

ALMA MATER STUDIORUM - UNIVERSITA' DI
BOLOGNA

CAMPUS DI CESENA
SCUOLA DI SCIENZE

CORSO DI LAUREA IN INGEGNERIA E
SCIENZE INFORMATICHE

Un prototipo di Caccia al tesoro per device mobili

Relazione finale in
Tecnologie Web

Relatore

Dott.ssa Silvia MIRRI

Presentata da

Leonardo MONTINI

Sessione II

Anno Accademico 2016/2017

Introduzione

Questa tesi nasce in parallelo con lo sviluppo del progetto *Caccia alla Ricerca - Chi ri-cerca trova*, che verrà presentato durante la *Notte dei Ricercatori 2017* a Cesena.

Lo scopo del progetto consiste nella realizzazione di un gioco, realizzato con una qualche tecnologia, che permetta ai partecipanti di poter visitare le varie esposizioni presenti durante la serata in una maniera un po' diversa da quella che può essere una semplice serata ad un evento culturale.

Il target di utenza del gioco sono i ragazzi delle scuole superiori, abituati alle classiche visite scolastiche. Per questo motivo infatti si è deciso di realizzare un gioco, il cui scopo non dovesse essere unicamente quello di far visitare più esposizioni possibili ai partecipanti in modo da imparare e scoprire nuove realtà, ma di permettergli di fare tutto ciò divertendosi. In conclusione di questa tesi verrà infatti affrontato un po' più nel dettaglio il termine *Edutainment*, ossia imparare divertendosi. C'è molta più produttività nel divertirsi mentre si fanno cose utili ed importanti, come appunto arricchire il proprio sapere.

Per loro, è stato pensato di poter sfruttare la certezza che avranno sicuramente in tasca uno *smartphone*, quindi la piattaforma sicuramente più comoda e già conosciuta dal target su cui costruire il nostro gioco sarà proprio il cellulare.

Oggetto di questa tesi sarà proprio la ricerca della tecnologia più adatta da utilizzare per la realizzazione di questo progetto, partendo dalla nascita del *World Wide Web*. Il motivo di questo punto di partenza è il seguente: così come a scuola ci fanno studiare la storia, per comprendere le ragioni che hanno portato determinate usanze o situazioni ad essere tali, per analizzare le scoperte del passato e per non ripetere gli errori già commessi, ho deciso di iniziare la mia analisi dagli inizi della condivisione di informazioni attraverso due terminali remoti.

La prima parte ripercorrerà alcune delle principali tappe nella storia del web e servirà ad approfondire le motivazioni per le quali da sempre si cerca il

modo migliore per raggiungere l'utente finale, rendendo i meccanismi per gli sviluppatori sempre più rapidi ed efficienti, per poi soffermarsi ed approfondire le tecnologie attuali e su cosa lo stato dell'arte mette a disposizione per lo sviluppo di un software pensato per i dispositivi mobili.

Fondamentale sarà la seconda parte nella ricerca delle ultime tecnologie offerte sul mercato, per quanto riguarda le tecnologie web ed il loro apporto all'ambiente mobile. Quando si pensa allo sviluppo tecnologico, in qualsiasi settore, tra le prime riflessioni che possono sorgere c'è sicuramente quella riferita alla *velocità* del progresso. Con il passare degli anni, il progresso tecnologico si muove sempre più velocemente. Non solo è in costante crescita, ma è anche in costante accelerazione ed il settore mobile non è da meno. Ogni anno nuove tecnologie, nuovi dispositivi con nuovi processori, sensori e funzionalità talvolta nemmeno mai immaginate fino all'anno prima. Per questo motivo sarà un ulteriore fattore di sfida, e di interesse, studiare ed analizzare tutto quel che il mercato offre al momento con la consapevolezza che tutto ciò che ora può venir considerato l'*ottimo*, domani potrebbe già non esserlo più.

Quanto appena detto rafforza ancora di più l'utilità dell'analizzare sempre le nuove tecnologie in relazione con le precedenti, poichè saranno sempre le fondamenta su cui vegono costruite le scoperte e le tecnologie future. Conoscere approfonditamente una tecnologia significa essere in grado di apprezzarne le qualità ed i difetti, ed è proprio questo il punto di partenza per poter sviluppare qualcosa di ancora migliore.

La conoscenza dei punti di forza servirà per avere degli agganci, dei punti di partenza su cui continuare a spingere il miglioramento, mentre sui punti deboli sarà necessario dedicare particolare attenzione per analizzare l'opzione di risolvere i problemi seguendo la strada già intrapresa, oppure valutare di riprogettare qualcosa con lo stesso scopo, ma con filosofie diverse se le precedenti sono state giudicate errate o comunque non ottimali.

Nel terzo capitolo verrà approfondito l'ambiente di sviluppo nel quale sarà poi sviluppato il progetto.

Infine il quarto capitolo, dopo aver ricercato e scelto la tecnologia che verrà utilizzata per la realizzazione del progetto, sarà dedicato ad un approfondimento delle caratteristiche del software per poi passare alla panoramica del

funzionamento, tecnico e applicativo, dell'applicazione da presentare durante la Notte dei Ricercatori.

Indice

1	World Wide Web	7
1.1	La nascita	7
1.2	Web 2.0	8
1.3	Layout	11
2	Applicazioni per dispositivi mobili	13
2.1	Responsive Design	14
2.2	Mobile First	16
2.3	Progressive Web App	19
2.4	Applicazioni Ibride	26
3	Tecnologie coinvolte: Ionic	33
3.1	Framework	33
3.2	Ionic v1.0	34
3.3	Ionic 2	36
3.4	Tecnologie di sviluppo	43
4	Caccia alla Ricerca	51
4.1	Introduzione	51

4.2	Funzionamento Tecnico	55
4.3	Edutainment	62
5	Conclusioni	65

Capitolo 1

World Wide Web

1.1 La nascita

Negli ultimi anni, ad una velocità sempre maggiore, la diffusione della tecnologia è sempre più alla portata di tutti. Siamo passati dal veder utilizzare i Computer solo dagli "addetti ai lavori", fino al Personal Computer che ha trovato posto nella maggior parte delle abitazioni familiari, in tutto il mondo.

Gli utilizzi del Personal Computer in casa erano molteplici, soprattutto grazie alle primitive connessioni internet. Bastava rinunciare all'utilizzo della linea telefonica per comunicare con altre persone per avere accesso al nuovo mondo di Internet tramite un modem, convertitore di segnale da analogico a digitale e viceversa.

La nascita del WWW (World Wide Web) avvenne grazie a Tim Berners-Lee, un informatico britannico che nel 1991 avviò il primo server web con pagine contenenti informazioni in formato testuale accessibili tramite browser. O meglio, *ipertestuale*, ossia pagine con testi e riferimenti ad altre pagine.

Per ancora diversi anni, nonostante una vasta diffusione, il web era limitato ad essere un'immensa biblioteca di pagine statiche, quasi senza nessuna interazione da parte dell'utente che doveva limitarsi a consultare i contenuti reperiti online. Ufficialmente, questo status cambia intorno al 2004 con la nascita di quello che viene chiamato *Web 2.0*.

1.2 Web 2.0

Tim O'Reilly nel 2005, in *What is Web 2.0* [20], fa un'attenta analisi sulle novità che concettualmente portano a distinguere quello che verrà etichettato come vecchio Web1.0, a discapito delle novità sotto il nome di Web 2.0. Secondo O'Reilly, per sopravvivere in questa nuova era, le aziende dovranno basarsi (a seconda della loro specializzazione) su questi sette principi fondamentali, dettati dal veloce cambiamento.

- Servizi piuttosto che software pre-confezionati.
- Controllo su fonti di dati uniche, difficili da riprodurre ed arricchite con l'utilizzo degli utenti.
- Dare fiducia agli utenti come co-sviluppatori.
- Dare importanza all'intelligenza della collettività.
- Fare leva sull'effetto *The Long Tail* [1] attraverso un customer self-service
- Mantenere il software ad un livello superiore rispetto al singolo dispositivo
- Interfaccia utente, modello di sviluppo e di business leggeri

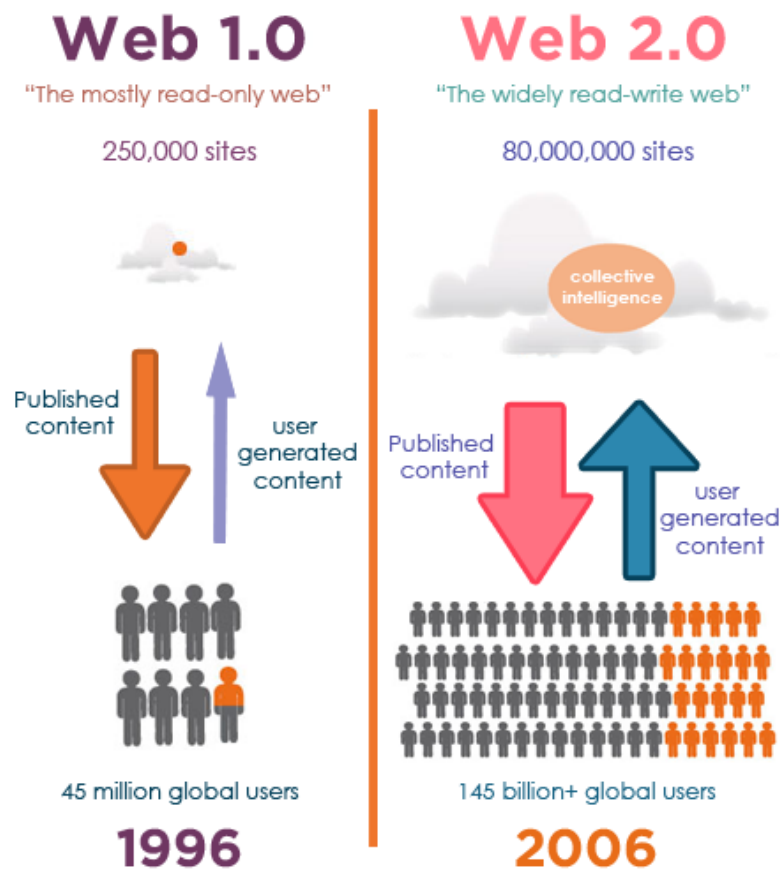


Figura 1.1: Illustrazione grafica sulla principale differenza tra 1.0 e 2.0, la partecipazione dell'utente come creatore di contenuti e non solo come lettore. [znetlive.com]

Peer to Peer

Un chiaro esempio pratico di come è cambiato un servizio, la distribuzione di file e contenuti non solo testuali (principalmente multimediali), è tra *Akamai* e *BitTorrent*. Entrambe hanno lo stesso scopo comune, ossia la distribuzione di contenuti per gli utenti del web, ma seguono due filosofie completamente diverse. La prima, fondata nel 1998 a Cambridge (Massachusetts) da tre studenti ed un docente del MIT, gestisce tutta la distribuzione dei contenuti tramite i server centrali, mantenendo quindi *passiva* l'interazione dell'utente finale, che può solamente richiedere e scaricare i contenuti. BitTorrent invece segue una filosofia completamente diversa chiamata *Peer to Peer*, spesso abbreviata come P2P. Secondo il P2P non è più il server principale a gestire

tutto il processo, ma garantisce un ruolo fondamentale ad ogni singolo utente, o peer.

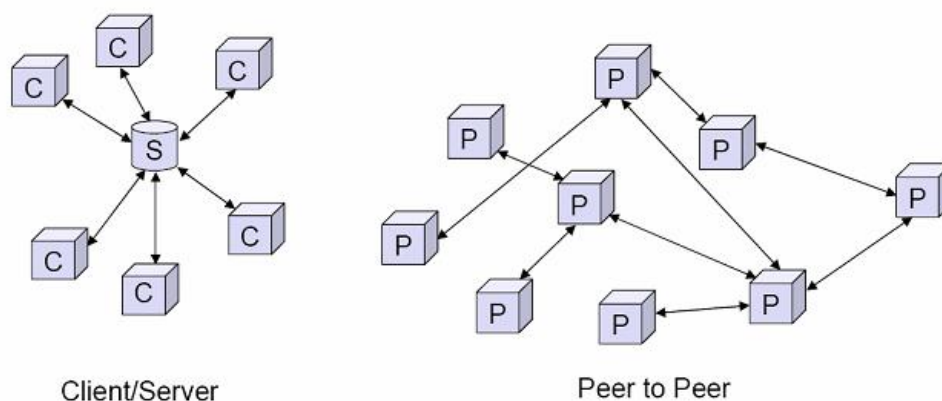


Figura 1.2: Strutture a confronto: Client-Server vs Peer to Peer. [ragazziweb.it]

Secondo la filosofia del Peer to Peer, ogni client è anche un server. Tutti gli utenti connessi al servizio si inviano e ricevono dati tra di loro, esplodendo i singoli file in più piccoli frammenti da poter poi diffondere attraverso il sistema, e ricostruire localmente alla richiesta dei file. Questo significa che più il file è popolare, maggiori saranno i possessori di frammenti dello stesso e più veloce sarà il processo di reperimento e download del file richiesto.

Più utenti significa migliori prestazioni, praticamente l'opposto rispetto ad una situazione di servizio centralizzato a cui tutti gli utenti si connettono, rallentandone le prestazioni in caso di numeri eccessivamente elevati.

Javascript

Un'altra forte svolta è stata la rapida e massiva introduzione di un nuovo linguaggio a supporto della dinamicità delle pagine web, molto più leggero del già presente Java, chiamato *Javascript*. Creato dalla Netscape, doveva originariamente servire da supporto a Java, per questo il nome originale LiveScript venne poi rinominato in onore della precedente tecnologia.

Tra le particolarità di questo linguaggio, troviamo:

- Non necessita di compilazione, ma basta semplicemente un interprete (che risiede all'interno del browser, lato client) per eseguire le azioni richieste.
- La sintassi è semplice e molto simile a C, C++ e Java.
- È un linguaggio debolmente tipizzato, rendendo più veloci (e meno controllati) alcuni processi.
- Ha tutte le funzionalità tipiche dei linguaggi di alto livello, come controlli di flusso tramite cicli ed if.
- È un linguaggio debolmente orientato agli oggetti, in quanto quello che in Java viene usato come oggetto, in Javascript è più una specie di array associativo, a cui è possibile associare variabili e funzioni.

Per via della particolare leggerezza e praticità, venne fin da subito utilizzato in maniera autonoma piuttosto che limitarsi a supportare le applet Java, diventando così un must in ogni pagina considerata *dinamica* con interazione tra utente e browser.

1.3 Layout

I primi sviluppatori di pagine web statiche non avevano grandi preoccupazioni su come venisse visualizzato il loro contenuto nel computer di qualcun altro in termini di proporzioni e posizione relative degli oggetti sullo schermo, questo perché l'unico dispositivo in grado di accedere a tali contenuti era il computer, la cui visualizzazione era affidata solitamente ad uno schermo quadrato e l'interazione con la pagina avveniva unicamente con mouse e tastiera. Con l'evoluzione tecnologica degli schermi però i primi problemi sono iniziati a sorgere. Ad esempio fissando due menu laterali di dimensioni fisse a sinistra e a destra dello schermo la resa in uno schermo quadrato potrebbe sembrare gradevole, ma visualizzando lo stesso identico contenuto in quello che poteva essere 10 anni fa un "moderno" schermo con risoluzione 16:9 il risultato potrebbe non essere lo stesso. I due menu risultano troppo piccoli e la lettura del contenuto centrale resta scomoda.

Nella migliore delle ipotesi, si crea un inutile spazio vuoto laterale rendendo la navigazione non molto gradevole, ma comunque funzionante come si può vedere in Figura 1.3.

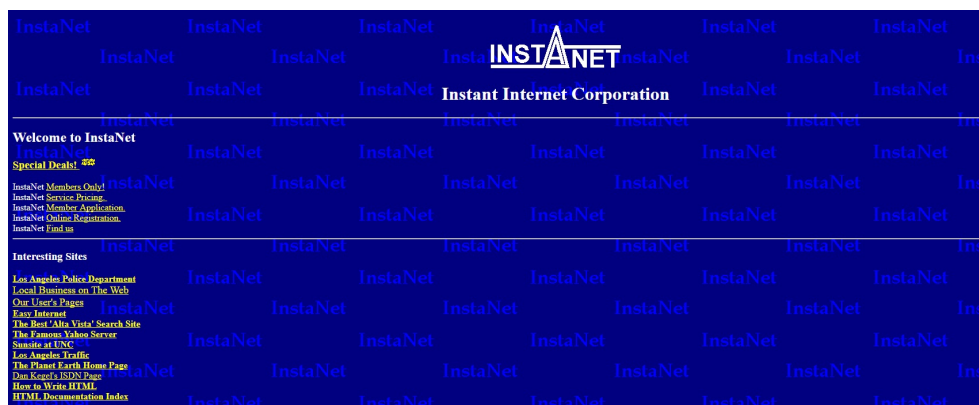


Figura 1.3: Sito sviluppato durante il periodo del web1.0. Si vede chiaramente come l'utilizzo sia una semplice fonte di dati, una biblioteca online, ed il layout risulta scombinato in uno schermo largo.

Se la dimensione dello schermo del computer può venir facilmente risolta evitando dimensioni fisse ma rendendo tutto relativo ad altezza e larghezza, il problema successivo è sicuramente l'arrivo degli smartphone. Le pagine web erano pensate per schermi di certe dimensioni non così ridotte, in via di espansione appunto con la diffusione di schermi widescreen, quindi per non perdere una grossa fetta di utenti e fruitori dei servizi web, gli sviluppatori si sono dovuti adeguare in qualche maniera.

Capitolo 2

Applicazioni per dispositivi mobili

Con l'inizio della diffusione degli smartphone, di dispositivi mobili dallo schermo di dimensioni molto ridotte ma capaci di navigare tra le pagine web, gli sviluppatori del settore hanno dovuto apportare grandi e sostanziali modifiche ai loro contenuti.

Alcuni hanno intrapreso la via della doppia versione del servizio, web e mobile, a seconda del dispositivo utilizzato per accedere al sito o dalle dimensioni rilevate dello schermo. Avere due differenti versioni è sicuramente la scelta migliore per sfruttare al massimo le caratteristiche di entrambe le tipologie di accesso al servizio. La cosiddetta versione *Desktop* avrà solitamente queste caratteristiche:

- Schermo molto grande, solitamente esteso in larghezza.
- Possibilità di suddividere i contenuti in più colonne.
- Menu di navigazione fisso.
- Ampi spazi per inserire banner pubblicitari.

Mentre per quanto riguarda la rispettiva versione *Mobile*, tra le varie caratteristiche si troveranno:

- Schermo molto piccolo, solitamente esteso in altezza ma con la possibilità di venir ruotato, estendendolo nuovamente in larghezza.

- Quasi obbligatorio l'utilizzo di una colonna unica. Utilizzarne due significherebbe forzare l'utente ad usare lo zoom per leggere i contenuti di una colonna alla volta.
- Menu di navigazione a scomparsa.

2.1 Responsive Design

L'alternativa alla creazione di due diverse versioni dello stesso servizio web è creare una struttura dinamica, *Responsive*, in grado di adattarsi autonomamente allo spazio concesso dalle dimensioni dello schermo.

Uno dei primi a parlare di *Responsive Web Design* è il web designer Ethan Marcotte in un libro pubblicato nel 2011. In un articolo pubblicato sul sito alistapart.com [16] riassume l'argomento a partire da esempi pratici nell'architettura. Molto ad effetto il paragone con una recente e molto particolare tipologia di vetro in grado di oscurarsi automaticamente alla presenza di persone all'interno della stanza, tramite una reazione dovuta alla densità dell'aria.

In maniera analoga, le pagine web dovrebbero essere in grado di adattarsi, e per farlo Marcotte in questo articolo parla delle *media query* introdotte con i CSS in versione 2.1, in grado di identificare soltanto la tipologia di device fisico in uso. Per una versione più stabile ed accolta dai principali browser però, è necessario attendere fino alla versione CSS3 dove l'ente W3C ha incluso le *media queries* come parte integrante dello standard. Questa nuova versione di *media queries* era in grado di ispezionare il dispositivo fisico in tutte le sue caratteristiche (utili per il caso in uso) fornendo quindi molte più informazioni e dando più possibilità allo sviluppatore di adattare il proprio contenuto al device.

Le *media query* sono esattamente quel che serve al browser per essere a conoscenza del dispositivo che sta visitando la pagina, come si può vedere in Figura 2.1 dove la stessa ipotetica pagina, prima visualizzata su uno schermo di un computer, viene poi adattata in due diversi modi su device di formato e dimensioni completamente differenti.

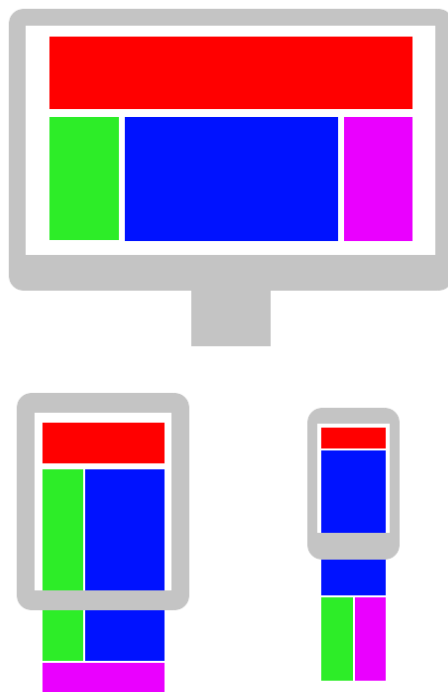


Figura 2.1: Due esempi di come un sito web reponsive può adattare i contenuti alla dimensione dello schermo del dispositivo utilizzato. [wikipedia.it]

Bootstrap

Per fortuna esiste la regola del non reinventare la ruota ogni volta! Nel 2010, direttamente dagli uffici di Twitter, due sviluppatori decidono di creare il framework che nel giro di poco tempo diventerà il più diffuso in assoluto, Bootstrap.

Questo framework permette di costruire pagine web, dalle più semplici alle più complesse, riducendo al minimo lo sforzo semplificando al massimo il processo della costruzione di una pagina web responsive. Grazie ad un intuitivo sistema di griglie e colonne ben istruite su come e dove muoversi al variare delle dimensioni dello schermo, la pagina si adatta quasi automaticamente. Tutti i componenti aggiuntivi che offre il framework sono già costruiti e pensati di default per adattarsi, quindi anche per lo sviluppatore alle prime armi sarà facile ottenere un discreto risultato.

Un altro fattore vincente di Bootstrap, oltre alla semplicità d'uso e all'efficacia, è senza dubbio la personalizzazione. Oltre a poter sovrascrivere le regole CSS a mano creando a cascata nuove regole legate alle classi preesistenti del framework, Bootstrap offre la possibilità di customizzare interamente lo stile tramite UI direttamente sul sito, permettendo quindi di generare e scaricare direttamente la versione personalizzata.

Per versione personalizzata non si parla solo di colori, dimensioni, proporzioni e spazi, ma anche di contenuti potendo aggiungere/togliere eventuali altre funzioni o componenti, rendendo il codice scaricato ed incluso nei propri progetti più veloce e leggero possibile, rimuovendo eventualmente componenti non utilizzati o non necessari. Questa scelta può essere molto conveniente nel caso si stia lavorando ad un progetto piuttosto ampio e pesante in termini di sforzo richiesto al dispositivo per usufruirne, e sicuramente alleggerire il carico rimuovendo contenuti non utilizzati non può far altro che avere un beneficio a livello di prestazioni.

2.2 Mobile First

Grazie alla praticità ed alla semplicità d'uso degli smartphone, come già detto più volte, la tipologia di utenza è in continuo cambiamento. Gli utilizzatori del computer stanno diminuendo in contrasto agli utilizzatori degli smartphone sempre più velocemente.

Basta pensare al fatto che ormai la maggior parte degli adulti, anche se non in grado di utilizzare al meglio il proprio computer (sempre che ne abbiano uno), hanno ora uno smartphone. E lo usano spesso. La breve ricerca di informazioni su internet è tutta un'altra cosa se fatta da uno smartphone. Sei una persona è in cucina ed ha bisogno di rinfrescarsi la memoria su una determinata ricetta, le opzioni sono due: andare in un'altra stanza, accendere il vecchio e lento computer, aspettare diversi minuti che si accenda per poi aprire il browser e cercare finalmente la ricetta, oppure estrarre dalla tasca il proprio smartphone, due tocchi sullo schermo e parte immediatamente la ricerca. Non c'è paragone.

Un altro caso: chi, magari per lavoro, ha particolari esigenze e necessi-

tà preferisce solitamente sedersi comodamente al computer, davanti ad uno schermo molto più grande ed una tastiera fisica, e rispondere alle email. Ma chi non ha nessuna particolare esigenza e sta solamente aspettando l'email di conferma per l'iscrizione ad un sito di viaggi, magari ad un'applicazione, non ha nessuna voglia di andare ad accendere il computer.

Ha ancora senso creare quindi contenuti originariamente per schermi desktop, che si limitano ad adattarsi (seppur con una certa qualità) ai piccoli schermi dei dispositivi mobili? Ecco che entra in gioco la filosofia del *Mobile First*.

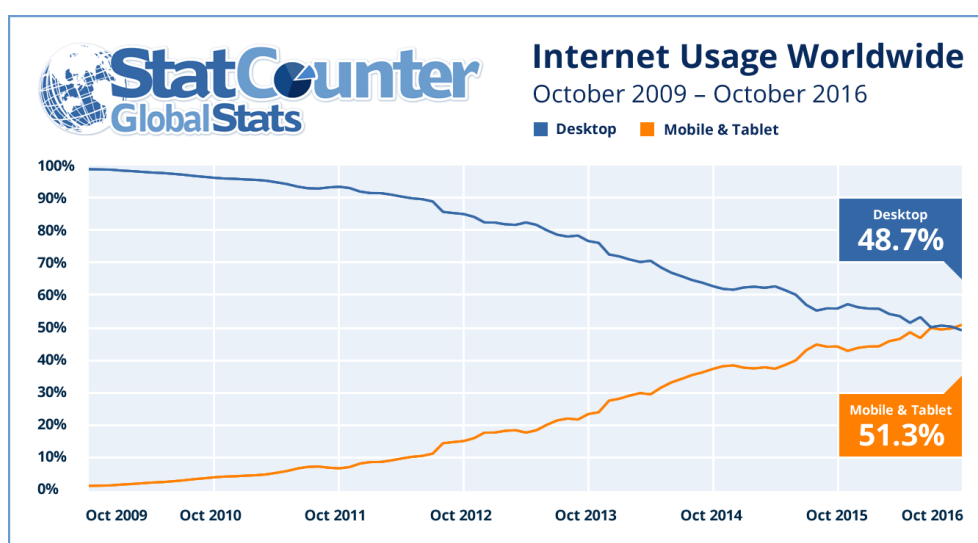


Figura 2.2: Quello che ci si aspettava è diventato realtà, l'utilizzo di internet da parte di dispositivi mobili ha superato nel 2016 quello da parte dei dispositivi mobile secondo statcounter.com. [statcounter.com]

In un interessante articolo [22], Luke Wroblewski Product Director a Google, già dal 2009 guarda verso il futuro ed intuisce come possa aver molto più senso progettare e sviluppare contenuti pensando prima di tutto alla loro visualizzazione su cellulare, ed in secondo luogo gestire, adattare ed espandere il contenuto per una visualizzazione gradevole e funzionale anche da uno schermo desktop.

Analizza la situazione in tre principali punti:

1. **La diffusione dei cellulari sta spopolando.** Già dal 2009 era palese come il mercato avrebbe completamente spostato la concentrazione ver-

so i dispositivi mobili piuttosto che computer desktop. Iniziare il prima possibile a concentrarsi su questa nuova piattaforma era logicamente una scelta vincente.

2. **Lo spazio ridotto di costringe a mettere a fuoco.** Sviluppare un sito web pensando all'utilizzo principale tramite cellulare, costringe lo sviluppatore a doversi concentrare solo ed esclusivamente sulle funzioni veramente importanti e necessarie. Nello sviluppo di un contenuto per desktop si ha molto spazio sullo schermo, e può essere una fonte di distrazione per l'utente così come una tentazione per lo sviluppatore di aggiungere contenuti su contenuti per riempire e sfruttare tutto lo spazio, confondendo solamente l'utilizzatore. Pensare ad un contenuto in uno spazio ridotto quindi rende anche la navigazione migliore per l'utente, che troverà immediatamente quel che cercava.

3. **Il browser mobile offre più possibilità.** Il Web inteso come World Wide Web negli anni 90 è stato costruito come una biblioteca, un insieme di testi e collegamenti, a cui sono poi state aggiunte immagini, formattazione e script per interagire e dare dinamicità. Nel mondo mobile ci sono molte più possibilità di funzioni non pensabili o insensate da vedere nella vecchia ottica di pagina web, tra le quali alcune usufruibili solamente da cellulari grazie a caratteristiche interamente assenti nei computer.

Alcuni esempi:

- **GPS.** A volte capita che alcuni servizi web richiedano la posizione. Se fatto da un computer fisso, sprovvisto di sensore GPS, la posizione viene calcolata in base ai dati forniti dalla connessione, che nella stragrande maggioranza dei casi risulta notevolmente inesatta. Richeidere la posizione ad un cellulare invece risulterà quasi sempre esatta con errori nell'ordine di pochi metri.
- **Posizione del dispositivo.** La combinazione accelerometro/giroscopio può tornare utile in servizi basati sulla geolocalizzazione, in combinazione al GPS, per guidare ad esempio un cliente verso il proprio punto vendita indicandogli la corretta direzione direttamente dal sito web.

- **Multi tocco.** Se tramite computer l'interazione con il puntatore arriva da una sola fonte, il mouse, tramite cellulare è possibile toccare (o cliccare) contemporaneamente in più punti dello schermo stravolgendo completamente la logica di utilizzo del servizio.

Questi tre semplici esempi mostrano come sviluppare un contenuti web pensando ai dispositivi mobili in primo luogo, apre un intero mondo ancora da esplorare a fondo con numerose nuove possibilità, inesistenti fino a pochi anni fa.

Con l'introduzione della filosofia *Mobile First* quindi, gli sviluppatori hanno iniziato a sfruttare al meglio le nuove possibilità offerte dalla nuova piattaforma di accesso ai contenuti e di conseguenza i browser mobile hanno subito e stanno subendo una grossa e rapida crescita in fatto di prestazioni e funzionalità, per permettere appunto la creazione di siti web sempre più interattivi, complessi e utilizzatori di tutti gli strumenti a disposizione.

2.3 Progressive Web App

La ricerca e l'innovazione non si fermano mai ed anche per i siti web, seppur impostati secondo la filosofia *Mobile First*, è stata introdotta una nuova metodologia per sfruttare ancora di più le risorse del dispositivo mobile in uso, andando ben oltre il limite del controllare le dimensioni dello schermo e qualche sensore o attuatore fisico. In questo caso si può parlare di *Progressive Web App*.

Come istanziare una classe secondo il pattern di programmazione *Decorator*, una *Progressive Web App* (PWA) non è altro che un semplice sito web mobile first, circondato da un'armatura piena di ulteriori funzionalità per sfruttare al 100% (compatibilità permettendo) il dispositivo in uso, andando ad inserire quello che era partito come un semplice sito web, tra le applicazioni installate. Un altro grosso passo avanti per le PWA rispetto ai normali siti web non riguarda strettamente la tecnologia in uso, ma soprattutto l'usabilità. Inserire un collegamento diretto alla nostra PWA tramite un'icona, in mezzo alle altre app, non è semplicemente un fattore grafico. Accedere ad un sito web significa dover digitare un indirizzo nella barra dell'URL e farlo con

la tastiera virtuale del telefono può risultare molto scomoda e macchinoso, mentre raggiungere il servizio con un click (o meglio, un tap) invoglia molto di più l'utente!

Se quindi le PWA non sono altro che siti web che vanno a confondersi tra le applicazioni installate nel dispositivo, perchè puntare direttamente ad un'applicazione standard? Le differenze in realtà sono parecchie e significative tra PWA e applicazione nativa, alcune tra quelle evidenziate in un articolo pubblicato su *mentormate.com* [9]:

- **Funziona anche offline.** Tramite un meccanismo chiamato *service worker* (che verrà approfondito più avanti) le PWA hanno la possibilità di fornire contenuti anche se il dispositivo non è al momento connesso alla rete, grazie ad una intelligente gestione della cache.
- **Codice sorgente unico da mantenere.** Purtroppo esistono diversi sistemi operativi in dotazione per i vari dispositivi mobile, con architetture completamente diverse ed incompatibili. Oltre ad i più famosi Android ed iOS ne esistono molti altri, nativi di alcuni brand di smartphone, che rendono ancora più complicata la programmazione di applicazioni 100% native. Grazie alle PWA è possibile scrivere un unico sorgente unico che verrà poi interpretato dal browser, permettendo quindi di funzionare nella stragrande maggioranza delle situazioni senza nessun problema. Compatibilità a parte, ma anche questo discorso verrà approfondito più avanti.
- **Si comporta come un'app, ma è costruita con tecnologie web.** Non è necessario utilizzare particolari linguaggi di programmazione diversi da quelli già utilizzati nell'ambito web. Sarà quindi possibile costruire layout in HTML ed animarlo con i più moderni SASS e Typescript.

Un elemento in comune tra PWA ed applicazioni native è il file manifest. In generale un file di questo tipo, presente per tutte le applicazioni in tutti i sistemi operativi, serve a dichiarare in anticipo alcune caratteristiche dell'app, fondamentali per il funzionamento della stessa. Solitamente i manifest vengono scritti in JSON o XML.

Sebbene le applicazioni native Android abbiano un file manifest molto più ampio, per una PWA può bastare questo esempio:

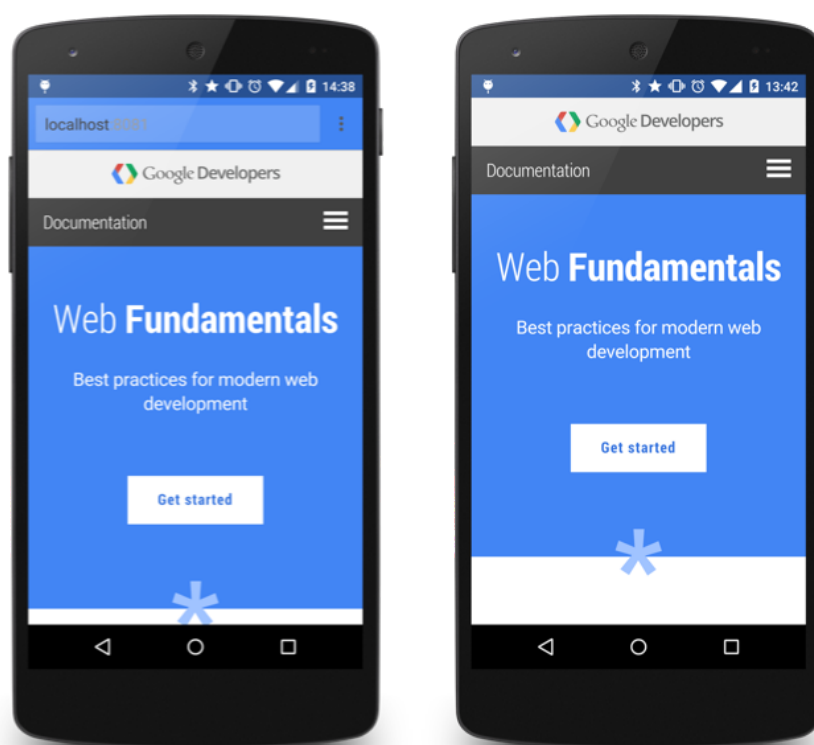
```
1 {
2   "short_name": "AirHorner",
3   "name": "Kinlan's AirHorner of Infamy",
4   "icons": [
5     {
6       "src": "launcher-icon-1x.png",
7       "type": "image/png",
8       "sizes": "48x48"
9     },
10    {
11      "src": "launcher-icon-2x.png",
12      "type": "image/png",
13      "sizes": "96x96"
14    },
15    {
16      "src": "launcher-icon-4x.png",
17      "type": "image/png",
18      "sizes": "192x192"
19    }
20  ],
21  "start_url": "index.html?launcher=true"
22 }
```

Codice 2.1: Esempio di file manifest

Molto semplicemente questo file andrà ad istruire il sistema operativo per quanto riguarda:

- `short_name`, il nome da mostrare nella home screen.
- `name`, il nome per esteso, mostrato ad esempio in liste o comunque viste che garantiscono più spazio.
- `icons`, le icone di varie dimensioni da mostrare a seconda del contesto, per lo stesso principio del nome e nome in breve.

- `start_url`, l'indicazione della pseudo pagina html da lanciare per prima dopo il caricamento dell'applicazione. Questo parametro è l'equivalente dell'indicare in un manifest Android o iOS la classe principale da cui far partire l'applicazione.
- `display`, la modalità di presentazione dell'applicazione all'interno del browser. Due dei parametri più comuni sono `browser` e `standalone` come mostrato in Figura 2.3. Altre due possibili opzioni sono `fullscreen` che permette di utilizzare il 100% dello schermo, rimuovendo ogni altro elemento non inerente all'app, e `minimal-ui` per lasciare all'utente la possibilità di visualizzare la barra dell'URL nel caso volesse copiarlo per salvarlo in memoria o inviarlo ad un altro utente.



"display": "browser"

"display": "standalone"

Figura 2.3: Esempio grafico della differenza del parametro `display`, tra `browser` che mostra la barra dell'URL e `standalone` che invece nasconde ogni riferimento al browser. [developers.google.com]

Logicamente è possibile estendere il file manifest con numerosi altri parametri, tra i più importanti c'è `orientation` per impostare un'orientamento di default, se orizzontale con la costante `landscape` o verticale specificando `portrait`.

Limitazioni

Sebbene le Progressive Web App siano ad oggi, metà 2017, una scelta molto valida e performante non sono ancora la soluzione definitiva, fermo restando un enorme potenziale destinato a palesarsi molto presto.

La principale azienda promotrice delle PWA è Google, questo significa che i prodotti Google quali Android e Chrome vengono ottimizzati di volta in volta per fornire costantemente un supporto ottimale alle PWA. Lo stesso purtroppo non si può dire per quanto riguarda Apple.

Nonostante Apple sia da sempre stata una forte promotrice dello sviluppo web, spingendo molto nel supporto dell'HTML5, è ancora relativamente indietro nel mondo delle Progressive Web App [3].

Nell'ottobre 2016, un anno fa, in un articolo pubblicato su cloudfour.com il fondatore analizza la scelta di alcune aziende di non accogliere ancora le PWA poichè non interamente supportate dai device iOS, giudicandola errata.

Le tre principali mancanze in ambiente iOS sottolineate nell'articolo sono le seguenti [11]:

- **service worker**, processi Javascript lanciati in background in grado di fornire supporto alle PWA e funzioni utili come la navigazione offline, tramite cache.
- **notifiche push**, le classiche notifiche che vengono visualizzate nella parte superiore dello schermo.
- **icona nella home** del proprio dispositivo, aggiungibile da un prompt mostrato direttamente durante la prima navigazione sul browser.

In effetti, se per la propria applicazione si può rinunciare a queste funzionalità, anche se in generale sono parte del core di una PWA, la maggior parte

delle altre features offerte è implementata, anche se a volte non in maniera ottimale.

Data comunque la rapida diffusione delle PWA, ad inizio agosto di quest'anno Apple ha lasciato intendere l'inizio dei lavori di implementazione dei service workers. Alcuni indizi sono la creazione di un bug report¹ con relativa patch in argomento service worker. Anche un dipendente Apple ha confermato via twitter l'apertura dei lavori (Figura 2.4). Pochi giorni dopo lo status della relativa feature è passato da *Under Consideration* a *In Development* nel sito ufficiale dello status delle feature Webkit (motore utilizzato dal browser Safari)².



Figura 2.4: Tweet di risposta da **Jonathan Davis**, *Web Technologies Evangelist* di Apple. [twitter.com]

La notizia è fresca di settimane quindi, ma era da molti solamente questione di tempo in quanto Apple non può di certo stare a guardare mentre Android prendersi la fetta più grossa della torta del mercato in fatto di Progressive Web App.

¹<https://bugs.webkit.org/attachment.cgi?id=317095&action=diff>

²<https://webkit.org/status/#specification-service-workers>

Progressive

Dopo aver evidenziato le limitazioni, è doveroso ricordare il significato della parola **Progressive**. Nella definizione delle PWA, Progressive sta per *Progressive Enhancement* [12], filosofia ben diversa dalla *Graceful Degradation* come si evince da un'analisi fatta quando ancora le PWA non erano nemmeno mai state nominate, ma ancora valida. In parole povere, anche se in ambiente iOS alcune funzioni vengono a mancare, così come in browser obsoleti, il fatto non è da considerarsi come un problema di una gravità eccessiva.

Secondi i paradigmi del Progressive Enhancement l'importante è non tanto la compatibilità ma il contenuto. Bisogna mirare a fornire ad un più vasto pubblico possibile tutto il contenuto di quanto vogliamo comunicare, e poi a seconda delle possibilità dell'utente, disporre e visualizzare il contenuto in maniera sempre più ottimale ed apprezzabile.

Anche un articolo di Chris Mills [17] pubblicato direttamente nella wiki del consorzio W3 pubblicato nel 2011 mostra un chiaro esempio di efficacia della filosofia del Progressive Enhancement, usando la contrapposizione alla Graceful Degradation per rendere il tutto molto più chiaro.

```
1 <p id="printthis">
2   <a href="javascript:window.print()">Print this page</a>
3 </p>
4 <noscript>
5   <p class="scriptwarning">
6     Printing the page requires JavaScript to be enabled.
7     Please turn it on in your browser.
8   </p>
9 </noscript>
```

Codice 2.2: In questo caso la gestione della mancanza di funzionalità del browser va a ricadere sull'utente al quale viene chiesto di abilitare autonomamente Javascript sempre che sia presente nel sistema.

```
1 <p id="printthis">Thank you for your order. Please print this
   page for your records.</p>
2 <script type="text/javascript">
3   (function() {
4     if(document.getElementById) {
```

```
5   var pt = document.getElementById('printthis');
6   if(pt && typeof window.print === 'function'){
7       var but = document.createElement('input');
8       but.setAttribute('type', 'button');
9       but.setAttribute('value', 'Print this now');
10      but.onclick = function(){
11          window.print();
12      };
13      pt.appendChild(but);
14  }
15  }
16  }) ();
17 </script>
```

Codice 2.3: In questo secondo caso non viene chiesto nulla all'utente semplicemente se è possibile stampare la pagina via Javascript viene visualizzato un bottone per farlo altrimenti resta all'utente decidere come stampare la pagina se lo desidera.

Infine, per concludere il discorso *service worker*, impostare la propria PWA per farne uso oggi, significa principalmente due cose:

1. Tutti gli utenti che utilizzano un browser che supporta tale tecnologia la potranno da subito utilizzare e trarne beneficio.
2. Quasi sicuramente quando verrà offerto il supporto anche in ambiente iOS, il prodotto sarà già funzionante a costo zero.

Senza dubbio un investimento che vale la pena fare.

2.4 Applicazioni Ibride

Un altro punto di incontro tra siti web costruiti secondo la filosofia Mobile First, Progressive Web App ed applicazioni interamente native costruite ad hoc per un unico dispositivo, si collocano le *Applicazioni Ibride*.

Da una pagina della guida ufficiale di Apache Cordova [4], uno dei principali framework per la costruzione di app ibride, vengono suggeriti tre scenari nelle quali scegliere questo tipo di approccio.

- Programmatore Mobile che vuole estendere la propria applicazione su più piattaforme, senza doverla re-implementare da capo per ogni linguaggio della specifica piattaforma, e tool set.
- Sviluppatore web che vuole diffondere la sua web app già confezionata per la distribuzione nei vari app store.
- Sviluppatore mobile interessato nel combinare applicazioni native con componenti web in una WebView, con accessi a livello di API del dispositivo.

Un articolo pubblicato su Telerik introduce molto chiaramente al concetto di Ibrido [6].

Una delle principali differenze rispetto alle applicazioni PWA è lo strumento tecnologico utilizzato per la visualizzazione dell'applicazione. Se nelle PWA veniva utilizzato un Web Browser mascherato da applicazione, nelle applicazioni ibride viene utilizzato un componente chiamato *WebView*.

Per *WebView* si intende un vero e proprio componente nativo del sistema, che permette di visualizzare contenuti in formato web, quindi in generale HTML, CSS e Javascript, ed allo stesso tempo sfruttare in maniera nativa i componenti del dispositivo.

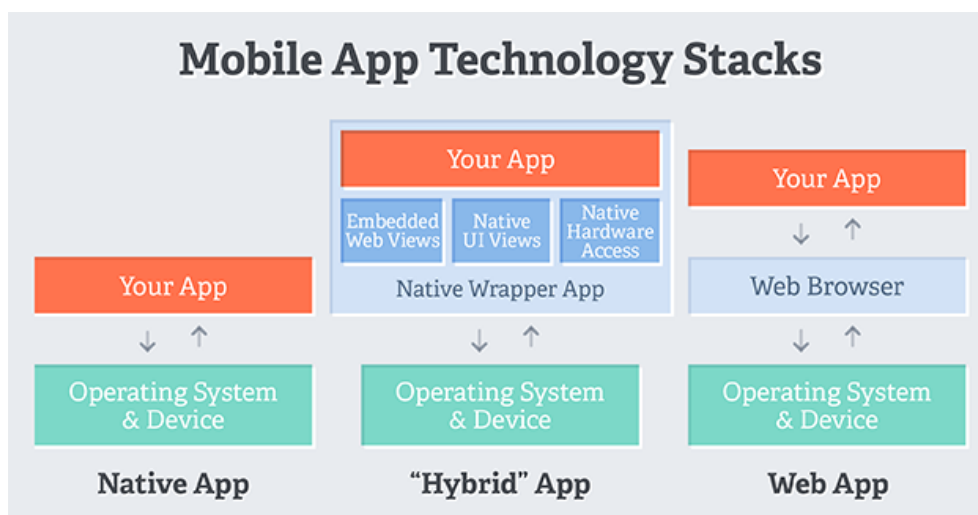


Figura 2.5: Rappresentazione grafica della differenza tra Applicazione Nativa Applicazione Ibrida e Progressive Web App. [myshadesofgray.wordpress.com]

Per sfruttare al meglio alcune specifiche funzionalità offerte dai dispositivi, bypassando i problemi nati dal diverso sistema operativo in uso, i principali framework per la programmazione di app ibride mettono a disposizione numerosi plugin richiamabili ed utilizzabili sempre tramite linguaggi web, contenenti al loro interno linguaggio nativo per le piattaforme per cui l'applicazione verrà compilata.

Infondo, così come le PWA, si cerca il più possibile di seguire il paradigma su cui la *Sun Microsystems* ha costruito Java, “*Write once, run everywhere*”.

In qualità di vere e proprie applicazioni, anche le app ibride hanno la necessità di istruire il sistema operativo su alcuni parametri fondamentali prima e durante l'avvio dell'applicazione. In parole povere, un manifest.

La scelta più utilizzata è quella di utilizzare un file chiamato `config.xml` appunto scritto in XML. La particolarità in questo caso è che il file utilizzato durante la scrittura dell'applicazione è unico, ma viene poi riadattato e ricompilato insieme al resto del codice a seconda del sistema operativo per cui si sta compilando, in modo da renderlo comprensibile e consistente con l'ambiente in cui verrà poi installata l'applicazione.

Agli occhi dell'utilizzatore finale, un'app ibrida è una vera e propria applicazione da installare sul proprio dispositivo e reperibile negli store insieme alle altre applicazioni, infatti dopo la compilazione del codice per le varie piattaforme, viene generato il relativo eseguibile che sarà poi lanciato per installare l'applicazione.

L'autore dell'articolo di Telerik citato in precedenza porta l'esempio di due vini di alta qualità paragonati ad un'applicazione ibrida ed una nativa ben fatte. Solamente un sommelier, vero intenditore, sarà in grado di notare le differenze. Per l'utente finale che nella maggior parte degli ambiti non sarà quasi sicuramente un esperto di programmazione, in contesti dove le performance sono importanti ma non indispensabili, entrambe le app risulteranno la stessa cosa.

Un punto a sfavore infatti delle applicazioni ibride contro le applicazioni native è il fattore performance. Così come alcuni programmi desktop dalle elevate richieste di performance possono venir ottimizzate scrivendo porzioni di codice in linguaggi più vicini al linguaggio macchina come *assembly*, le

applicazioni native programmate appositamente con i linguaggi e le strutture migliori per l'ambiente specifico avranno dei vantaggi rispetto ad applicazioni scritte in un linguaggio diverso ed adattate all'occasione.

Apache Cordova

Come citato in precedenza, *Apache Cordova* è uno dei principali framework per la costruzione di Applicazioni Ibride. Il progetto nasce dall'acquisizione da parte di **Adobe** di un progetto di successo avviato da un'azienda canadese, *Nitobi Software*, chiamato **PhoneGap**.

Successivamente, come spiega Brian LeRoux [14], impiegato a PhoneGap, il codice sorgente è stato donato alla fondazione *Apache Software Foundation* per mantenere al meglio un sorgente pulito, ben documentato e trasparente, permettendo anche ad altre grandi organizzazioni di contribuire.

Per motivi legali è stato necessario dare ai due progetti due nomi diversi, in modo da poter distinguere *Apache Cordova* come la sorgente, il motore su cui *PhoneGap* ed in futuro altri framework verranno poi costruiti e mantenuti.

Apache Cordova attualmente supporta i seguenti sistemi operativi: *Android*, *iOS*, *Blackberry*, *Bada*, *Tizen* e *Windows Phone*, anche se per utilizzare funzioni più avanzate e specifiche dei sistemi, alcuni rimarranno incompatibili.

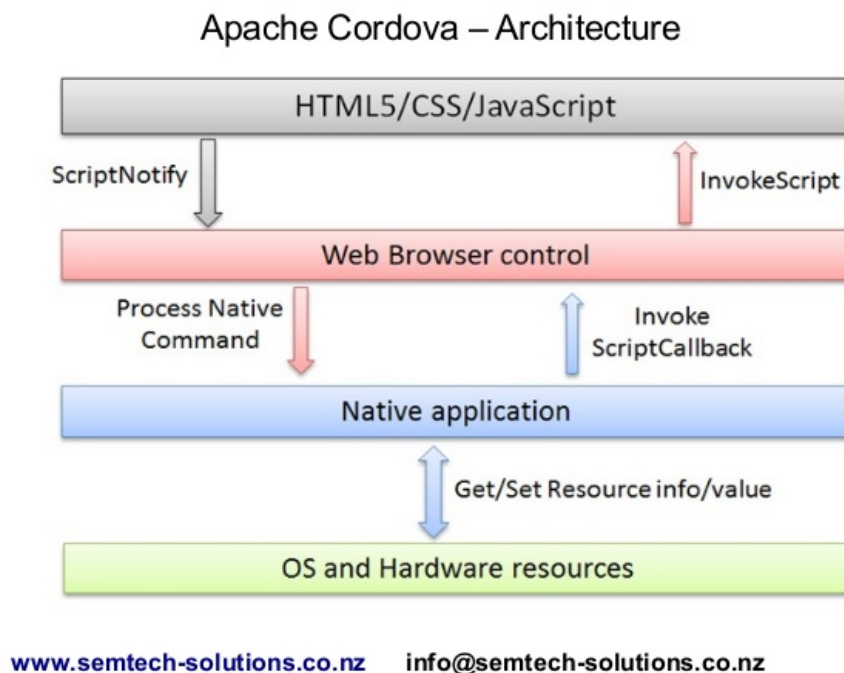


Figura 2.6: Architettura del funzionamento di Apache Cordova. [semtech-solutions.co.nz]

Un particolare da non trascurare nella progettazione e nel design di applicazioni ibride viene fatto notare da Andrew Trice [21], Technical Evangelist di Adobe: Apple può negare la pubblicazione di determinate applicazioni nello store ufficiale, se non rispettano alcuni requisiti.

- La user experience deve dare la sensazione di un'app.
- Il design e la struttura devono corrispondere agli standard delle app nell'ecosistema iOS.
- Mostrare delle chiare differenze con un'esperienza da mobile web app

L'articolo è stato scritto a seguito di alcune accuse da parte di sviluppatori di app con PhoneGap dopo aver ricevuto appunto la negazione da parte di Apple per la pubblicazione delle loro app.

Da Adobe spiegano però come esistano molte altre applicazioni costruite con layout HTML o comunque web based, ma vengono comunque accettate

grazie alla loro costruzione pensata per un'applicazione piuttosto che per un semplice sito web installato nel dispositivo, invitando quindi all'attenzione a questo fondamentale dettaglio quando si sviluppa un'applicazione ibrida.

Un semplice esempio che differenzia un'applicazione da un sito web è la barra di navigazione nella parte superiore dello schermo. Le applicazioni ne hanno quasi sempre una, con sul lato sinistro il pulsante per tornare indietro ed il titolo della schermata centrato nella barra. In un sito web ci sono molti link, in un'applicazione la navigazione tra le schermate dovrà essere gestita da bottoni, non da testi cliccabili.

Capitolo 3

Tecnologie coinvolte: Ionic

3.1 Framework

Uno dei più diffusi ed attualmente mantenuti framework costruiti sfruttando le possibilità di Apache Cordova è **Ionic Framework** (anche conosciuto semplicemente come *Ionic*).

Per essere più precisi, più che un "semplice" framework, sarebbe più corretto definirlo *SDK* (Software Development Kit) in quanto Ionic non fornisce solamente delle strutture a supporto dello sviluppo dell'applicazione, ma aiuta a gestire l'intero processo a partire dalla creazione del progetto, passando per il framework per la gestione frontend fino al momento della compilazione finale e la creazione dell'eseguibile pronto per il deploy sul dispositivo o negli store.

Il lancio ufficiale di Ionic risale al 2013 con una versione che verrà poi completamente ristrutturata e modernizzata con le tecnologie più recenti annunciata nel 2016, uscita poi a gennaio 2017. Dall'uscita della seconda versione, quando si parla di Ionic si intende *Ionic 2*, mentre la precedente è stata rinominata a *Ionic v1.0*.

Per lo svolgimento del progetto è stato scelto di analizzare ed utilizzare Ionic come oggetto della ricerca. La scelta è stata principalmente condizionata dal fatto che Ionic è attualmente uno dei framework più utilizzati e diffusi sul web, volendo quindi approfondire l'ambiente delle applicazioni ibride,

la scelta fatta confida in una semplice reperibilità di materiale come guide e documentazione, così come esempi e supporto nel *troubleshooting*.

3.2 Ionic v1.0

Un co-fondatore di Ionic, *Adam Bradley*, spiega in un post [5] datato 2013 nel blog ufficiale del framework le filosofie dietro la nascita di Ionic. Il post viene aperto riferendosi ad Ionic con la domanda "*Dove può trovare posto nel processo? Come mi porterà beneficio?*" a cui risponde molto chiaramente:

“Ionic’s ultimate goal is to make it easier to develop native mobile apps with HTML5, also known as Hybrid apps”

Lo scopo finale di Ionic è rendere più semplice lo sviluppo di applicazioni native in HTML5, anche conosciute come app Ibride.

La parola chiave in questa frase è **HTML5**. Framework come PhoneGap e Cordova già esistevano nel 2013 ma i creatori di Ionic hanno sentito il bisogno di sviluppare il loro framework, tra l’altro costruito sulle fondamenta di Apache Cordova, in grado di sfruttare al meglio le ultime tecnologie offerte dallo stato dell’arte per rendere il prodotto finale sempre migliore, in termini sia di performance ma anche di features e possibilità fornite agli sviluppatori.

Viene anche fatto un paragone con il framework web che all’epoca si chiamava *Twitter Bootstrap*, ora conosciuto solo come *Bootstrap*, in quanto entrambi puntano alla diffusione delle conoscenze e dell’open-source, rendendo tutto il codice pulito e documentato in modo da fornire un doppio servizio alla comunità

- Per sviluppatori dalle conoscenze avanzate è possibile fare "proprie" le conoscenze condivise e documentate per poter contribuire alla crescita progetto fornendo alla comunità un prodotto di qualità sempre migliore.
- Per chi invece è meno avanzato, o semplicemente non interessato al progetto, è possibile utilizzare il prodotto che mette a disposizione una serie di pattern e *best practice* per la creazione di un prodotto di qualità, senza dover partire da zero e reinventare la ruota.

Oltre a permettere di creare le interfacce grafiche con le più recenti funzioni include nell'HTML5, logicamente decorato ed animato con l'uso del CSS, Ionic è in grado di fornire supporto dinamico ed interattivo alle pagine (che saranno schermate di un'applicazione) grazie ad *AngularJS*. AngularJS è un vero e proprio framework scritto in *Javascript* che dona al vocabolario dell'HTML5 un vasta gamma di tag e possibilità di rendere la schermata dinamica, animata ed interattiva.

Per mostrare le potenzialità aggiunte a Javascript da AngularJS, basta questo semplice esempio che viene proposto direttamente nella homepage di AngularJS:

```
1 <!doctype html>
2 <html ng-app>
3   <head>
4     <script src="https://ajax.googleapis.com/ajax/libs/
      angularjs/1.6.5/angular.min.js"></script>
5   </head>
6   <body>
7     <div>
8       <label>Name:</label>
9       <input type="text" ng-model="yourName" placeholder="
      Enter a name here">
10    <hr>
11    <h1>Hello {{yourName}}!</h1>
12  </div>
13 </body>
14 </html>
```

Codice 3.1: Codice dimostrativo di AngularJS

Già dalla **seconda riga** risalta l'attributo `ng-app` all'interno del tag `html`, che dichiara subito la presenza di AngularJS all'interno della pagina.

Dalla **riga 9** si possono apprezzare le prime, fondamentali, differenze. Con l'attributo `ng-model` viene specificato che il contenuto testuale dell'input sarà automaticamente assegnato alla variabile `yourName`. Questo significa che in entrambe le direzioni, modificando il contenuto dell'input la variabile cambierà valore, e viceversa modificando il valore della variabile ad esempio in una funzione all'interno del codice aggiornerà automaticamente il contenuto dell'input.

Lo stesso discorso vale per la **riga 11**, dove l'identificatore formato dalle doppie graffe con all'interno il nome della variabile, permette di visualizzare il contenuto della variabile costantemente aggiornata.

Il risultato finale è che scrivendo del testo all'interno dell'input, verrà automaticamente aggiornato nell'header `h1` senza aver scritto nemmeno una riga di codice `javascript` che altrimenti avrebbe richiesto:

- Assegnare un ID o comunque un chiaro riferimento per rintracciare l'input e l'header su cui visualizzare il testo
- Mettere l'input in ascolto sull'evento `onchange` ed assegnargli una funzione che:
 - Prende il valore dell'input (ed opzionalmente lo assegna alla variabile, se necessario per altre funzioni).
 - Modifica il testo all'interno dell'header, facendo attenzione nel gestire la parte fissa (*Hello* prima della variabile, seguito dal punto esclamativo immediatamente dopo) inserendo il contenuto della variabile con una concatenazione di stringhe.

Semplicemente non ci sono paragoni.

La principale motivazione che ha spinto i creatori di Ionic nel fare forte affidamento su AngularJS è per quanto appena dimostrato, avere un framework così potente all'interno del progetto dovrà servire poi per rendere il contenuto programmato con tecnologie web più simile e familiare possibile a quello che poi dovrà venir compilato in un prodotto pensato per un ambiente diverso dal web navigabile via browser, con l'aggiunta e la gestione della navigazione direttamente interna all'app con menu, tab ed altre strutture tipiche delle applicazioni native e non delle applicazioni web.

3.3 Ionic 2

Il 25 gennaio 2017 viene pubblicata la prima versione stabile di **Ionic v2.0.0** [15], che come già anticipato si chiamerà semplicemente *Ionic* e non ci sarà più nel nome il numero della versione.

Con l'introduzione di questa nuova versione sono stati effettuati molti cambiamenti radicali e non retrocompatibili, per questo la versione precedente verrà chiamata *Ionic v1.0*.

Il cambiamento più importante è sicuramente il passaggio da *Javascript* a *Typescript* per quanto riguarda il frontend dell'applicazione.



Figura 3.1: Prompt in stile *Pokémon* per avvisare di un'imminente evoluzione. Il passaggio da Javascript a Typescript. [blog.ionic.io]

Visto il perfetto tempismo con l'uscita della nuova versione di AngularJS, completamente ristrutturata e rinominata in *Angular*, i due team di sviluppo hanno stretto la loro collaborazione essendo Ionic uno dei principali progetti *Open Source* basato fortemente su AngularJS è stato deciso di adottare la nuova versione, **Angular 2**.

Novità

Con l'aggiornamento alla nuova e stabile versione di Ionic, sono stati introdotti numerosi nuovi componenti al già esteso set presente nella versione precedente, oltre a nuovi plugin costruiti con le ultime tecnologie supportate ed ora utilizzabili all'interno del framework

- Numerosi **componenti** sono stati aggiunti alla parte del framework dedicato alla User Interface. Tra questi spiccano i *FAB (Floating Action Button)* introdotti con il *Material Design*¹.
- Una nuova serie di **plugin** è stata implementata sotto il nome di *Ionic Native*. Questi plugin si occupano di gestire le funzionalità native nei

¹<https://material.io/guidelines/components/buttons-floating-action-button.html>

vari dispositivi. Principalmente l'utilizzo di componenti hardware o di interazione tra sistemi interni al cellulare (come ad esempio postare una foto su Facebook tramite la relativa app).

- Nuovi **temi** gestiti e modificabili tramite SASS, in modo da rispettare gli standard dei vari ambienti (come discusso in precedenza, iOS ad esempio ha norme molto restrittive) e poter comunque aggiungere a cascata eventuali modifiche custom per l'applicazione specifica. Questa pratica viene chiamata dai creatori di Ionic *Platform Continuity*. I tre temi principali che Ionic supporta sono infatti:

- **Material Design**, utilizzato principalmente in ambiente Android.
- **iOS**, per i dispositivi Apple, che richiedono standard molto particolari.
- **Windows**, che adotta un particolare stile molto regolare, con angoli retti per formare quadrati e rettangoli, come si può notare facilmente nel tema principale di Windows 10.

- Una **documentazione** ancora migliorata ed aggiornata per tenere traccia di tutte le ultime tecnologie in uso.
- Le **performance** in generale sono state migliorate. Come già detto le app ibride in alcuni casi possono avere problemi di performance, uno di questi è lo scorrimento di lunghe liste. In questo scenario, implementando con codice nativo, sia Android che iOS gestiscono il caricamento degli oggetti delle liste solamente su richiesta quando è necessario visualizzarli. Questo purtroppo non è possibile farlo con *Javascript*.

Per ovviare a questo problema, con l'ausilio di *Typescript*, Ionic utilizza un metodo chiamato *Virtual Scroll*². Così come nelle soluzioni native, il *Virtual Scoll* gestisce un'ipotetica lista con un numero di elementi indefinito, renderzzati solamente quando necessario.

Un'ulteriore estensione di questo meccanismo viene fornita dal tag `<ion-img>` che secondo le stesse filosofie carica l'immagine solamente se la vista si ferma su di essa. Specialmente se l'immagine proviene da una richiesta HTTP, tale accorgimento può incrementare notevolmente le performance.

²<http://ionicframework.com/docs/api/components/virtual-scroll/VirtualScroll>

- **Nuovi strumenti** a supporto dello sviluppo sono stati aggiunti a questa versione. Tra i quali:
 - **Gestione degli errori.** Ora gli errori runtime vengono mostrati direttamente nell'applicazione sotto forma di una schermata che appare sopra quella corrente, con un bottone per chiuderla e continuare l'esecuzione se l'errore lo permette.

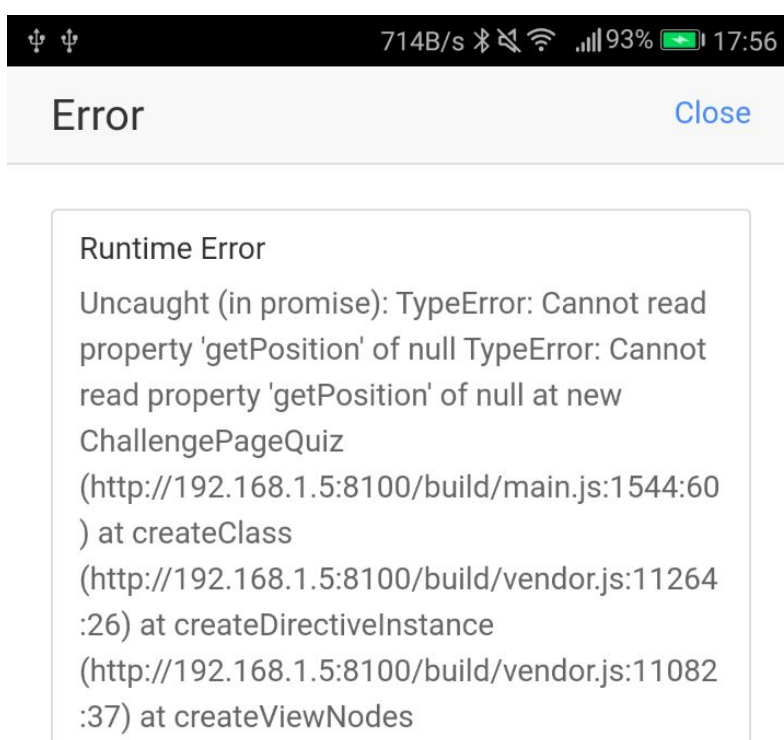


Figura 3.2: Esempio di errore runtime visualizzato direttamente sull'app, evitando di dover controllare la console.

- **Ionic Serve Lab.** Questo strumento permette di lanciare sul proprio browser tre diverse istanze dell'applicazione con i temi di default dei tre principali sistemi operativi supportati, *Android*, *iOS* e *Windows*, permettendo allo sviluppatore di confrontare contemporaneamente la resa e l'interazione della propria app nei tre differenti ambienti, senza dover avere a disposizione tre diversi dispositivi.

Ionic 3

Dopo soli 4 mesi dall'uscita della versione 2, viene annunciata nel blog ufficiale la versione 3.0.0 di Ionic [7].

Premessa importante, da questa versione in poi verrà utilizzato il *Versionamento Semantico 2.0.0*³.

L'incremento delle versioni, nel formato MAJOR.MINOR.PATCH segue questi criteri:

1. Versione MAJOR quando vengono fatte delle modifiche che rendono l'ultima versione **incompatibile** con la precedente.
2. Versione MINOR quando vengono aggiunte nuove significative funzionalità, ma **compatibili** con la versione precedente.
3. Versione PATCH quando l'aggiornamento è **compatibile** con la versione precedente e contiene principalmente risoluzione di bug o piccole modifiche.

Una curiosità riguardo il *SemVer* è che tra le regole per rispettare lo standard non sono concessi zeri alla sinistra dei numeri, fatta eccezione per le versioni in *pre-release* (non ufficialmente in produzione) che possono e devono iniziare con 0.Y.Z. Questo significa che ad esempio a seguito della versione 3.9.5 ci sarà la versione 3.10.0.

La modifica è stata necessaria in quanto essendosi il progetto molto ingrandito ed utilizzato da migliaia di sviluppatori, era necessario fare chiarezza su ogni tipo di aggiornamento in futuro.

In generale la maggior parte dei sistemi di versionamento si affida a criteri *simili* a questi, ma avere sigle e abbreviazioni solamente simili significa non necessariamente *consistenti*, e quindi non del tutto sicure a fraintendimenti che a livello di produzione possono costare cari. Ad esempio non identificare chiaramente una situazione di incompatibilità con la versione precedente potrebbe improvvisamente compromettere il funzionamento di un sistema, causando anche gravi danni se non gestito immediatamente.

³<http://semver.org/>

Un altro motivo che ha portato all'adozione di *SemVer* è l'aggiornamento ad *Angular 4*, dopo che anch'esso ha adottato lo stesso sistema di versionamento, a seguito di una grossa release che ha portato appunto alla versione 4.0.0.

Fondamentale l'introduzione del *Lazy Loading* [13] all'interno dell'ambiente Ionic. Questa tecnica consiste nel gestire nell'inizializzare gli oggetti solamente quando necessario. Un'applicazione web può richiedere numerose risorse ottenute da altrettante numerose fonti, come immagini, script e fogli di stile. Il *Lazy Loading* consiste nel reperire appunto tali risorse non prima del loro utilizzo, permettendo quindi di diminuire notevolmente i tempi di startup dell'applicazione, andando a distribuire progressivamente il carico durante la navigazione.

Supporto RTL

In questa release di Ionic è stato aggiunto il supporto ai linguaggi *RTL* [18]. La sigla sta per *Right To Left* che a differenza del solito *Left To Right* serve a supportare appunto le scritture come ad esempio l'arabo che iniziano da destra verso sinistra all'interno della pagina.

Importante specificare che il supporto RTL non è stato introdotto direttamente dal team Ionic ma da uno sviluppatore come tanti altri che usa il framework ed ha deciso di contribuire al progetto open source, ed anziché spendere tempo per aggiungere questo supporto unicamente ad i suoi progetti, ha condiviso con tutta la community il sorgente per farlo introdurre nativamente all'interno di Ionic. Questo ha dato il via per numerosi altri utenti nel partecipare a questo progetto, essendo il team di sviluppo di Ionic formato da persone che usano linguaggi LTR, sarebbe stato più difficile per loro pensare e sviluppare un ambiente inverso a quello solito.

Circa ad ogni MINOR release di Ionic (quindi 3.x.x secondo *SemVer*) sono state fatte delle modifiche e delle aggiunte al framework per supportare entrambe le direzioni. Una modifica necessaria è stata la rimozione delle parole chiave `left` e `right` in situazioni come l'allineamento degli elementi, i margini i bordi ed altre strutture gestite dal CSS. Questo perchè nel momento in cui si cambia lingua ad un'applicazione spostando da sinistra a destra

l'inizio della scrittura, quello che prima iniziava da `left` dovrebbe iniziare da `right` e questo non può funzionare automaticamente, in quanto non sarebbe più consistente. Per questo sono stati introdotti `start` ed `end`. In questo caso ha sempre senso poichè quando si seleziona una lingua LTR lo `start` sarà normalmente a sinistra, e nulla è cambiato, mentre nel momento in cui si seleziona una lingua RTL lo `start` si troverà nella parte destra dello schermo e l'`end` nella parte sinistra.

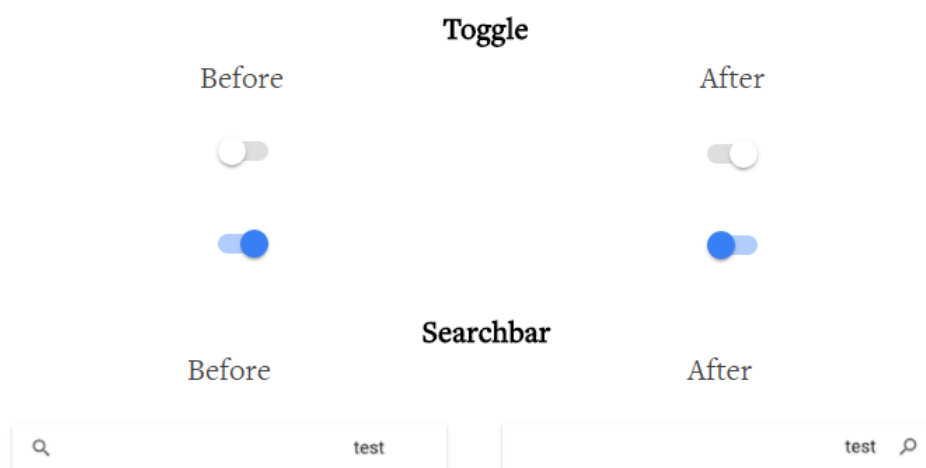


Figura 3.3: Due esempi di come la User Interface debba adattarsi al sistema RTL, a destra nell'immagine. [blog.ionic.com]

Nonostante questa nuova funzionalità possa risultare molto utile, se non indispensabile, in determinati contesti, nel semplice sviluppo di applicazioni che supportano linguaggi LTR unicamente non è strettamente necessario, per questo di default il supporto sarà disabilitato nelle applicazioni Ionic. Basta pensare che ad esempio *Apple* ha iniziato a supportare i linguaggi RTL solamente dalla versione 9 di iOS (Settembre 2015).

In ambiente Ionic è necessario impostare una variabile globale *Sass*, `$app-direction`, che di default è impostata su `ltr`. I valori consentiti sono appunto `ltr`, `rtl` e `multi` per supportare entrambi i versi di scrittura.

3.4 Tecnologie di sviluppo

Ad oggi, settembre 2017, Ionic è alla versione 3.6.0. Il progetto è ancora molto seguito e costantemente sviluppato, escono infatti nuove MINOR release circa ogni mese e PATCH all'occorrenza, senza una precisa cadenza.

Per funzionare, Ionic si basa su diverse tecnologie.

Node.js

Una breve ma completa descrizione di quel che può fare *Node.js* viene direttamente fornita nella homepage del sito ufficiale:

“Node.js® è un runtime Javascript costruito sul motore JavaScript V8 di Chrome. Node.js usa un modello I/O non bloccante e ad eventi, che lo rende un framework leggero ed efficiente. L'ecosistema dei pacchetti di Node.js, npm, è il più grande ecosistema di librerie open source al mondo.”

In parole povere, un framework pensato per lo sviluppo del lato backend e progettato in *JavaScript*. Grazie alle sue principali caratteristiche, viene ampiamente utilizzato in Ionic, sia per quanto riguarda la distribuzione interna a Ionic, gestendo pacchetti, aggiornamenti, strutture interne e plugin, sia per quanto riguarda il prodotto finale, cioè le applicazioni scritte dagli utilizzatori di Ionic.

Una differenza sostanziale di Node rispetto ai soliti linguaggi utilizzati per la gestione backend di server come PHP o ASP.NET riguarda la gestione del flusso, che in questo caso sarà *asincrona*. In situazioni di gestione del flusso *sincrona*, le azioni vengono eseguite una dopo l'altra, al terminare della precedente. Node invece, sfruttando i meccanismi di callback offerti da JavaScript, permette di effettuare più chiamate simultanee senza doversi preoccupare della gestione e soprattutto della sincronizzazione dei thread. Logicamente a basso livello viene sfruttato il *multithreading*, ma questo viene lasciato nascosto durante la programmazione in quanto gestito autonomamente dal sistema.

Il meccanismo sfruttato è quello delle *callback*. Quando vengono richieste risorse non presenti, non è necessario avviare manualmente nuovi thread o mettere in pausa quello corrente, semplicemente la chiamata si metterà in attesa di una callback per poi riprendere, mentre il flusso principale può proseguire autonomamente. Ad orchestrare l'esecuzione delle callback ci pensa un *Main Loop*, risiedente in un unico thread, che tiene traccia di uno *stack* di chiamate da gestire. Quando una chiamata viene presa in carico, il thread la esegue e se necessita nuovamente di una risorsa esterna viene effettuata la richiesta per poi passare alla chiamata successiva nello stack. Autonomamente all'arrivo del callback verranno riposizionati i risultati in fondo alla coda.

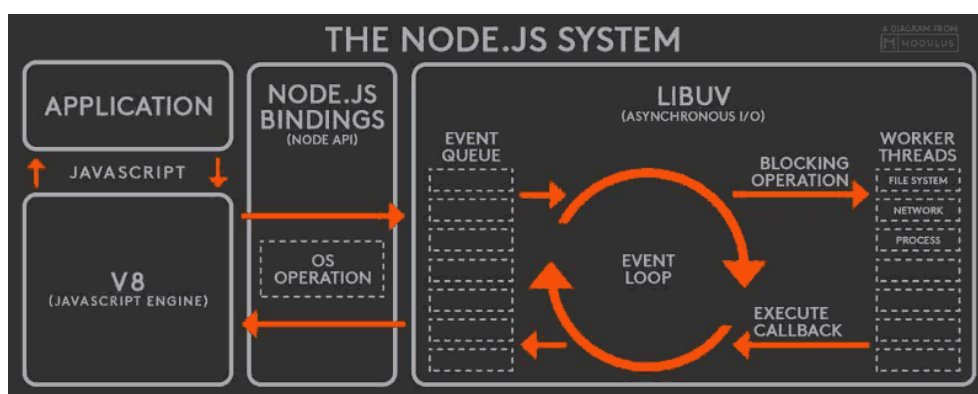


Figura 3.4: Spiegazione grafica di come *Node* è in grado di gestire le chiamate all'interno dell'*event loop*. [stackoverflow.com]

Per quanto riguarda il lato applicativo, a livello di codice significa chiamare una funzione che resterà in attesa del risultato richiesto mentre il thread principale, l'unico, continua il suo processo. Questo meccanismo può essere molto utile e potente, ma bisogna fare particolare attenzione a quello che viene chiamato *Callback Hell* [19].

```

1 step1(function (value1) {
2     step2(value1, function(value2) {
3         step3(value2, function(value3) {
4             step4(value3, function(value4) {
5                 // Do something with value4
6             });
7         });
8     });

```

```
9 })
```

Codice 3.2: Esempio di Callback Hell con quattro funzioni chiamate a cascata

Per risolvere questo inconveniente ci sono diverse *Best Practise*, ma ancora meglio Node supporta l'utilizzo delle *Promise*, in modo da rendere l'esecuzione del codice molto più lineare supportando anche una corretta gestione degli errori.

```
1 Q.fcall(promisedStep1)
2 .then(promisedStep2)
3 .then(promisedStep3)
4 .then(promisedStep4)
5 .then(function (value4) {
6     // Do something with value4
7 })
8 .catch(function (error) {
9     // Handle any error from all above steps
10 })
11 .done();
```

Codice 3.3: Come le *Promise* rendono il codice più lineare.

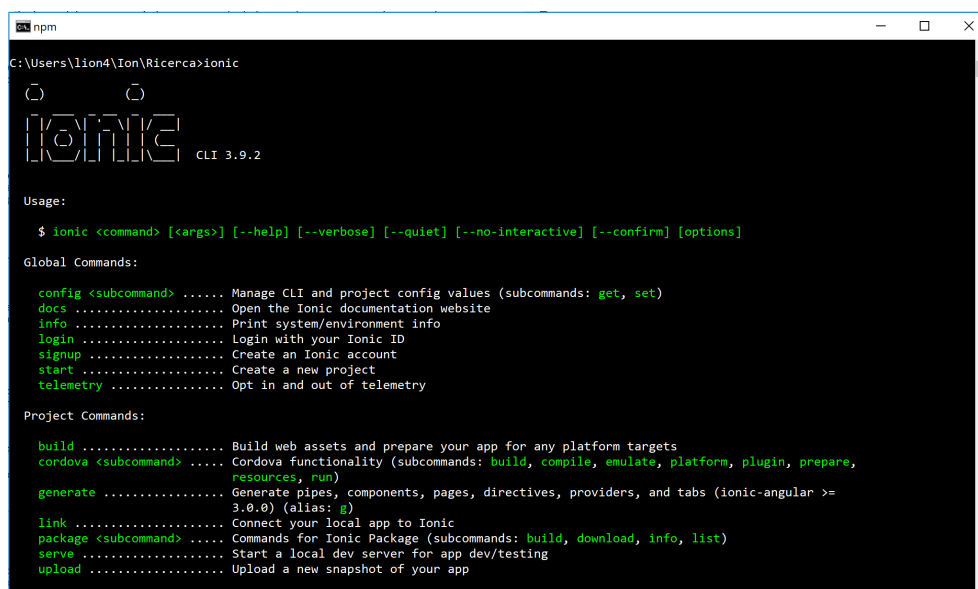
Npm e Ionic CLI

Il download e l'installazione di Ionic, insieme ai numerosi pacchetti e plugin aggiuntivi, viene costruito su npm. npm è il *Package Manager* di Node, ed è un'applicazione di sistema installata insieme a Node. Tramite npm è possibile scaricare l'intero codice sorgente di Ionic, insieme ad Apache Cordova per farlo funzionare correttamente, semplicemente lanciando da terminale il comando `npm install ionic` per un'installazione nella cartella corrente, oppure `npm install -g ionic` per un'installazione globale.

La differenza consiste nel dove npm andrà a generare il file `package.json` contenente una specie di riassunto in formato JSON di tutti i pacchetti installati e le relative dipendenze, e la cartella `node_modules` con appunto i sorgenti di quanto scaricato ed installato.

Insieme all'installazione di Ionic, viene fornito il potente tool nativo per la gestione specifica del singolo progetto, anch'essa usufruibile tramite linea

di comando dalle logiche molto simili a npm, chiamata *Ionic CLI*, dove CLI sta per (*Command Line Interface*).



```
C:\Users\lion4\Ion\Ricerca>ionic
Ionic
CLI 3.9.2

Usage:
  $ ionic <command> [<args>] [--help] [--verbose] [--quiet] [--no-interactive] [--confirm] [options]

Global Commands:
  config <subcommand> ..... Manage CLI and project config values (subcommands: get, set)
  docs ..... Open the Ionic documentation website
  info ..... Print system/environment info
  login ..... Login with your Ionic ID
  signup ..... Create an Ionic account
  start ..... Create a new project
  telemetry ..... Opt in and out of telemetry

Project Commands:
  build ..... Build web assets and prepare your app for any platform targets
  cordova <subcommand> ..... Cordova functionality (subcommands: build, compile, emulate, platform, plugin, prepare,
resources, run)
  generate ..... Generate pipes, components, pages, directives, providers, and tabs (ionic-angular >=
3.0.0) (alias: g)
  link ..... Connect your local app to Ionic
  package <subcommand> ..... Commands for Ionic Package (subcommands: build, download, info, list)
  serve ..... Start a local dev server for app dev/testing
  upload ..... Upload a new snapshot of your app
```

Figura 3.5: Output della console di windows all'esecuzione del comando `ionic`, che mostra il messaggio di default con la presentazione della Ionic CLI, lanciato all'interno della cartella di un progetto.

Come si può vedere in Figura 3.5 la versione è leggermente più avanti rispetto a quella di Ionic stesso, infatti la CLI viene aggiornata molto spesso per renderla uno strumento sempre più utile per lo sviluppatore.

I comandi eseguiti all'interno di Ionic servono ad esempio a creare un nuovo progetto, con `ionic start <type> <name>`, dove inserendo il parametro `<type>` è possibile installare alcuni template di applicazioni interamente funzionanti e ben documentati da utilizzare come punto di partenza per esplorare al meglio le diverse funzionalità offerte.

Come già stato detto, Ionic è un framework costruito sulle solide basi di Apache Cordova. Per mantenere chiara la distinzione tra i due ambienti dalla versione 3.0.0 della CLI i comandi strettamente collegati alla compilazione, gestione di plugin e deploy dell'applicazione vengono chiamati con il prefisso `ionic cordova`.

Angular

La struttura principale su cui viene costruita un'applicazione Ionic è quella dettata dalla versione corrente di *Angular*. In ogni app sarà quindi presente il percorso `src/app` con all'interno i file che serviranno da punto di partenza per l'intera applicazione. In un progetto standard realizzato con le più recenti versioni dei vari tool forniti avrà questi file:

- `app.component.ts` contiene il componente principale dell'applicazione. Il codice HTML relativo a questo componente, che sia incluso in questo file o in un file separato `app.component.html`, sarà quello visualizzato alla base dell'applicazione. In questo file solitamente viene gestito il menu laterale a scomparsa, tipico componente delle applicazioni mobile, che sarà presente in tutte le schermate dell'app.
- `app.module.ts` è la raccolta di tutti i moduli presenti all'interno dell'applicazione con il relativo percorso. Solamente ciò che è specificato in questo file sarà utilizzabile all'interno del progetto.
- `app.scss` è il file di stile in cui è possibile dichiarare regole di stile globali che saranno poi seguite in tutta l'applicazione.
- `main.ts` tramite la funzione `platformBrowserDynamic()` avvia l'applicazione tenendo conto della piattaforma in cui è stata lanciata.

All'avvio dell'applicazione viene lanciata una "comune" pagina `index.html` con all'interno del tag `<head>` solamente alcuni dati fondamentali come il titolo, alcuni meta tag per la dimensione dello schermo e le varie inclusioni del codice generato durante la compilazione. All'interno del `<body>` è contenuto solamente un tag particolare che si apre e si chiude immediatamente, `<ion-app>`. Qui sarà inserito il codice preso da `app.component.ts` e partirà l'applicazione.

Con l'introduzione di *Ionic 3* logicamente non poteva mancare il supporto alla più recente versione disponibile di *AngularJS*. Il percorso seguito da *Angular* e *Ionic* è stato quasi parallelo. Se *Ionic* è passato dalla versione 1 alla versione 2 quasi cambiando nome a seguito delle grosse modifiche apportate al core del sistema, la stessa cosa è successa per *Angular* che dalla prima versione, appunto *AngularJS*, è passato semplicemente ad *Angular* [8].

Con la release della versione 4 anche Angular ha adottato il sistema di versionamento semantico SemVer, saltando dalla versione 2 alla versione 4 poichè il pacchetto `@angular/router` era già alla versione 3.3.0 permettendo di allineare tutti i componenti ad un'unica versione, appunto con il numero 4.

Oltre ad aver alleggerito il peso delle applicazioni [8] ed aver incrementato le prestazioni, sono state effettuate alcune migliorie all'estensione del supporto ai tag HTML5 permettendo veri e propri `if/else` all'interno del codice, insieme all'assegnazione di variabili locali. A seguire un esempio:

```
1 <div *ngIf="userList | async as users; else loading">
2   <user-profile *ngFor="let user of users; count as count;
3     index as i" [user]="user">
4     User {{i}} of {{count}}
5   </user-profile>
6 </div>
7 <ng-template #loading>Loading...</ng-template>
```

Codice 3.4: Esempio di codice HTML5 arricchito dalle potenti funzionalità offerte da *Angular 4*

La breve porzione di codice qui sopra serve a generare una lista di elementi la cui UI è definita in un altro componente, `userprofile`, scaricata in maniera asincrona. Durante il download e finchè i dati non sono pronti per la visualizzazione viene invece mostrata la scritta *Loading...* grazie al costrutto `if/else` dichiarato nella prima riga, che se valutato come falso andrà a visualizzare il blocco identificato da `#loading`, altrimenti il blocco corrente. All'interno del componente `userprofile` viene passato di volta in volta un oggetto chiamato `user` presente nella lista appena scaricata, e in questo specifico caso viene mostrato come ottenere e mostrare l'indice dell'elemento in corso ed il numero totale di elementi, rispettivamente `i` e `count`.

Typescript

Le applicazioni ibride sfruttano meccanismi web, quindi il linguaggio di programmazione in ambiente web per eccellenza è sicuramente *JavaScript*, linguaggio supportato praticamente da qualsiasi browser e sistema seppur con alcune minime differenze e compatibilità.

Per sfruttare al meglio le potenzialità di JavaScript solitamente si utilizza il framework *JQuery*, che solitamente per la costruzione di semplici siti web è più che sufficiente, ma per costruire un progetto più grande ed ambizioso come un'applicazione mobile non basta, per questo Angular ha deciso di utilizzare un linguaggio più ricco e potente, TypeScript, che verrà quindi utilizzato anche nello sviluppo di applicazioni con Ionic.

Il codice scritto in *TypeScript* al momento della compilazione genera del codice JavaScript, pronto per venir interpretato da qualsiasi browser che supporta almeno la versione dello standard *ECMAScript* selezionato in fase di compilazione. Scrivere del codice JavaScript sintatticamente corretto e funzionante ed inviarlo ad un compilatore TypeScript, ritornerà in output lo stesso identico codice in input, questo significa che è possibile utilizzare la sintassi JavaScript quando si lavora in ambiente TypeScript.

Con l'utilizzo di TypeScript, si guadagna l'accesso a strutture più complesse comunemente utilizzate nella programmazione *Object Oriented* come *classi* ed *interfacce*. In Javascript è comunque possibile creare degli oggetti a cui assegnare campi e metodi, ma l'utilizzo di vere e proprie classi rende tutto più pulito ed organizzato, oltre a permettere l'ereditarietà.

Un'altra sostanziale aggiunta è l'opzionalità della dichiarazione del tipo delle variabili. In Typescript è ancora possibile dichiarare variabili ed affidarne il riconoscimento del tipo direttamente al momento dell'interpretazione del codice, ma viene anche offerta la possibilità di specificarlo in partenza, come ad esempio:

```
1 function add(left: number, right: number): number {  
2   return left + right;  
3 }
```

Codice 3.5: Dichiarazione di una funzione con i tipi delle variabili esplicitamente scritti nella *signatore* della funzione sia per i parametri in input sia per il risultato in output

Come illustrato, i numeri non sono distinti tra interi e numeri con la virgola come in altri linguaggi, ma semplicemente marcati come `number`.

Capitolo 4

Caccia alla Ricerca

4.1 Introduzione

Il progetto nasce come evento partecipante alla **Notte dei Ricercatori 2017** e consiste in un'applicazione ibrida scritta in ambiente *Ionic* con oggetto una caccia al tesoro allo scopo di permettere ai partecipanti, ragazzi delle scuole superiori, di avere un filo conduttore che li veicolerà attraverso alcune delle esposizioni presenti durante la serata.

La realizzazione del progetto è stata svolta insieme ad un'altra laureanda, *Rossi Sofia*, che si è occupata principalmente della parte *Backend*, quindi dialogo tra applicazione e server/database, mentre io mi sono occupato della parte di *Game Design* (logiche di gioco) e *Frontend*, quindi l'interazione tra i giocatori e l'applicazione.

Il gioco

All'avvio dell'applicazione, prima di procedere con il login, viene consigliata al giocatore la lettura di un brevissimo tutorial che introduce i principali aspetti del gioco, come si vede in Figura 4.1.

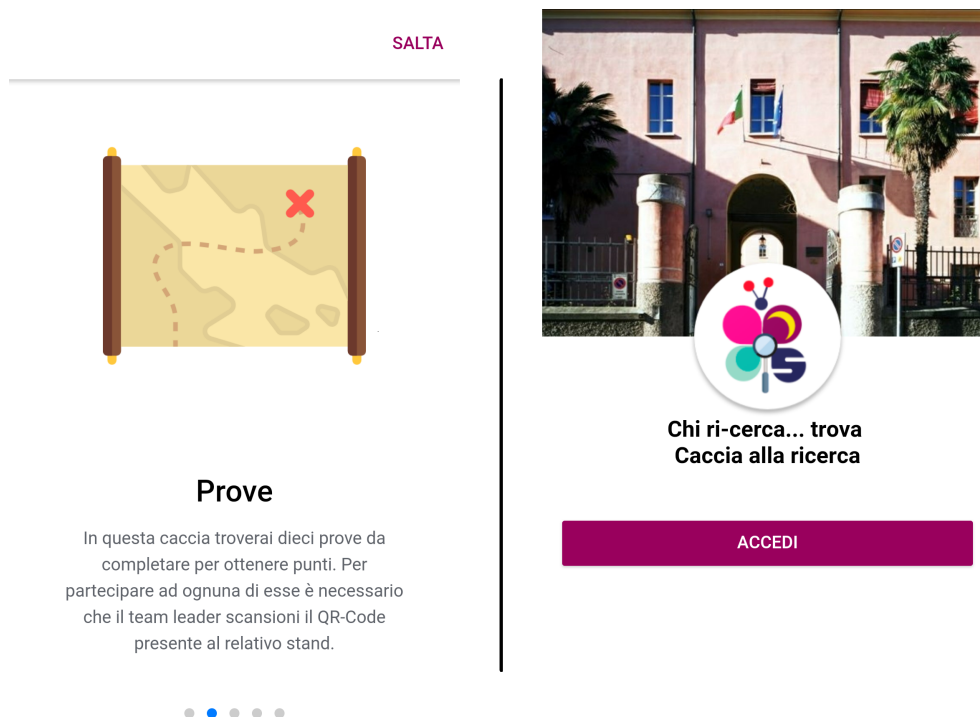


Figura 4.1: Una delle pagine del tutorial (a sinistra) e la schermata di accesso (a destra).

I giocatori sono suddivisi in squadre formate da 3 o 4 componenti che durante lo svolgimento del gioco dovranno restare quasi sempre uniti. Per ogni esposizione inclusa nel progetto (attualmente 10) ogni giocatore dovrà svolgere una prova per ottenere punti per la propria squadra, ma una sola prova alla volta potrà essere attiva per tutta la squadra, quindi il gruppo dovrà muoversi insieme.

Per attivare una prova sarà necessario che il capogruppo scansioni un *QR Code* fisicamente posizionato nel luogo dello svolgimento, che sarà appunto l'esposizione oggetto della prova. A seguito della lettura del codice, se considerato valido dall'applicazione, la prova verrà impostata come *In Corso* a tutti i membri del gruppo e questo sarà visibile direttamente nei loro dispositivi.

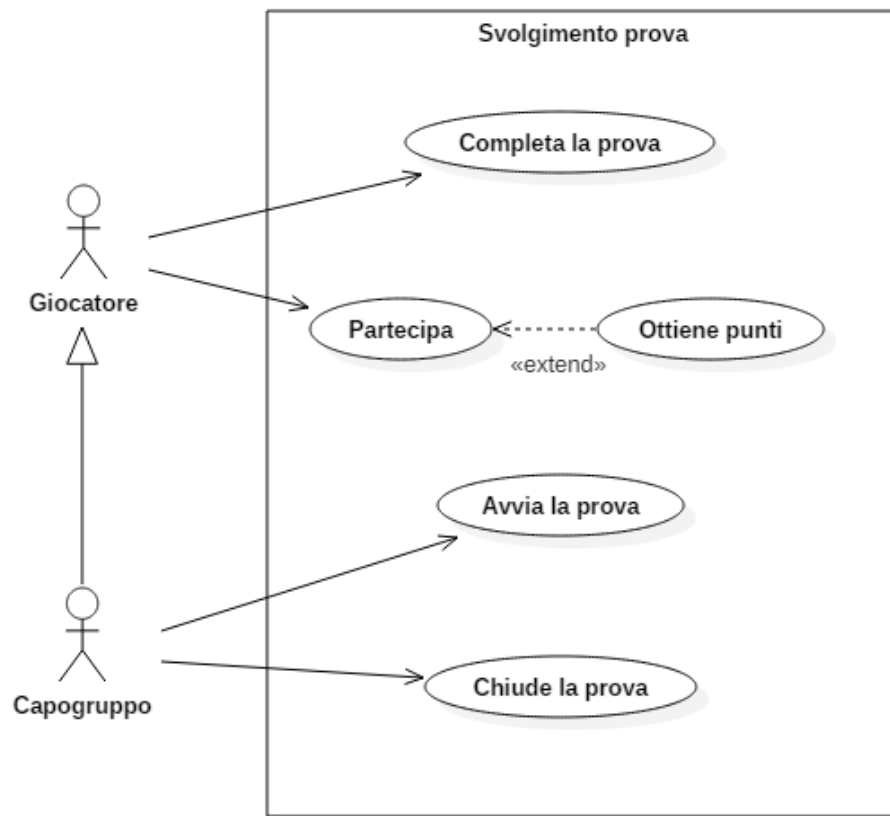


Figura 4.2: *Use Case Diagram* per illustrare come possono interagire giocatore e capogruppo durante la prova in corso.

Una volta scansionato il codice, apparirà ai giocatori una tabella con le statistiche come mostrato in Figura 4.3, aggiornate automaticamente in tempo reale, del progresso dei componenti del proprio gruppo. Oltre a permettere a ciascuno di visualizzare i punti e lo stato (prova completata o meno) dei compagni di squadra, servirà al capogruppo per sapere quando potrà dichiarare chiusa definitivamente la prova, per poterne poi avviare un'altra scansionando un nuovo QR Code.

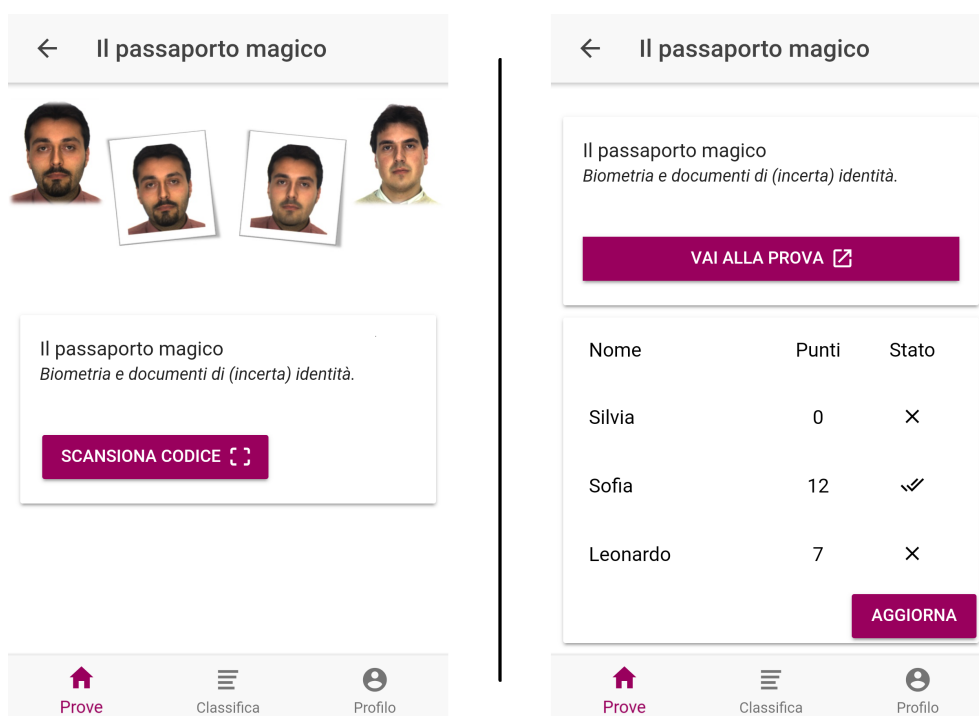


Figura 4.3: Schermata principale di una prova (a sinistra) e tabella di aggiornamnto status squadra (a destra).

Le prove possono essere di varie tipologie:

- **Quiz:** Ad ogni membro del gruppo apparirà sul proprio dispositivo una domanda con 4 possibili risposte, una corretta e tre sbagliate.
- **Lista:** Ogni membrò avrà una diversa lista di circa 10 elementi. Da questa lista dovrà selezionare tutti gli elementi suggeriti da un'affermazione (ad esempio: *Seleziona tutti i numeri primi dall'elenco*). Ogni risposta esatta darà un punto, ogni risposta errata meno uno.
- **Slide:** Un professore ha nascosto una parola chiave in alcune slide della sua presentazione. Il capogruppo dovrà trovare questa parola ed inserirla nell'applicazione aiutato dalla squadra.
- **Puzzle:** Delle immagini saranno tagliate in parti rettangolari ed ogni membro del gruppo dovrà ricomporre il proprio puzzle per far ottenere dei punti alla propria squadra.
- **Shelf:** Questa è la prova probabilmente più "fisica". Ad ogni membro del gruppo sarà assegnato un libro da ricercare all'interno della *Bi-*

biblioteca Malatestiana e dovrà inserire in un apposito form all'interno dell'applicazione alcuni dati reperiti sul posto, come il numero dello scaffale ed il colore della copertina del libro.

Come in ogni gioco a punti è necessaria una classifica. Accedendo alla seconda tab del menu principale infatti è possibile ottenere informazioni sui punti delle altre squadre in corso. Anche l'aggiornamento della classifica avviene in tempo reale con i dati contenuti ed aggiornati nel database, così che ogni squadra può sempre essere a conoscenza della propria posizione rispetto a quella delle altre squadre. In Figura 4.4 si può vedere appunto la classifica, affiancata dalla schermata principale del gioco con l'elenco delle prove.

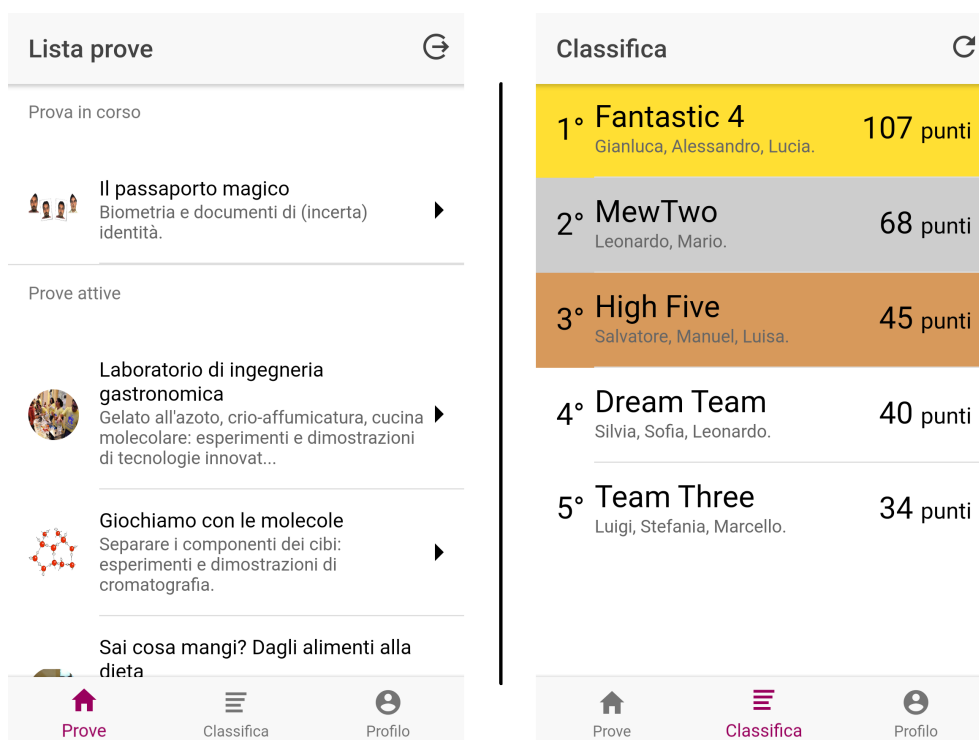


Figura 4.4: Schermata principale con l'elenco delle prove (a sinistra) e pagina della classifica, con evidenziate le prime tre posizioni (a destra).

4.2 Funzionamento Tecnico

In questa sezione andrò ad analizzare nel dettaglio le componenti dell'applicazione principalmente da me curate, tenendo conto che logicamente du-

rante l'intero periodo di sviluppo ci siamo entrambi consultati per collaborare ed aiutarci attivamente nelle rispettive parti.

Autenticazione

L'unica componente backend che ho sviluppato in prima persona riguarda il processo di autenticazione, gestione della password e del sistema di *Token*. Al momento della registrazione la password viene immediatamente cifrata tramite la funzione `password_hash` offerta nativamente da PHP. La funzione chiede in input due parametri oltre alla password in chiaro: una costante per identificare il tipo di algoritmo da utilizzare ed eventuali opzioni. Ho deciso di utilizzare l'algoritmo `CRYPT_BLOWFISH` come suggerito dalla documentazione per avere un hash di lunghezza costante, 60 caratteri. La forza di questo algoritmo è la generazione di un *Salt* casuale ad ogni utilizzo della funzione, generando hash diversi per la medesima password.

In questo modo le password registrate all'interno del database non sono in chiaro, quindi per effettuare il login è necessario l'utilizzo della funzione duale sempre offerta da PHP, `password_verify`, che prende in input la password in chiaro inserita dall'utente e la confronta con l'hash salvato sul database. La funzione restituirà `true` se la password corrisponde, `false` altrimenti, e da qui è possibile proseguire restituendo al frontend la struttura con i dati dell'utente loggato se la password è stata accettata, un errore altrimenti.

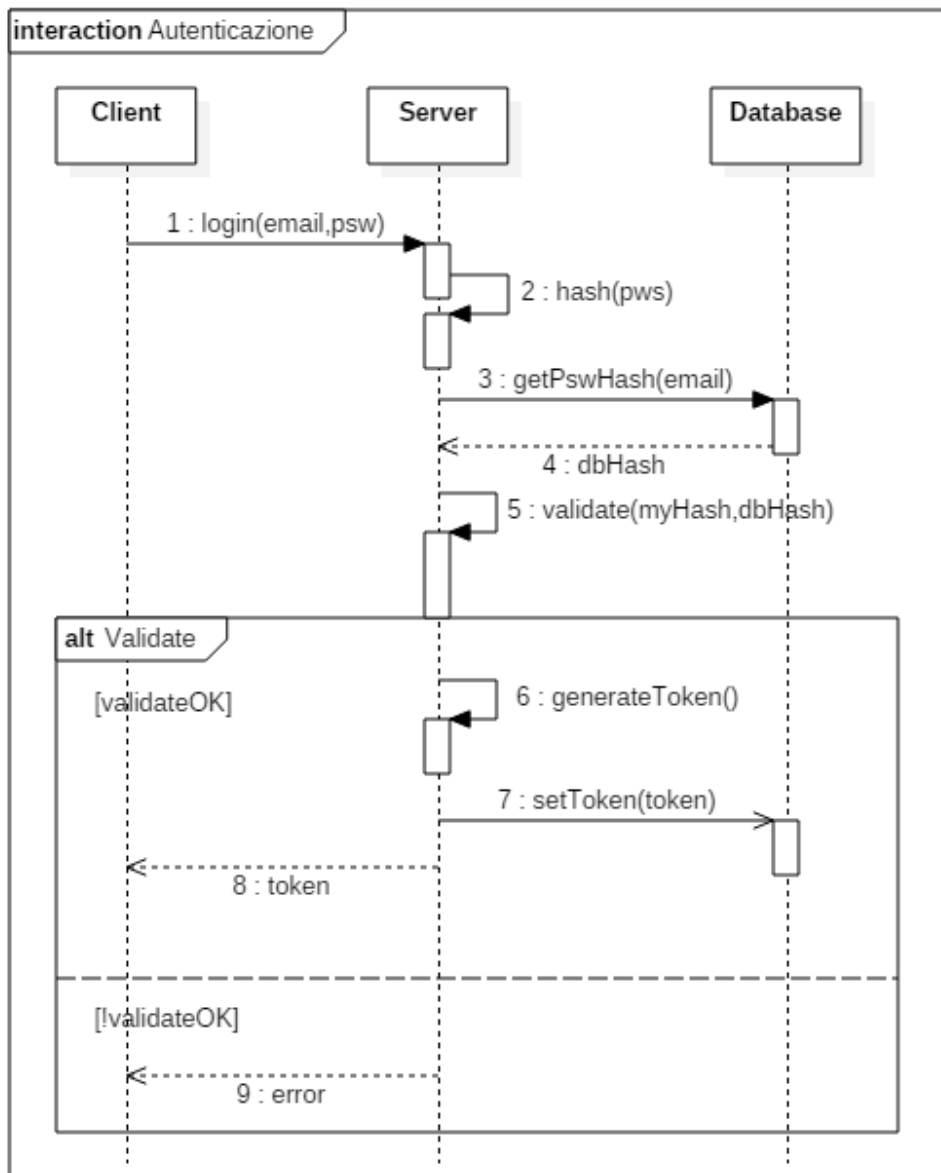


Figura 4.5: *Sequence Diagram* per illustrare i passaggi effettuati per ottenere l'accesso all'applicazione con il token di ritorno che servirà successivamente per identificarsi ad ogni chiamata.

Per garantire l'identità del chiamante ad ogni richiesta di informazioni (sia input che output) con il server l'applicazione invierà un *Token* generato dal server ad ogni accesso e comunicato unicamente durante login. Questo token, formato da 12 caratteri alfanumerici, verrà ogni volta confrontato con quello salvato nel database. Solo se il token inviato corrisponde a quello salvato

verrà accettata la richiesta al server. Questo meccanismo permette anche di evitare eventuali errori nel sistema, in quanto se viene per sbaglio passato un identificativo errato al posto di quello corretto, la verifica del token fallirà e non verranno scritti od ottenuti dati relativi ad un altro giocatore.

Lifecycle di una prova

All'interno dell'applicazione, le prove vengono gestite da una classe denominata *Challenge*. Un'istanza di *Challenge* può assumere, in fila, quattro valori di *State*:

1. **HIDDEN**: La prova è nascosta e non visibile nell'elenco.
2. **ACTIVE**: La prova è visibile ed è possibile attivarla, tramite la lettura del corrispondente *QR Code*.
3. **CURRENT**: L'unica prova in corso, posizionata in cima alla lista.
4. **CLOSED**: La prova è stata completata e conclusa, resta nella lista per vedere i punti.

La prima valutazione dello stato di una prova avviene al momento del login, quando viene richiesto al database lo stato in corso da cui iniziare. Nel caso la prova non sia ancora stata avviata (**CURRENT**) o già conclusa (**CLOSED**) si rientra nel caso base, descritto da dallo *StateChart Diagram* in figura Figura 4.6.

Come si può intuire dal diagramma in Figura 4.6, una prova allo stato iniziale può essere nascosta. Questo significa che non comparirà nella lista insieme alle altre fino ad un evento prestabilito, una notifica che avvisa il giocatore della comparsa di una prova aggiuntiva. A seguito della notifica, la prova passerà dallo stato **HIDDEN** allo stato **ACTIVE**.

Per il passaggio da **ACTIVE** a **CURRENT** è necessario che non ci sia nessun'altra prova attualmente in stato **CURRENT**, in questo modo i giocatori sono forzati nello svolgere una prova alla volta. Se quindi non c'è nessuna prova attualmente in corso, il capogruppo avrà la possibilità di avviare la fotocamera del proprio dispositivo per procedere con la lettura di un *QR Code*

che se ritenuto valido andrà ad impostare lo stato della prova selezionata su `CURRENT` ed andrà ad aggiungere questa informazione nel database. Tramite un processo in background i device degli altri membri del gruppo si aggiorneranno di conseguenza, impostando anche nel loro client l'avvenuto cambio di stato della prova.

A questo punto tutti i componenti del gruppo avranno la possibilità di premere un bottone per accedere alla schermata dove si svolgerà la prova, ancora segnata come incompleta. Dopo aver svolto quanto richiesto dalla prova verrà registrato dall'applicazione il completamento della prova stessa, automaticamente in alcune tipologie mentre manualmente (richiesto al giocatore) in altre tipologie di prove, e comunicato al database.

Dopo che tutti i membri del gruppo, capogruppo compreso, avranno completato la prova e registrato il punteggio sul database il capogruppo avrà la possibilità di chiudere definitivamente la prova impostandola localmente su `CLOSED` ed informando il database dell'avvenuta chiusura della prova. A seguire anche gli altri membri del gruppo troveranno la prova conclusa nel proprio dispositivo, pronti in ascolto per ricevere informazioni sulla prova successiva che passerà a `CURRENT`.

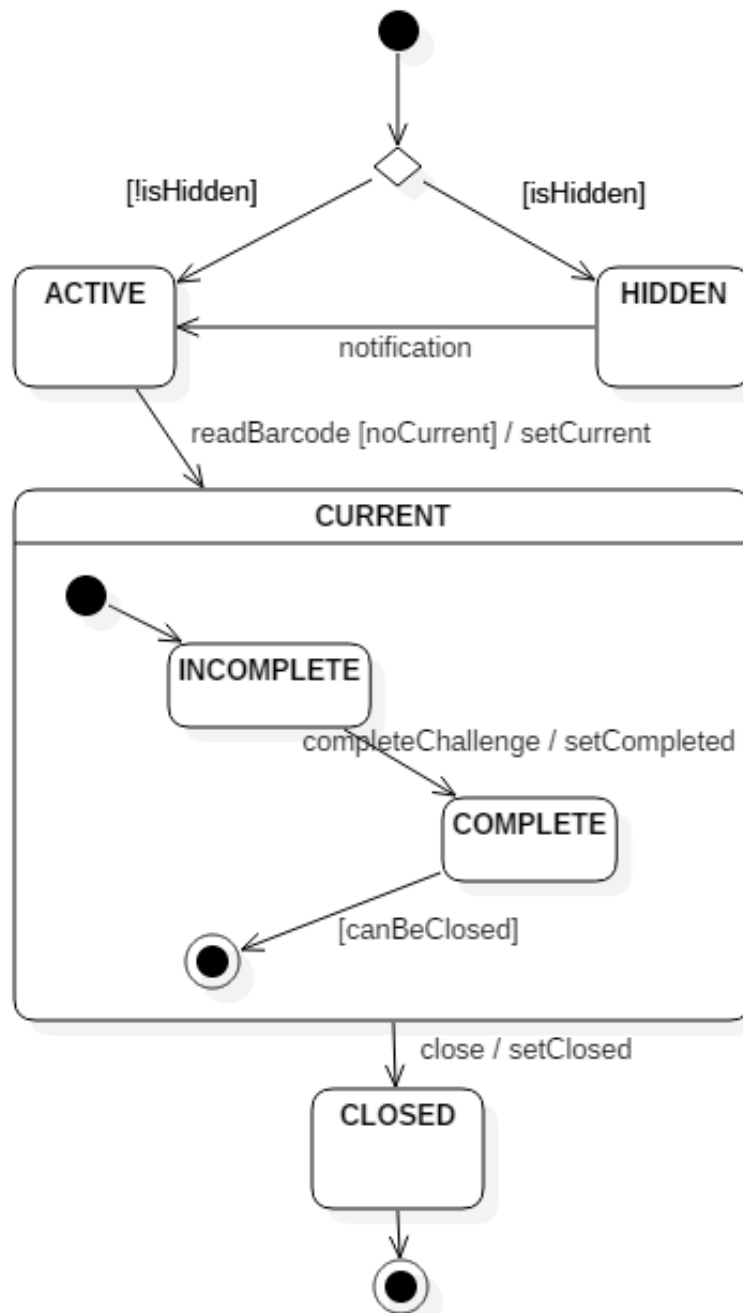


Figura 4.6: *StateChart Diagram* per illustrare i passaggi di stato della classe `Challenge`, per tenere traccia dello stato di svolgimento delle prove.

Gestione dei dati: i provider

Per gestire la fonte da cui reperire i dati per le singole prove è stata utilizzata una classe `provider` prendendo spunto dalla documentazione Ionic. Lo scopo di questa classe è incapsulare al suo interno le strutture dati necessarie in formato JSON relative ad una specifica situazione, per poi fornire su richiesta il dato necessario dall'applicazione. Lo stesso meccanismo è stato utilizzato per la gestione iniziale della lista di `Challenge`, così come appunto per fornire a `View` e `Controller` i dati necessari per impostare ed interagire con la prova.

I dati informato JSON possono essere reperiti da diverse fonti, come ad esempio un database/server remoto, un file con estensione `.json` all'interno del progetto o direttamente con una stringa appositamente formattata che crea automaticamente gli oggetti.

Una grande comodità di Typescript, in quanto esteso da Javascript che non è un linguaggio fortemente tipizzato, è la possibilità di dichiarare l'istanza di un oggetto del tipo primitivo `any`. Molto simile al concetto di `Object` nella comune programmazione ad oggetti, è possibile dichiarare un oggetto di tipo `any` ed al suo interno inserire un oggetto formato da una struttura qualsiasi, per poi richiamare i valori salvati all'interno nel formato `chiave.valore` come fossero campi pubblici. Due grossi vantaggi nell'usare `any` sono la possibilità di poter stabilire una specifica struttura senza la necessità di dover creare un oggetto esplicito (utilizzato per le strutture per le singole prove), così come il poter creare un'istanza di una classe esplicita a partire da un semplice `any` con una struttura compatibile (utilizzato per istanziare le `Challenge` reperite dal database durante il login).

Ambiente di sviluppo

La gestione del progetto Ionic è stata svolta come suggerito tramite la *Ionic CLI*, per generare il progetto base ed ogni volta compilare ed eseguire il codice generato su browser e sui dispositivi tramite i comandi offerti dalla *Command Line Interface*.

Lo sviluppo per la mia parte è stato quasi interamente effettuato da una

macchina Windows 10 Pro (Surface Pro 4) testando l'applicazione su un dispositivo Android (Huawei P9). A scopo di verifica anche in ambiente Apple è stata testata l'applicazione sfruttando la possibilità di deploy di *Xcode* su un MacBook Air con macOS Sierra 10.12, ed un iPhone 6 come target.

Come IDE (*Integrated Development Environment*) è stato scelto *Atom* con l'aggiunta del plugin `atom-typescript` per supportare appunto la programmazione in Typescript con highlight del testo e controllo statico degli errori. Per gestire invece i file caricati sul server *Notepad++* con il plugin per il protocollo FTP (*File Transfer Protocol*).

Importante precisare che per l'accesso ai file sul server di Unibo è stato usato il protocollo SFTP attraverso una VPN (*Virtual Private Network*) dedicata per l'accesso tramite IP dell'ateneo.

La gestione del database *MySQL* è stata affidata al software *phpmyadmin* caricato sul server.

Per coordinare lo sviluppo dell'applicazione e la gestione degli aggiornamenti è stato deciso l'utilizzo di *BitBucket* per depositare e rendere disponibile il repository creato e gestito con *Mercurial*.

4.3 Edutainment

Lo scopo principale di questo progetto, *Caccia alla Ricerca*, non si limita ad essere unicamente una ricerca sulle applicazioni ibride, così come non sarà solamente un passatempo per gli studenti delle scuole superiori che utilizzeranno l'applicazione per partecipare alla caccia al tesoro durante la notte dei ricercatori.

Dietro l'ideazione di questo progetto ci sono dei fondamenti della recente, circa, filosofia dell'*Edutainment* [10]. Il gioco della caccia al tesoro deve servire per permettere ad i ragazzi di avvicinarsi ed osservare il maggior numero di esposizioni possibili durante la serata, senza però che siano forzati da qualcuno ma semplicemente perchè di loro spontanea volontà saranno interessati ad ottenere punti per vincere il gioco e l'unico modo per farlo sarà quello di imparare qualcosa di nuovo per poter risolvere le prove.

Per *Edutainment* si intende *Educazione* ed *Intrattenimento*. Al giorno d'oggi imparare può sembrare una pratica noiosa fin da piccoli, dove si viene costretti ad imparare seguendo noiose lezioni a scuola, magari anche contro voglia, perchè è difficile nei primi anni di vita riuscire a capire l'importanza del sapere.

Qualsiasi cosa fatta contro voglia o senza un minimo interesse, già dal principio si sa che non sarà mai fruttuosa come dovrebbe. Imparare annoiandosi non funziona e richiede molto più tempo di quello che sarebbe necessario in una situazione di interesse, con uno stimolo proveniente direttamente in prima persona da **vuole** imparare, perchè importante, e divertente.

Un chiaro esempio di Edutainment si può trovare già in una pubblicazione del 2009 di tre docenti del dipartimento di informatica dell'università di Bari [2] nella quale descrivono un progetto molto simile a *Caccia alla Ricerca* pensato per siti archeologici con le tecnologie offerte dai cellulari nel 2009. Il loro progetto, *Explore!*, consiste in una caccia al tesoro a tappe sequenziali dove i gruppi di ragazzi (dagli 11 ai 13 anni) possono muoversi liberamente all'interno del sito e sono chiamati ad immaginarsi e ragionare su come dovesse essere ad esempio lo stile di vita in quel luogo all'epoca in cui è stato costruito.

Costantemente due fasi di gioco e riflessione si alterneranno per permettere ai ragazzi di discutere e fissare meglio i concetti scoperti ed imparati nella fase di gioco appena terminata.

In questa maniera, anche quella che può sembrare una noiosa visita ad un sito archeologico può diventare un'occasione di divertimento per i ragazzi che quasi involontariamente concluderanno la giornata di gioco e divertimento con l'aver appreso nozioni storiche attraverso quello che può essere definito un gioco per cellulari con una diretta interazione con il mondo che li circonda.

Importante sottolineare che questo esempio è di ben 8 anni fa, quando le prime rappresentazioni 3D iniziavano a diventare diffuse. Al giorno d'oggi con la possibilità di sfruttare tecnologie come la realtà aumentata le possibilità di sviluppare applicazioni simili, molto più complesse e piene di funzionalità e divertimento, sviluppare applicazioni pensando all'edutainment è veramente facile e produttivo ed i ragazzi ne potranno trarre sicuramente un immenso

beneficio.

Capitolo 5

Conclusioni

Il percorso intrapreso per la stesura di questa tesi ha portato diversi risultati. L'analisi passo per passo dell'evoluzione delle tecnologie web è stata necessaria per poi poter affrontare la decisione su quale tecnologia utilizzare per la realizzazione del progetto, offrendo anche una conoscenza più approfondita per tenere in considerazione le altre opzioni nell'eventualità una situazione simile in futuro dovesse ripresentarsi, dovendo scegliere tra applicazioni native, ibride o PWA.

La decisione finale nell'utilizzo di Ionic è stata dettata principalmente dal fatto che il progetto inizialmente avrebbe dovuto utilizzare più componenti nativi possibili (fotocamera, geolocalizzazione, notifiche push e sensori) anche se nel prodotto finale non tutti questi elementi compariranno. Sarebbe stato molto interessante sfruttare la geolocalizzazione nella realizzazione di una caccia al tesoro, ma essendo tutte le prove (tranne una, alla biblioteca malatestiana) localizzate all'interno della sede dell'evento, palazzo Mazzini Marinelli, è stato necessario scartare questa funzionalità da quelle utilizzate all'interno del software.

Non avendo grandi esigenze di prestazioni è stato pensato quindi che un'applicazione ibrida potesse risultare adatta, e così è stato. Inoltre, per la natura di Ionic, è possibile utilizzare lo stesso codice compilato per la creazione degli eseguibili nativi anche per la versione *browser*, trattata dal framework come se fosse un sistema operativo a parte, avendo di fatto in mano contemporaneamente l'applicazione ibrida e Progressive Web App.

5. Conclusioni

L'inconveniente del dover installare l'applicazione non sarà un problema nel caso specifico, in quanto ai giocatori sarà espressamente indicato di scaricare l'applicazione per giocare, ma per altri tipi di software sicuramente questo può risultare sconveniente. La parte più scomoda è quella precedente al download, ossia l'upload dei file eseguibili (.apk per Android e .ipa per iOS). Alla luce dell'esperienza maturata si è sviluppata una consapevolezza sull'idea che le PWA prenderanno una buona parte di mercato e faranno la storia dell'ambiente mobile grazie alla loro semplicità ed immediatezza, sia per lo sviluppatore che per l'utente finale.

Poter sviluppare una vera e propria applicazione per browser, gestita dal dispositivo come app ma programmata come servizio web, ha dei grandissimi potenziali grazie allo sviluppo che sta ottenendo e al supporto dalle principali aziende come Google ed Apple.

In conclusione quindi, le applicazioni ibride hanno sicuramente molti lati positivi, ma molto presto saranno superate dalle PWA, finché l'ennesima novità non stravolga completamente il mercato, come ogni tanto accade.

Bibliografia

- [1] C. Anderson. The long tail. *wired.com*, Oct. 2004.
- [2] C. Ardito, M. F. Costabile, and R. Lanzilotti. Enhancing user experience while gaming in archaeological parks with cellular phones. In *Proceedings of the 8th International Conference on Interaction Design and Children*, IDC '09, pages 270–271, New York, NY, USA, 2009. ACM.
- [3] M. Asay. Apple could lose billions on progressive web apps, but it has no choice. *techrepublic.com*, Nov. 2016.
- [4] D. Blotsky. Apache cordova. *cordova.apache.org*, Nov. 2015.
- [5] A. Bradley. Where does the ionic framework fit in? *blog.ionic.io*, Oct. 2013.
- [6] J. Bristowe. What is a hybrid mobile app? *developer.telerik.com*, Mar. 2015.
- [7] B. Carney. Ionic 3.0 has arrived! *blog.ionic.io*, Apr. 2017.
- [8] S. Fluin. Angular 4.0.0 now available. *http://angularjs.blogspot.it*, Mar. 2017.
- [9] E. Genco. Harnessing modern web application architecture with progressive web apps. *mentormate.com*, Mar. 2017.
- [10] B. George. Education vs. edutainment. *huffingtonpost.com*, Oct. 2015.
- [11] J. Grigsby. ios doesn't support progressive web apps, so what? *cloudfour.com*, Oct. 2016.

-
- [12] A. Gustafson. Understanding progressive enhancement. *alistapart.com*, Oct. 2008.
- [13] S. koirala. Can you explain lazy loading? *codeproject.com*, Sept. 2013.
- [14] B. LeRoux. Phonegap, cordova, and what's in a name? *phonegap.com*, Mar. 2012.
- [15] M. Lynch. Announcing ionic 2.0.0 final. *blog.ionic.io*, Jan. 2017.
- [16] E. Marcotte. Responsive web design. *alistapart.com*, May 2010.
- [17] C. Mills. Graceful degradation versus progressive enhancement. *w3.org*, July 2011.
- [18] A. Moryossef. Ionic and rtl. *blog.ionic.io*, June 2017.
- [19] M. Ogden. Callback hell. *callbackhell.com*, July 2012.
- [20] T. O'Reilly. What is web 2.0, design patterns and business models for the next generation of software. *oreilly.com*, Sept. 2005.
- [21] A. Trice. Phonegap advice on dealing with apple application rejections. *adobe.com*, Oct. 2012.
- [22] L. Wroblewski. Mobile first. *lukew.com*, Nov. 2009.