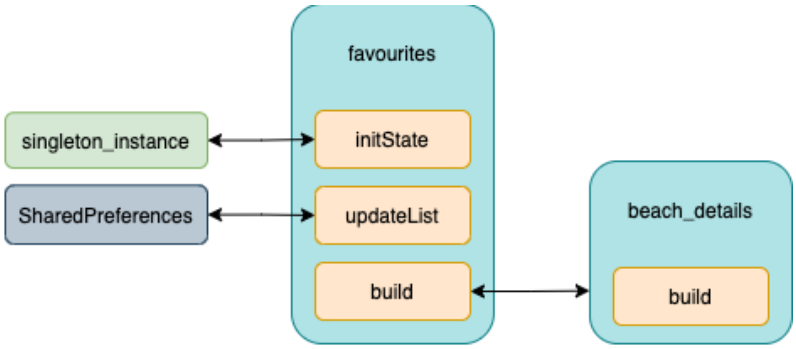

Figura 8.6 - Architettura ramo “Preferiti” app cross-platform

La classe “favourite” è responsabile della creazione e della gestione della pagina dei preferiti.
All’interno del suo metodo initState( ) essa invoca l’oggetto globale con il fine di crearne una copia in locale. A questo punto effettua una chiamata al metodo updateList, responsabile della generazione della lista delle spiagge indicate come preferite dall’utente.
Quest’ultima funzione Consulta ogni volta che viene invocata le SharedPreferences per valutare quale sito di rilevazione è contrassegnato come preferito. Inoltre, questo metodo viene invocato all’interno del metodo build( ), cosicché ogni volta che la pagina viene rigenerata la lista dei preferiti viene aggiornata.
Questa pagina mostra come item della lista il Widget prodotto della classe “beach_details”. Viene da sé che tutte le interazioni già descritte per questa classe valgono anche in questo caso e non vengono riportate per evitare di rendere tale elaborato ripetitivo.

8.2.8 Schema di architettura – Ramo “Classifica” e ramo “Info”

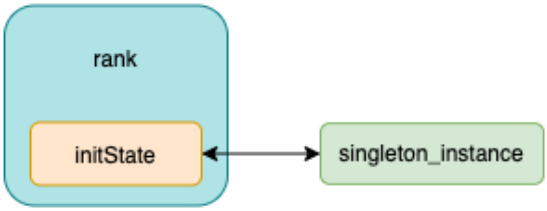

Figura 8.7 – Architettura Ramo “Classifica” app cross-platform

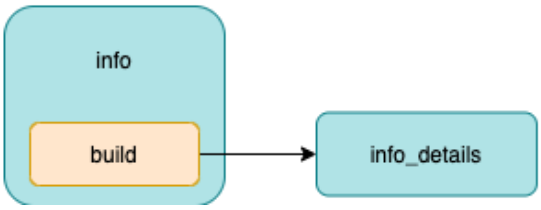

Figura 8.8 – Architettura Ramo “Info” app cross-platform

A causa della loro forte somiglianza e semplicità questi due rami vengono trattati insieme.
Il primo, riguardante la sezione “Classifica”, oltre ad un collegamento con la classe “beach_details” non riportato per i motivi già espressi nel paragrafo precedente, contiene all’interno del suo metodo initState( ) un riferimento all’oggetto “singleton_spiagge” così da crearne una copia in locale.
Il ramo “Info” invece, contiene all’interno del proprio build( ) una chiamata alla classe “info_details”. Questa chiamata si verifica nel caso in cui l’utente decida di aprire una delle preview generate dalla classe “info”.

8.3 Frammenti di codice salienti

In questo paragrafo vengono riportati i frammenti di codice su cui occorre porre l'attenzione a causa delle soluzioni adottate al loro interno.

8.3.1 Fase iniziale

```
Future<List<SpiaggiaRidotto>> getSingleton(){
  var singleton = new SingletonSpiagge();
  final spiagge = singleton.spiagge;
  return spiagge;
}
@override
void initState() {
  super.initState();
  getSingleton().then((spiagge){
    Navigator.pushNamed( context,
      '/bottomNavBar',
    );
  });
}
```

Figura 8.9 – Codice splash-screen

In figura viene riportata la sezione di codice, presente all'interno della classe "_SplashScreenState", contenente il metodo initState( ) dello SplashScreen ed il metodo getSingleton( ).
La prima funzione contiene al suo interno una componente comune a tutti i metodi initState( ) delle classi che creano in locale una copia dell'oggetto globale "singleton_spiagge".
Dato che la chiamata a tale oggetto attraverso getSingleton( ) è di tipo asincrono, occorre incapsulare le istruzioni che dipendono da tale chiamata all'interno del costrutto "then"[32].
Tale costrutto permette l'accesso alle istruzioni contenute al suo interno solo quando il metodo su cui è stato chiamato ritorna il valore richiesto. In questo modo si evita che vengano eseguite delle istruzioni riferite a tale valore prima che esso venga istanziato generando così una chiusura dell'applicazione.

32 https://api.flutter.dev/flutter/dart-async/Future/then.html

```
class SingletonSpiagge {
  static final SingletonSpiagge instance =
  new SingletonSpiagge._internal(ServicesRidotto.getSpiaggeRidotto());
  final Future<List<SpiaggiaRidotto>> spiagge;
  factory SingletonSpiagge() => instance;

  SingletonSpiagge._internal(this.spiagge);
}
```

Figura 8.10 – Codice oggetto globale "SingletonSpiagge"

Nel frammento di codice contenuto in *Figura 8.10* viene mostrato il processo di generazione dell'oggetto SingletonSpiagge. Utilizzando i modificatori "static final" si fa in modo che non possa essere creata più di una istanza di tale oggetto.
L'attributo al suo interno viene popolato mediante la chiamata al metodo getSpiaggeRidotto( ) del service ServicesRidotto.
Esiste un metodo che svolge tale funzionalità per ogni service, di seguito viene preso in analisi il metodo relativo al ServicesStorico.

```
class ServicesStorico {

  static const String urlProva = 'https://pastebin.com/raw/y5aVYmE0';
  static const String urlNull = 'https://pastebin.com/raw/9xqGCRAF';

  static Future<List<Rilevazione>> getStorico(String pollutant, String idSpiaggia) async {

    final idPollutant = pollutant == ostreopsisName ? 0 : (pollutant == escherichiaName ? 1 : 2);

    final url = "http://193.205.129.120:63434/spiagge/${idSpiaggia}/${idPollutant}";

    try{
      final response = await http.get(Uri.parse(url));
      if (200 == response.statusCode){
        final List<Rilevazione> storico = rilevazioneFromJson(response.body);
        return storico;
      }else{
        return <Rilevazione>[];
      }
    }catch(e){
      return <Rilevazione>[];
    }
  }

}
```

Figura 8.11 - Codice ServicesStorico

Nel frammento di codice appena riportato si può notare nel dettaglio l'interazione tra model e service già enunciata nei paragrafi precedenti.
Altra componente interessante è la creazione della stringa contenente l'url attraverso il quale è possibile raggiungere l'applicazione web "AQUA".
Esso è composto da una componente statica e da due componenti che variano in base a che spiaggia stiamo considerando e a quale inquinante stiamo facendo riferimento.

8.3.2 Marker e zoom

In questa sezione viene analizzato nel dettaglio il meccanismo che permette all'applicativo di aggiornare la distribuzione dei marker in base allo zoom della spiaggia. La mappa utilizzata in tale progetto è quella gestita dal pacchetto "google_map_flutter"[33].

```
const List<String> firstLevelMarker = [
  'IT011041019001',
  'IT011041044024',
  'IT011041013004',
  'IT011041013018',
  'IT011042045004',
  'IT011042027001',
  'IT011042018009',
  'IT011042002018',
  'IT011042002013',
  'IT011042032010',
  'IT011043043006',
  'IT011043013002',
  'IT011109006004',
  'IT011109006005',
  'IT011044017002',
  'IT011044023007',
  'IT011044007001',
  'IT011043017001',
  'IT011043051001',
  'IT011043012001',
];
```

Figura 8.11 – Elenco marker primo livello di zoom

In figura è mostrato l'elenco degli id delle spiagge da contrassegnare con un marker quando il livello dello zoom è nell'intorno del livello di default con cui viene mostrata la mappa all'utente.
Insieme all'elenco dei marker relativi al secondo livello di zoom, questi dati sono salvati all'interno di un file, denominato "marker_visible.dart", nella cartella "values" nel progetto dell'applicativo.

33 https://pub.dev/packages/google_maps_flutter

```
onCameraMove: (position) {
  setState(() {
    _currentZoom = position.zoom;
  });
},
onCameraIdle: (
updateMarkers()
),
```

Figura 8.12 – Codice stato zoom

In figura viene mostrato il codice responsabile della rilevazione dei cambiamenti dello zoom contenuto nella classe "_HomePageState".
Con il metodo "onCamerMove" viene settato lo stato della variabile "_currentZoom" con il valore attuale di zoom. Tale metodo si attiva quando avviene un movimento della camera che visualizza la mappa.
"onCameraIdel", invece, viene attivato quando il suddetto movimento si interrompe e invoca il metodo "updateMarkers( )".

```
if (_currentZoom<secondLevelZoom){
  setState(() {
    _markers.clear();
  });
  bool visibility = false;
  setState(() {
    for (SpiaggiaRidotto elem in _spiagge){
      visibility = false;
      if (firstLevelMarker.contains(elem.id)) {
        visibility = true;
      }
      setMarker(elem, visibility);
    }
  });
}
```

Figura 8.13 – Codice updateMarker

Nell'immagine è riportato un frammento del codice del metodo appena citato. Quest'ultimo è composto da una serie di blocchi if come quello visibile nella *Figura 8.13*.
Ad ogni spostamento tutti i marker vengono istrutti e ricreati. L'unica variazione sta nel parametro "visibile" del marker che verrà settato in base a quale livello di zoom appartiene.

8.3.3 Barra di navigazione

```
void _selectTab (String tabItem, int index){
  if (tabItem== _currentPage){
    _navigatorKeys[tabItem]!.currentState!.popUntil((route) => route.isFirst);
  } else{
    setState(() {
      _currentIndex = index;
      _currentPage = pageKeys[index];
    });
  }
}
```

Figura 8.14 – Codice tap bottoni barra di navigazione

In figura viene mostrato come vengono gestiti i tap sui bottoni della barra di navigazione; il codice appartiene alla classe "_BottomNavBar".
Cliccando su un bottone il primo controllo che viene effettuato valuta se tale pulsante è quello corrispondente alla schermata già visualizzata. In questo caso, si verrà reindirizzati alla pagina alla base dello stack di navigazione. Altrimenti verrà recuperato l'indice dell'ultima pagina visualizzata nello stack relativo al bottone cliccato e si verrà reindirizzati su tale schermata non perdendo lo stato della navigazione.

8.3.4 Stack Home

```
return WillPopScope(
  child: Stack(
        children: <Widget> [
          HomaPage(),
          SideBar(),
        ]  // <Widget>[]
  ),  // Stack
  onWillPop: () {
    return Future.value(false);
  },
);  // WillPopScope
```

Figura 8.15 – Codice Stack home

In figura viene mostrata l'implementazione dello stack contenuto nella schermata home, appartenente alla classe "Home".
In questo modo i due Widget risultati dalle classi HomePage e SideBar potranno sovrapporsi graficamente all'interno della pagina home.

8.3.5 Lista preferiti

```
@override
void initState(){
  super.initState();
  getSingleton().then((spiagge){
    setState(() {
      _spiagge = spiagge;
    });
    setState(() {
      listFav = updateList();
    });
  });
}

int _current = 0;
@override
Widget build(BuildContext context){
  WidgetsBinding.instance!.addPostFrameCallback((_) =>
      setState(() {
        listFav = updateList();
  }));
```

Figura 8.16 – Codice Preferiti

Nella figura 8.16 sono riportate le due chiamate al metodo responsabile di effettuare l'aggiornamento della lista dei preferiti, entrambe contenute nella classe "_FavouritesState".
La prima viene fatta all'interno del metodo initState( ) per inizializzare la lista di visualizzare. Mentre, la seconda viene effettuata ogni qualvolta il metodo build viene triggerato. Questa scelta progettuale è stata presa per far si che la lista venisse aggiornata anche durante il funzionamento dell'applicazione e non solo al suo avvio, cosicché se l'utente decidesse di variare le sue preferenze in merito ai preferiti visualizzerebbe immediatamente i risultati delle sue scelte.

Conclusioni

In questo elaborato di tesi si è discusso della progettazione e realizzazione di un'applicazione per il monitoraggio della situazione di inquinamento relativa alle acque che bagnano la costa marchigiana.
A seguito di un incontro con il ricercatore, riconducibile alla figura del committente dell'applicativo, e di un'analisi approfondita della tecnologia già in uso dall'ARPAM si è giunti alla stesura della specifica dei requisiti, documento che ha guidato il team di sviluppo per tutta la durata del progetto.
È stata riportata la relazione della fase di progettazione grafica e funzionale dell'app; quindi, sono state prodotte le prime due versioni dell'applicativo "AQUA".
La prima, disponibile solo su piattaforma Android, è caratterizzata dalla presenza al suo interno di dati verosimili alla realtà ma del tutto simulati.
La seconda, invece, è in grado di lavorare con i dati rilevati periodicamente grazie alla web app "AQUA" che mette a disposizione tali informazioni recuperandole dal sito internet dell'ARPAM.
Per entrambe le applicazioni sono stati presentati l'aspetto grafico, la struttura del progetto, l'architettura applicativa, i frammenti di codice salienti.
L'intero progetto "AQUA" ha ancora molta strada da fare: ci sono numerose funzioni aggiuntive che verranno implementate nelle versioni successive.

Ringraziamenti

A conclusione di questo elaborato, desidero menzionare con un ringraziamento tutte le persone senza le quali tale lavoro di tesi, probabilmente, non esisterebbe nemmeno.
Chi mi conosce sa che non sono affatto bravo con le parole ma proverò a fare del mio meglio per esprimere il senso di gratitudine che ho dentro.
Ringrazio il mio relatore, il prof. Storti, che, in questi mesi di lavoro, ha saputo guidarmi, con suggerimenti pratici, nello sviluppo dell'applicazione e nella stesura dell'elaborato.
Ringrazio i miei genitori, per non avermi mai fatto mancare il sostegno e l'affetto di cui, in questo periodo come in altri, ho avuto bisogno; per avermi cresciuto con dei sani principi ed avermi dato l'esempio nell'affrontare la vita. Spero, con questo traguardo, di rendervi fieri di me.
Ringrazio i miei nonni, per avermi insegnato l'importanza e l'umiltà del lavoro, per ricordarmi che nella vita vale la pena fare dei sacrifici per mirare a ciò che davvero si vuole.
Ringrazio tutta la mia famiglia, gli zii, i cugini, per essermi stati accanto e per aver creato sempre un clima sereno intorno a me.
Ringrazio la mia ragazza Chiara, per l'amore con cui quotidianamente mi sostiene. Grazie per essermi stata vicino nei momenti più difficili e per aver festeggiato con me delle piccole vittorie lungo questo percorso. Sei un grandissimo esempio per me in tutto quello che fai, per la passione e la dedizione con cui affronti ogni aspetto della tua quotidianità. Sono orgoglioso di averti al mio fianco ad affrontare questa lungo percorso chiamato vita.
Ringrazio la tua famiglia per avermi fatto sentire parte di essa sin dal primo momento, sappiate che per qualunque cosa potrete contare sempre su di me.
Ringrazio Giordano, il mio compagno in questo lungo viaggio universitario e non solo. Grazie, perché affrontare questi tre anni, soprattutto questi ultimi mesi, con te ha reso questo percorso notevolmente più semplice e leggero.
Ringrazio Roberto ed Antonio per aver condiviso con me, anche se in parte, la fatica e le gioie che hanno distinto questi ultimi anni.
Ringrazio tutti i miei amici. Per essermi stati accanto ed aver alleggerito con risate, scherzi e consigli i momenti più difficili.

Ringrazio il mio gruppo scout e tutta la comunità capi: la mia seconda famiglia. Grazie per avermi reso un uomo della partenza e per ricordarmi ogni giorno che questa non è la strada di chi parte e già vuole arrivare ma di chi parte ed arriva per partire.
Ringrazio i ragazzi del mio reparto per darmi una continua dimostrazione di come, con pochi materiali ma con tanta fantasia e spirito di sacrificio, si possano realizzare delle imprese, in tutti i sensi. Sono grato di poterli accompagnare nella loro vita anche solo per un piccolo tratto.
Ringrazio Dio per le splendide persone che mi ha messo di fianco e per la vita che mi ha donato. Farò di tutto per non sprecare neanche un secondo del tempo che mi è stato affidato.
Ed infine, non posso che ringraziare me stesso. Per non aver mollato anche quando sembrava impossibile, per aver sempre puntato in alto senza mai accontentarmi, per aver avuto l'umiltà di chiedere aiuto quando ce n'era bisogno. Sono fiero di dove sono arrivato, tra i sacrifici e i sorrisi, le difficoltà e le soddisfazioni. Ma so al tempo stesso che ora non posso accontentarmi di quello che ho; quindi, è già arrivato il momento di puntare alla prossima meta: senza fretta ma senza sosta.

Bibliografia

[1] C. Totti, S. Accoroni, F. Cerino, E. Cucchiari, T. Romagnoli, «Ostreopsis ovata bloom along the Conero Riviera (northern Adriatic Sea): Relationship with environmental conditions and substrata» Harmful algae, 2010, pp. 233-239.

[2] G. Rovanato, «JSON: cos'è, formato, esempio» 2011. [Online]. Available: https://www.html.it/articoli/introduzione-a-json/.

[3] Google LLC, «Google Developers» 2005. [Online]. Available: https://developers.google.com.

[4] ZOE S.R.L.S., «3 TIPOLOGIE DI APP PER DISPOSITIVI MOBILE. QUALE È ADATTA AL TUO PROGETTO?» 2020. [Online]. Available: https://www.zoewebsolutions.it/3-tipologie-di-app-per-dispositivi-mobile-quale-%C3%A8-adatta-al-tuo-progetto.

[5] Google LLC, «Android Studio» 2020. [Online]. Available: https://developer.android.com/studio.

[6] JetBrains s.r.o., «IntelliJ IDEA» 2020. [Online]. Available: https://www.jetbrains.com/idea/.

[7] A. Inc, «Xcode» 2021. [Online]. Available: https://developer.apple.com/it/support/xcode.

[8] C. Garau, «Git e il controllo versione distribuito» 2016. [Online]. Available: https://www.html.it/pag/52561/git-e-il-controllo-di-versione-distribuito/.

[9] Google LLC. , «Android» 2008. [Online]. Available: https://developer.android.com/.

[10] JetBrains, «Kotlin» 2011. [Online]. Available: https://kotlinlang.ore/.

[11] Square, Inc, «OkHttp» 2011. [Online]. Available: https://square.github.io/okhttp/.

[12] Google LLC, «Gson» 2008. [Online]. Available: https://github.com/google/gson.

[13] M. Lopez, «CircularProgressBar» 2015. [Online]. Available: https://github.com/lopspower/CircularProgressBar.

[14] Google LLC, «Flutter» 2021. [Online]. Available: https://flutter.dev/.

[15] Google LLC, «Dart» 2020. [Online]. Available: https://dart.dev/.

[16] Google LLC., «Material design» 2014. [Online]. Available: https://material.io/components.