

Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA

CORSO DI LAUREA IN INFORMATICA



## Sviluppo di applicazioni native per iOS in JavaScript

*Tesi di laurea triennale*

*Relatore*

Prof. Claudio Enrico Palazzi

*Laureando*

Giacomo Manzoli

---

ANNO ACCADEMICO 2014-2015



# Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage, della durata di circa trecento ore, dal laureando Giacomo Manzoli presso l'azienda WARDA S.r.l.. L'obiettivo di tale attività di stage è l'analisi dei vari framework disponibili per lo sviluppo di applicazioni native utilizzando il linguaggio JavaScript, al fine di creare un'applicazione nativa per iOS simile alla gallery sviluppata dall'azienda.



*“Fuck it, i did it”*

— Confucius

# Ringraziamenti

*Bla bla bla*

*Bla bla bla*

*Bla bla bla*

*Even more bla bla bla*

...

*Padova, Dec 2015*

Giacomo Manzoli



# Indice

<b>1</b>	<b>Il contesto aziendale</b>	<b>3</b>
1.1	L'azienda . . . . .	3
1.1.1	WARDA - Work on what you see . . . . .	3
1.2	Il progetto . . . . .	4
<b>2</b>	<b>Framework analizzati</b>	<b>5</b>
2.1	Considerazioni generali . . . . .	5
2.1.1	Differenze con le applicazioni ibride . . . . .	6
2.2	Tabris.js . . . . .	6
2.2.1	Come funziona . . . . .	7
2.2.2	Pregi e difetti . . . . .	7
2.2.3	Prototipo . . . . .	7
2.3	NativeScript . . . . .	8
2.3.1	Come funziona . . . . .	8
2.3.2	Pregi e difetti . . . . .	10
2.3.3	Prototipo . . . . .	10
2.4	React Native . . . . .	11
2.4.1	Come funziona . . . . .	11
2.4.2	Pregi e difetti . . . . .	12
2.4.3	Prototipo . . . . .	12
2.5	Confronto finale . . . . .	13
2.6	Framework scelto . . . . .	14
<b>3</b>	<b>Strumenti e tecnologie utilizzate</b>	<b>15</b>
3.1	React Native . . . . .	15
3.1.1	La sintassi JSX . . . . .	16
3.1.2	Oggetti JavaScript per la definizione dello stile . . . . .	16
3.1.3	JavaScript ES6 e ES7 . . . . .	17
3.1.4	Componenti esterni . . . . .	17
3.2	Flux . . . . .	18
3.2.1	Sequenza delle azioni . . . . .	18
3.2.2	Differenze con MVC . . . . .	19
3.3	React Native CLI . . . . .	20
3.4	Atom e Nuclide . . . . .	20
3.5	Xcode . . . . .	20
3.6	Google Chrome Dev Tools . . . . .	21
<b>4</b>	<b>Analisi dei Requisiti</b>	<b>23</b>

4.1	Applicazione attuale . . . . .	23
4.1.1	Pagina di visualizzazione della gallery . . . . .	23
4.1.2	Pagina di dettaglio di un asset . . . . .	25
4.2	Requisiti individuati . . . . .	26
4.2.1	Requisiti Funzionali . . . . .	27
4.2.2	Requisiti di Vincolo . . . . .	29
4.3	Riepilogo requisiti . . . . .	29
<b>5</b>	<b>Progettazione</b>	<b>31</b>
5.1	API REST di WARDA . . . . .	31
5.2	Architettura generale dell'applicazione . . . . .	31
5.2.1	Model . . . . .	31
5.2.2	Stores . . . . .	31
5.2.3	Actions . . . . .	31
5.2.4	Pages . . . . .	31
5.2.5	Components . . . . .	31
5.2.6	Utils . . . . .	31
5.3	Diagrammi di attività . . . . .	31
5.3.1	Navigazione nella gallery . . . . .	31
5.3.2	Applicazione di un filtro . . . . .	31
<b>6</b>	<b>Realizzazione</b>	<b>33</b>
6.1	Dal prototipo alla gallery . . . . .	33
6.2	Sistema di navigazione . . . . .	33
6.3	Sistema dei filtri . . . . .	33
6.4	Visualizzazione di dettaglio . . . . .	33
6.5	Animazioni . . . . .	33
6.6	Gestione degli errori . . . . .	34
<b>7</b>	<b>Conclusioni</b>	<b>35</b>
7.1	Valutazione del risultato . . . . .	35
7.1.1	Requisiti soddisfatti . . . . .	35
7.2	Aspetti critici e possibili estensioni . . . . .	35
7.3	Conoscenze acquisite . . . . .	35



# Elenco delle figure

1.1	Logo di WARDA S.r.l. . . . .	3
2.1	Screenshot del prototipo realizzato con Tabris.js . . . . .	8
2.2	Schema rappresentante il runtime di NativeScript . . . . .	9
2.3	Architettura di NativeScript . . . . .	10
2.4	Screenshot del prototipo realizzato con NativeScript . . . . .	11
2.5	Screenshot del prototipo realizzato con React Native . . . . .	13
3.1	Diagramma del pattern Flux . . . . .	18
3.2	Funzionamento del pattern Flux . . . . .	19
3.3	Tools di debug di Xcode . . . . .	21
3.4	Debug di un'applicazione con Google Chrome . . . . .	22
4.1	Screenshot della gallery dell'applicazione attuale . . . . .	24
4.2	Dettaglio della griglia degli assets dell'applicazione attuale . . . . .	25
4.3	Popover che mostra i dettagli di un asset . . . . .	26
4.4	Requisiti per importanza . . . . .	30
4.5	Requisiti per tipologia . . . . .	30

# Elenco delle tabelle

2.1	Tabella comparativa dei framework analizzati . . . . .	14
4.1	Requisiti Funzionali . . . . .	29
4.2	Requisiti di Vincolo . . . . .	29
4.3	Numero di requisiti per importanza . . . . .	29
4.4	Numero di requisiti per tipologia . . . . .	29



# Notes

■ Valutare se inserire le informazioni riguardo l'esecuzione di un'applicazione	5
■ Trovare un nome migliore . . . . .	13
■ AskAlberto: Immagine della lista dei nodi, dove si vede più di qualche nodo	24
■ AskAlberto: Quanti assets vengono caricati per volta dalla griglia? . . . . .	25
■ AskAlberto: Immagine della lista dei filtri . . . . .	25
■ AskAlberto: immagine della visualizzazione di dettaglio, sia con la sidenav aperta, sia con la sidenav chiusa . . . . .	26
■ Descrizione dell'importanza dei requisiti . . . . .	27
■ discorso introduttivo . . . . .	33
■ Descrizione dell'adattamento del prototipo alla gallery . . . . .	33
■ Aggiunta di funzionalità come lo scroll infinito . . . . .	33
■ Descrizione dell'aggiunta del sistema di navigazione tra i nodi della gallery .	33
■ Come sono stati aggiunti i filtri alla navigazione . . . . .	33
■ Come sono stati aggiunti i filtri alla navigazione . . . . .	33
■ Navigazione tra pagine . . . . .	33
■ Swipe . . . . .	33
■ Immagine con indicatore di caricamento . . . . .	33
■ Animazione delle comparse/scomparsa . . . . .	33
■ Animazione dello swipe . . . . .	33
■ Animazione del pinch-to-zoom non implementata . . . . .	33
■ Errori di comunicazione . . . . .	34
■ Commento sul risultato ottenuto e sulle presentazioni dell'applicazione . . .	35
■ Tabellina riassuntiva dei requisiti soddisfatti . . . . .	35
■ Del framework e dell'applicazione . . . . .	35
■ Storia del log-in e dell'interfaccia grafica . . . . .	35
■ Tecnologie studiate . . . . .	35



# Capitolo 1

## Il contesto aziendale

### 1.1 L'azienda

WARDA S.r.l. è una startup avviata di recente dai due soci Marco Serpilli e David Bramini, con sede a Padova.

L'azienda è nata come spin-off di Visionest S.r.l, una società che si occupa di consulenza informatica e dello sviluppo di software gestionali dedicate alla aziende.

A differenza di Visionest, WARDA S.r.l. si focalizza sulle aziende che operano nel mercato della moda e del lusso, offrendo un sistema per la gestione delle risorse e dei processi aziendali, ottimizzato per le aziende del settore.



**Figura 1.1:** Logo di WARDA S.r.l.

#### 1.1.1 WARDA - Work on what you see

WARDA è un sistema software che si occupa di Digital Asset Management (DAM), cioè un sistema di gestione di dati digitali e informazioni di business, progettato per il mercato del lusso, fashion e retail.

In un sistema DAM, le immagini, i video ed i documenti sono sempre legati alle informazioni di prodotto, costituendo così i digital assets: un'immagine prodotto riporta le caratteristiche della scheda prodotto, un'immagine marketing le informazioni della campagna, un'immagine della vetrina i dati del negozio, del tema, dell'ambiente, e così via.

Memorizzando questi digital assets in un unico sistema centralizzato, WARDA permette di riutilizzare infinite volte le immagini, i video e i documenti durante le attività aziendali evitando così ogni duplicazione.

Il materiale digitale è sempre associato alla scheda prodotto gestita nel sistema informativo aziendale, rappresentando così l'unico "catalogo digitale" di tutti i beni/prodotti aziendali.

In questo modo WARDA diventa il centro dell'ecosistema digitale aziendale, coordinando e organizzando la raccolta e la condivisione dei digital assets attraverso processi ben definiti e istanziati secondo le prassi aziendali.

WARDA è disponibile sia come applicazione Web, sia come applicazione per iPad.

## 1.2 Il progetto

Lo scopo del progetto è quello di valutare se, mediante l'utilizzo di framework differenti, sia possibile migliorare l'esperienza d'uso del client per iPad di WARDA.

Secondo l'azienda la causa di alcuni problemi del client attuale derivano dal framework che è stato utilizzato per svilupparlo, infatti, l'applicazione attuale è un'applicazione di tipo ibrido, cioè composta da un'applicazione web ottimizzata per lo schermo dell'iPad e racchiusa in un'applicazione nativa utilizzando Cordova.

Di conseguenza, si ritiene che l'utilizzo di un'altra tipologia di framework che permette lo sviluppo di applicazioni native utilizzando JavaScript possa portare dei miglioramenti all'applicazione attuale.

Pertanto, la prima parte del progetto riguarda la ricerca e l'analisi di framework che permettono di sviluppare applicazioni native con JavaScript utilizzando un'interfaccia grafica nativa al posto di una pagina web.

La seconda parte invece, prevede lo sviluppo di un'applicazione analoga a al client per iPad attuale utilizzando uno dei framework individuati.

## Capitolo 2

# Framework analizzati

Questo capitolo inizia con una descrizione generale della tipologia di framework analizzati, per poi andare a descrivere più in dettaglio il funzionamento dei singoli framework, seguito da una sintesi dei pregi e difetti e un breve resoconto riguardo il prototipo realizzato. Infatti, per valutare al meglio ogni framework è stata realizzata un'applicazione prototipo che visualizza una gallery di immagini ottenuta mediante chiamate ad API REST.

La struttura del prototipo è stata scelta in modo tale da verificare le seguenti caratteristiche:

- possibilità di disporre le immagini in una griglia;
- possibilità di implementare lo scroll infinito, cioè il caricamento di ulteriori dati una volta visualizzati tutti quelli disponibili;
- fluidità dello scroll, specialmente durante il caricamento dei dati;
- possibilità di personalizzare la barra di navigazione dell'applicazione.

Valutare se inserire le informazioni riguardo l'esecuzione di un'applicazione

### 2.1 Considerazioni generali

Le applicazioni realizzate con questa tipologia di framework hanno alla base lo stesso funzionamento: una macchina virtuale interpreta il codice JavaScript e mediante un “*ponte*” viene modificata l'interfaccia grafica dell'applicazione, che è realizzata con i componenti offerti dall'SDK nativo. Ognuno dei framework analizzati utilizza un “*ponte*” diverso, il cui funzionamento verrà descritto più in dettaglio nell'apposita sezione.

Un'altra caratteristica di questa tipologia di framework è l'assenza del DOM. Infatti, l'interfaccia di un'applicazione è composta da componenti grafici del sistema operativo che vengono creati e composti durante l'esecuzione dell'applicazione e non da elementi HTML come nelle applicazioni ibride.

Nel complesso si ottengono due grossi vantaggi:

- L'esecuzione del codice JavaScript e il rendering dell'interfaccia grafica avviene su due thread distinti, rendendo l'applicazione più fluida;

- Utilizzando i componenti grafici nativi si ottiene un'esperienza utente più simile a quella che si ottiene con un'applicazione realizzata in Obj-C/Java.

### 2.1.1 Differenze con le applicazioni ibride

Un'applicazione ibrida consiste in un'applicazione nativa composta da una `WebView`<sup>1</sup> che visualizza un'insieme di pagine web realizzate utilizzando HTML5, JavaScript e CSS3, che replicano l'aspetto di un'applicazione nativa.

Trattandosi quindi di un'applicazione web è possibile utilizzare tutti i framework disponibili per il mondo web, come Angular, jQuery, Ionic, ecc. Inoltre, se è già disponibile un'applicazione web per desktop, la creazione di un'applicazione mobile ibrida risulta rapida.

Questo approccio viene utilizzato da qualche anno e permette di creare applicazioni che possono accedere ad alcune funzionalità hardware del dispositivo come il giroscopio o la fotocamera e che possono essere commercializzate nei vari store online.

Per supportare questo processo di sviluppo sono stati sviluppati dei framework come Cordova/PhoneGap che si occupano di gestire la `WebView` e di fornire un sistema di plug-in per accedere alle funzionalità native.

Questa tipologia di applicazioni ha però delle limitazioni:

- **Prestazioni:** trattandosi di una pagina web renderizzata all'interno di un browser, non è possibile sfruttare al massimo le potenzialità della piattaforma sottostante, come il multi-threading. Questo comporta che il codice JavaScript e il rendering dell'interfaccia grafica vengano eseguiti nello stesso thread, ottenendo così un'interfaccia poco fluida.
- **Esperienza d'uso:** una delle caratteristiche principali delle applicazioni native sono le gesture, l'utente è abituato ad interagire con le applicazioni sviluppate in linguaggio nativo che sfruttano un sistema complesso di riconoscimento delle gesture e che non è replicabile in ambito web.
- **Funzionalità:** non tutte le funzionalità che può sfruttare un'applicazione nativa sono disponibili in un'applicazione ibrida.

Il funzionamento dei framework analizzati in questo capitolo permette di risolvere i problemi principali delle applicazioni ibride, in quanto sfruttando i componenti nativi, non è presente il problema dell'interfaccia grafica che viene renderizzata nello stesso thread che si occupa dell'esecuzione del JavaScript. Sempre per il fatto che vengono utilizzati componenti nativi, è possibile sfruttare lo stesso sistema di riconoscimento delle gesture e le stesse animazioni, rendendo l'esperienza d'uso più simile a quella offerta da un'applicazione realizzata con l'SDK nativo.

## 2.2 Tabris.js

Framework pubblicato da EclipseSource<sup>2</sup> nel Maggio 2014, che permette di controllare mediante JavaScript i componenti dell'interfaccia grafica nativa, sia di iOS, sia di Android.

---

<sup>1</sup>Componente delle applicazioni native che consiste in un browser web con funzionalità ridotte e che permette di visualizzare pagine web scaricate da internet oppure memorizzate in locale.

<sup>2</sup><http://eclipsesource.com/en/home/>



### 2.2.1 Come funziona

Tabris.js utilizza come “*ponte*” una versione modificata di Cordova e dei plug-in personalizzati per incapsulare i componenti grafici offerti dai vari SDK nativi.

Trattandosi di un framework derivato da Cordova è possibile utilizzare i plug-in di Cordova già esistenti per aggiungere nuove funzionalità, oltre a quelle offerte framework, come per esempio l'utilizzo della fotocamera.

L'unica condizione per il corretto funzionamento dei plug-in esterni è che non dipendano dal DOM, dal momento che il DOM non è presente durante l'esecuzione di un'applicazione.

### 2.2.2 Pregi e difetti

Uno dei pregi di Tabris.js è quello che il codice JavaScript scritto è indipendente dalla piattaforma, questo permette di utilizzare lo stesso codice sorgente e lo stesso layout sia per iOS sia per Android, si occupa il framework di eseguire tutte le operazioni specifiche per le varie piattaforme.

Un altro pregio deriva dall'estensibilità, è infatti possibile utilizzare sia dei plug-in di Cordova, sia i moduli disponibili su npm, con la condizione che questi non dipendano dal DOM.

Le criticità di Tabris.js riguardano per lo più il layout che deve essere fatto in modo imperativo, prima definendone posizione e dimensione, e poi combinandoli tra loro in modo gerarchico.

Sempre per quanto riguarda il layout, la personalizzazione è limitata in quanto risulta complesso se non impossibile definire dei componenti grafici composti o con layout particolari, come la visualizzazione a griglia.

Infine, il framework non impone né suggerisce alcun pattern architetturale da adottare, lasciando completa libertà al programmatore, con il rischio che il codice sorgente dell'applicazione diventi complesso e difficile da mantenere.

### 2.2.3 Prototipo

Nel realizzare l'applicazione prototipo sono state riscontrate varie problematiche in particolare riguardanti il layout. Ad esempio, non è stato possibile disporre gli elementi in una griglia e la personalizzazione della barra di navigazione si è rilevata essere molto limitata.

Un altro problema emerso durante la realizzazione del prototipo è stata la scarsa disponibilità di materiale on line, infatti oltre alle risorse messe a disposizione dai creatori e alla documentazione ufficiale, non è stato possibile trovare altre informazioni riguardanti il framework. Tuttavia, l'applicazione realizzata è molto fluida e l'esperienza d'uso è paragonabile a quella di un'applicazione nativa.

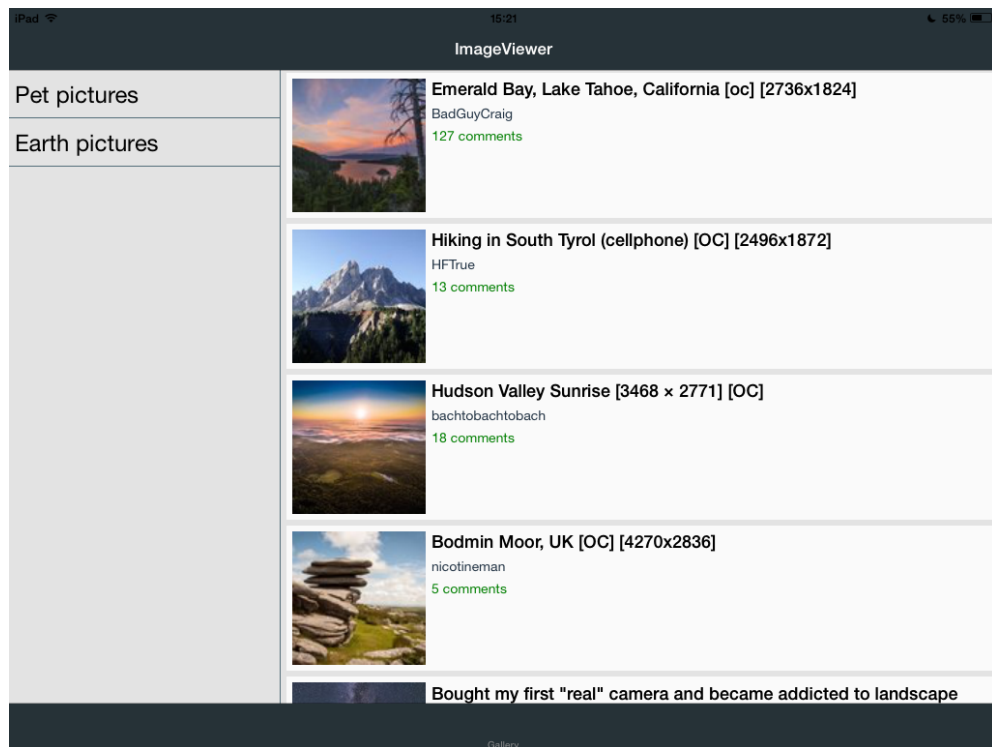


Figura 2.1: Screenshot del prototipo realizzato con Tabris.js

## 2.3 NativeScript

Framework rilasciato da Telerik nel Maggio 2015 che permette di realizzare applicazioni mobile native sia per iOS che per Android rendendo possibile utilizzare tutte le API native mediante JavaScript.

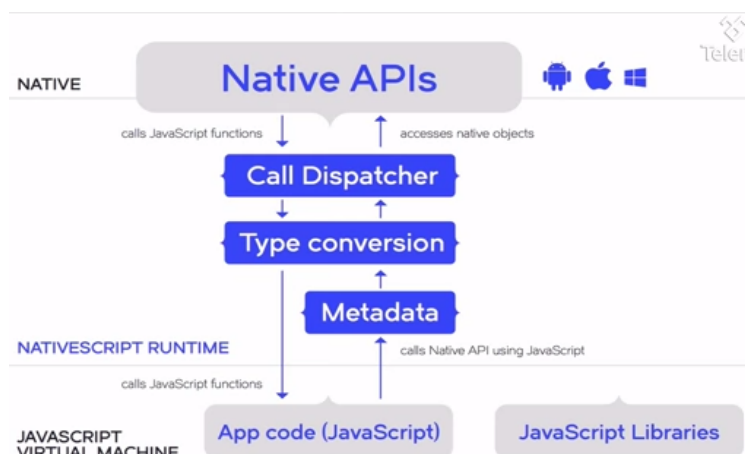
### 2.3.1 Come funziona

NativeScript utilizza come “*ponte*” una virtual machine appositamente modificata che all’occorrenza usa il C++ per invocare le funzioni scritte in linguaggio nativo (Obj-C, Java). Questa virtual machine deriva da V8 se l’applicazione viene eseguita su Android o da JavaScriptCore nel caso l’applicazione venga eseguita su iOS.

Le modifiche subite dalla virtual machine riguardano:

- la possibilità di intercettare l’esecuzione di una funzione JavaScript ed eseguire in risposta del codice C++ personalizzato;
- l’iniezione di meta dati che descrivono le API native;

In questo modo il runtime di NativeScript può utilizzare i meta dati per riconoscere le funzioni JavaScript che richiedono l’esecuzione di codice nativo ed eseguire il corrispettivo codice nativo invocando le funzione con delle istruzioni scritte in C++.



**Figura 2.2:** Schema rappresentante il runtime di NativeScript

Di conseguenza il runtime di NativeScript permette di creare degli oggetti JavaScript che funzionano da proxy rispetto ad oggetti nativi, specifici della piattaforma. Un esempio del funzionamento è dato dal seguente codice che su iOS crea un oggetto di tipo `UIAlertView`:

```
1 var alert = new UIAlertView();
```

**codice 2.1:** Esempio di creazione di un oggetto nativo

All'esecuzione del JavaScript la virtual machine riconosce, grazie ai meta dati iniettati, che la funzione JavaScript deve essere eseguita come una funzione nativa.

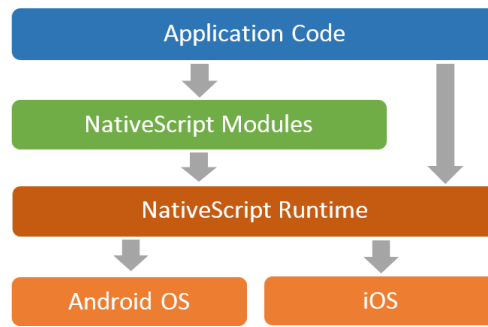
Di conseguenza, utilizzando del codice C++, invoca la corrispondente funzione Obj-C che in questo caso istanzia un oggetto UIAlertView e salva un puntatore all'oggetto nativo in modo da poter eseguire delle funzioni su di esso.

Alla fine crea un oggetto JavaScript che funziona come un proxy dell'oggetto nativo precedentemente creato e lo ritorna in modo che possa essere memorizzato come oggetto locale.

I meta dati che vengono iniettati nella virtual machine e descrivono tutte le funzioni offerte dalle API native della piattaforma e vengono ricavati durante il processo di compilazione dell'applicazione utilizzando la proprietà di riflessione dei linguaggi di programmazione.

Il vantaggio di questa implementazione è che tutte le API native sono invocabili da JavaScript e anche le future versioni delle API saranno supportate subito, così come possono essere supportate librerie di terze parti scritte in Java/Obj-C.

Per permettere il riuso del codice NativeScript fornisce dei moduli che aggiungo un livello di astrazione ulteriore rispetto alle API native, questo livello contiene sia componenti grafici sia funzionalità comuni ad entrambe le piattaforme come l'accesso al filesystem o alla rete. Questo livello di astrazione è opzionale ed è sempre possibile effettuare chiamate alle API native.



**Figura 2.3:** Architettura di NativeScript

### 2.3.2 Pregi e difetti

Il pregio principale di NativeScript è che rende disponibili ad un'applicazione nativa realizzata in JavaScript tutte le funzionalità che possono essere presenti su un'applicazione realizzata con l'SDK nativo.

Inoltre, l'interfaccia grafica di un'applicazione viene realizzata in modo dichiarativo mediante XML e CSS<sup>3</sup>, in un modo analogo a quello utilizzato per le applicazioni web.

Tuttavia, le funzionalità offerte dal livello di astrazione sono limitate e di conseguenza è necessario utilizzare le API native. Questo comporta la diminuzione del codice riusabile su più piattaforme e un notevole aumento della complessità, allontanandosi così dallo scopo principale dell'utilizzo del JavaScript per lo sviluppo di applicazioni native, che è quello di sviluppare in modo semplice e senza utilizzare le API native.

### 2.3.3 Prototipo

Sfruttando quasi solamente i componenti generici offerti da NativeScript è stato possibile ottenere un layout simile a quello dell'applicazione attuale.

Tuttavia per realizzare la visualizzazione a griglia delle immagini è stato necessario utilizzato un componente **Repeater** all'interno di una **ScrollView**, questa implementazione risulta poco efficiente e poco fluida, in quanto non essendo mappata su un componente nativo non gode delle stesse ottimizzazioni.

Sono state individuate soluzioni alternative sfruttando librerie native di terze parti, che non sono state adottate in quanto ritenute troppo complesse dal momento che facevano uso largo di codice specifico per iOS.

---

<sup>3</sup>NativeScript implementa un sotto insieme limitato di CSS in quando le istruzioni CSS devono essere applicate sui componenti nativi.

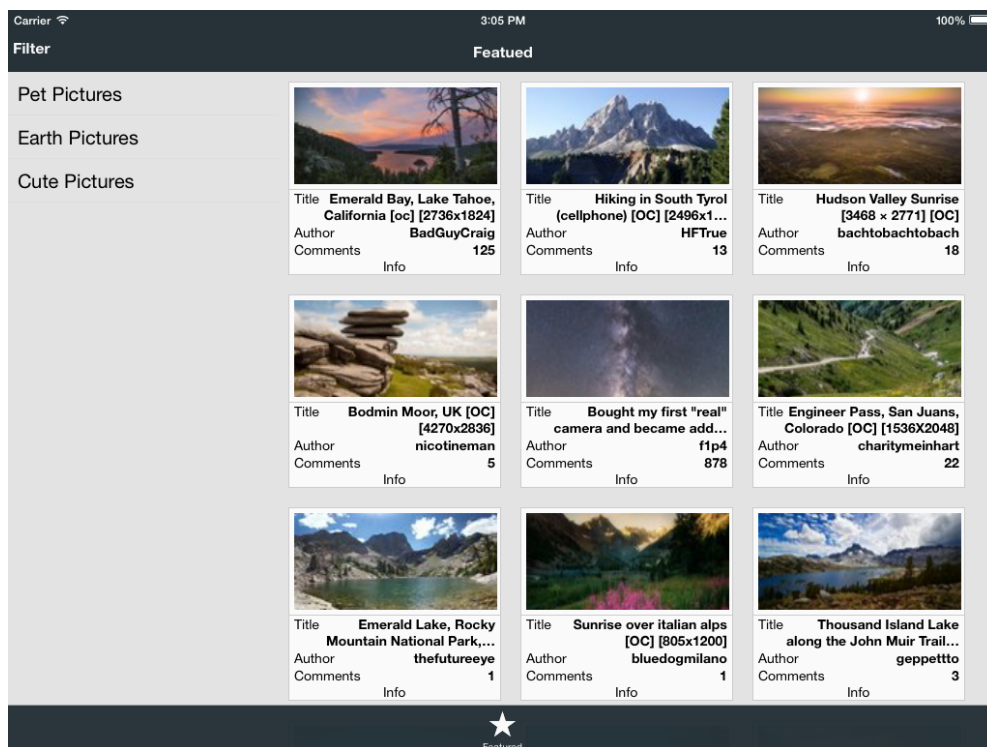


Figura 2.4: Screenshot del prototipo realizzato con NativeScript

## 2.4 React Native

Framework sviluppato da Facebook come progetto interno per la realizzazione di applicazioni native per iOS sfruttando il JavaScript e con un funzionamento analogo a quello di React<sup>4</sup>.

React Native è stato successivamente rilasciato come progetto open source nel Marzo 2015.

### 2.4.1 Come funziona

React Native è composto da una parte scritta in Obj-C e un'altra parte scritta in JavaScript. La parte realizzata in Obj-C comprende:

- una serie di classi che definiscono il “*ponte*” che permette di alla virtual machine di invocare codice nativo;
- un'insieme di macro che permettono alle classi Obj-C di esportare dei metodi in modo che questi possano essere invocati dal “*ponte*”;
- un'insieme di classi che derivano dai componenti nativi di uso comune e che utilizzano le macro per esportare alcune funzionalità.

La parte realizzata in JavaScript consiste in un livello di astrazione, organizzato in moduli, che nascondere l'interazione con le componenti native, viene inoltre fornito

<sup>4</sup>Framework per lo sviluppo di applicazioni web pubblicato da Facebook.

il modulo `NativeModules` che permette di interagire direttamente con il “*ponte*” in modo da utilizzare oggetti nativi personalizzati.

Quando viene avviata un’applicazione con React Native, viene eseguito del codice Obj-C che istanzia la virtual machine che andrà ad interpretare il JavaScript e il “*ponte*” tra la virtual machine e il codice nativo.

Durante l’esecuzione del codice JavaScript è possibile richiedere l’esecuzione di codice nativo usando il modulo `NativeModules`, questo modulo fornisce all’oggetto “*ponte*” le informazioni necessarie per permettergli di identificare la porzione di codice nativo da eseguire. Per poter essere eseguito il codice nativo deve utilizzare le macro fornite da React Native, altrimenti il “*ponte*” non riesce ad identificare il metodo da invocare.

Al fine di ottenere prestazioni migliori, la virtual machine che esegue il JavaScript viene eseguita su un thread diverso rispetto a quello che si occupa dell’esecuzione del codice nativo e del rendering dell’interfaccia grafica.

Inoltre, la comunicazione tra questi due thread viene gestita in modo asincrono ed eseguita in blocchi, in questo modo è possibile ridurre le comunicazioni tra i due thread ed evitare che l’interfaccia grafica si blocchi durante l’esecuzione del codice JavaScript.

### 2.4.2 Pregi e difetti

React Native fornisce un buon livello di astrazione rispetto la piattaforma nativa, rendendo possibile ad uno sviluppatore web di realizzare un’applicazione nativa completa senza conoscere nulla riguardo il funzionamento della piattaforma sotto stante.

A causa di questa astrazione non tutte le funzionalità native sono disponibili, tuttavia è possibile adattare classi Obj-C già esistenti mediante le macro messe a disposizione dal framework, anche se questo processo prevede una buona conoscenza del linguaggio Obj-C.

Tra gli altri pregi di React Native c’è la community di sviluppatori creatasi attorno a framework, infatti, nonostante si tratti di un framework pubblicato recentemente, si è già creata una community numerosa ed è già possibile trovare dei moduli open source che estendono le funzionalità base del framework.

Infine il flusso di lavoro per lo sviluppo di un’applicazione con React Native risulta molto veloce, in quanto grazie all’utilizzo dei WebSocket, non è necessario eseguire la build dell’applicazione ad ogni modifica del codice sorgente, portando un notevole risparmio di tempo.

### 2.4.3 Prototipo

Il prototipo realizzato è stato in grado di soddisfare la maggior parte delle caratteristiche ricercate, infatti è stato possibile ottenere una visualizzazione a griglia, dotata di scroll infinito e fluido.

Sono stati invece incontrati dei problemi riguardanti la personalizzazione della barra di navigazione, in quanto il componente offerto dal framework non permette la presenza di pulsanti personalizzati nella barra di navigazione.

Tuttavia sono state individuate alcune soluzioni che prevedono l’utilizzo di moduli open source che forniscono una barra di navigazione maggiormente personalizzabile.

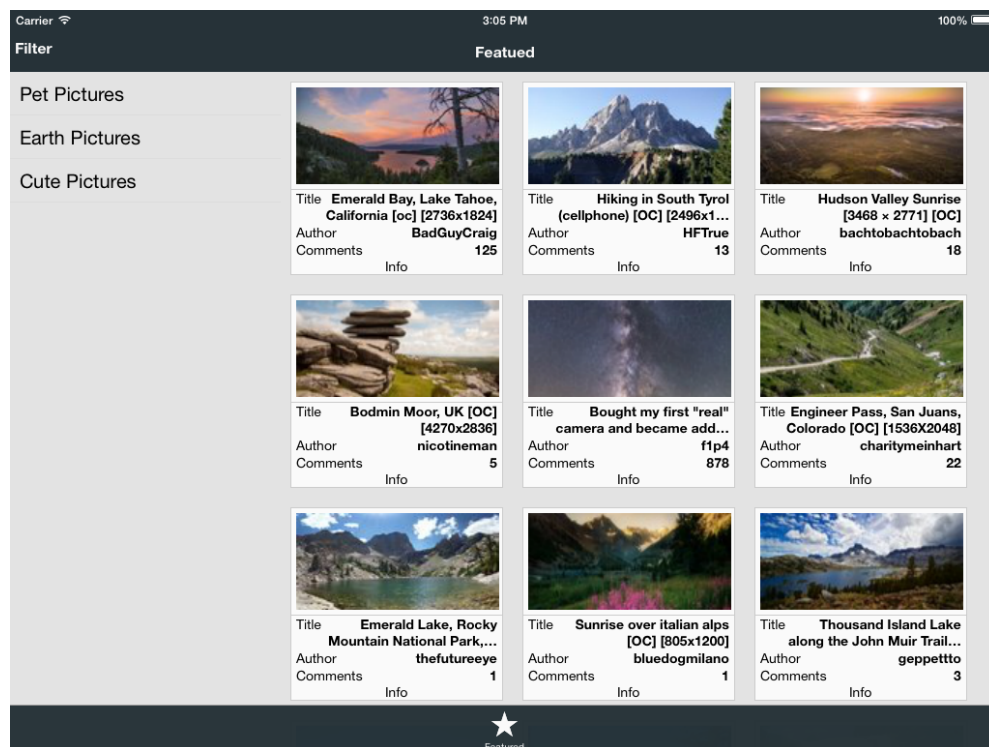


Figura 2.5: Screenshot del prototipo realizzato con React Native

## 2.5 Confronto finale

Trovare un nome migliore

-	Tabris.js	NativeScript	React Native
Funzionamento	Utilizza dei plug-in di Cordova per rendere accessibili i componenti grafici nativi via JavaScript	Utilizza una VM modificata e dei meta dati relativi alle API native per renderne possibile l'utilizzo via JavaScript	Utilizza delle classi Obj-C per creare un ponte tra la VM che esegue il JavaScript e le componenti native
Possibilità di personalizzazione	Limitata a quanto offerto dal framework	Come se l'applicazione fosse scritta in linguaggio nativo	Limitata a quanto offerto dal framework
Estensibilità	Mediante plug-in di Cordova	Mediante librerie native di terze parti	Estendendo librerie native di terze parti con le macro offerte dal framework
Piattaforme supportate	iOS e Android	iOS e Android	iOS

**Tabella 2.1:** Tabella comparativa dei framework analizzati

## 2.6 Framework scelto

Dopo aver esaminato e confrontato i tre framework individuati si è scelto di utilizzare React Native nella seconda parte dello stage.

Questo perché React Native si è dimostrato un framework molto potente anche se non offre l'accesso completo alle API native come fa NativeScript. Inoltre, la diffusione già ampia del framework e il supporto da parte di Facebook, che sta usando React Native per sviluppare le proprie applicazioni, fornisce al framework un'ampia possibilità di crescita, mediante l'estensione di nuove funzionalità o il supporto di ulteriori sistemi operativi, come Android.

Per quanto riguarda NativeScript, è stato scartato principalmente perché non raggiunge l'obiettivo di nascondere la complessità dello sviluppo di un'applicazione nativa con Obj-C o Swift. Infatti, l'obiettivo che si vuole raggiungere sviluppando un'applicazione nativa in JavaScript è quello di riutilizzare il più possibile le competenze derivate dallo sviluppo web senza dover imparare tutti i dettagli dello sviluppo con l'SDK nativo.

Infine, Tabris.js è stato scartato perché è risultato troppo limitato e non è stato in grado di soddisfare alcune delle caratteristiche ricercate.



## Capitolo 3

# Strumenti e tecnologie utilizzate

Il contenuto di questo capitolo contiene una descrizione più dettagliata delle tecnologie e degli strumenti utilizzati per sviluppare l'applicativo oggetto dello stage.

Come anticipato nel precedente capitolo, si è scelto di utilizzare React Native come framework principale per lo sviluppo dell'applicazione, il che vincola la scelta di alcuni strumenti e tecnologie.

### 3.1 React Native

Trattandosi di un framework per la definizione di interfacce grafiche, React Native prevede di strutturare l'applicazione secondo *componenti*, ognuno dei quali viene definito combinando componenti standard offerti dalla libreria o altri componenti definiti dallo sviluppatore.

React Native infatti offre una serie di componenti basilari che permettono di definire l'interfaccia grafica dell'applicazione e che vengono poi tradotti in componenti nativi, questi componenti possono sia semplici come `View`, `Text` o `Image`, sia componenti più complessi come `TabBarIOS` o `MapView`.

Ogni componente di un'applicazione realizzata con React Native ha sempre due proprietà:

- **state**: un oggetto che contiene le informazioni riguardanti lo stato del componente, i cambiamenti fatti a questo oggetto prevedono il re-rendering dell'interfaccia grafica. Tipicamente viene utilizzato per memorizzare gli oggetti che contengono le informazioni da visualizzare sull'interfaccia grafica.
- **props**: un oggetto che contiene le informazioni che il componente riceve dal componente che lo contiene, spesso consistono i dati di visualizzare o in delle funzioni di callback da invocare al verificarsi di determinati eventi. Tipicamente i campi dati di questo oggetto vengono considerati immutabili in modo da evitare *side effect* indesiderati.

Questi due oggetti portano ad un pattern comune nella progettazione dei componenti detto *Smart & Dumb*, che prevede la divisione dei componenti in due categorie, quelli *smart* che svolgono il ruolo di *controller* dell'applicazione e quelli *dumb* sono più simili a dei template.

Tipicamente un componente *dumb* non ha un proprio stato e si limita a visualizzare i dati ricevuti mediante l'oggetto **props** o ad invocare delle callback al verificarsi di

determinati eventi. In questo modo si ottengono dei componenti generici, indipendenti dall'applicazione che possono essere testati e riutilizzati facilmente.

Un componente *smart* invece è tipicamente composto da più componenti, sia *dumb* che *smart*, ed è dotato di un proprio stato, i cui dati vengono passati ai componenti figli. Inoltre contiene la definizione delle funzioni che si occupano della gestione degli eventi.

Nel caso l'applicazione segua il design pattern Flux (§3.2), i componenti *smart* sono quelli che si occupano di recuperare lo stato dagli *stores* e di creare le varie *actions*.

Un altro pattern comune nelle applicazioni sviluppate con React Native è quello di definire un singolo componente per ogni file di codice che deve contenere tutto il codice del componente, sia quello riguardante la logica di funzionamento, sia quello riguardante la logica di layout. Questo è reso possibile dal fatto che le informazioni riguardanti lo stile sono definite come oggetti JavaScript e il layout viene definito utilizzando la sintassi JSX.

### 3.1.1 La sintassi JSX

La sintassi JSX permette di inserire all'interno del codice JavaScript alcuni pezzi di codice XML, che devono essere poi trasformati in JavaScript normale per poter essere eseguiti.

Il vantaggio offerto da questa sintassi è quello di poter definire in modo dichiarativo come i vari componenti dell'applicazione si compongono tra loro, semplificando così la definizione dell'interfaccia grafica da parte del programmatore.

```
1 <View style={styles.container}>
2   <Text style={styles.welcome}>
3     Welcome to React Native!
4   </Text>
5   <Text style={styles.instructions}>
6     To get started, edit index.ios.js
7   </Text>
8   <Text style={styles.instructions}>
9     Press Cmd+R to reload,{'\n'}
10    Cmd+D or shake for dev menu
11  </Text>
12 </View>
```

**codice 3.1:** Esempio della sintassi JSX di React Native

Nel caso di React Native, la traduzione da JSX a JavaScript viene poi fatta dal *packager* prima della compilazione dell'applicazione (§3.3).

### 3.1.2 Oggetti JavaScript per la definizione dello stile

Come anticipato la definizione dello stile dei componenti di un'applicazione viene effettuato utilizzando degli oggetti JavaScript che hanno dei campi dati che ricordano le proprietà dei CSS.

Questa scelta è stata effettuata perché il team di sviluppo di React Native ha dovuto implementare un sistema che trasformi le proprietà CSS in attributi dei componenti nativi.

Nell'implementare questo sistema è stato scelto di utilizzare degli oggetti JavaScript al posto dei CSS puri perché, in questo modo vengono risolti alcuni problemi dei CSS, come la località dei nomi delle classi e la gestione delle variabili. Inoltre, non è più necessario riferirsi ad una determinata classe CSS utilizzando una stringa, in quanto

basta usare un campo `data` di un oggetto JavaScript, evidenziando così eventuali errori riguardanti il nome della classe errato.

```
1 var styles = StyleSheet.create({
2   container: {
3     flex: 1,
4     justifyContent: 'center',
5     alignItems: 'center',
6     backgroundColor: '#F5FCFF',
7   },
8   welcome: {
9     fontSize: 20,
10    textAlign: 'center',
11    margin: 10,
12  },
13  instructions: {
14    textAlign: 'center',
15    color: '#333333',
16    marginBottom: 5,
17  },
18 });
```

**codice 3.2:** Esempio della definizione dello stile di un componente di React Native

Le impostazioni dello stile sono analoghe a quelle offerte dai CSS ed includono alcune funzionalità che non sono ancora pienamente supportate nell'ambito web come il sistema di layout flexbox. Questo sistema prevede che un componente dell'applicazione possa andare a modificare le dimensioni dei componenti che contiene, in modo da occupare al meglio lo spazio disponibile e di allineare in vari modi i componenti contenuti.

### 3.1.3 JavaScript ES6 e ES7

Il codice JavaScript prodotto utilizzando React Native segue lo standard ES5 dal momento che è lo standard supportato dalla versione attuale di JavaScriptCore<sup>1</sup>.

Tuttavia è possibile utilizzare alcune funzionalità specifiche degli standard ES6 e ES7, come la destrutturazione degli oggetti e l'utilizzo delle classi, dal momento che il *packager* di React Native (§3.3), prima di compilare l'applicazione nativa, compila il codice JavaScript utilizzando Babel<sup>2</sup>, un compilatore per JavaScript che trasforma la sintassi ES6 e ES7 in modo che sia conforme allo standard ES5.

### 3.1.4 Componenti esterni

React Native supporta la gestione dei moduli secondo lo standard CommonJS<sup>3</sup>, questo permette di utilizzare npm per la gestione delle dipendenze con i componenti esterni.

Sfruttando le possibilità offerte da npm, la community di sviluppatori ha già iniziato a pubblicare alcuni componenti per le applicazioni di React Native, che possono essere facilmente integrati nella propria applicazione. Una raccolta di questi componenti può essere trovata sul sito React Parts<sup>4</sup>, il quale contiene componenti sia per React che per React Native.

<sup>1</sup> JavaScriptCore è la virtual machine che interpreta il JavaScript a runtime.

<sup>2</sup> <https://babeljs.io/>

<sup>3</sup> <http://requirejs.org/docs/commonjs.html>

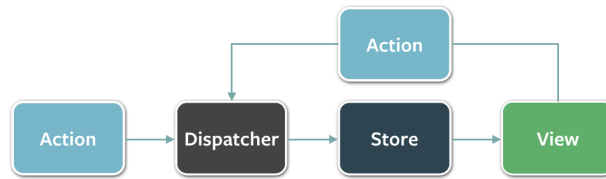
<sup>4</sup> <https://react.parts/native-ios>

## 3.2 Flux

Flux è un pattern architetturale per le applicazioni sviluppate con React e React Native proposto da Facebook.

Questo pattern sfrutta il sistema di composizione delle view di React costruendo un flusso di dati unidirezionale, che a partire da un oggetto *store* diventano lo stato di un oggetto *view-controller*, che a sua volta li fornisce ai propri componenti.

L'unico modo per modificare i dati presenti in uno *store* è mediante un *action*, quando un *view-controller* vuole modificare i dati a seguito di un evento scatenato dall'utente, crea un *action* che, mediante un *dispatcher*, viene ricevuto dai vari *store* dell'applicazione.

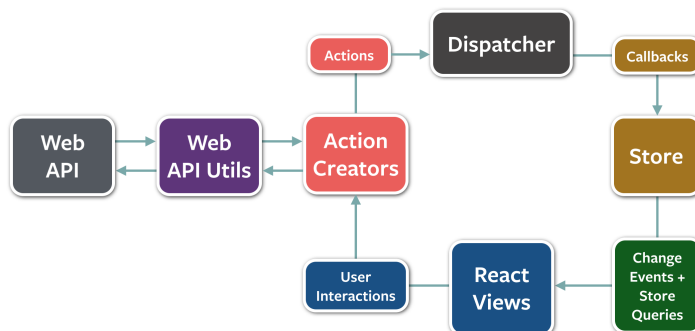


**Figura 3.1:** Diagramma del pattern Flux

Come anticipato, Flux prevede tre tipologie principali di componenti:

- **Stores:** sono dei *singleton* che contengono i dati dell'applicazione, forniscono solamente dei metodi *getter*, per interagire con i dati contenuti è necessario usare le *actions*.
- **Actions:** sono degli oggetti che contengono delle informazioni riguardante alle varie operazioni che possono essere eseguite dagli *stores* dell'applicazione. Tipicamente vengono create dai *view-controller* di React e contengono già i dati necessari agli *store* per aggiornarsi, nel caso di operazioni asincrone i *view-controller* creano l'azione che verrà comunicata al *dispatcher* solamente quando sono stati caricati i dati.
- **Dispatcher:** oggetto che riceve un *action* e ne esegue il broadcast verso tutti gli *stores* registrati dell'applicazione. Fornisce delle funzionalità che permettono ai vari *stores* di registrarsi e di specificare eventuali dipendenze verso altri *stores* dell'applicazione in modo che l'aggiornamento di un determinato *store* venga effettuato una volta completato l'aggiornamento degli *stores* da cui dipende, evitando così di ottenere uno stato inconsistente.

### 3.2.1 Sequenza delle azioni



**Figura 3.2:** Funzionamento del pattern Flux

1. L'utente esegue un'azione sulla view.
2. Il gestore dell'evento crea un *action* e la comunica al *dispatcher*.
3. Il *dispatcher* manda a tutti gli *stores* registrati l'oggetto *action* ricevuto.
4. Ogni *store* esamina l'oggetto *action* e se necessario si aggiorna.
5. Gli *stores* che hanno subito modifiche emettono un evento per comunicare ai componenti React in ascolto che si devono aggiornare.
6. I componenti React richiedono agli *stores* i dati per aggiornarsi.

### 3.2.2 Differenze con MVC

Nonostante Flux ed MVC possano sembrare due pattern totalmente diversi, in realtà Flux è una variante del MVC classico con delle modifiche che lo adattano al funzionamento di React e React Native.

Infatti, con MVC, i *controllers* interagiscono con il *model* e le *view* dell'applicazione visualizzano i dati presenti al suo interno. Quando il *model* viene modificato, le *view* vengono notificate e recuperano i dati aggiornati dal *model*.

Mentre con Flux, i *view-controller* aggiornano il *model*, definito dagli *stores*, in un modo più strutturato utilizzando le *actions* e, una volta che l'aggiornamento degli *store* è completato, i *view-controller* vengono notificati in modo che possano recuperare i nuovi dati.

Considerando che per React e React Native una *view* e il relativo *controller* sono lo stesso oggetto diventa chiaro che la logica di base è la stessa, l'unica differenza è come viene effettuato l'aggiornamento dei dati, che con Flux deve passare attraverso delle *actions*.

Questo vincolo imposto dall'utilizzo delle *actions* permette di circoscrivere la logica di aggiornamento del *model* all'interno del *model* stesso, limitando la complessità dell'applicazione, che nel caso di grandi applicazioni può diventare ingestibile.

Un altro vantaggio che viene dall'adozione di Flux con React riguarda l'aggiornamento dell'interfaccia grafica a seguito di una modifica dei dati, in quanto sia React che Flux ragionano a stati: con React l'interfaccia grafica visualizza uno stato dell'applicazione, il cambiamento dello stato comporta il re-rendering dell'interfaccia, mentre

con Flux l'insieme degli *stores* rappresenta lo stato dell'applicazione e l'esecuzione di un'azione comporta il cambiamento dello stato.

Di conseguenza è possibile collegare direttamente lo stato definito dagli *stores*, con lo stato dei componenti grafici, limitando il numero di operazioni intermedie.

### 3.3 React Native CLI

React Native è dotato di un'interfaccia a riga di comando che può essere installata via npm e che comprende una serie di tool per lo sviluppo di supporto allo sviluppo delle applicazioni.

In particolare contiene il *packager*, un programma che permette di creare dei bundle JavaScript contenenti il codice dell'applicazione.

Quando richiesto, il *packager* esegue la conversione da JSX in JavaScript, trasformando anche il JavaScript ES6 in JavaScript ES5, e successivamente crea il bundle, unendo tutti i file JavaScript necessari al funzionamento dell'applicazione.

Il bundle deve poi essere aggiunto al progetto Xcode in modo che sia disponibile sul dispositivo.

Durante lo sviluppo di un'applicazione è inoltre possibile configurare il progetto Xcode in modo che recuperi il bundle JavaScript dell'applicazione da un server presente sul computer con il quale si sta sviluppando. Così facendo è possibile eseguire ogni volta il codice aggiornato senza dover reinstallare l'applicazione sul dispositivo o sul simulatore. L'avvio e la gestione di questo server viene gestita in modo automatico dai tool che vengono installati con l'interfaccia a riga di comando.

### 3.4 Atom e Nuclide

L'editor di testo prescelto per lo sviluppo di un'applicazione con React Native è Atom<sup>5</sup>, un editor open source sviluppato di GitHub, che può essere personalizzato mediante dei pacchetti.

È stato scelto Atom perché Facebook ha rilasciato Nuclide<http://nuclide.io/>, una serie di pacchetti da installare su Atom che aggiungono alcune funzionalità di supporto allo sviluppo con React Native, come l'auto-completamento delle keyword e l'evidenziazione della sintassi JSX.

### 3.5 Xcode

Nonostante il codice JavaScript possa essere scritto con qualsiasi editor di testo, è necessario utilizzare Xcode per compilare l'applicazione finale in modo da poterla installare sul simulatore di iOS o su un dispositivo Apple.

Xcode rende disponibili una serie di tool per lo sviluppo delle applicazioni native che possono essere riutilizzati durante lo sviluppo di un'applicazione con React Native. Ad esempio durante l'esecuzione dell'applicazione sul simulatore di iOS è possibile controllare il consumo della memoria, il traffico dati e l'utilizzo della CPU.

---

<sup>5</sup><https://atom.io/>

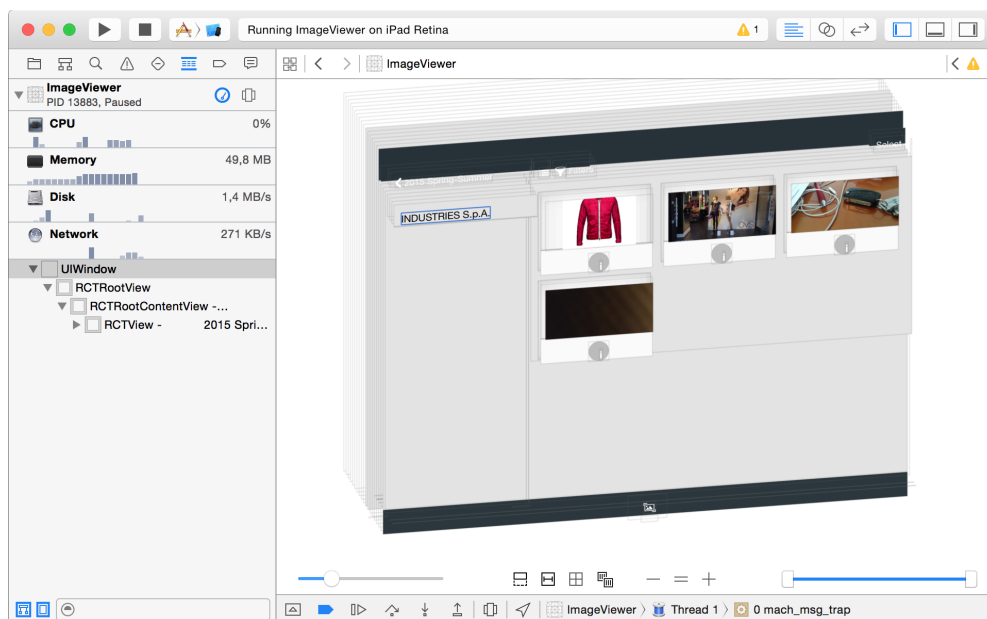


Figura 3.3: Tools di debug di Xcode

## 3.6 Google Chrome Dev Tools

Durante lo sviluppo di un'applicazione con React Native è possibile utilizzare i developers tool di Google Chrome per effettuare il debug del codice JavaScript.

In particolare è possibile utilizzare il debugger di Chrome per effettuare inserire dei break point nel codice dell'applicazione ed effettuare l'esecuzione passo passo del codice, oppure è possibile stampare sulla console di Chrome mediante `console.log`.

Al momento non è possibile utilizzare tutti i tools in quanto la modalità debug di React Native modifica la virtual machine che esegue JavaScript.

Infatti, durante l'esecuzione normale il JavaScript viene interpretato da JavaScriptCore, mentre, durante il debug, viene utilizzata la virtual machine V8 presente all'interno di Chrome, al quale comunica con l'applicazione nativa mediante WebSocket.

In questo modo Chrome riesce a controllare l'esecuzione del JavaScript, ma non riesce ad accedere alle funzionalità che vengono eseguite dai componenti nativi, come l'utilizzo delle risorse di rete o la gestione della memoria.

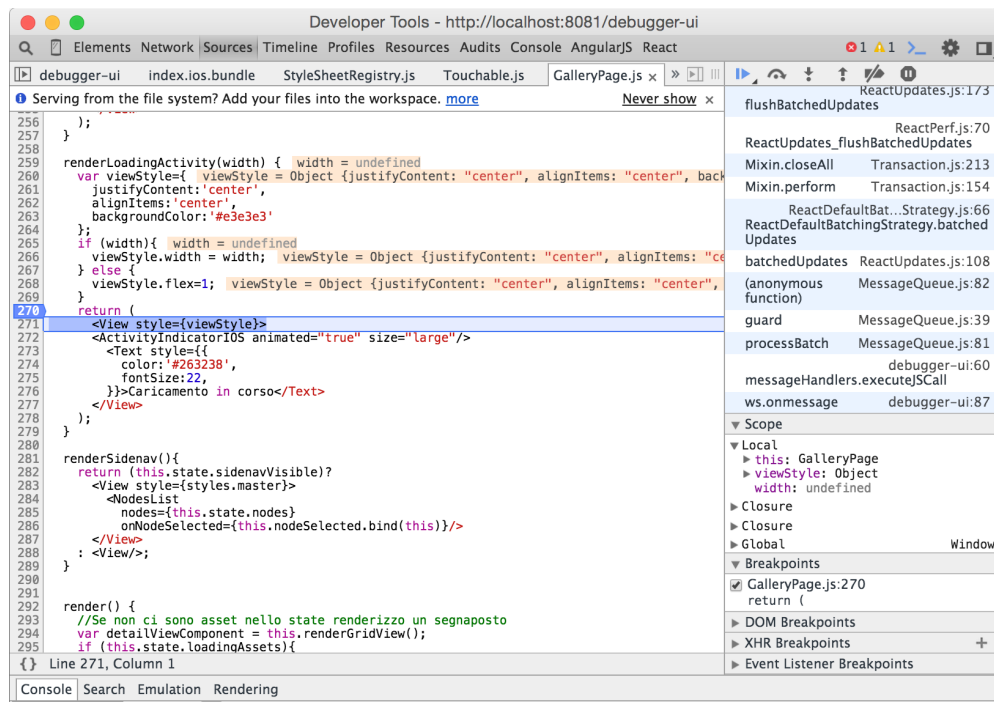


Figura 3.4: Debug di un'applicazione con Google Chrome



## Capitolo 4

# Analisi dei Requisiti

Questo capitolo contiene i requisiti dell'applicazione che sono stati individuati sia discutendo con il tutor aziendale, sia analizzando l'attuale client per iPad di WARDA.

Il contenuto di questo capitolo è il risultato dei vari sprint effettuati, in quanto l'intero processo di sviluppo del prodotto è stato svolto in modo simile a quanto previsto dalla metodologia agile Scrum.

Dopo un'analisi iniziale dell'applicazione attuale, effettuata in modo da identificare il dominio applicativo, tutte le attività sono state svolte secondo degli *sprint*, ognuno dei quali mirato ad implementare una determinata caratteristica dell'applicazione.

Ogni *sprint* è stato caratterizzato da un breve riunione con il tutor aziendale, seguita da un'analisi dei nuovi requisiti emersi, per poi passare alla progettazione e all'implementazione delle nuove funzionalità.

### 4.1 Applicazione attuale

L'applicazione attuale per iPad di WARDA permette di visualizzare il contenuto di una gallery che si trovata sul server principale dell'applicazione.

Oltre alla visualizzazione della gallery è possibile anche creare nuovi asset, recuperando un'immagine dalla libreria interna del dispositivo o dalla fotocamera e accedere alla funzionalità collaborative offerte dalla piattaforma WARDA.

Per il progetto dello stage, l'azienda è interessata solamente alla componente gallery dell'applicazione, in quanto si tratta della parte della applicazione che richiede la maggior quantità di risorse e che attualmente soffre di alcuni problemi prestazionali.

La struttura dati alla base di una gallery realizzata con WARDA è un albero, composto da vari nodi, ognuno dei quali contenete un'insieme di assets e dei possibili filtrinzion Per WARDA un'asset è un'immagine a cui vengono associati dei meta dati che la descrivono, questi meta dati vengono poi utilizzati dal sistema di filtri definiti sul nodo per permettere all'utente di visualizzare un sotto insieme degli assets contenuti nel nodo.

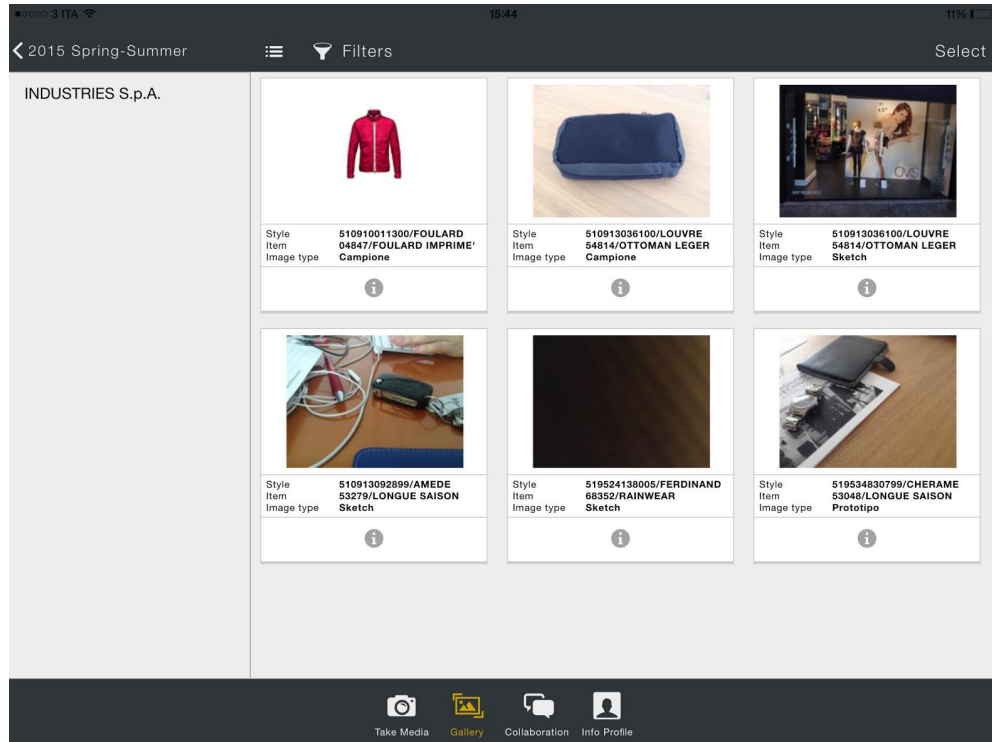
#### 4.1.1 Pagina di visualizzazione della gallery

La pagina dell'applicazione che visualizza la gallery è composta da tre parti principali:

- Una lista che visualizza i nodi figli del nodo corrente;
- Una griglia che visualizza gli asset presenti nel nodo corrente;

- Una lista di filtri che visualizza i filtri che possono essere applicati sugli assets contenuti nel nodo corrente.

Ognuno di questi componenti sarà descritto nelle seguenti sotto sezioni.



**Figura 4.1:** Screenshot della gallery dell'applicazione attuale

### Lista dei nodi

**AskAlberto:** Immagine della lista dei nodi, dove si vede più di qualche nodo

La lista dei figli del nodo corrente visualizza, per ogni nodo, il titolo e quando l'utente seleziona un nodo dalla lista, la lista viene aggiornata in modo che visualizzi i nodi figli del nodo selezionato, inoltre, viene anche aggiornata la griglia degli assets in modo che visualizzi gli assets contenuti nel nodo selezionato.

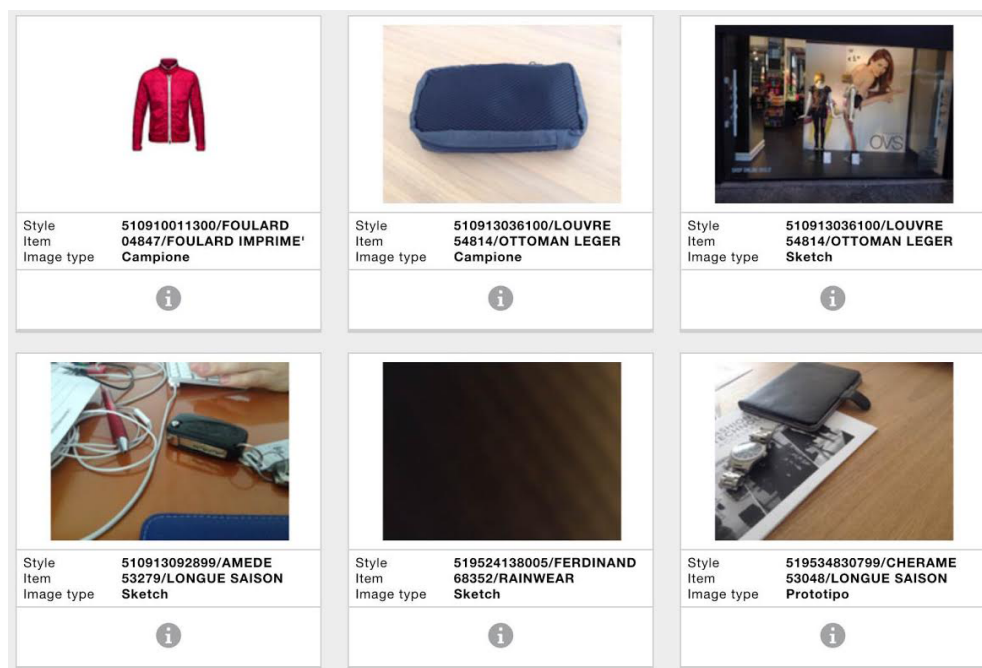
Durante il caricamento dei dati della lista viene visualizzato un indicatore di attività per fornire all'utente un feedback riguarda l'operazione di caricamento dei dati in corso.

Sopra la lista dei nodi è presente un pulsante che permette di tornare al nodo padre del nodo correntemente visualizzato.

Questo pulsante è caratterizzato da una freccia indietro e dal titolo del nodo correntemente visualizzato, nel caso il nodo corrente sia il nodo radice della gallery, il pulsante non deve essere visibile.

### Griglia degli assets

La griglia degli assets rappresenta il componente principale dell'applicazione, questa griglia permette di visualizzare per ogni assets contenuto nel nodo corrente un'immagine



**Figura 4.2:** Dettaglio della griglia degli assets dell'applicazione attuale

di anteprima e un pulsante che permette all'utente di visualizzare i dettagli dell'asset mediante un popover.

Se l'utente esegue un tap sull'immagine di un asset, viene visualizzata una pagina dell'applicazione contenente i dettagli dell'asset e un'immagine ingrandita, maggiori informazioni riguardo questa pagina sono disponibili nella sezione §4.1.2.

**AskAlberto:** Quanti assets vengono caricati per volta dalla griglia?

Per motivi prestazionali, la griglia non carica subito tutti gli assets contenuti nel nodo corrente ma si limita a visualizzare solo i primi 25 assets, i successivi assets contenuti nel nodo vengono caricati man mano che l'utente prosegue nella visualizzazione della griglia, secondo il sistema dello scroll infinito.

### Lista dei filtri

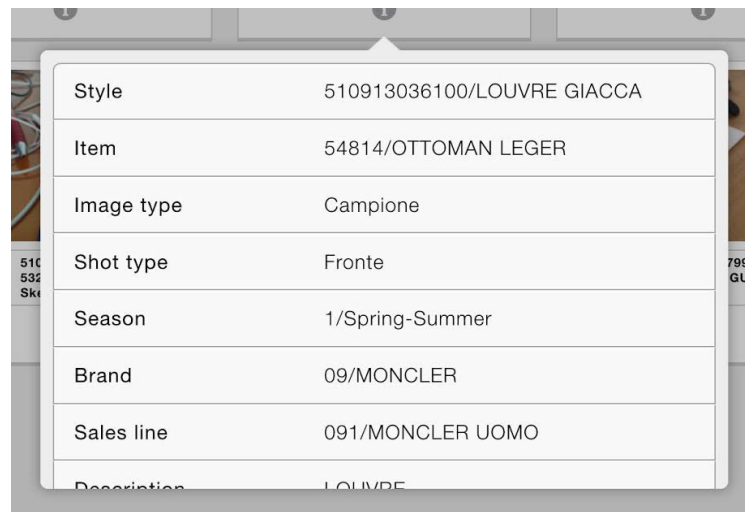
**AskAlberto:** Immagine della lista dei filtri

La lista dei filtri compare come pop up quando l'utente esegue un tap sul pulsante "Filters" presente nella barra di navigazione.

Per ogni filtro presente nel nodo corrente, viene visualizzata una lista dei possibili valori che possono essere assegnati al filtro ed una casella di testo che permette all'utente di filtrare i valori presenti nella lista.

### 4.1.2 Pagina di dettaglio di un asset

Questa pagina contiene un'immagine ingrandita di un asset e una lista con tutti i dettagli dell'asset.



Style	510913036100/LOUVRE GIACCA
Item	54814/OTTOMAN LEGER
Image type	Campione
Shot type	Fronte
Season	1/Spring-Summer
Brand	09/MONCLER
Sales line	091/MONCLER UOMO
Description	LOUVRE

**Figura 4.3:** Popover che mostra i dettagli di un asset

Mediante un apposito pulsante l'utente può nascondere o rendere visibile la lista dei dettagli, in modo da lasciare più spazio all'immagine.

Sull'immagine l'utente può eseguire alcune gesture:

- Swipe verso sinistra, per visualizzare l'asset successivo contenuto nel nodo visualizzato dalla gallery.
- Swipe verso destra, per visualizzare l'asset precedente contenuto nel nodo visualizzato dalla gallery.
- Pinch-to-zoom, per ingrandire ulteriormente l'immagine.

Infine, nella barra di navigazione della pagina è presente un pulsante che permette all'utente di tornare alla pagina con la visualizzazione a griglia.

AskAlberto: immagine della visualizzazione di dettaglio, sia con la sidenav aperta, sia con la sidenav chiusa

## 4.2 Requisiti individuati

I requisiti individuati dall'analisi dell'applicazione attuale e dalle riunioni con il tutor aziendale sono stati categorizzati secondo il codice:

$$R[T][I][C]$$

dove:

- **Tipo:** specifica la tipologia del requisito e può assumere i seguenti valori:
  - **F** - *funzionale*, cioè che determina una funzionalità dell'applicazione;
  - **V** - *vincolo*, che riguarda un vincolo che il prodotto deve rispettare.
- **Importanza:** specifica l'importanza del requisito e può assumere i seguenti valori:

- **O** - *obbligatorio*, indica che il requisito deve essere soddisfatto per garantire il corretto funzionamento dell'applicazione;
  - **D** - *desiderabile*, indica che il requisito non è fondamentale per il funzionamento minimo dell'applicazione, ma che comunque rappresenta un aspetto importante dell'applicazione;
  - **F** - *facoltativo*, indica che il requisito fornisce del valore aggiunto all'applicazione
- **Codice**: rappresenta un codice che identifica il requisito all'interno di una gerarchia. Questo codice è definito in modo che il requisito  $RTI.x.y$  sia un requisito che va a definire con un grado maggiore di dettaglio alcuni degli aspetti del requisito  $RTI.x$ .

Descrizione dell'importanza dei requisiti

### 4.2.1 Requisiti Funzionali

Id Requisito	Descrizione
RFO1	L'utente deve poter visualizzare una gallery dell'applicazione WARDA a partire dal nodo radice della gallery
RFO1.1	L'utente deve poter visualizzare la lista dei nodi contenuti nel nodo correntemente visualizzato
RFD1.1.1	Durante il caricamento della lista dei nodi, deve essere presente un indicatore di attività per evidenziare l'attività in corso
RFD1.2	L'utente deve poter rendere visibile la lista dei nodi contenuti nel nodo corrente
RFF1.2.1	La comparsa della lista dei nodi deve avvenire in modo animato
RFD1.3	L'utente deve poter nascondere la lista dei nodi contenuti nel nodo corrente
RFF1.3.1	La scomparsa della lista dei nodi deve avvenire in modo animato
RFO1.4	L'utente deve poter spostarsi tra i nodi presenti in una gallery
RFO1.4.1	L'utente deve poter selezionare un nodo contenuto nel nodo corrente per visualizzarne i contenuti
RFO1.4.2	L'utente deve poter ritornare al nodo precedente visualizzato
RFF1.4.3	Il cambiamento degli elementi presente nella lista dei nodi deve essere animato
RFO1.4.4	Lo spostamento da un nodo all'altro deve comportare l'aggiornamento della lista dei nodi figli e della lista degli assets
RFO1.5	L'utente deve poter visualizzare la lista degli assets contenuti nel nodo correntemente visualizzato
RFD1.5.1.	Durante il caricamento degli elementi della lista, deve essere presente un indicatore di attività per evidenziare l'attività in corso
RFO1.5.2.	La lista contenente gli assets deve avere un layout a griglia
RFO1.5.3.	La lista deve visualizzare un numero limitato di assets ed essere dotata di un sistema di scroll infinito

<b>Id Requisito</b>	<b>Descrizione</b>
RFO1.5.3.1	Una volta che l'utente deve visualizza tutti gli contenuti della griglia, se presenti, devono essere caricati ulteriori assets
RFO1.5.3.2	Durante il caricamento degli ulteriori assets deve essere presente un indicatore di attività
RFO1.5.4.	Gli elementi della lista degli assets devono essere composti da un'immagine di anteprima dell'asset e da un pulsante "Info"
RFO1.5.4.1	Quando l'utente esegue un tap sull'immagine di anteprima di un asset, deve essere visualizzata la pagina di dettaglio dell'asset
RFO1.5.4.2	Quando l'utente esegue un tap sul pulsante Info di un asset, deve essere visualizzato un popover contenente le informazioni di dettaglio dell'asset
RFO1.6	L'utente deve poter visualizzare la lista dei filtri disponibili per il nodo correntemente visualizzato
RFO1.6.1	Per ogni filtro disponibile, l'utente deve poter visualizzare tutti i valori che può assumere il filtro
RFD1.6.1.1	L'utente deve poter cercare un determinato valore all'interno della lista dei valori che può assumere il filtro
RFO1.6.1.2	L'utente deve poter selezionare un valore per il filtro da applicare
RFO1.6.2	L'utente deve poter applicare più filtri contemporaneamente
RFO1.6.3	L'utente deve poter rimuovere un filtro
RFO1.6.4	L'utente deve poter rimuovere tutti i filtri applicati
RFD1.6.5	L'utente deve poter visualizzare il numero di filtri applicati
RFO1.6.6	I filtri devono rimanere attivi anche se l'utente si sposta su un altro nodo
RFD1.6.7	La lista dei filtri deve essere visualizzata come un pop-up
RFO2	L'utente deve poter visualizzare una pagina contenente i dettagli di un asset
RFO2.1	L'utente deve poter tornare alla pagina contenente la gallery
RFO2.2	L'utente deve poter visualizzare un'immagine ingrandita dell'asset
RFD2.2.1	Durante il caricamento dell'immagine, deve essere presente un indicatore di attività che visualizzi la percentuale di caricamento
RFF2.2.2	L'utente deve poter effettuare il pinch-to-zoom sull'immagine
RFO2.2.3	L'utente deve poter effettuare uno swipe da destra verso sinistra sull'immagine, per visualizzare in dettaglio l'asset successivo secondo l'ordine del contenuto della gallery
RFF2.2.3.1	Allo swipe deve essere associata un'animazione che sposti l'immagine da destra verso sinistra seguendo il movimento effettuato dall'utente
RFF2.2.3.2	Nel caso non sia presente un'asset successivo da visualizzare, l'animazione dello swipe deve essere interrotta
RFO2.2.4	L'utente deve poter effettuare uno swipe da sinistra verso destra sull'immagine, per visualizzare in dettaglio l'asset precedente secondo l'ordine del contenuto della gallery
RFF2.2.4.1	Allo swipe deve essere associata un'animazione che sposti l'immagine da sinistra verso destra seguendo il movimento effettuato dall'utente
RFF2.2.4.2	Nel caso non sia presente un'asset precedente da visualizzare, l'animazione dello swipe deve essere interrotta

Id Requisito	Descrizione
RDO2.3	L'utente deve poter visualizzare una lista contenente i dettagli dell'asset visualizzato
RFD2.4	L'utente deve poter rendere visibile la lista contenente i dettagli dell'asset visualizzato
RFF2.4.1	La comparsa della lista contenente i dettagli deve avvenire in modo animato
RFD2.5	L'utente deve poter nascondere la lista contenente i dettagli dell'asset visualizzato
RFF2.4.1	La scomparsa della lista contenente i dettagli deve avvenire in modo animato
RFF3	L'utente deve visualizzare un messaggio d'errore nel caso l'applicazione non riesca a connettersi con il server

Tabella 4.1: Requisiti Funzionali

### 4.2.2 Requisiti di Vincolo

Id Requisito	Descrizione
RVD1	L'applicazione deve essere dotata di un file di configurazione che permette di impostare: l'indirizzo del server a cui connettersi, l'id della gallery Rda visualizzare, l'username e password con i dati da utilizzare per effettuare l'accesso e il numero di assets da visualizzare in una singola pagina
RVO2	L'applicazione deve essere realizzata con React Native
RVO3	L'applicazione deve essere compatibile con iPad di seconda generazione
RVD4	L'interfaccia grafica dell'applicazione deve essere fluida e non bloccarsi

Tabella 4.2: Requisiti di Vincolo

## 4.3 Riepilogo requisiti

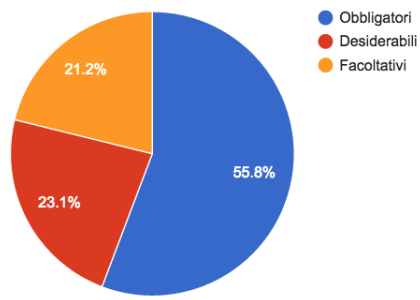
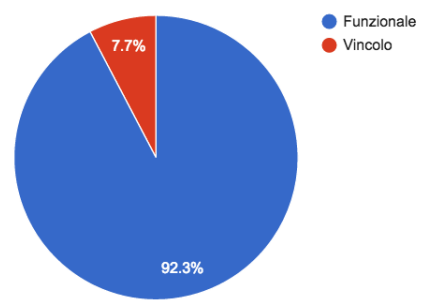
In totale sono stati individuati 52 requisiti, ripartiti tra le varie tipologie secondo quanto riportato nelle seguenti tabelle.

Importanza	#
Obbligatoria	29
Desiderabili	12
Facoltativi	11
Totale	52

Tabella 4.3: Numero di requisiti per importanza

Tipologia	#
Funzionali	48
Vincolo	4
Totale	52

Tabella 4.4: Numero di requisiti per tipologia

**Figura 4.4:** Requisiti per importanza**Figura 4.5:** Requisiti per tipologia



## Capitolo 5

# Progettazione

### 5.1 API REST di WARDA

### 5.2 Architettura generale dell'applicazione

#### 5.2.1 Model

#### 5.2.2 Stores

#### 5.2.3 Actions

#### 5.2.4 Pages

#### 5.2.5 Components

#### 5.2.6 Utils

### 5.3 Diagrammi di attività

#### 5.3.1 Navigazione nella gallery

#### 5.3.2 Applicazione di un filtro



## Capitolo 6

# Realizzazione

discorso introduttivo

### 6.1 Dal prototipo alla gallery

Descrizione dell'adattamento del prototipo alla gallery

Aggiunta di funzionalità come lo scroll infinito

### 6.2 Sistema di navigazione

Descrizione dell'aggiunta del sistema di navigazione tra i nodi della gallery

### 6.3 Sistema dei filtri

Come sono stati aggiunti i filtri alla navigazione

Come sono stati aggiunti i filtri alla navigazione

### 6.4 Visualizzazione di dettaglio

Navigazione tra pagine

Swipe

Immagine con indicatore di caricamento

### 6.5 Animazioni

Animazione delle comparse/scomparsa

Animazione dello swipe

Animazione del pinch-to-zoom non implementata

## 6.6 Gestione degli errori

Errori di comunicazione

## Capitolo 7

# Conclusioni

### 7.1 Valutazione del risultato

Commento sul risultato ottenuto e sulle presentazioni dell'applicazione

#### 7.1.1 Requisiti soddisfatti

Tabellina riassuntiva dei requisiti soddisfatti

### 7.2 Aspetti critici e possibili estensioni

Del framework e dell'applicazione

Storia del log-in e dell'interfaccia grafica

### 7.3 Conoscenze acquisite

Tecnologie studiate

e



**Riferimenti bibliografici**

**Siti Web consultati**