UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Dipartimento di Scienze Fisiche, Informatiche e Matematiche

Corso di Laurea in Informatica

Progetto e Sviluppo dell'item gallery di mobile.yoox.com, con tecnologia AngularJS

Francesco Nicoli

Tesi di Laurea

Relatore:
Prof. Riccardo Martoglia

Anno Accademico 2015/2016

RINGRAZIAMENTI

Ringrazio l'Ing.Riccardo Martoglia, mio relatore, per la sua disponibilità, i suoi consigli e la sua assistenza durante questo percorso.

Ringrazio inoltre l'azieda YOOX-NET-A-PORTER per avermi dato l'opportunità sia di entrare in contatto con una realtà leader nel settore e-commerce, sia di avermi dato l'occasione di lavorare su questo progetto.

Ringrazio infine i miei genitori che in questi anni mi hanno sostenuto e sopportato.

Indice

Introduzione

Parte 1: Tecnologie utilizzate

Capitolo 1: JavaScript Framework

1.1 JavaScript Framework: cosa sono, perché utilizzarli
1.2 MVC Architecture
1.3 JavaSscript MVC Framework: vantaggi

Capitolo 2: Scegliere il framework JavaScript

2.1 Community
2.2 Framework: dimensioni
2.3 Template
2.4 Framework a confronto

Capitolo 3: Conoscere AngularJS

3.1 Data Binding
3.2 Dove inserire lo script AngularJS
3.3 Inizializzare un'applicazione AngularJS
3.4 Moduli
3.5 AngularJS Scope
3.6 Dirty Checking e $watch
3.7 Integrazione con il ciclo di eventi del browser, $digest Loop e $apply
3.8 Dirty Checking
3.9 Integrazione con il ciclo di eventi del browser e AngularJS
3.10 AngularJS Controller
3.11 Espressioni AngularJS
3.13 Direttive
3.14 Dependency Injection (DI)
3.15 Servizi

Capitolo 4: Tecnologia utilizzata in mobile.yoox.com

4.1 RESTful Web Service
4.2 ASP.NET
4.3 Razor
4.4 JSON
4.5 CDN

Parte 2: Progettazione e implementazione

Capitolo 5: Progettazione della soluzione client-side con AngularJS

5.1 Il sito web yoox.com: versioni desktop e mobile
5.2 Perché ho scelto AngularJS e in quale versione
5.3 Il caso di studio: mobile.yoox.com
5.4 Infinite Scrolling
5.5 Lazy Loading
5.6 Infinite Scrolling e Lazy Loading
5.7 Ciclo di vita di una richiesta web per mobile.yoox.com
5.8 Ciclo di vita di una richiesta web per mobile.yoox.com, con tecnologia AngularJS
5.9 Dati relativi agli oggetti: scelta del tipo di dato
5.10 Il pacchetto JSON di yoox.com
5.11 Come implementare l'Infinite Scrolling
5.12 URL & Navigazione
5.13 Funzionalità "segnalibro" del browser
5.14 Funzionalità del bottone "indietro" del browser
5.15 Navigazione tramite parametro URL "page"
5.16 SEO e rendering client-side di pagine web
5.17 I filtri di ricerca

Capitolo 6: Implementazione della soluzione client-side con AngularJS

6.1 Il template Razor
6.2 AngularJS Template
6.3 AngularJS View Logic
6.4 AngularJS Business Logic
6.5 Il Template AngularJS e le Direttive

Capitolo 7: Performance e miglioramenti

7.1 WebPagetest
7.2 Ambiente locale
7.3 Infinite Scrolling: il collo di bottiglia
7.4 Pre-caricamento del file JSON

Parte 3: Conclusioni e sviluppi futuri

Giudizio sul framework AngularJS
Giudizio riguardo la soluzione client-side con AngularJS
Sviluppi futuri

Introduzione

L'interesse maturato negli ultimi tempi nell'ambito del web design e web development mi ha portato a scegliere l'attività di tirocinio presso l'azienda YOOX NET-A-PORTER GROUP.

YOOX NET-A-PORTER GROUP, il partner globale di Internet retail per i principali brand della moda e del lusso, si è affermato tra i leader di mercato con gli online store multi-brand yoox.com, thecorner.com, shoescribe.com ed i numerosi online store mono-brand tutti "Powered by YOOX NET-A-PORTER GROUP".

yoox.com, fondato nel 2000, è lo store online di lifestyle leader nel mondo: moda, design e arte. Grazie a consolidate relazioni dirette con designer, produttori e dealer autorizzati, yoox.com offre una selezione infinita di prodotti: un'ampia scelta di capi d'abbigliamento e accessori difficili da trovare, dei più importanti designer al mondo, capsule collection esclusive, proposte di moda eco-friendly, un assortimento unico di oggetti di design, rari capi vintage, originali libri, e una ricercata collezione di accattivanti opere d'arte.

Un utente, arrivando su yoox.com, può compiere ricerche per diverse categorie di prodotti, utilizzare filtri e parole chiave.
Una volta compiuta una ricerca, ad esempio per categoria, l'utente può navigare la galleria di oggetti che viene restituita come risultato di ricerca.

Ogni volta che l'utente si inoltra nella navigazione della galleria di oggetti, attraverso i bottoni per cambiare pagina nella versione desktop, o l'Infinite Scrolling nella versione mobile di yoox.com, viene restituita la galleria di oggetti corrispondente.

L'attività di tirocinio prevedeva lo studio della soluzione software per la creazione di pagine web dinamiche, implementata a livello di server per il sito yoox.com, lato desktop e mobile.
Successivamente avrei dovuto trovare un'alternativa a tale soluzione software, non a livello di server, ma bensì a livello di client.
Infine avrei implementato questa nuova soluzione client-side.
La re-ingegnerizzazione del sito mobile di yoox.com con tecnologia AngularJS si è sviluppata in più fasi, alcune delle quali hanno richiesto approfondimenti in linguaggi o discipline oltre il contesto JS e gli insegnamenti previsti dal corso di Laurea, ma comunque legati all'ambiente di frontend: parliamo di HTML (HyperText Markup Language), CSS (Cascading Style Sheets) e SEO (Search Engine Optimization).

L’obiettivo di questa re-ingegnerizzazione è stato analizzare pregi e difetti dell’utilizzo di una tecnologia differente da quella attualmente in vigore, di tipo server-side, e che invece eseguisse a livello di client.

Parte I

Tecnologie utilizzate

Capitolo 1

Javascript Framework

1.1 JS Framework: cosa sono, perché utilizzarli

Il linguaggio JavaScript è oggigiorno il più popolare, data la sua versatilità nei vari ambiti di programmazione: dalle applicazioni web fino ad arrivare, recentemente, a quelle desktop (Ionic, Electron).

Le applicazioni crescono in complessità e in numero di servizi: la quantità di codice è in continua espansione ed è di importanza cruciale scrivere codice riusabile e mantenibile. A questo scopo nascono i design patterns e, successivamente, i framework: ma perché scegliere un framework invece della classica applicazione con struttura server-side?

In termini semplici, i JS framework migliorano la User Experience (UX): il view layer è spostato client-side ed è a sè stante, quindi nel momento in cui un utente fa una richiesta per un certo contenuto, non deve essere scaricata l'intera pagina web (HTML + CSS + JS + contenuto), ma solo i dati relativi al contenuto.

Un uso classico che si fa dei JS framework è quello legato alle Single Page Application (SPA): una SPA è una applicazione web costituita da una sola pagina html, che, su richiesta dell'utente o allo scatenarsi di eventi JS, carica una o più "viste" (view) diverse.

L'uso di un SPA apporta benefici di grande importanza su 3 fronti:

1) risparmio in termini di traffico dati;

2) un quantità minore di dati in download significa un tempo minore per il caricamento della pagina;

3) tempi di caricamento minori determinano un'applicazione web più reattiva all'azione dell'utente.

Ci si potrebbe chiedere se non si possa replicare una applicazione web del genere con tecnologie alternative: si potrebbe ad esempio utilizzare una libreria JS come jQuery combinata con Underscore (template library).

Ma librerie e framework sono due cose ben diverse: mentre le prime forniscono solo funzionalità al nostro programma, i framework (oltre a definire funzionalità aggiuntive) danno una struttura ben precisa al programma e definiscono il modo e l'ordine di come le parti del programma vengono eseguite e interagiscono tra di loro.

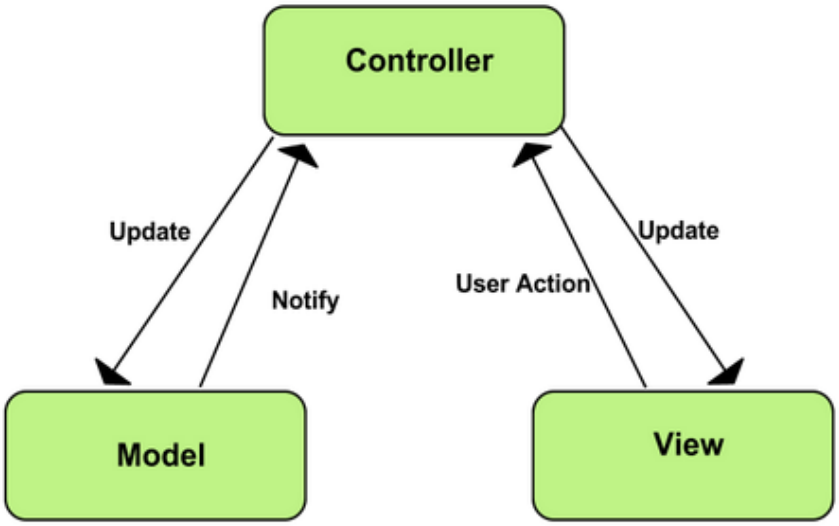
Inoltre, per un'applicazione web avente come requisito fondamentale un'elevata interazione con l'utente, è conveniente seguire un approccio MVC: un'approccio che, se implementato attraverso le librerie, porterebbe a codice poco strutturato, non mantenibile, non riusabile e non testabile.

Anche in questo caso ci vengono in aiuto i framework, spesso implementando il design pattern pattern MVC per ovviare all'organizzazione del codice e ad un uso intensivo dell'interfaccia utente (UI).

Vediamo ora quali sono i casi ottimali per l'utilizzo di un framework MVC:

- l'applicazione web richiede una connessione asincrona con il comparto backend;
- l'applicazione web ha funzionalità che non devono essere compromesse dal ricarimento della pagina web (Infinite Scrolling);
- una parte consistente di codice legata alla visualizzazione e manipolazione dei dati può essere/ deve essere spostata client-side;
- i dati della pagina web devono essere presentati più volte, in modo differente;
- l'applicazione web prevede l'invio frequente di dati attraverso elementi della UI (bottoni, input, switch …).

1.2 MVC Architecture



Il design pattern MVC è formato da 3 componenti:

1. View (vista): mostra il contenuto della pagina con uno stile, una formattazione del testo e un'impostazione di pagina ben definiti. Rappresenta la parte di interazione con l'utente, la UI.

   E' codificata in HTML, CSS, JavaScript e spesso fa utilizzo di template.

2. Model (modello): è adibito alla gestione dei dati e alla business logic.

   E' la parte in cui vengono salvati i dati necessari per l'esecuzione dell'applicazione. Il Model non è a conoscenza né delle viste, né dei controller: quando avviene un cambiamento nel Model vengono avvisate le componenti software "osservatori" del Model.

3. Controller: riceve le richieste dell'utente e fa una chiamata al Model per gestire la richiesta.

   Svolge la funzione di collegamento tra la View e il Model: aggiorna la View sia quando il Model cambia, e viceversa, quando la View cambia (per user input) viene aggiornato il Model. A questo scopo vengono aggiunti alla View componenti software in attesa di eventi.

Ricapitolando, questi sono i principali vantaggi del design pattern MVC:

- codice riusabile, estendibile e mantenibile;
- separazione della View logic dalla business logic;
- permette agli sviluppatori di lavorare simultaneamente su più livelli, dal momento che i livelli sono indipendenti l'uno dall'altro;
- modularità.

1.3 JS MVC Framework: vantaggi

Analizziamo ora i vantaggi di un a JS MVC Framework:

- Two-way Binding tra HTML e client-side JavaScript object model;
- View Template;
- Model: tracciamento dei cambiamenti;
- Data Storage (in locale/ un web server che comunica con un database);

1.3.1 Data Binding

E' la caratteristica saliente di un JS Framework. Cambiare il testo di un elemento HTML come `<input>` vuol dire che l'oggetto JS collegato a tale `<input>` viene subito aggiornato.

1.3.2 View Template

Il JS Model deve essere interallacciato con la parte HTML, perché avvenga il data binding, e i JS framework offrono principalmente due opportunità: template a livello di stringa (string-based) e template a livello di DOM (DOM-based).

Utilizzare un JS template è un modo per separare la struttura HTML (template) della pagina web, dai dati (JS model) in essa contenuti.

I template string-based ricevono in input delle parole o un template di testo e sostituiscono le parti dinamiche con i dati del modello.
(Backbone, Ember)

I template DOM-based si rifanno alla natura dichiarativa del mark-up e la parte HTML è annotata con attributi aggiuntivi per definire il data binding e gli eventi/ comportamenti associati. Questo tipo di template (coadiuvato dal template engine del framework) richiede meno codice da scrivere ed è molto più performante nei browser moderni. (AngularJS)

1.3.3 Model: tracciamento dei cambiamenti

Alcuni framework (Backbone) sono più focalizzati sul Model: allo sviluppatore è lasciata carta bianca riguardo alla gestione degli aggiornamenti tra Model e View. Bisogna estendere il JS object model e accedere alle sue proprietà con metodi quali `.set()` e `.get()`.

Altri framework (AngularJS) eseguono un processo chiamato "Dirty Checking" su tutti gli elementi del DOM collegati al Model: gli aggiornamenti Model – View avvengono solo per gli elementi HTML e per i dati del Model che sono stati modificati.

Questo processo permette allo sviluppatore di essere più produttivo e dedicarsi allo sviluppo delle funzionalità della sua applicazione, specialmente nelle fasi iniziali di apprendimento di un framework.

1.3.4 Data Storage

I framework offrono modalità differenti di salvare i dati del Model su un server:

- automaticatimente sincronizzandosi con servizi RESTful;
- chiedono allo sviluppatore di implementare chiamate ajax ai servizi web, i quali inviano una risposta in formato json;
- permettono entrambe le possibilità.

REST

Alcuni framework (Backbone) assumono che esista di default sul server un servizio web di tipo JSON RESTful e che lo sviluppatore sia in grado di gestire il trasferimento dati in modo asincrono, così che la UI sia reattiva alle azioni dell'utente. Internamente, questi framework usano jQuery o altre librerie per inviare le richieste AJAX al server. Quando avvengono cambiamenti nel Model, la parte di sincronizzazione viene notificata e questa invia gli aggiornamenti al servizio RESTful, mantenendo il Model aggiornato sul server.

Do-it-yourself (DIY)

Questi framework (AngularJS) chiedono allo sviluppatore di usare chiamate AJAX ai servizi web per soddisfare il bisogno di data storage.

Capitolo 2

Scegliere il Framework JavaScript

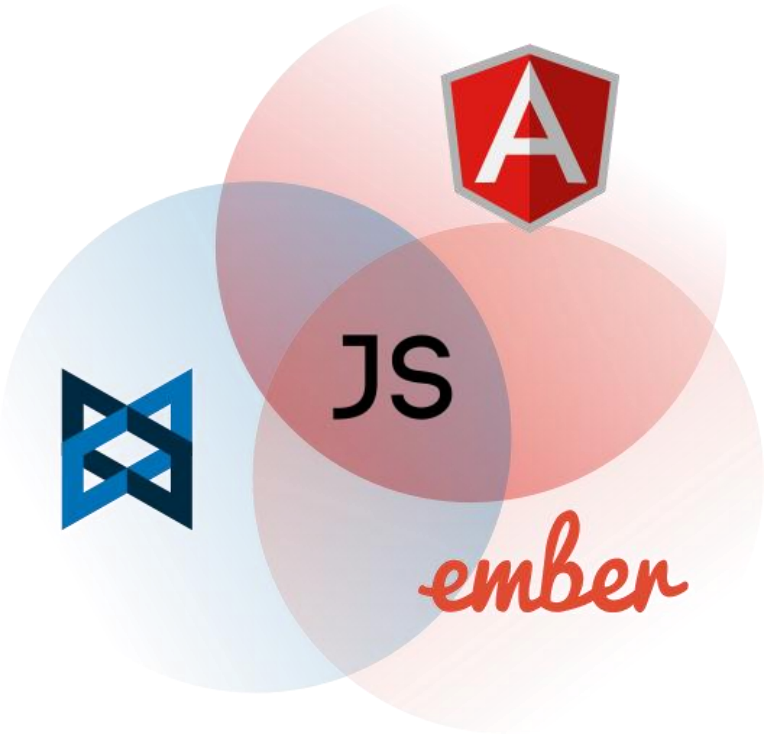
Introduzione

Procediamo ad esplorare i 3 framework JS più promettenti e popolari tra gli sviluppatori: AngularJS, Backbone, Ember. Analizziamo prima 3 fattori principali per il successo di un framework: la community, la dimensione in KB del framework, l'utilizzo di template per la View.

Sebbene esistano molti JS framework, ne sono stati individuati 3 che spiccano sugli altri: Backbone, AngularJS, Ember.

Condividono 3 caratteristiche:

1. il codice è open source, rilasciato su licenza MIT;
2. sono basati su architettura MVC;
3. adempiono alla funzionalità di SPA con l'aiuto del pattern MV*

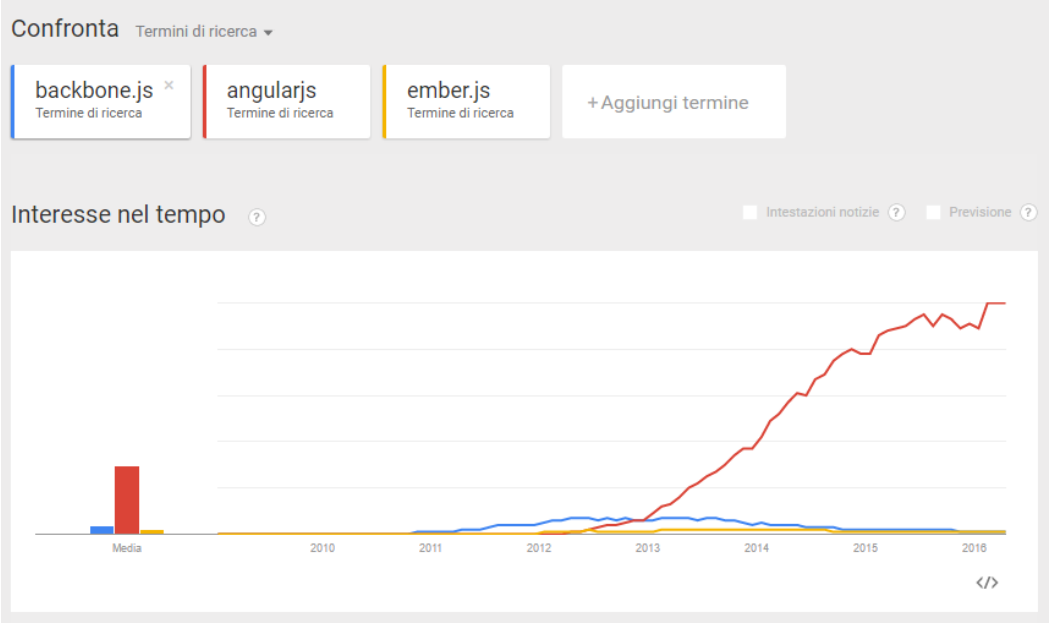


2.1 Community

La community è uno dei fattori principali nella scelta di un framework: un'ampia community significa infatti più risposte ai problemi, più moduli di terze parti, più documentazione, tutorial e video su Youtube. Come si può vedere dalla griglia, AngularJS è di gran lunga il più popolare.

Misura	*AngularJS*	*Backbone.js*	*Ember.js*
Stelle su Github	40.2k	18.8k	14.1k
Moduli di terze parti	1488 ngmodules	256 backplugs	1155 emberaddons
Domande su StackOverflow	104k	18.2k	15.7k
Risultati di YouTube	~93k	~10.6k	~9.1k
Collaboratori su GitHub	96	265	501
Utenti di estensioni Chrome	275k	15.6k	66k
Problemi senza risposta	922	13	413
Problemi risolti	5,520	2,062	3,35

Non solo AngularJS ha la più vasta community e molto più contenuto online ma è anche sostenuto da Google.



2.2 Framework: dimensioni

Il tempo di caricamento di una pagina web è cruciale nel successo di un sito web e 2 sono i fattori che lo influenzano, nel caso di utilizzo di un framework: la dimensione in KB del framework e il tempo richiesto per l'avvio di tale framework.

I file JS dei framework sono solitamente minificati e in formato compresso.

Framework	*Dimensione*	*Dimensione + dipendenze*
AngularJS 1.2.22	39.5 KB	39.5 KB
Backbone.js 1.1.2	6.5 KB	43.5 KB (jQuery + Underscore) / 20.6 KB (Zepto + Underscore)
Ember.js 1.6.1	90 KB	136.2 KB (jQuery + Handlebars)

2.3 Template

AngularJS ed Ember includono un template engine, Backbone invece lascia allo sviluppatore la scelta di usare ciò che egli preferisce.

La parte di HTML contiene sia gli elementi standard che quelli specifici del framework in uso.

```
<ul>
    <li ng-repeat="framework in frameworks" title="{{framework.description}}">
                    {{framework.name}}
    </li>
</ul>
```

AngularJS template engine

```
<ul>
    {{#each frameworks}}
        <li {{bind-attr title=description}}>
            {{name}}
        </li>
    {{/each}}
</ul>
```

Ember.js template engine

2.2 Framework a confronto

Procediamo ora ad esaminare in dettaglio pregi e difetti dei 3 framework: AngularJS, Backbone, Ember.

2.2.1 AngularJS

AngularJS, nato come progetto sperimentale nel 2009 (3 anni dopo jQuery), ha una certa somiglianza con il linguaggio Java. Di seguito una breve lista di noti siti web che sfruttano le potenzialità di questo framework: Lego, MSNBC, Tinder, GoPro, Ford, Renault, Amazon, Indiegogo e molti altri a seguire.

Per consultare la lista intera visitare madewithangular.com

Vantaggi

- Modularizzazione: si può organizzare logicamente un'applicazione in più parti, dove ogni parte è un modulo a sè stante e utilizzare questi moduli in più progetti o in punti diversi dell'applicazione;

- Two-way data binding: i dati inseriti da un utente e collegati al JS object model aggiornano quest'ultimo; vale anche il viceversa;

```
<label>Please input your name:</label>
<input type="text" ng-model="username"/>
<div>{{ 'Hello, ' + username}}</div>
```

- Espressioni AngularJS: consentono di manipolare i dati direttamente dal template, cioè dalla View. Ad esempio possono essere funzioni JS che restituiscono dei dati;

- Template: si può suddividere una pagina in più parti, conosciute come Direttive e utilizzarle laddove ce ne sia il bisogno. Tali Direttive possono essere dichiarate come attributi o elementi HTML;

- Validazione dei form: AngularJS incorpora la validazione dei form, anche estendibile. E' così possibile, con poche righe di codice, notificare l'utente di input non inseriti o non validi.

```
<form  ng-app="myApp"  ng-controller="validateCtrl"
name="myForm" novalidate>

<p>Username:<br>
 <input type="text" name="user" ng-model="user" required>
 <span style="color:red" ng-show="myForm.user.$dirty && myForm.user.
$invalid">
 <span ng-show="myForm.user.$error.required">Username is required.</
span>
 </span>
</p>

<p>Email:<br>
 <input type="email" name="email" ng-model="email" required>
 <span style="color:red" ng-show="myForm.email.$dirty && myForm.email.
$invalid">
 <span ng-show="myForm.email.$error.required">Email is required.</span>
 <span ng-show="myForm.email.$error.email">Invalid email address.</span>
 </span>
</p>

<p>
 <input type="submit"
 ng-disabled="myForm.user.$dirty && myForm.user.$invalid ||
 myForm.email.$dirty && myForm.email.$invalid">
</p>

</form>
```

- SPA;

- Filtri: permettono di di filtrare i dati da mostrare, e ciò può anche avvenire nel View layer. Potrebbe essere una selezione o una esclusione di dati, un filtro sulla lunghezza in caratteri dei dati o una trasformazione in caratteri maiuscoli. Ad ogni modo, oltre ai filtri già disponibili, è comunque possibile definire un filtro ad hoc;

- Dependency Injection: AngularJS gestisce le dipendenze delle parti di una applicazione attraverso una "inclusione" operata da un "injector". Si possono così includere in Moduli, Servizi, Controller, etc. tutte le dipendenze necessarie (Moduli, Servizi, etc.);

- Interceptor: questo strumento permette compiere operazioni di pre-processing e post-processing per chiamate e risposte http, sia sincrone che asincrone, anche in caso di errori e fase di autenticazione;

- Supporto integrato per AJAX;

- Debugging attraverso gli Unit Test.

- Dirty Checking: questo processo, agendo automaticamente ed essendo ottimizzato, permette allo sviluppatore di concentrarsi sulle funzionalità dell'applicazione. Qualora una proprietà di un oggetto "bindato" cambi, allora il framework si prenderà in carico il compito di aggiornare tutto ciò ad esso collegato.

- Supportato da Google

Svantaggi

- Il processo di Two-way data-binding è implementato attraverso il "digest cycle", il quale monitora tutti gli oggetti "bindati" (collegati) e periodicamente esegue gli aggiornamenti.

  Nel caso di una grande quantità di dati, la probabilità che debba essere fatto un aggiornamento aumenta notevolmente: gli oggetti collegati verranno controllati anche nel caso di minimo cambiamento.

- Tutto ciò che viene fatto all'interno dell'applicazione AngularJS deve essere compiuto da AngularJS. Questo significa che usando componenti di terze parti come jQuery e cambiando, tramite tale libreria, il valore di una variabile "bindata", allora non avverrà alcun Two-way data binding: la contro parte (nel Model o nella Miew) non verrà aggiornata. In questo caso bisogna provvedere a lanciare manualmente un'esecuzione del "Digest Cycle".

- Una parte rilevante nello sviluppo di applicazioni AngularJS deriva dalla difficoltà di debugging, per certi aspetti. Scrivere erroneamente il nome di una Direttiva o dell'applicazione AngularJS o del controller; chiamare una funzione non esistente nello `$scope` di AngularJS; le eccezioni non sollevate dalle espressioni AngularJS: tutto ciò non si risolve in alcun messaggio di errore.

- AngularJS è spesso criticato per la complessità della API delle Direttive. La gerarchia dello `$scope` usa l'ereditarietà prototipale di JavaScript: i diversi tipi di `$scope` (transclusion/ isolate/ child) e tutte le altre configurazioni per le Direttive richiedono non poco tempo per essere capite e utilizzate nel modo appropriato.

- le Espressioni AngularJS, utilizzate nella View, possono portare lo sviluppatore a definire una parte di business logic nella View, rendendo così il codice poco leggibile e complesso da testare;

```
<button ng-click="(oldPassword && checkComplexity(newPassword) &&
oldPassword != newPassword) ? (changePassword(oldPassword, newPassword)
&& (oldPassword=(newPassword='''))) : (errorMessage='Inserisci una nuova
password con i seguenti requisiti: ' + passwordRequirements)">
Cliccami
</button>
```

- bisogna ricordarsi di chiamare il metodo `$digest()` quando si compiono delle modifiche in un contesto diverso da quello AngularJS: in questo modo viene avviato il "Digest Cycle", che si preoccupa della fase di "Dirty Checking" e aggiornamento dei dati.
  Inoltre è necessario porre attenzione nel non causare fasi lente di "Dirty Checking" e infiniti di "Digest Cycle".
  In generale, per pagine web con molti elementi "bindati", AngularJS diventa molto lento. Tutt'ora la regola da seguire è di non eccedere il numero di 2,000 "data-binding" per View;

- una curva di apprendimento abbastanza ripida;

- la mancanza di server-side rendering è comune quasi a tutti i client-side framework. E ci si potrebbe chiedere perché volere una tale funzionalità se avendo scelto un framework lato client. Il motivo è la SEO (Search Engine Optimization).

  Allo stato attuale i motori di ricerca non sono capaci di analizzare una pagina web dinamica: ciò che vedono è solo un template e i riferimenti a certe variabili; succede quindi che viene indicizzata tale pagina, vuota, senza contenuto. Esistono dei modi per ovviare a questo problema, ma occorre analizzarli e studiarli.

2.2.2 Backbone

Rilasciato nel 2010, è utilizzato anch'esso in molti siti web:Twitter, Hulu, Pinterest, Sony Entertainment Network, LinkedIn, Foursquare, Airbnb, Stripe, Trello e molti altri.

Per consultare la lista intera visitare Projects and Companies using Backbone

Vantaggi

- piccolo in dimensioni (KB), è veloce;
- curva di apprendimento lineare (il sorgente può essere letto in un'ora);
- buona documentazione, il codice è di facile comprensibilità;
- sulla base di Backbone si possono costruire altri framework, ad esempio: Backbone UI, Chaplin, Geppetto, Marionette, LayoutManager, Thorax, Vertebrae;
- si può scegliere di usare un framework di terze parti;
- non ha un proprio template engine (a parte quello incluso in Underscore), lasciando agli sviluppatori la scelta di utilizzare ciò che preferiscono;

- date le sue piccole dimensioni (KB), Backbone funziona al meglio quando usato in semplici progetti dove la velocità è una priorità assoluta, come nelle SPA di Twitter e Pinterest.

Svantaggi

- Backbone non fornisce alcuna struttura alle applicazioni. Esso, invece, fornisce gli strumenti per creare tale struttura, lasciando la scelta allo sviluppatore. Ci sono molte scelte da fare utilizzando Backbone, infatti molte funzionalità possono essere implementate attraverso plugins alternativi: capire quale di questi si addice più alla nostra applicazione, richiede una ricerca e ciò prende tempo. Ricordiamo che uno dei vantaggi principali nell'utilizzo di un framework è proprio quello di risparmiare tempo nella creazione dello scheletro di una applicazione;

- è necessario porre attenzione alla gestione della memoria. La gestione della "View lifecycle" lascia i cambiamenti di rotta/ stato incline a memory leaks, a meno che lo sviluppatore non si prenda in carico di rimuovere i collegamenti non necessari;

- Backbone è più adatto per sviluppatori JS con esperienza: il suo minimalismo è sia un punto di forza che a sfavore, dipende dalle capacità dello sviluppatore. Lo stesso discorso vale per il il fatto che si debbano combinare tra loro molte librerie e plugins tra una miriade di varietà;

- non viene supportato il "Two-way data binding";

- la View in Backbone manipola direttamente il DOM: è complicato eseguire su di essa un unit-tes e quindi risulta più fragile e meno riusabile. La pratica comune consiste nel navigare il DOM attraverso i selettori CSS: cambiando il nome a una classe CSS rende l'applicazione inusabile;

- la mancanza di server-side rendering.

2.2.3 Ember

Rilasciato nel 2011, come AngularJS somiglia al linguaggio Java ma ancora di più al framework Ruby on Rails.

Alcuni siti web costruiti con Ember: Qualcomm, Nest Store, Vine, Twitch, PlayStation Now, NBC News, Apple Music e molti altri.

Per consultare la lista intera visitare builtwithember.io

Vantaggi

- al contrario degli altri 2 framework, Ember ha un data layer ben sviluppato che si integra perfettamente lato backend con Ruby on Rails o una JSON API di tipo RESTful.

- La performance è stata un obiettivo principale nella progettazione di Ember. Il "Run Loop" (equivalente del Digest Cycle) assicura che il DOM venga aggiornato una sola volta nel caso una variabile cambi più volte in un breve periodo; le proprietà calcolate vengono messe in cache; la pre-compilazione del template (HandleBars), in fase di build o lato server. Tutto questo aiuta il caricamento e l'esecuzione dell'applicazione;

- Richiede tempi di configurazione minimi. Ciò che è necessario per lo sviluppo di un'applicazione è integrato nel framework: template, routing, etc.

Svantaggi

- la più grande preoccupazione è il fatto che le cose debbano essere fatte alla "maniera Ember". Questo aspetto è presente anche in AngularJS, ma in Ember è ancora più radicato. Ember aspira a fornire una soluzione completa;

- è difficile integrare librerie/ plugins di terze parti a causa dell'elevata specificità: tutto deve essere fatto alla "maniera Ember";

- il numero di strumenti di testing è limitato;

- ha inizialemente una curva di apprendimento ripida: il framework mira a fornire una soluzione completa, di conseguenza le funzionalità integrate sono da capire quando e come utilizzare, e sono molte;

- API instabile, poichè molto è cambiato prima che entrasse in versione stabile. C'è molta documentazione in rete che fa riferimento a codice non più funzionante, rendendo difficile l'apprendimento per i principianti;

- La libreria Handlebars relativa al template "sporca" il DOM con una gran numero di tag `<script>`, usati per tenere aggiornata la View con il Model;

  Ciò porta a una minore leggibilità del codice e può inoltre compromettere l'integrazione con altri framework, quale jQuery UI Sortable.

Capitolo 3

Conoscere AngularJS

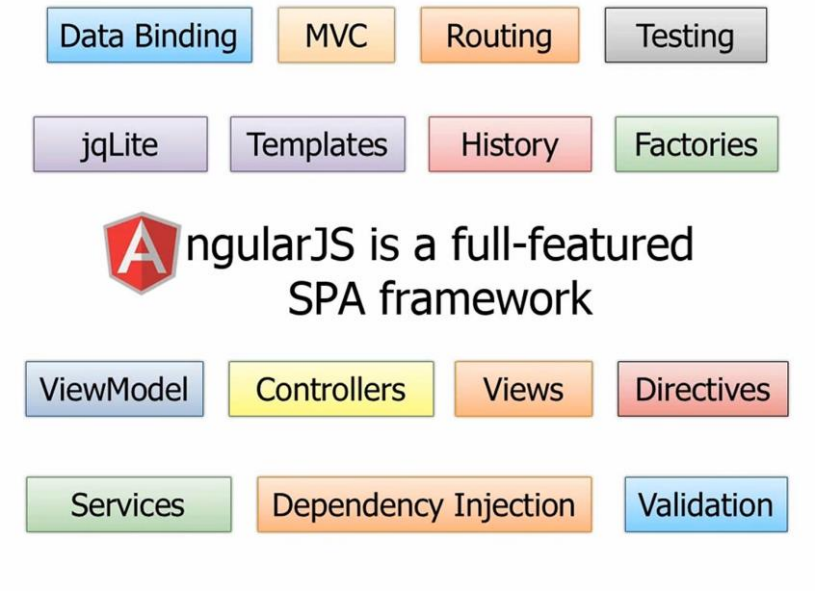
AngularJS 1.2 dichiara di supportare i seguenti browser: Chrome, Firefox, Safari, iOS, Android, IE8+, mentre dalla versione 1.3 decade il supporto per IE8.

AngularJS divide ogni applicazione in più parti: Controller, Direttive, Filtri, Servizi e View (template).
Questi sono organizzati in moduli, i quali possono dipendere l'uno dall'altro.

La View costiuisce la UI, i Controller fanno funzionare la logica che sta dietro alla UI, i Servizi rappresentano lo strumento con cui condividere dati tra le varie parti dell'applicazione e sono la parte di codice riusabile in unapplicazione AngularJS; le Direttive permettono di creare componenti riusabili ed estendono l'HTML attraverso la definizione di nuovi elementi e attributi.

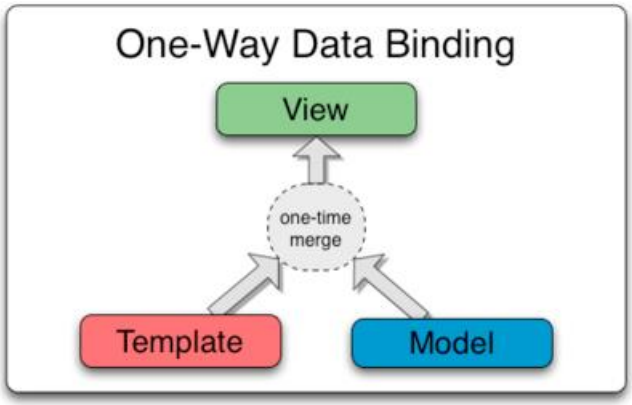
Concetto	*Descrizione*
Template	HTML con markup aggiuntivo.
Direttiva	Estende l'HTML tramite elementi e attributi creati ad hoc.
Model	Tutti i dati, o una parte, che vengono mostrati tramite la View.
Scope	Il contesto in cui viene definite il modello dei dati che deve essere mostrato nella View. Ciò che è nello `$scope` viene visto da tutti gli altri componenti dell'applicazione.
Espressione	Variabile o funzione dello `$scope`: interpolata a livello di View tramite {{variabile}} o utilizzata come attributo di una Direttiva.
Compilatore	Analizza il template e istanzia Direttive ed Epressioni.
Filtro	Esegue formattazioni o selezioni sui dati in fase di visualizzazione.
View	Ciò che l'utente vede (DOM).

Data Binding	Sincronizzazione dei cambiamenti tra Model e View.
Controller	Permette di legare la logica della UI alle funzioni corrispondenti (View Logic) e di gestire la Business Logic. Funge da contenitore dello `$scope`.
Dependency Injection	Gestisce la risoluzione delle dipendenze.
Injector	Si occupa della Dipendency Injection.
Module	Un contenitore per parti differenti dell'applicazione.
Service	La parte riusabile di codice.



3.1 Data Binding

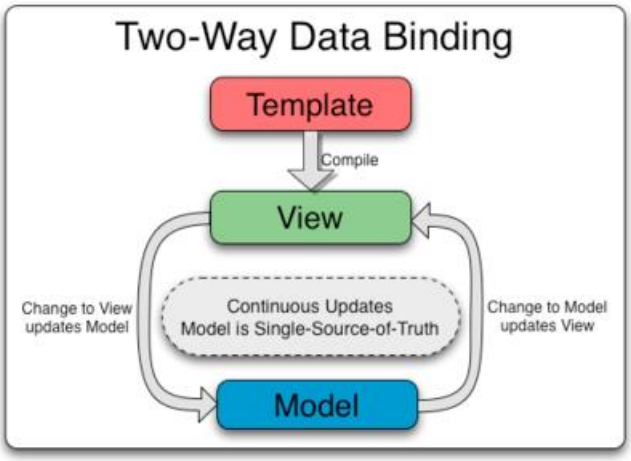
Metodo Ordinario



Successivamente alla fase di fusione tra Model e Template, i cambiamenti non vengono sincronizzati.

Ogni cambiamento che avviene nel Model non viene riflesso nella View, e viceversa.

Metodo Angular (default)



Dapprima il template è compilato: questo processo si conclude con una Live View.

Tutti i cambiamenti che avvengono nel Model vengono riflessi nella View; è vero anche il contrario. La View non è che una proiezione del Model.

3.2 Dove inserire lo script AngularJS

Sebbene sia consigliato di porre gli scripts, nelle applicazioni web, alla fine dell'elemento `<body>` (per non bloccare la renderizzazione della pagina web), è raccomandabile caricare il framework AngularJS al termine dell'elemento `<head>` o all'inizio dell'elemento `<body>`.

Questo perchè le chiamate ad `angular.module` possono essere compilate solo dopo aver caricato il framework.

3.3 Inizializzare un'applicazione AngularJS

Occorre utilizzare la direttiva `ng-app` per eseguire l'inizializzazione di una applicazione AngularJS. Questa direttiva designa l'elemento radice dell'applicazione.

```
<div ng-app="angularApp">
    <!-- applicazione AngularJS -->
</div>
```

Alcune considerazioni da tenere a mente per l'uso della direttiva `ng-app`:

- le applicazioni AngularJS non possono essere annidate l'una nell'altra;
- per eseguire più applicazioni AngularJS, entro la stessa pagina web, bisogna manualmente inizializzarle utilizzando `angular.bootstrap` .

3.4 Moduli

Si può pensare a un modulo come a un contenitore per le diverse parti dell'applicazione AngularJS: Controller, Servizi, Filtri, Direttive, etc.

La maggior parte delle applicazioni ha un metodo/ funzione principale che istanzia gli oggetti e collega le diverse parti dell'applicazione. AngularJS non ha questo metodo principale: i moduli dichiarano le loro dipendenze e come l'applicazione deve essere avviata.

Di seguito alcuni vantaggi derivanti da questo approccio:

- il processo dichiarativo di dipendenza è più intuitivo;
- si può organizare il codice in moduli riutilizzabili;
- gli Unit Test devono solo caricare i moduli necessari, il che rende la fase di testing più veloce.

Per creare, registrare e caricare in memoria i moduli (anche di terze parti) occorre utilizzare il metodo `angular.module`.

Metodo

```
angular.module(name, [requires], [configFn]);
```

Parametri

Parametro	Tipo	Dettagli
name	string	Il nome del modulo da creare o caricare nell'applicazione AngularJS.
requires *(opzionale)*	!Array.<string>=	Una lista di moduli (di tipo string) che l'injector carica in memoria prima che il modulo stesso, dichiarato con `name`, venga caricato.
configFn *(opzionale)*	Function=	Funzione di configurazione del modulo.

Ritorna

angular.Module un modulo con la API di `angular.module`.

```
// Crea un nuovo modulo, nessuna dipendenza
var newModule = angular.module('newModule', []);
```

```
// Crea un nuovo modulo, con una dipendenza
var mySecondModule = angular.module('mySecondModule', ['newModule']);
```

```
// Carica in memoria un modulo
var otherModule = angular.module('otherModule');
```

3.4.1 angular.Module API

Interfaccia per la configurazione dei moduli AngularJS.

Metodi

Metodo

`factory(name, providerFunction);`

Parametri

Parametro	Tipo	Dettagli
name	string	Nome del servizio
providerFunction	Function	Funzione per creare una nuova istanza del servizio

Metodo

```
service(name, constructor);
```

Parametri

Parametro	Tipo	Dettagli
name	string	Nome del servizio
constructor	Function	Funzione, costruttrice del servizio, che verrà istanziata

Metodo

```
filter(name, filterFactory);
```

Parametri

Parametro	Tipo	Dettagli
name	string	Nome del filtro
filterFactory	Function	Funzione di tipo Factory per creare una istanza del filtro

Metodo

```
controller(name, constructor);
```

Parametri

Parametro	Tipo	Dettagli
name	stringObject	Nome del Controller, o un oggetto che mappa i nomi dei Controller con le rispettive funzioni costruttrici
constructor	Function	Funzione costruttrice del Controller

Metodo

```
directive(name, directiveFactory);
```

Parametri

Parametro	Tipo	Dettagli
name	stringObject	Nome della Direttiva, o un oggetto che mappa i nomi delle Direttive con le rispettive funzioni costruttrici di tipo Factory
directiveFactory	Function	Funzione di tipo Factory per creare una instanza della Direttiva

Metodo

```
config(configFn);
```

Utilizzare questo metodo al fine di compiere operazioni prima del caricamento delle dipendenze del modulo.

Parametri

Parametro	Tipo	Dettagli
configFn	Function	La funzione eseguita prima del caricamento delle dipendenze modulo.

Metodo

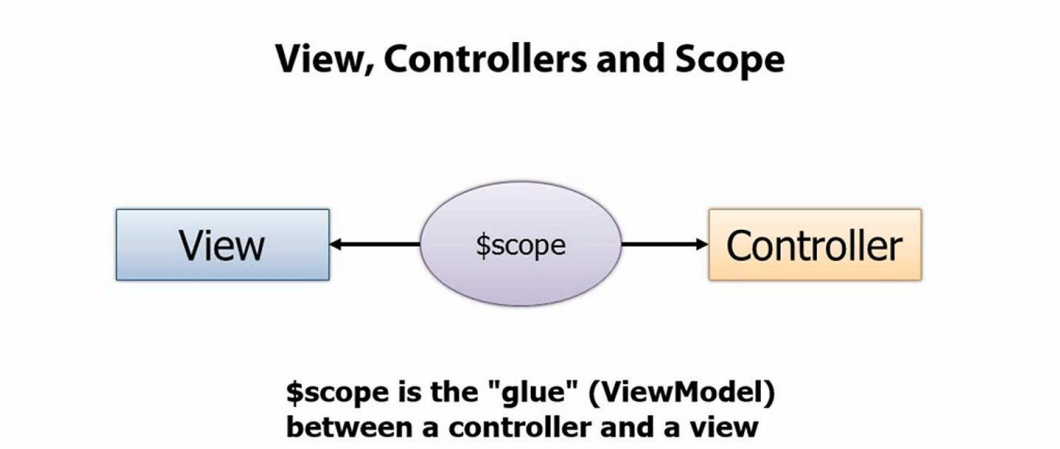
```
run(initializationFn);
```

Utilizzare questo metodo al fine di compiere operazioni successivamente al caricamento delle dipendenze del modulo.

Parametri

Parametro	Tipo	Dettagli
initializationFn	Function	La funzione eseguita dopo il caricamento delle dipendenze del modulo. Utile per inizializzare i dati dell'applicazione.

3.5 AngularJS Scope



3.5.1 $scope come View-Model

Lo `$scope` di un'applicazione AngularJS può essere interpretato come il Model visibile dalla View, infatti possiamo utilizzare nella View le proprietà definite nello `$scope`; non solo, è anche l'oggetto in cui viene definita la logica che agisce a livello di View, quando utilizziamo nella View le funzioni definite nello `$scope`.

Lo `$scope`, inoltre, è il contesto di esecuzione delle espressioni AngularJS.

La definizione di proprietà e funzioni utilizzabili nella View avviene nello `$scope`, definito all'interno di un Controller; può essere anche interpretato, quindi, come il legame che unisce il Controller alla View.

L'oggetto `$scope` è di tipo POJO (Plain Old JavaScript Object): possiamo aggiungere e cambiarne le proprietà liberamente.

Lo `$scope` ha le seguenti funzioni:

- resta in osservazione dei cambiamenti del Model;
- propaga i cambiamenti del Model a tutta l'applicazione;
- possono essere annidati, cosicchè possano venire isolate funzionalità e proprietà del Model;
- rappresenta il contesto in cui le espressioni AngularJS vengono valutate.

3.5.2 Gerarchia dello $scope

Ogni applicazione AngularJS ha esattamente uno `$scope` radice dell'applicazione, chiamato `$rootScope`, ma è possibile creare oggetti `$scope` figli di altri oggetti `$scope/ $rootScope` .
La creazione di oggetti `$scope`, diversi da `$rootScope`, deriva dall'uso delle Direttive.

Quando vengono creati nuovi oggetti `$scope`, questi assumono il ruolo di figli nella gerarchia degli `$scope`: questo meccanismo crea una struttura parallela a quella del DOM, ai cui elementi sono collegati.

Quando AngularJS valuta l'espressione `{{name}}`, dapprima compie una ricerca per la proprietà `name` nello `$scope` corrente. Se la ricerca per tale proprietà non dà esito positivo, allora la ricerca procede nello `$scope` padre, e così fino a `$rootScope` finchè non viene trovata tale proprietà.

In JavaScript questo comportamento è conosciuto come Ereditarietà Prototipale, ovvero gli oggetti `$scope` figli ereditano le proprietà dall'oggetto `$scope` padre.

3.5.3 Il ciclo di vita dello $scope

Creazione dello $scope

Il `$rootScope` viene creato durante la fase di avvio dell'applicazione per mezzo dell'oggetto `$injector`. Quando si crea un Controller o una Direttiva, AngularJS tramite `$injector` crea un nuovo oggetto `$scope` figlio e passa questo nuovo `$scope` al Controller o la Direttiva creata; tutto questo avviene a livello di esecuzione.

Registrazione di Watcher

Quando lo `$scope` viene collegato alla View, tutte le Direttive che creano uno `$scope` registrano i loro Watcher sullo `$scope` padre. Questi Watcher notano i cambiamenti che avvengono nelle proprietà dello `$scope` e propagano questi cambiamenti dalla View alle Direttive.

Cambiamento del Model

Affichè vengano notati i cambiamenti, bisognerebbe compiere le modifiche alle proprietà dello `$scope` solo all'interno del metodo `$scope.$apply()` . Tuttavia la API di AngularJS svolge questo compito implicitamente, in caso di operazioni sincrone e asincrone compiute all'interno del contesto AngularJS.

Osservazione dei cambiamenti

Al termine dell'esecuzione di un `$apply`, AngularJS compie un `$digest` cycle partendo dal `$rootScope`, propagandosi poi verso tutti gli oggetti `$scope` figli. Durante questa fase di `$digest` cycle, tutte le espressioni AngularJS a cui è stato assegnato uno Watcher vengono controllate (`$watch`) e in caso di cambiamento il `$watch` listener viene chiamato.

Distruzione di uno $scope

Quando gli `$scope` figli non sono più utilizzabili, è compito dello `$scope` padre distruggerli attraverso il metodo `$scope.$destroy()`. Questo permette al garbage collector di recuperare la memoria utilizzata da tale `$scope` figlio.

3.5.4 Direttive che creano $scope

Precedentemente è stato citato che le Direttive creano uno `$scope`: non è del tutto vero. In genere non ne creano uno proprio, ma ciò dipende dal tipo di Direttiva: `ng-controller` ed `ng-repeat` ad esempio creano uno `$scope` figlio e lo associano all'elemento del DOM in cui sono state dichiarate.

3.5.5 Direttive e $scope

Durante la fase di compilazione, il compilatore combina le Direttive con le rispettive controparti nel template. Le Direttive possono essere di due tipi:

- Direttive che stanno in osservazione, come l'espressione AngularJS `{{expression}}`. Registrano un oggetto che sta in osservazione tramite il metodo `$watch()`. Questo tipo di Direttiva deve essere notificata quando l'espressione AngularJS cambia, così che possa aggiornare il valore nella View.

- Direttive che rimangono in ascolto, come `ng-click`. Registrano un oggetto che sta in ascolto sull'elemento del DOM e quando tale oggetto notifica la Direttiva, questa esegue l'espressione AngularJS associata e aggiorna la View tramite il metodo `$apply()`.

Quando accade un evento o un'operazione esterna al contesto di AngularJS che cambia una proprietà dello `$scope`, AngularJS non si accorge di alcuna modifica. Per ovviare a tale inconveniente, occorre che la modifica venga eseguita all'interno del metodo `$scope.$apply()`.

3.6 Dirty Checking e $watch

Il meccanismo di Dirty Checking (controllo delle modifiche) può essere effettuato secondo 3 diverse strategie: per riferimento, per valore di una collezione di oggetti, e per valore.

Le 3 strategie differiscono per il modo in cui le modifiche vengono individuate e nelle performance.

- Compiere `$watch` per riferimento, tramite `($scope.$watch(watchExpression, listener)`), permette di individuare le modifiche a una variabile di tipo semplice: non vengono individuati cambiamenti se viene modificato uno degli elementi di un vettore o le proprietà di un oggetto.

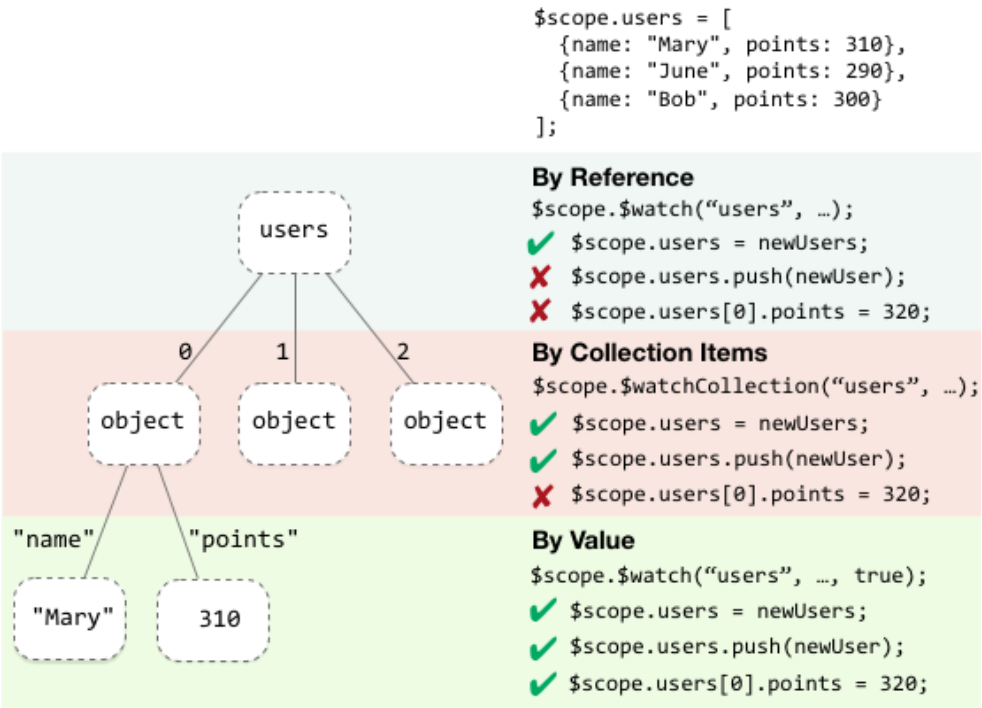
  A livello di performance è la strategia più efficiente.

- Compiere `$watch` sul valore di una collezione di dati, tramite (`$scope.$watchCollection(watchExpression, listener)`), permette di individuare anche le modifiche ai valori degli elementi di un vettore e alle proprietà di un oggetto. Ciò include l'inserimento, l'eliminazione di nuovi elementi e il riordinamento.

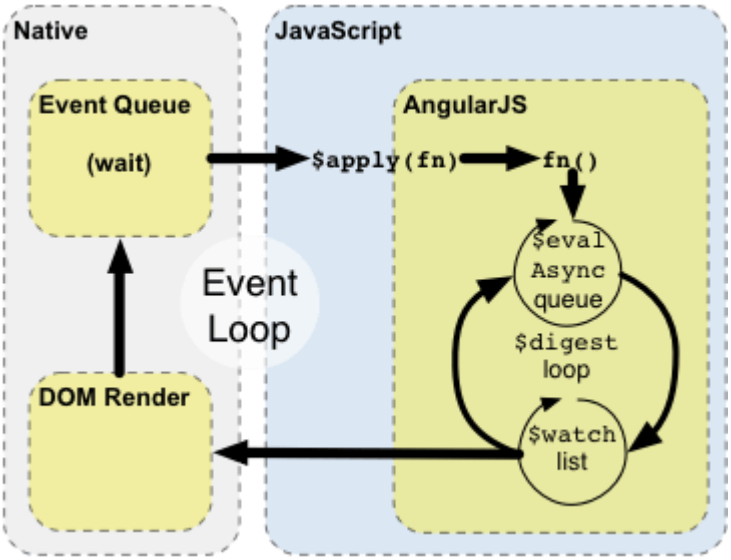
  Questa strategia tuttavia non permette di individuare modifiche in altri collezioni di dati annidati in questo vettore/ oggetto. A livello di performance è più onerosa, poichè occorre mantenere in memoria le copie dei valori della collezione di oggetti.

- Compiere `$watch` per valore, tramite (`$scope.$watch(watchExpression, listener, true)`), permette di individuare tutti i cambiamenti di una struttura dati, indipendentemente dal livello in cui è annidata.

  E' la strategia più potente ma anche la peggiore a livello di performance. Devono essere mantenute in memoria le copie di tutte le strutture dati annidate, e, ad ogni `$digest` cycle, l'intera struttura dati deve essere attraversata e controllata.



3.7 Integrazione con il ciclo di eventi del browser, $digest Loop e $apply



1. Il browser esegue il ciclo degli eventi: rimane in attesa del verificarsi di un evento.
   Un evento può essere un'interazione con la UI, l'esecuzione della funzione di un timer, o un evento proveniente dalla rete (una risposta asincrona ad esempio).

2. La funzione associata all'evento viene eseguita, si entra nel contesto JS: può succedere che in questa fase vengano apportare modifiche alla struttura del DOM.

3. Una volta che la funzione è stata eseguita, il browser esce dal contesto JS e renderizza nuovamente la pagina web, basandosi sulle modifiche apportate al DOM.

4. Il browser torna al punto 1 .

Quando il browser esegue una funzione JS che fa parte del contesto di esecuzione di AngularJS, ovvero il `$digest` loop, allora lo `$scope` riflette i cambiamenti delle modifiche avvenute.

Se la funzione JS viene eseguita fuori dal contesto AngularJS, ad esempio quando si tratta di una funzione di una libreria di terze parti, allora il framework non è

consapevole delle modifiche compiute allo `$scope` da tale funzione.

Per ovviare a tale inconveniente, occorre che tale funzione (o le modifiche allo `$scope`) venga eseguita all'interno del metodo `$scope.$apply()`.

Per capire meglio il contesto di esecuzione di AngularJS, il `$digest` loop, diamo un'occhiata alle sue parti costituenti:

- `$watch` list
- `$evalAsync` list

3.7.1 $watch List

Per tracciare il verificarsi di un evento nella View, bisogna registrare la rispettiva funzione, eseguita in caso di tal evento.

```
<body>
        <input ng-model="name" type="text" placeholder="Your name">
        <h1>Hello {{ name }}</h1>
</body>
```

Ogniqualvolta l'utente aggiorna il campo dell'elemento `<input>`, `{{ name }}` si aggiorna: i cambiamenti vengono riflessi poichè abbiamo collegato il campo dell'elemento `<input>` con la proprietà `$scope.name`.

Per aggiornare la View, AngularJS deve tracciare il cambiamento: a questo fine, esso aggiunge una funzione `$watch` alla `$watch` list, per tutto ciò che è presente nella View e fa riferimento a una proprietà dello `$scope`.

Queste `$watch` list vengono controllate nella fase di `$digest` loop, attraverso il processo di Dirty Checking.

3.7.2 $evalAsync() List

La seconda operazione compiuta dal `$digest` loop è l'esecuzione della `$$asyncQueue`.

Il metodo `$evalAsync()` è un modo per programmare, nel futuro prossimo, la risoluzione delle espressioni AngularJS dello `$scope` corrente.

Durante il `$digest` loop, `$$asyncQueue` viene vuotata ad ogni ciclo attraverso il meccanismo Dirty Checking, il che ci porta a due conclusioni per ogni funzione chiamata tramite `$evalAsync`:

- La funzione verrà eseguita successivamente alla funzione che l'ha programmata.
- Passerà almeno un `$digest` cycle prima che l'espressione AngularJS venga valuatata.

3.8 Dirty Checking

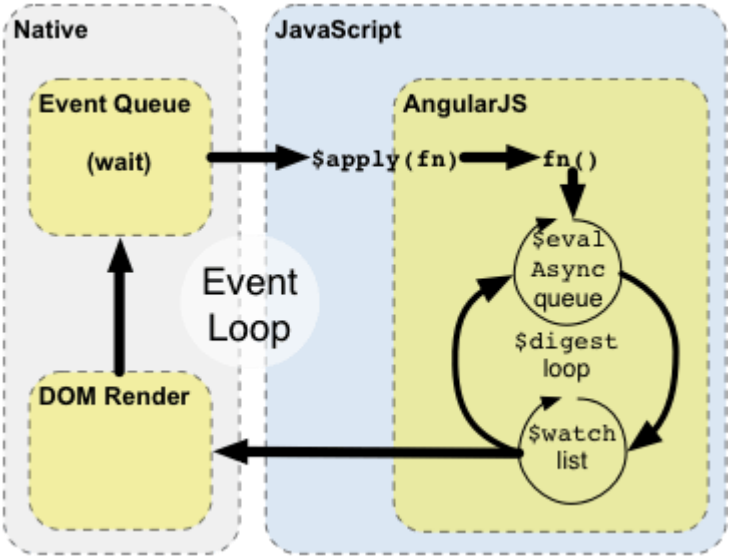
Il meccanismo di Dirty Checking è un meccanismo alla base di un semplice concetto: controlla se un valore che ha subito modifiche è stato sincronizzato in tutta l'applicazione AngularJS.

Viene analizzata la `$watch` list durante il `$watch` loop, e per ogni `$watch`, se sono state apportate modifiche viene registrato il nuovo valore.

Una volta analizzata la `$watch` list, viene rieseguito `$watch` loop, fintanto che ci sono modifiche da sincronizzare.
Ci si potrebbe chiedere il perchè di questi `$watch` loop ulteriori al termine di una analisi della `$watch` list.
Se venisse aggiornato un valore della `$watch` list, il quale ne aggiorna un altro della `$watch` list, AngularJS non individuerebbe quest'ultima modifica, a meno che non venga eseguito un altro `$watch` loop.

Nota: Se il `$watch` loop viene eseguito 10 o più volte, AngularJS solleva un'eccezione e l'applicazione termina.
Nel caso non venisse lanciata alcuna eccezione, l'applicazione potrebbe finire in un infinito `$watch` loop.

3.9 Integrazione con il ciclo di eventi del browser e AngularJS



1. Si entra nell'ambiene di esecuzione di AngularJS tramite `$scope.$apply(fn)`, dove `fn` è la funzione che si vuole svolgere nel contesto di AngularJS.

2. AngularJS esegue la funzione `fn()`.

3. AngularJS esegue il `$digest` loop, costituito di 2 cicli che processano rispettivamente su `$evalAsync` queue e `$watch` list. Se viene notato un cambiamento allora viene chiamata una funzione `$watch` che sincronizza il DOM con il nuovo valore.

   Il `$digest` loop continua la sua esecuzione fintanto che ci sono modifiche da sincronizzare, ovvero `$evalAsync` queue non diventa vuota e/o il `$watch` loop individua delle modifiche.

4. `$evalAsync` queue consente di programmare operazioni che non devono avvenire subito, ma prima che venga renderizzata la pagina web.

5. Una volta che il `$digest` loop finisce l'esecuzione, viene abbandonato il contesto AngularJS e JS. Il browser renderizza nuovamente il DOM.

3.10 AngularJS Controller

Il Controller è definito da una funzione JS e viene utilizzato per contenere e definire/ cambiare lo `$scope` di AngularJS.
Quando un Controller viene collegato al DOM, attraverso la Direttiva `$ng-controller`, AngularJS instanzia un nuovo oggetto Controller, utilizzando dietro le quinte la funzione per la creazione di un Controller. Verrà creato un nuovo oggetto `$scope` figlio, disponibile come parametro di nome `$scope` nella funzione di creazione del Controller.

```
var myApp = angular.module('myApp',[]);


myApp.controller('myController', function($scope) {
  $scope.saluto = 'ciao!';
});
```

Dapprima creiamo un modulo, `myApp`, per l'applicazione: successivamente aggiungiamo a tale modulo un Controller, utilizzando il metodo `.controller()` per la creazione di un Controller.

```
<div ng-controller="myController">
  {{saluto}}
</div>
```

Abbiamo collegato il Controller appena creato, `myController`, all'elemento `<div>` del DOM atraverso la Direttiva `ng-controller`. La proprietà `saluto` definita nello `$scope` può ora essere collegata alla View (template).

Per reagire al verificarsi di eventi, o eseguire operazioni più complesse nella View, dobbiamo definire nell'oggetto `$scope` i metodi che ci forniranno tali funzionalità. Questi metodi saranno così invocabili dalla View.

L'utilizzo di un Controller deve essere limitato a:

- definire lo stato iniziale dell'oggetto `$scope`;
- definire il comportamento dell'oggetto `$scope`, attraverso la definizione di metodi.

Lo stato di uno `$scope` è definito dalle sue proprietà; tali proprietà definiscono il View-Model (il Model visibile dalla View).

3.10.1 Gerarchia dei Controller

E' possibile avere più di un Controller in una applicazione AngularJS, purchè essi siano collegati a livelli differenti del DOM. Dal momento che la Direttiva `ng-controller` crea un nuovo `$scope` figlio, otteniamo un gerarchia di `$scope` (utilizzando più Controller ad esempio) in cui ogni `$scope` figlio eredita le proprietà e i metodi dello `$scope` padre.

3.11 Espressioni AngularJS

Le espressioni AngularJS sono frammenti di codice JS posti tra doppie parentesi graffe, come in `<span titolo="{{ attrBinding }}">{{ textBinding }}</span>`, o come attributo di una Direttiva, ad esempio `ng-click="functionExpression()"`.

Di seguito una breve lista di espressioni AngularJS:

- 1+2
- a+b
- user.name
- items[index]

Le espressioni AngularJS hanno le seguenti caratteristiche:

- sono eseguite all'interno di uno `$scope` e pertanto possono accedere alle sue variaibli;
- non sollevano eccezioni in caso di TypeError o ReferenceError;

- non permettono di eseguire istruzioni condizionali (`if/else`);
- non accettano filtri.

3.11.1 One-time Binding

Un'espressione AngularJS che comincia con il simbolo `::` subisce l'effetto del one-time binding: ciò siginfica che non verrà più calcolata dopo la prima volta, ossia dopo dopo il primo `$digest` loop, se l'espressione si risolve in un valore definito.

Lo scopo di questo meccanismo è quello di fornire un modo per far fronte all'utilizzo di dati statici nella View; di conseguenza vengono de-registrati gli `$watch` relativi a tali espressioni e vengono liberate le rispettive risorse in memoria.

Si ricorda che il numero delle espressioni AngularJS da controllare incide sul `$watch` loop e di conseguenza sul `$digest` loop: ridurre tale numero permette un `$digest` loop più veloce e di mostrare un numero maggiore di informazioni (dinamiche Vs dinamiche e statiche, ricordando il limite di 2,000 data-bindings).

3.13 Direttive

Una Direttiva può essere spiegata come una funzione eseguita su un particolare elemento del DOM.

A livello di View, le Direttive vengono dichiarate presso un elemento del DOM (come attributo, elemento, commento o classe CSS): questo permette al compilatore HTML di AngularJS (`$compile`) di collegare a tale elemento la funzione definita dalla Direttiva, o anche di trasformare ed estendere tale elemento del DOM.

Abbiamo visto quindi che c'è una fase di compilazione di AngularJS, in cui le Direttive vengano combinate all'HTML: per questo la View viene chiamata template. Questa compilazione avviene durante la fase di inizializzazione dell'applicazione AngularJS.

Anche in questo caso si possono creare Direttive ad hoc.

3.13.1 Direttive integrate in AngularJS

Di seguito sono elencate alcune Direttive di uso più frequente, già integrate nel framework.

Direttiva	Crea `$scope` figlio	Utilizzo
`ng-app`	✓	Necessaria per inizializzare il framework AngularJS: l'elemento del DOM, associato alla Direttiva, rappresenta l'inizio del `$rootScope`.
`ng-controller`	✓	Utilizzata per collegare un Controller alla View, associando la Direttiva a un elemento del DOM.
`ng-include`	✓	Utilizzata per caricare, compilare, e includere una parte di HTML (anche da file esterno) internamente all'elemento del DOM a cui è collegata la Direttiva.
`ng-if`	✓	Valuta l'espressione AngularJS ad essa associata: se valutata `false`, allora l'elemento del DOM a cui è associata la Direttiva viene eliminato dal DOM (compresi gli elementi discendenti).
`ng-show`	✓	Valuta l'espressione AngularJS ad essa associata: se valutata `false`, allora l'elemento del DOM a cui è associata viene nascosto tramite la proprietà CSS `display: none !important`
`ng-hide`	✓	Come `ng-show`, ma a parti invertite: se valutata `true`, allora l'elemento del DOM a cui è associata viene nascosto tramite la proprietà CSS `display: none`
`ng-repeat`	✓	Utilizzata per iterare su una collezione di oggetti dello `$scope`. Per ogni oggetto di tale collezione viene create un'istanza con uno `$scope` a sè stante.
`ng-init`	✓	Utilizzata per impostare lo stato iniziale dello `$scope` di una Direttiva, quando questa viene invocata.
`ng-href`		Permette ad AngularJS di attendere l'interpolazione con il dato dinamico; successivamente abilita il comportamento del link.

ng-src		Permette ad AngularJS di attendere l'interpolazione con il dato dinamico; successivamente carica l'immagine relativa a quel dato URL.
ng-bind	✓	Qunado viene usata la sintassi {{ }} relativa alle espressioni AngularJS, c'è un lasso di tempo, anche breve, in cui i dati dinamici sono in caricamento e non ancora renderizzati nella pagina web: come risultato, nel frattempo, le espressioni dinamiche sono visibili nella pagina web. Possiamo evitare questo comportamento utilizzando la Direttiva ng-bind, collegando una espressione AngularJS (senza {{ }}) come attributo della Direttiva.
ng-cloak	✓	Una alternativa ad ng-bind.
ng-model	✓	Utilizzata per collegare il campo di un element HTML di tipo input, select, textarea, o altri elementi di form, ad una proprietà dello $scope.
ng-change	✓	Utilizzata per valutare un'espressione AngularJS quando l'input, a un elemento di form, è cambiato. Deve essere utilizzata in combinazione con la Direttiva ng-model.
ng-bind-template	✓	Come ng-bind, ma per collegare più espressioni alla View.
ng-class	✓	Utilizzata per aggiungere, dinamicamente e condizionalmente, una o più classi CSS a un elemento HTML, attraverso un'espressione AngularJS che rappresenta tutte le classi da aggiungere.
ng-style		Utilizzata per aggiungere, dinamicamente e condizionalmente, lo stile CSS a un elemento HTML.
ng-click	✓	Utilizzata per eseguire un metodo dello $scope o un'espressione AngularJS, al verificarsi di un click sull'elemento HTML presso cui è definita.

3.13.2 Connettere le Direttive all'HTML

Un elemento HTML viene collegato a una Direttiva se quest'ultima è dichiarata insieme ad esso. AngularJS, in fase di compilazione del template, procede a "normalizzare" il nome e gli attributi degli elementi HTML, per determinare quali elementi HTML sono collegati a una Direttiva.

Il processo di "normalizzazione" prevede 2 operazioni:

- elimina i prefissi `x-` e `data-` dagli elementi e attributi;
- converte i nomi delimitati dai simboli `:`, `-`, or `_` in una sintassi camelCase. (ng-model -> ngModel).

3.13.3 Tipi di Direttive

Il servizio `$compile`, adibito alla compilazione di un template, può collegare le Direttive come elemento HTML, come attributo di un elemento HTML, come nome di una classe CSS o come commento HTML.

Ad esempio, questi di seguito, sono tutti modi equivalenti per collegare la Direttiva `my-directive` al template:

```
<my-directive></my-directive>
<div my-directive></div>
<div class="my-directive"></div>
<!-- directive: my-directive -->
```

3.13.4 Creare le Direttive

Per creare una nuova Direttiva, occorre usare l'API di `module.directive`, la quale prende in input il nome "normalizzato" della Direttiva seguito da una funzione di tipo Factory. Quest'ultima ritorna un oggetto con differenti proprietà, corrispondenti alle opzioni di una Direttiva: queste opzioni insegnano a `$compile` come definire il comportamento di una Direttiva quando viene collegata al DOM.

La funzione Factory viene invocata sono una volta, in fase di compilazione del template.

3.13.5 Opzioni di configurazione

La creazione di una nuova Direttiva, con alcune tra le opzioni di configurazione più utilizzate, appare come qui di seguito.

```
   angular.module('myApp', [])
      .directive('myDirective', function() {
            // in questo spazio si compiono operazioni di
            // inizializzazione di una Direttiva
            return {
                  // la definizione di una Direttiva consiste
                  // nella configurazione delle sue opzioni,
                  // che in questo sovrascriviamo
                  restrict: ‘E’,
                  priority : 10,
                  terminal : false,
                  template: '<div><h3>{{title}}</h3></div>',
                  templateUrl : 'myDirective.html',
                  replace : true,
                  scope : true,
                  controller : function ($scope, $element, $attrs)
                  {},
                  require : 'myDirective'
            };
});
```

Opzione di configurazione: Restrict

Quando si crea una Direttiva, essa è limitata (o che dir si voglia, può essere collegata) ad elementi HTML e attributi di elementi HTML. Per cambiare questo comportamento bisogna sovrascrivere l’opzione di configurazione `restrict`, con uno/ una combinazione dei seguenti valori.

- `'A'` – è limitata ad attributi di elementi HTML
- `'E'` – è limitata ad elementi HTML
- `'C'` – è limitata a nomi di classi CSS
- `'M'` – è limitata ai commenti

La combinazione di tutti i valori è: `'AECM'`.

Opzione di configurazione: Priority

Permette di specificare in quale ordine le Direttive debbano essere eseguite, nel caso siano presenti più Direttive presso lo stesso elemento HTML. Il valore di default è 0: le Direttive con valori più alti vengono eseguite prima.

Opzione di configurazione: Terminal

Stabilisce se l'esecuzione della Direttiva sia limitata entro il valore dell'opzione di configurazione `restrict`. Siccome non è definito l'ordine di esecuzione delle Direttive aventi lo stesso valore `restrict`, l'esecuzione della Direttiva continua con `restrict: 'yes'` fintanto che non viene raggiunto un valore successivo a quello di `priority`; ovvero finchè non sono state eseguite tutte le Direttive per quel valore di `priority` .

Opzione di configurazione: Template, TemplateUrl

E' possibile definire parte del template (direttive ed espressioni AngularJS) o semplice HTML come valore dell'opzione `template` o definirli in un file esterno e utilizzare l'URL di tale file come valore per l'opzione `templateUrl` .
Nel momento in cui avviene la compilazione del template, viene recuperata la struttura HTML definita da questa opzione e posta come elemento HTML discendente della Direttiva, nel file template in cui quest'ultima è dichiarata.

Opzione di configurazione: Replace (Deprecato)

Utilizzata per sostituire l'elemento in cui è dichiarata la Direttiva, con la struttura HTML definita da `template` o `templateUrl`.

Opzione di configurazione: Scope

scope: false

E' il valore di default. La Direttiva non crea alcuno `$scope` figlio, ma invece condivide quello dell'elemento HTML padre.

scope: true

La Direttiva crea un nuovo `$scope` figlio, che eredita prototipalmente dallo `$scope` padre.

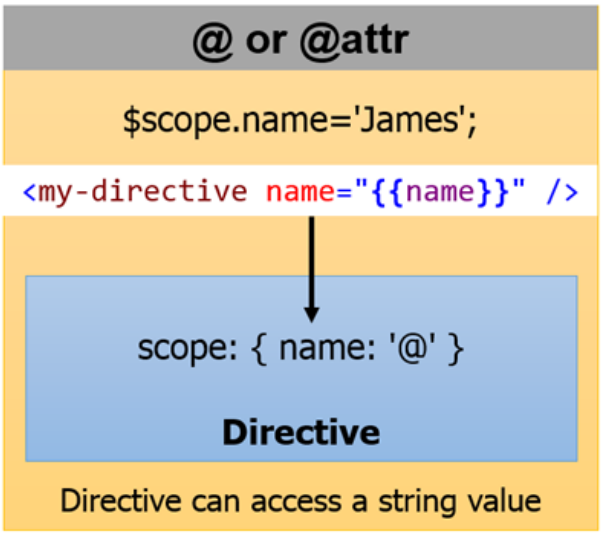
scope: { }

La Direttiva crea uno `$scope` isolato, che non eredita alcunchè dallo `$scope` padre, ma al quale si può accedere tramite `scope.$parent`.

Lo `$scope` isolato prende in input un oggetto, il quale permette di collegare le proprietà dello `$scope` padre allo `$scope` locale, che è isolato. Esistono 3 modi per collegare queste proprietà:

- '@'

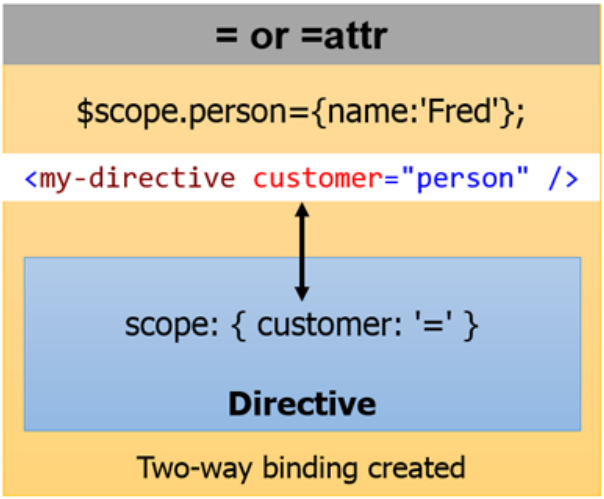
  Utilizzata per accedere a valori di stringhe definite esternamente alla Direttiva.



  Copia il valore della proprietà (`name`) dello `$scope` padre presso lo `$scope` locale e se avviene una modifica di tale proprietà nello `$scope` padre, il cambiamento viene riflesso nello `$scope` locale. Tuttavia se all'interno della Direttiva viene modificata tale proprietà, il cambiamento non viene riflesso nello `$scope` padre.

- ‘=’ (esempio {name:’=’})

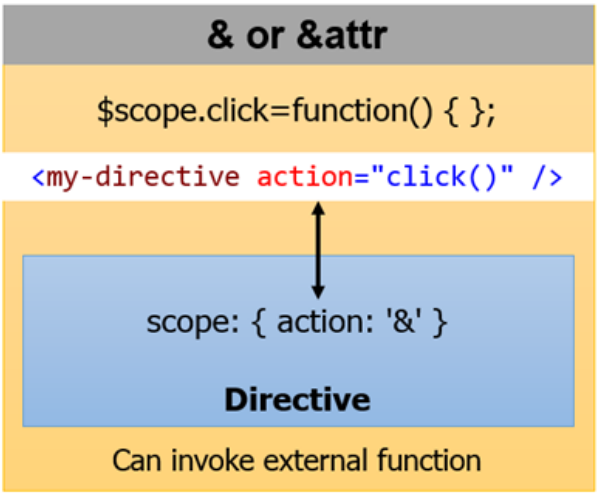
  Utilizzata per creare two-way data binding tra le proprietà di `$scope` padre e `$scope` figlio.



  Collega il valore della proprietà (`name`) dello `$scope` padre presso lo `$scope` locale: l’aggiornamento della copia locale avvieve per Two-way data binding, quindi le proprietà di `$scope` padre e `$scope` figlio restano sincronizzate.

- ‘&’

  Utilizzata per collegare funzioni definite esternamente allo `$scope` locale.



Viene creata una proprietà (`action`) nello `$scope` locale: quando tale proprietà (`action`) viene invocata, la funzione (`click`) ad essa associata viene invocata.

3.13.6 Compilazione delle Direttive

La compilazione del template avviene in 3 fasi:

- Il servizio `$compile` analizza il DOM alla ricerca di Direttive.
  Se il compilatore trova un elemento che corrisponde a una Direttiva, tale Direttiva viene aggiunta alla lista delle Direttive per quell'elemento del DOM. Un singolo elemento può avere più Direttive.

- Fase di compilazione

  Una volta identificate tutte le Direttive per un elemento del DOM, il compilatore ordina la loro l'esecuzione a seconda dell'opzione di configurazione `priority`. Durante questa fase, prima che avvenga il data-binding, le Direttive (come `ng-repeat`) manipolano il DOM.

  Per ogni Direttiva viene eseguita la funzione `compile`, se definita come opzione di configurazione, la quale ritorna una funzione di tipo `link`. Queste funzioni sono combinate in una funzione `link` composta. La funzione `compile` viene eseguita solo una volta, indipendentemente da quante volte la Direttiva compaia nel template.

- Fase di linking (collegamento)

  `$compile` collega il template allo `$scope` chiamando la funzione `link` composta (la quale invoca ogni funzione `link`), la quale registra "listeners" sugli elementi HTML e imposta `$watch` relativi allo `$scope` di ogni Direttiva.

3.14 Dependency Injection (DI)

E' un software design pattern che riguarda il modo in cui i componenti gestiscono le loro dipendenze; permette di aggiungere o rimuovere le dipendendenze a tempo di esecuzione.

La DI permette di creare ambienti isolati, ideali per la fase di testing.

In AngularJS, `$injector` è il servizio responsabile per la creazione dei componenti, la risoluzione delle relative dipendenze, e, quando richiesto, l' "iniezione" di tali componenti in altri, come dipendenze.

A tempo di esecuzione, un `$injector` crea le istanze delle dipendenze, poi le passa al componente che ne ha fatto richiesta.

Vediamo un'esempio di come agisce il servizio `$injector`.

```
<div ng-controller="MyController">
  <button ng-click="Saluta()">Ciao</button>
</div>
```

In fase di compilazione viene processata la Direttiva `ng-controller`, la quale fa richiesta ad `$injector` di creare un'istanza del Controller `MyController`, insieme alle sue dipendenze. E' da notare che è la Direttiva a fare la richiesta ad `$injector`; `MyController` non è a conoscenza né di `$injector` né delle proprie dipendenze.

```
injector.instantiate(MyController);
```

3.15 Servizi

I Servizi costituiscono il modo per condividere i dati tra i componenti dell'applicazione; inoltre permettono la comunicazione tra Controller differenti.

Sono lo strumento adatto in cui memorizzare le parti di codice che vogliamo rendere riutilizzabili: per fare un paragone con altri linguaggi di programmazione, è come se fossero delle funzioni/ metodi.

I Servizi AngularJS hanno 2 caratteristiche:

- Lazy loading – AngularJS istanzia un Servizio solo in caso di dipendenza da esso;
- Singleton – Ogni componente AngularJS che dipende da un Servizio, ottiene un riferimento a un'istanza di quel Servizio, generata da un Servizio Factory.

AngularJS ha alcuni Servizi integrati: il nome di questi comincia con il simbolo `$`.

Per creare un Servizio il modo più semplice è utilizzare il metodo `factory` dell'API di angular.module:

```
var myModule = angular.module('myModule', []);
myModule.factory('serviceId', ['$http', function($http) {
  var shinyNewServiceInstance = function()  {
        // corpo del Servizio
  };
  // il corpo della funzione factory che restituisce
     shinyNewServiceInstance, il nostro Servizio
  return shinyNewServiceInstance;
}]);
```

Il servizio `serviceId` contiene un metodo richiamabile dai componenti della nostra applicazione.

Questa funzione `factory` crea un oggetto, simbolo del servizio, che rimarrà in memoria per l'intero ciclo di vita dell'applicazione. Quando verrà caricato in memoria questo Servizio, come risultato di una dipendenza, il Servizio eseguirà questa funzione factory e attende di ricevere l'oggetto singleton del Servizio.

3.15.1 Usare i Servizi

Per usare un Servizio occorre dichiararlo ocme dipendenza per il componente all'interno del quale verrà utilizzato: Controller, Direttiva, etc. A tempo di esecuzione verrà istanziato il Servizio e verranno risolte le dipendenze del Servizio.

A questo punto è possibile eseguire qualsiasi metodo definito nel Servizio.

Capitolo 4

Tecnologia utilizzata in mobile.yoox.com

Introduzione

Di seguito un breve excursus sulle tecnlogie utilizzate per le pagine servite da mobile.yoox.com, ossia per la soluzione server side: ASP.NET e il view engine Razor, il RESTful Web Service, il file JSON e la CDN.

4.1 RESTful Web Service

Un Web Service è un sistema software progettato per supportare un'interazione tra applicazioni sviluppate con linguaggi di programmazione diversi, che girano su sistemi operativi eterogenei, utilizzando le tecnologie e gli standard Web.

Questo meccanismo consente di realizzare porzioni di funzionalità in maniera indipendente e su piattaforme potenzialmente incompatibili facendo interagire i vari pezzi tramite tecnologie Web e creando un'architettura facilmente componibile.

Un approccio possibile alla creazione di Web Service è il REST (REpresentational State Transfer), ispirato alle linee guida architetturali del Web. I principi che rendono il Web adatto a realizzare Web Service secondo l'approccio REST sono i seguenti:

- identificazione delle risorse;
- utilizzo esplicito dei metodi http;
- risorse autodescrittive;
- collegamenti tra risorse;
- comunicazione senza stato.

Identificazione delle risorse

In ambito Web, il meccanismo più naturale per individuare una risorsa è dato dal concetto di URI (*http://www.miodominio.com/ordini/2016, http://www.miodominio.com/ordini/2016?cliente=rossi*), cioè una stringa che identifica una risorsa.

Il principale beneficio nell'adottare lo schema URI per identificare le risorse consiste nel fatto che esiste già, è ben definito e collaudato.

Utilizzo esplicito dei metodi http

In un contesto RESTful sappiamo già a priori come ottenere la rappresentazione di una risorsa. In altre parole, questo principio REST stabilisce una mappatura uno a uno tra le tipiche operazioni CRUD e i metodi HTTP.

Metodo HTTP	Operazione CRUD	Descrizione
POST	Create	Crea una nuova risorsa
GET	Read	Ottiene una risorsa esistente
PUT	Update	Aggiorna una risorsa o ne modifica lo stato
DELETE	Delete	Elimina una risorsa

Risorse autodescrittive

Le risorse di per sè sono concettualmente separate dalle rappresentazioni restituite al client. Ad esempio, un Web Service non invia al client direttamente un record del suo database, ma una sua rappresentazione in una codifica dipendente dalla richiesta del client e/o dall'implementazione del servizio.
I principi REST non pongono nessun vincolo sulle modalità di rappresentazione di una risorsa.

Il tipo di rappresentazione inviata dal Web Service al client è indicato nella stessa risposta HTTP tramite un tipo MIME, così come avviene nella classica comunicazione tra Web server e browser.

Collegamenti tra risorse

Un altro vincolo dei principi REST consiste nella necessità che le risorse siano tra loro messe in relazione tramite link ipertestuali.
In sostanza, tutto quello che un client deve sapere su una risorsa e sulle risorse ad essa correlate deve essere contenuto nella sua rappresentazione o deve essere accessibile tramite collegamenti ipertestuali.

Comunicazione senza stato

E' una delle caratteristiche principali del protocollo HTTP, cioè ciascuna richiesta non ha alcuna relazione con le richieste precedenti e successive.
La principale ragione di questa scelta è la scalabilità: mantenere lo stato di una sessione ha un costo in termini di risorse sul server.

È importante sottolineare che sebbene REST preveda la comunicazione stateless, non vuol dire che un'applicazione non debba avere stato. La responsabilità della gestione dello stato dell'applicazione non deve essere conferita al server, ma rientra nei compiti del client.

4.2 ASP.NET

ASP.NET è un web framework utilizzato per la costruzione di siti e applicazioni web tramite l'uso di HTML, CSS, e JavaScript.

Si basa sul framework .NET di Microsoft.

ASP.NET si divide in 3 framework: Web Forms, ASP.NET MVC, ASP.NET Web Pages.

4.3 Razor

Razor permette di includere la sintassi di un linguaggio di programmazione, quale C# o Visual Basic .NET, all'interno di una pagina web. In una pagina web che usa la sintassi Razor, ci sono due tipi di contenuto: la pagina web, codificata in HTML, CSS, JS e il codice di un linguaggio di programmazione tra C# e Visual Basic .NET, che chiamaeremo codice server.

Se una pagina web da servire a un client contiene del codice server, esso viene eseguito dal server prima che la pagina web sia inviata al client. Questo codice server può creare dinamicamente la pagina web che sarà poi servita al client.

Le pagine web create con ASP.NET Web Pages hanno un'estensione del file che termina con *.cshtml* o *.vbhtml*. Il server rionosce queste estensioni ed esegue il codice relative alla sintassi Razor: successivamente invia la pagina al client.

La sintassi Razor syntax è basata su ASP.NET.

ASP.NET codifica le espressioni e i blocchi di codice server preceduti dal simbolo `@`.

Variabili e tipi di dato

```
@{
    // Assigning a string to a variable.
    var greeting = "Welcome!";
}
```

Conversione e testing di tipi di dato

```
@{
    var num1 = Request["text1"];
    var num2 = Request["text2"];
    // Convert the entered strings into integers numbers and add.
    total = num1.AsInt() + num2.AsInt();
}
```

Supporto degli operatori (+ - * / = == != < > <= >= + +=-= . () [] ! && ||)

Blocchi condizionali e cicli

```
<ul>
@foreach (var myItem in Request.ServerVariables)
{
    <li>@myItem</li>
}
</ul>
```

```
@for(var i = 10; i < 21; i++)
{
    <p>Line #: @i</p>
}
```

```
@{
  var showToday = false;
  if(showToday)
  {
    @DateTime.Today;
  }
  else
  {
    <text>Sorry!</text>
  }
}
```

Oggetti e collezioni di dati (array e dictionary)

```
@{
    var path = Request.FilePath;
    string[] teamMembers = {"Matt", "Joanne", "Robert", "Nancy"};
    var myScores = new Dictionary<string, int>();
    myScores.Add("test1", 71);
    myScores.Add("test2", 82);
}
```

Invocazione di metodi con parametri

```
@{
    // Pass parameters to a method using positional parameters.
    var myPathPositional = Request.MapPath("/scripts", "/", true);
}
```

Gestione di errore tramite costrutto Try-Catch

```
@{
    try {
        fileContents = File.ReadAllText(@"c:\batafile.txt");
    }
    catch (FileNotFoundException ex) {
        errMsg = ex.Message;
        userErrMsg = "Bad file path";
    }
}
```

4.4 JSON

JSON (JavaScript Object Notation) è un semplice formato per lo scambio di dati. Per le persone è facile da leggere e scrivere, mentre per le macchine risulta facile da generare e analizzarne la sintassi. Si basa su un sottoinsieme del linguaggio JavaScript, lo Standard ECMA-262 Terza Edizione, utilizzato per la definizione, creazione e utilizzo degli oggetti JS.

Ha una sintassi minimale e ed è portabile poiché si basa su 2 strutture fondamentali:

- Un insieme di coppie nome/valore. In diversi linguaggi, questo è realizzato come un oggetto, un dizionario, una tabella hash, un elenco di chiavi o un array associativo.

- Un elenco ordinato di valori. Nella maggior parte dei linguaggi questo si realizza con un array, un vettore, un elenco o una sequenza.

Queste caratteristiche fanno di JSON un linguaggio ideale per lo scambio di dati.

4.5 CDN

Una rete CDN è un sistema di nodi, di solito server, collegati in rete attraverso Internet, che collaborano in maniera trasparente, sotto forma di sistema distribuito, per distribuire contenuti agli utenti finali ed erogare servizi di streaming audio e video.

Per servire le richieste web provenienti da diverse zone geografiche nel mondo, l’utilizzo della CDN risulta essere indispensabile: per questo i nodi della CDN sono geograficamente distribuiti, spesso connessi a diverse dorsali.

Il vantaggio della CDN è di diminuire i tempi di latenza avvicinando i siti e le applicazioni agli utenti, diminuendo il carico sui server e l'utilizzo della banda passante che li collega al Web.

Lo schema che riassume le varie transazioni è il seguente:

1. Il client interroga il DNS per ottenere l'indirizzo IP del web server;

2. Il NS Autoritativo per il dominio delega la risoluzione ad un NS autoritativo della CDN;

3. È proprio quest'ultimo ad eleggere il miglior Edge Server per l'utente;

4. Al termine della query DNS, il client ottiene l'indirizzo IP del miglior Edge Server;

5. Il client si connette all'Edge Server

A questo punto ci troviamo di fronte a 3 casi possibili

1. I file richiesti non sono ancora disponibili in cache nel POP (Point of Presence, il nodo) della CDN, la CDN li recupera sui tuoi server, li salva nella cache del POP e li rinvia dal POP al tuo utente. Quando un utente richiede ancora questa pagina, il punto CDN più vicino distribuisce il contenuto statico direttamente dalla sua cache, senza richiedere nulla all'hosting.

2. I file richiesti sono disponibili in cache nel POP e sono reinviati dalla cache POP senza interrogare i tuoi server

3. I file/pagine richiesti sono fuori dalle regole di cache impostate. La CDN sposta la richiesta verso il tuo server affinchè possa reinviare i contenuti richiesti all'utente.

Quando il TTL di un file scade, la CDN lo rimuove dalla cache; il file verrà aggiunto di nuovo se richiesto da un altro visitatore.

E' possibile inoltre mettere in cache dei file in modo permanente. Con questa tecnica, è il webmaster a scegliere quali risorse far gestire alla CDN (solitamente risorse statiche). È una tecnica chiamata URL Rewriting proprio perchè si devono riscrivere gli URL delle risorse scelte, sostituendoli con quelli forniti dalla CDN.

Parte 2

Progettazione e implementazione

Capitolo 5

Progettazione della soluzione client-side con AngularJS

Introduzione

Ci inoltriamo nella fase di studio della soluzione di mobile.yoox.com: ci muoveremo dalla situazione di partenza, cercando di capire come funziona la soluzione server-side, quali sono le funzionalità richieste e come sono implementate. Passo per passo cercheremo di progettare una soluzione client-side.

Osserveremo il comportamento dell'Infinite Scrolling dell'item gallery e del Lazy Loading del contenuto; capiremo come avviene il ciclo di vita di una richiesta web per mobile.yoox.com e cercheremo di adattarla alal soluzione client-side.

Avremo a che fare con i dati relativi agli oggetti dell'item gallery e alla scelta del tipo di dato ricevuto dal server; daremo uno sguardo alle funzionalità "segnalibro" e bottone "indietro" del browser; vedremo i problemi relativi alal SEO dovuti al client-side rendering.

5.1 Il sito web yoox.com: versioni desktop e mobile

Il portale e-commerce yoox.com presenta due soluzioni software differenti nella modalità di navigazione tra le pagine di una ricerca, a seconda della piattaforma: desktop o mobile.

Mentre la versione desktop del sito yoox.com presenta dei bottoni per spostarsi tra le pagine, la versione mobile presenta il paradigma di Infinite Scrolling.

Il sito web yoox.com attraverso della logica lato server individua il device di navigazione di un utente che arriva sul sito e, se nota che arriva da un dispositivo mobile, lo reindirizza a mobile.yoox.com.

La re-ingegnerizzazione della soluzione esistente, attraverso una tecnologia di rendering client-side, ha avuto luogo solo per la parte mobile del sito web, ovvero per mobile.yoox.com, poichè presentava un'affinità perfetta con il design pattern di Single Page Application.

5.2 Perché ho scelto AngularJS e in quale versione

Ciò che mi ha portato a scegliere AngularJS, rispetto ai framework concorrenti Ember e Backbone, è stata la sua enorme popolarità tra le ricerche Google: le domande su StackOverflow, i libri, i tutorial, i blog, una lunga documentazione.

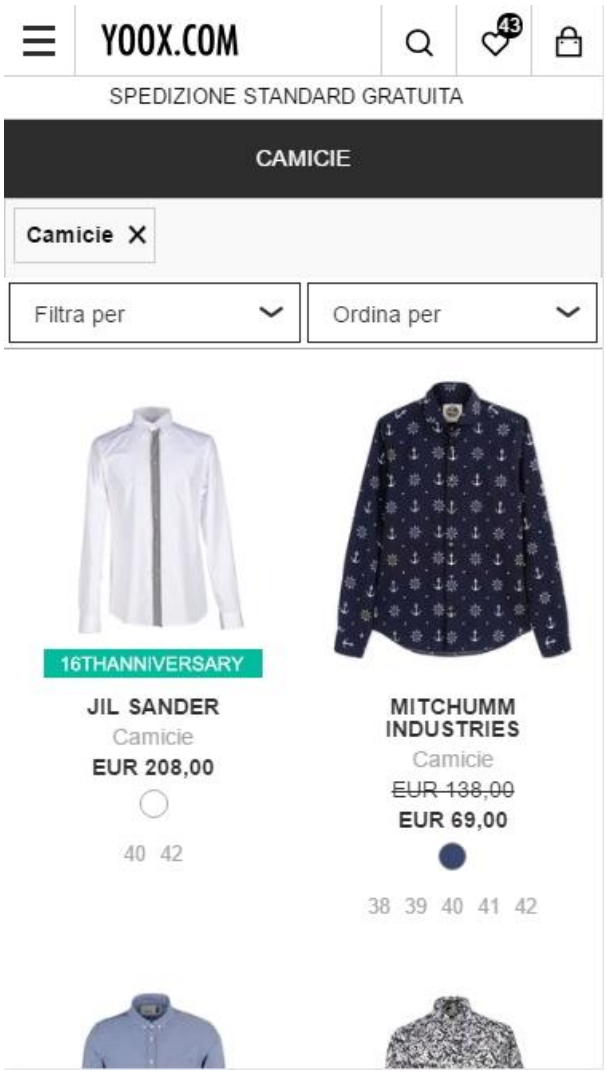
In secondo luogo è stato decisivo il fattore "dipendenze" e "soluzioni integrate": AngularJS è un framework di grandi dimensioni, basti guardare la documentazione; gli strumenti a disposizione sono di una quantità tale da fornire una soluzione alle più tipiche funzionalità del web dinamico.

Infine, e non di poco conto, il fatto che esistano molteplici plugins (o moduli di terze parti) a colmare le lacune del framework o aggiungere funzionalità, come, in questo caso, l'Infinite Scrolling.

Il framework AngularJS presenta 2 versioni: la 1, stabile, e la 2, in beta.
Dal momento che ero alle prime armi con il linguaggio JavaScript e rispettivi framework, avevo più bisogno di robustezza e affidabilità del software che non di nuove funzionalità a livello sperimentale.

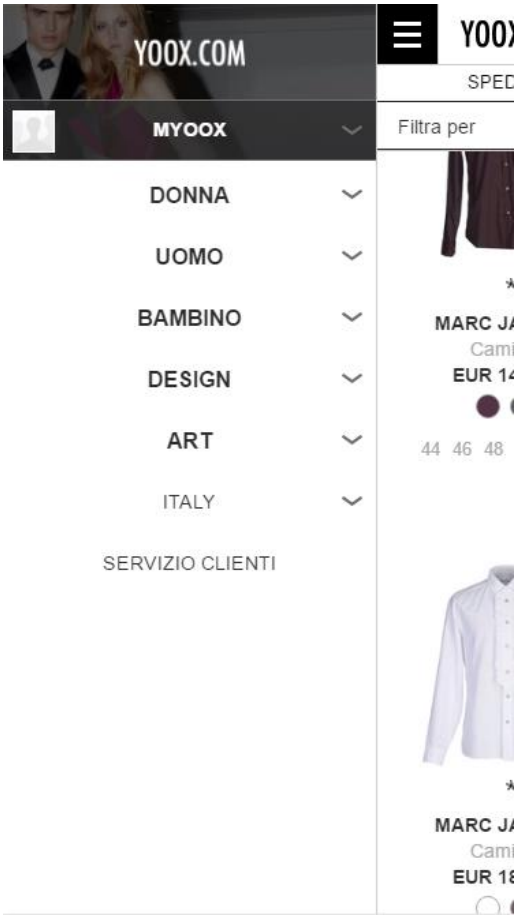
Successivamente la scelta da fare è stata tra le versioni 1.2 o 1.3+
Infatti, mentre la versione 1.2 supporta ancora IE8, dalla versione 1.3 questo supporto decade, ma viene aggiunta una funzionalità di importanza fondamentale per una SPA di un portale e-commerce: il one-time binding.
Quindi si è deciso di proseguire lo sviluppo con la versione 1.5.0 .

5.3 Il caso di studio: mobile.yoox.com



Analizziamo brevemente la UI di mobile.yoox.com:

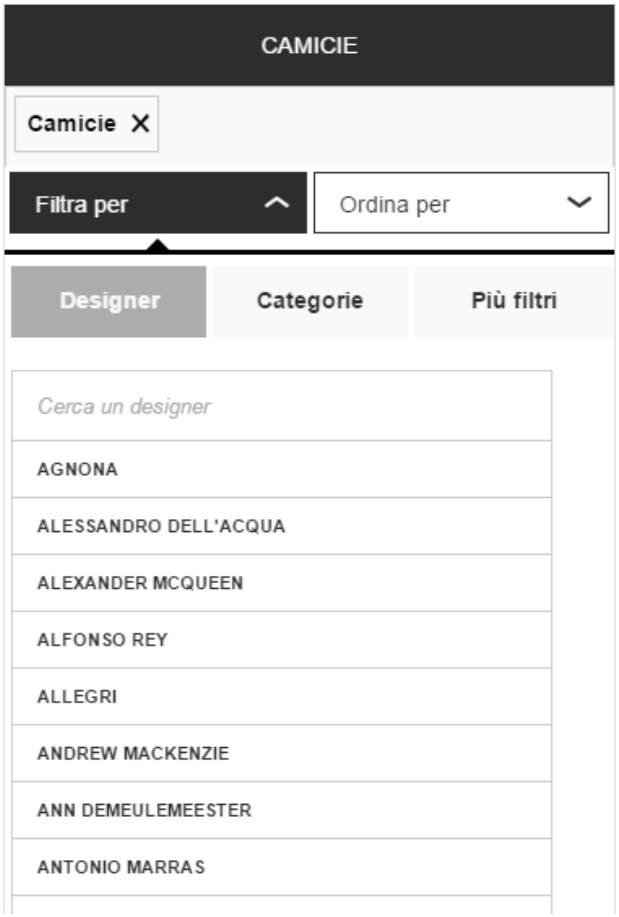
- L'icona "hamburger" che, con una transizione da sinistra a destra, porta l'utente alle categorie di yoox.com



- L'icona per la ricerca nel catalogo di yoox.com
- L'icona relativa alla "dreambox" dell'utente
- L'icona relativa alla shopping bag
- Un avviso relativo alle promozioni ("Spedizione Standard Gratuita")

- La categoria di ricerca attuale (il box nero con il titolo “Camicie”)

- I filtri attualmente impostati per la ricerca (“Camicie”) e una parte interattiva per impostare filtri ulteriori per la ricerca corrente: Designer, Categorie, Più filtri (Prezzo, Taglie, Colori, Stampa, Materiali); Ordinamento per: Ultimi arrivi, Prezzo Alto, Prezzo Basso.

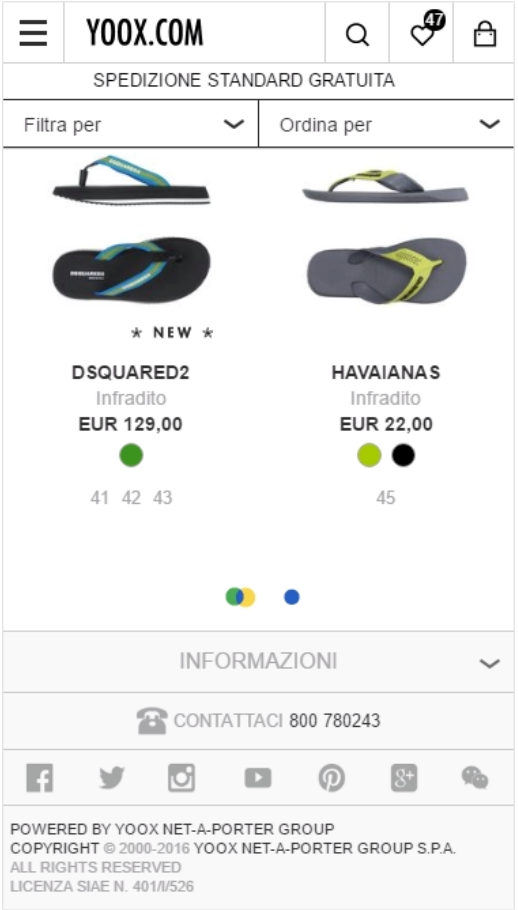


- Infine il nostro oggetto di studio, l’item gallery per la ricerca effettuata, di cui vengono subito all’occhio 2 caratteristiche:

    - È stata implementata la tecnica di navigazione chiamata “Infinite Scrolling”: non esiste un metodo di paginazione a disposizione dell’utente per la fruizione dell’item gallery, non ci si sposta di pagina in pagina attraverso dei bottoni. Invece, scrollando verso il basso, vengono caricati i nuovi oggetti: è come se le pagine venissero collocate una di seguito all’altra.

- Scrollando più e più volte ci si rende conto che gli oggetti per una ricerca non sono già tutti presenti nella pagina corrente.

  E' stata implementata la tecnica di Lazy Loading: la navigazione è stata suddivisa in pagine, ognuna di lunghezza pari a 20 oggetti. Quando l'utente, scrollando, arriva a fine pagina, viene scatenato un evento che fa partire una richiesta web per la pagina successiva. In questo frangente di tempo, ha luogo a fine pagina l'animazione di una GIF, a indicare lo stato di attesa per il caricamento di nuovi oggetti, a fine pagina.



5.4 Infinite Scrolling

Con Infinite Scrolling si intende una pagina web in cui per muoversi tra il contenuto occorre scrollare verso il basso. Il termine "Infinite" è stato scelto a

indicare la percezione che l'utente ha per la navigazione della pagina, che sembra non aver fine.
Non si tratta solo di una pagina di considerevole lunghezza: indica che la lunghezza di tale pagina può essere potenzialmente infinita.

L'Infinite Scrolling viene quasi sempre combinato con la tecnica di Lazy Loading.

5.5 Lazy Loading

Con il termine Lazy Loading (caricamento pigro) si intende il download di file on demand, cioè soltanto quando sono necessari per il prosieguo dell'applicazione.

E' una tecnica spesso utilizzata nel caricamento di file, come una serie di immagini, al verificarsi di un evento: in questo modo non si obbliga il client a scaricare file che potenzialmente non verranno mai visti, come una galleria di immagini a fine pagina, o utilizzati, come gli script.

Sostanzialmente è l'opposto dell'image preloading (o prefetching), in cui le immagini vengono caricate preventivamente in memoria per un uso successivo.

Analizziamo brevemente i vantaggi di questa tecnica:

- l'inizializzazione dell'applicazione è più rapida, perché non vengono scaricati sul client tutti i file potenzialmente fruibili dall'utente;
- il carico globale di banda è inferiore, poichè vengono scaricati solo i file necessari;
- l'applicazione occupa meno memoria, particolarmente utile nei browser di dispositivi mobile
- le richieste web ai server diminuiscono

L'unico svantaggio deriva dal codice che diventa più complesso.

5.6 Infinite Scrolling e Lazy Loading

La tecnica di Infinite Scrolling, in combinazione con il Lazy Loading, può essere implementata in una pagina web attraverso un evento che viene scatenato quando viene raggiunta la fine della pagina.

Occorre prima di tutto definire una “finestra” di spazio, oltre la quale deve essere scatenato l’evento relativo al Lazy Loading.

L’implementazione si suddivide quindi in 4 fasi:

1. Quando l’utente, durante la navigazione, raggiunge il limite inferiore della “finestra” di spazio, allora viene scatenato l’evento di Lazy Loading e si entra in una fase di caricamento. Bisogna salvare lo stato dell’evento, ad esempio “on”, tra 2 opzioni possibili, come “on” ed “off”, per evitare il verificarsi dello stesso evento durante la fase di caricamento.

   Se non venisse registrato lo stato dell’evento e l’utente continuasse a scrollare verso il basso, allora verrebbero scatenati tanti eventi, e create tante fasi di caricamento, quanto il numero di scroll che raggiunge il limite inferiore della “finetra” di spazio.

2. Si compie una chiamata AJAX per ricevere il contenuto da aggiungere alla pagina web.

3. Si attende il completamento della chiamata AJAX: a questo punto, in caso di successo, la risposta ricevuta ha al suo interno il contenuto che di andrà ad aggiungere in fondo alla pagina web.

4. Si imposta lo stato dell’evento ad “off”.

5.7 Ciclo di vita di una richiesta web per mobile.yoox.com

Il sito web yoox.com, sia in ambito desktop che mobile, si basa su Service Web RESTful per le richieste web relative alle pagine del portale e-commerce.

Vediamo in dettaglio il percorso che compie una richiesta web.

1.

Quando un utente compie una ricerca, all'interno del portale yoox.com, uno script JS o il click su un bottone modificano l'URL che identifica la risorsa per tale ricerca.

Ad esempio, un URL valido per il genere "Uomo" (gender=U), pagina numero 4 (page=4), Categoria "Scarpe" (dept=shoesmen), Sottocategoria "Mocassini" (ctgr%27%3a%5b%27mcssn%27%5d%7d) si presenta come:

http://mobile.yoox.com/it/uomo/shoponline/mocassini_c#/dept=shoesmen&gender=U&page=4&attributes=%7b%27ctgr%27%3a%5b%27mcssn%27%5d%7d&season=X&infiniteScroll=true

Il parametro "&infiniteScroll=true" viene automaticamente aggiunto se la ricerca avviene da dispositivo mobile, a indicare che la navigazione e le richieste web sono gestite secondo il paradigma di Infinite Scrolling implementato per mobile.yoox.com .

2.

Successivamente la richiesta dovrebbe arrivare su uno dei server di yoox.com: tuttavia il portale yoox.com si appoggia al servizio di CDN (Content Delivery Network) di Akamay.

3.

Se la risorsa non viene trovata presso la cache di un POP, quindi il POP inoltra la richiesta viene inoltrata ai server di yoox.com.

A questo punto la risorsa deve essere creata dinamicamente, dal server, in base all'URL:

http://mobile.yoox.com/it/uomo/shoponline/mocassini_c#/dept=shoesmen&gender=U&page=4&attributes=%7b%27ctgr%27%3a%5b%27mcssn%27%5d%7d&season=X&infiniteScroll=true

La parte dell'url che viene analizzata è la seguente, detta "path" (percorso):

it/uomo/shoponline/mocassini_c#/dept=shoesmen&gender=U&page=4&attributes=%7b%27ctgr%27%3a%5b%27mcssn%27%5d%7d&season=X&infiniteScroll=true

In questa fase, internamente al dominio mobile.yoox.com, un server si occupa della creazione di tale pagina: dal momento che deve creare dinamicamente la pagina a seconda del "path", esso si appoggia su Razor, un view engine per ASP.NET.

4.

La pagina web viene creata sulla base dell'url, ma il server non dispone dei dati. Quindi fa una chimata a un Web Service interno a yoox.com, il quale gli restituisce i dati per tale url, in formato JSON.
A questo punto il server, poichè Razor ha i dati necessari, può creare dinamicamente la pagina e mandarla come risposta al POP che ne ha fatto richiesta.

5.

Questo POP salva la pagina in cache, insieme a relativo TTL, e a sua volta la invia al client che ne aveva fatto richiesta.

5.8 Ciclo di vita di una richiesta web per mobile.yoox.com, con tecnologia AngularJS

Usando AngularJS viene spostata la fase di creazione della View (della pagina web richiesta) da livello server a livello client.

Questo sta a significare che il lavoro necessario per creare dinamicamente la pagina lato Server, tramite Razor, non avrà luogo poichè verrà compiuto lato client. Ricordiamo che Razor, per costruire una pagina web, si basa su un template (in cui può essere invocata la business logic) e sui dati che andranno a popolare tale template.

In una soluzione server-side ci troviamo con 2 pagine web: una già esistente lato client e un'altra che viene creata lato server e posta sotto a quella già esistente presso il client.

In una soluzione client-side c'è una sola pagina web, dal momento che è qua a livello client che si trova il template AngularJS e viene compiuto il binding dinamico dei dati.

Quindi ciò di cui si ha bisogno, per proseguire la navigazione e caricare altri oggetti, sono solo i dati relativi agli oggetti.

5.9 Dati relativi agli oggetti: scelta del tipo di dato

Una chiamata web basata su REST permette di ricevere in risposta dati di qualsiasi tipo, tuttavia la scelta è ricaduta sul tipo di dato JSON per 2 motivi.

1) Sostanzialmente perchè il formato JSON è riconosciuto nativamente dal linguaggio JS; ha un peso intermini di KB inferiore rispetto ai concorrenti (XML);

2) Secondariamente, ci si è appoggiati al Web Service che yoox.com utilizza internamente per recuperare e restituire i dati per un url (rappresentante una ricerca) in formato JSON.

Sebbene avessimo bisogno di questo Web Service di yoox.com, per ricevere i dati relativi alle ricerche, questo non era esposto al web, infatti come si è detto viene utilizzato internamente.

Quindi si è proceduto modificando l'URL di una richiesta web, aggiungendo una parametro in coda all'URL, "json=true", per indicare il tipo di dati che si vuole in risposta.

http://mobile.yoox.com/it/uomo/shoponline/mocassini_c#/dept=shoesmen&gender=U&page=4&attributes=%7b%27ctgr%27%3a%5b%27mcssn%27%5d%7d&season=X&infiniteScroll=true&json=true

5.10 Il pacchetto JSON di yoox.com

Internamente a yoox.com i dati per le ricerche vengono trasmessi tra server, che ricevono le richieste web, e Web Service in formato JSON. Quindi procediamo a vedere come si presenta il pacchetto JSON per la soluzione server-side.

Come si è visto, un pacchetto JSON è formato da una serie di campi costiuiti da una coppia chiave/ valore.

Internamente a yoox.com questi campi sono organizzati in classi: ricordiamo che Razor si basa su ASP.NET: la programmazione server-side avviene in linguaggio C#, che include la creazione e l'utilizzo di classi.

Viene creata una variabile per ogni classe, vengono popolati tutti gli attributi (che sarebbero le chiavi dei campi in un pacchetto JSON) e infine si compone il file JSON unendo tutti gli attributi delle varie classi, dove le chiavi dei campi corrispondono al nome degli attributi e i valori corrispondono ai valori degli attributi.

Le classi utilizzate sono 3: `SearchResultViewModel`, `ItemViewModel`, `SearchResultPaginationViewModel`.

```
public class SearchResultViewModel
{
    public long TotalResults { get; set; }
    public RefinementsViewModel Refinements { get; set; }
    public IEnumerable<ItemViewModel> Items { get; set; }
    public SearchRequestViewModel SearchRequest { get; set; }
    public SearchResultPaginationViewModel Pagination { get; set; }
    public string BasePathAjaxRequest { get; set; }
    public string SiteCode { get; set; }
    public AnalyticsSearchResultViewModel Analytics { get; set; }
    public SwitchSeasonBarViewModel SwitchSeason { get; set; }
    public string Section { get; set; }
    public string Area { get; set; }
    public string Department { get; set; }
    public string DeptCssPath { get; set; }
    public MetaTagViewModel Title { get; set; }
    public MetaTagViewModel MetaDescription { get; set; }
    public MetaTagViewModel MetaKeywords { get; set; }
    public SeoTag H1 { get; set; }
    public string MetaRobots { get; set; }
    public string CanonicalUrl { get; set; }
    public string FirstPage { get; set; }
```

```
    public string NextPage { get; set; }
    public string PrevPage { get; set; }
    public bool IsTextSearchEnabled { get; set; }
    public string TextSearchSelectedDepartment { get; set; }
    public bool InconsistentPagination { get; set; }
    public string InconsistentPaginationErrorMessage { get; set; }
}
```

```
public class ItemViewModel
{
    public string Cod10 { get; set; }
    public string Brand { get; set; }
    public string Title { get; set; }
    public string MicroCategory { get; set; }
    public string MicroCategoryPlural { get; set; }
    public string AlternativeMicro { get; set; }
    public int FullPrice { get; set; }
    public int DiscountedPrice { get; set; }
    public bool IsSoldout { get; set; }
    public IEnumerable<ItemSizeViewModel> Sizes { get; set; }
    public string SizeType { get; set; }
    public IEnumerable<ItemColorViewModel> Colors { get; set; }
    public string BadgeName { get; set; }
    public bool HasFlipSide { get; set; }
    public string Author { get; set; }
    public BadgeSearchResult BadgeSearchResult { get; set; }
    public int DiscountedPriceEur { get; set; }
    public string MicroCategoryCode { get; set; }
}
```

```
public class SearchResultPaginationViewModel
{
    public int TotalPages { get; set; }
    public int SelectedPage { get; set; } /* added in r10 */
    public List<SearchResultPageUrlsViewModel> Pages { get; set; }
    public SearchResultPageUrlsViewModel CurrentPage { get; set; }
    public SearchResultPageUrlsViewModel FirstPage { get; set; }
    public SearchResultPageUrlsViewModel PrevPage { get; set; }
```

```
    public SearchResultPageUrlsViewModel NextPage { get; set; }
    public SearchResultPageUrlsViewModel LastPage { get; set; }
    public ItemPerPageSelectorViewModel ItemPerPageSelector { get; set;
}
    public List<SearchResultSortingUrlsViewModel> SortingSelector { get;
set; }
    public string SortingSelectorLabelSelected { get; set; }
    public SeoTag H1 { get; set; }
}
```

Ora c'è una considerazione da fare: il pacchetto JSON contiene un gran numero di campi il cui utilizzo è riservato internamente a yoox.com e non hanno alcuna relazione con i dati che vengono utilizzati nella View.

Quindi si è proceduto ad eliminare tutti i campi che non sarebbero mai stati utilizzati.

```
public class SearchResultViewModel
{
    public IEnumerable<ItemViewModel> Items { get; set; }
    public SearchResultPaginationViewModel Pagination { get; set; }
    public string Department { get; set; }
    public string Area { get; set; }
}
```

```
public class ItemViewModel
{
    public string Cod10 { get; set; }
    public string Brand { get; set; }
    public string Title { get; set; }
    public string MicroCategory { get; set; }
    public string MicroCategoryPlural { get; set; }
    public string AlternativeMicro { get; set; }
    public int FullPrice { get; set; }
    public int DiscountedPrice { get; set; }
    public bool IsSoldout { get; set; }
    public string SoldoutLabel { get; set; }
    public IEnumerable<ItemSizeViewModel> Sizes { get; set; }
    public IEnumerable<ItemColorViewModel> Colors { get; set; }
    public string Author { get; set; }
    public BadgeSearchResult BadgeSearchResult { get; set; }
    public int DiscountedPriceEur { get; set; }
    public string MicroCategoryCode { get; set; }
```

```
    public string Href { get; set; }
    public string Src { get; set; }
    public string FullPriceStr { get; set; }
    public string DiscountedPriceStr { get; set; }
}
```

```
public class SearchResultPaginationViewModel
{
    public int TotalPages { get; set; }
    public SearchResultPageUrlsViewModel CurrentPage { get; set; }
    public SearchResultPageUrlsViewModel NextPage { get; set; }
}
```

Come si può notare, le variabili eliminate sono molte e questo si traduce in una serie di pregi:

- un tempo di elaborazione, a carico dei server, minore per la creazione del pacchetto JSON;
- un peso minore del pacchetto JSON che viene inviato al client: questo si traduce in una risposta più veloce al client e un'occupazione di memoria del browser inferiore.

5.10.1 Esempio di pacchetto JSON ricevuto dal server

Per motivi di spazio gli oggetti elencati (all'interno del vettore "Items") sono solo 2, sebbene una pagina web, quindi un file JSON, ne possa contenere fino a 20.


```
{
   "Items":[
      {
         "Cod10":"39584644AE",
         "Brand":"VIKTOR \u0026 ROLF",
         "Title":null,
         "MicroCategory":"Felpa",
         "MicroCategoryPlural":"Felpe",
         "AlternativeMicro":null,
         "FullPrice":328,
         "DiscountedPrice":328,
```

```
         "IsSoldout":false,
         "SoldoutLabel":"",
         "Sizes":[
            {
               "Text":"38"
            }
         ],
         "Colors":[
            {
               "Cod10":"39584644AE",
               "Rgb":"000000",
               "Description":"Nero"
            }
         ],
         "Author":"",
         "BadgeSearchResult":{
            "IsActive":true,
            "Description":"PLV_JUST IN",
            "PromotionLabel":"Questo articolo è escluso da tutte l
e promozioni.",
            "Img":"http://media.yoox.biz/yoox14/sections/sr/badge/
badge_new.png"
         },
         "DiscountedPriceEur":328,
         "MicroCategoryCode":"flp1",
         "Href":"/it/39584644AE/item#dept=clothingwomen\u0026sts=S
earchResult",
         "Src":"http://images.yoox.com/39/39584644ae_10_f.jpg",
         "FullPriceStr":"EUR 328,00",
         "DiscountedPriceStr":"EUR 328,00"
      },
      {
         "Cod10":"39578493IE",
         "Brand":"MICHAEL KORS",
         "Title":null,
         "MicroCategory":"Pullover",
         "MicroCategoryPlural":"Pullover",
         "AlternativeMicro":null,
         "FullPrice":293,
         "DiscountedPrice":293,
         "IsSoldout":false,
         "SoldoutLabel":"",
         "Sizes":[
            {
               "Text":"XS"
            },
            {
               "Text":"S"
            },
            {
               "Text":"M"
            },
            {
               "Text":"L"
            }
         ],
```

```
         "Colors":[
            {
               "Cod10":"39578493IE",
               "Rgb":"FFFFFF",
               "Description":"Bianco"
            }
         ],
         "Author":"",
         "BadgeSearchResult":null,
         "DiscountedPriceEur":293,
         "MicroCategoryCode":"pllvr",
         "Href":"/it/39578493IE/item#dept=clothingwomen\u0026sts=S
earchResult",
         "Src":"http://images.yoox.com/39/39578493ie_10_f.jpg",
         "FullPriceStr":"EUR 293,00",
         "DiscountedPriceStr":"EUR 293,00"
      }
   ],
   "Pagination":{
      "TotalPages":941,
      "CurrentPage":{
         "Page":"2",
         "Type":null,
         "Selected":false,
         "SeoHref":null,
         "Href":"http://mobile.yoox.lcl/it/donna/shoponline/maglie
 e felpe_mc/2#/dept=clothingwomen\u0026gender=D\u0026page=2\u0026a
ttributes=%7b%27ctgr%27%3a%5b%27mglr%27%5d%7d\u0026season=X",
         "OnClick":"dept=clothingwomen\u0026gender=D\u0026page=2\u
0026attributes=%7b%27ctgr%27%3a%5b%27mglr%27%5d%7d\u0026season=X"
      },
      "NextPage":{
         "Page":"3",
         "Type":null,
         "Selected":false,
         "SeoHref":null,
         "Href":"http://mobile.yoox.lcl/it/donna/shoponline/maglie
 e felpe_mc/3#/dept=clothingwomen\u0026gender=D\u0026page=3\u0026a
ttributes=%7b%27ctgr%27%3a%5b%27mglr%27%5d%7d\u0026season=X",
         "OnClick":"dept=clothingwomen\u0026gender=D\u0026page=3\u
0026attributes=%7b%27ctgr%27%3a%5b%27mglr%27%5d%7d\u0026season=X"
      }
   },
   "Department":"clothingwomen",
   "Area":"fashion"
}
```

5.11 Come implementare l'Infinite Scrolling

Per replicare il funzionamento di mobile.yoox.com occorreva implementare l'Infinite Scrolling. La soluzione più semplice e intuitiva, dal momento che si utilizzava il framework AngularJS, è stata quella di cercare un plugin compatibile con AngularJS.

La scelta è quindi ricaduta su ngInfiniteScroll.

Esploriamo ngInfiniteScroll

Dalla documentazione si capisce che è una Direttiva che viene utilizzata per implementare l'Infinite Scrolling in una applicazione AngularJS: ciò avviene semplicemente dichiarando come attributo di tale direttiva la funzione che si desidera attivare una volta che viene raggiunta la fine della pagina web.

Come si usa ngInfiniteScroll

ngInfiniteScroll richiede, oltre AngularJS, anche la dipendenza da jQuery.
Per fare uso di ngInfiniteScroll occorre includere le dipendenze esattamente in questo ordine:

```
<script type='text/javascript'
src='path/to/jquery.min.js'></script>
<script type='text/javascript'
src='path/to/angular.min.js'></script>
<script type='text/javascript'
src='path/to/ng-infinite-scroll.min.js'></script>
```

`1)` Includere la dipendenza da ngInfiniteScroll nel modulo in cui è presente la funzione che verrà associate alla direttiva. Specificare `infinite-scroll` come dipendenza.

```
angular.module('myApplication', ['infinite-scroll']);
```

2) La direttiva viene utilizzata come attributo `infinite-scroll` su un elemento HTML.

```
<div infinite-scroll=""></div>
```

3) Essendo la direttiva un attributo, bisogna specificare un valore. Questo valore è la funzione che viene invocata una votla raggiunta la fine dela pagina web.

```
<div infinite-scroll="myPagingFunction()"></div>
```

Alcuni parametri utili, opzionali, da tenere in considerazione:

- `infinite-scroll-distance` (optional) - {number} – Un numero che rappresenta a quale distanza viene valutata l'espressione associata alla direttiva `infinite-scroll`. Questo paramentro rappresenta un multiplo della lunghezza della viewport: se il parametro viene settato a 2, la viewport è 1000px, l'espressione verrà valutata quando ci si trova nella navigazione a distanza di 2000px dalla fine della pagina. Di default il valore è 0, e l'evento viene fatto scattare quando viene raggiunta la fine della pagina web.

- `infinite-scroll-disabled` (optional) - {boolean} – Un'espressione booleana che, quando valutata `true`, indica che non deve essere valutata l'espressione associata a `infinite-scroll`, indipendentemente da altre condizioni. Questo attributo viene spesso utilizzato per mettere in pausa l'Infinite Scrolling, nell'attesa che che i dati siano caricati a fine pagina.

5.12 URL & Navigazione

Quando l'utente naviga e arriva a fine pagina abbiamo già detto che viene inviata una richiesta per la pagina successiva. La differenza tra soluzione server-side e quella nuova client-side, a livello di URL, è solo il parametro "json=true".

Ma come viene fatta partire una richiesta? Analizziamo brevemente come è stato implementato l'Infinite Scrolling, sia per la soluzione vecchia, server-side, che quella nuova, client-side.
In entrambi i casi viene effettuata e gestita dietro le quinte, presso lato JS.

Nella soluzione server-side, uno script JS cambia l'URL della finestra del browser. Il nuovo URL è già presente nella pagina su cui si trova l'utente, di preciso è contenuto nell'attributo `href` dell'elemento HTML `<a id="infiniteScrollLink">`.

```
<div id="infiniteScrollBlock">
    <a id="infiniteScrollLink"
href="http://mobile.yoox.com/it/uomo/shoponline/infradito_c/2#/dep
t=shoesmen&amp;gender=U&amp;page=2&amp;attributes=%7b%27ctgr%27%3a
%5b%27nfrdt%27%5d%7d&amp;season=X"
name="&amp;lid=browse_top_numbers&amp;lpos=browse_top"
rel="address:dept=shoesmen&amp;gender=U&amp;page=2&amp;attributes=
%7b%27ctgr%27%3a%5b%27nfrdt%27%5d%7d&amp;season=X&amp;infiniteScro
ll=true" address="true"> 
    </a>
</div>
```

Il browser intercetta questo cambiamento di URL grazie all'evento `window.onhashchange`, che viene scatenato proprio quando vengono apportati cambiamenti alla parte di URL oltre il simbolo '#' (hash), detta "anchor path".

L'evento `window.onhashchange` è supportato dai browser: Chrome 5.0+, Explorer 8.0+, Firefox 3.6+, Safari 5.0+, Opera 10.6+.

A sua volta il browser invoca la funziona associata a tale evento, definita lato JS dal programmatore: questa funzione mette in moto la richiesta web e resta in attesa (in modo asincrono) della risposta, ovvero una pagina web contenente i nuovi oggetti che verrà posizionata alla fine della pagina web su cui si trova l'utente.

Per quanto riguarda invece la soluzione client-side, l'approccio adottato è differente. Osserviamo da dove proviene l'URL per la pagina successiva: abbiamo detto dall'elemento HTML `<a id="infiniteScrollLink">`. Ricordiamo che i dati, tra cui l'attributo `href`, provengono dal pacchetto JSON.
Questo vuol dire che per quanto riguarda la soluzione client-side, una volta ricevuto il pacchetto JSON, non abbiamo alcun bisogno di impostare l'attributo href dell'elemento `<a id="infiniteScrollLink">`: tutte le informazioni sono già disponibili nel pacchetto JSON.

Da qui la scelta di eliminare l'elemento HTML `<a id="infiniteScrollLink">` e mantenere solo l'elemento padre `<div id="infiniteScrollBlock">`.

Successivamente viene effettuata una chiamata lato JS e si resta in attesa della risposta, che stavolta è un pacchetto JSON. A questo punto si potrebbe dire di aver concluso, visto che l'URL non ha alcuno scopo: invece no.

Sebbene l'URL non abbia alcun fine dal punto di vista dell'infinite Scrolling, ha una rilevanza fondamentale per l'implementazione di 2 funzionalità:

- salvare un URL come segnalibro;
- far funzionare il bottone "indietro" in un SPA.

Per questi due motivi, che possono essere riassunti in "salvare lo stato di navigazione dell'utente", una volta ricevuto il pacchetto JSON viene aggiornato l'URL.

5.13 Funzionalità "segnalibro" del browser

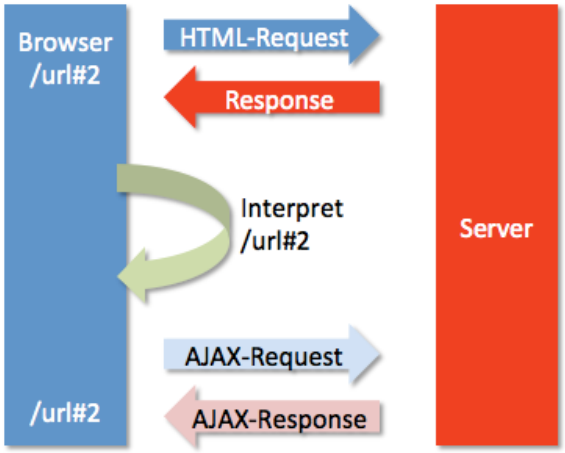
Prendiamo in analisi un'applicazione sviluppata come SPA, senza supporto ai segnalibri. Se un utente, trovandosi alla pagina 5, decidesse di salvare la pagina corrente come segnalibro, il risultato non sarebbe quello desiderato. L'URL, durante la navigazione dalla pagina iniziale fino alla pagina 5, non è mai cambiato.

Il segnalibro sarebbe conterrebbe così un URL alla prima pagina.

Per salvare lo stato di navigazione, occorre registrare nell'URL gli spostamenti di pagina in pagina che l'utente compie. Semplicemente, tutte le volte che viene cambiata pagina, si cambia l'URL con un "anchor path" differente: in questo modo ogni pagina ha un URL diverso.

Poniamo che l'URL del segnalibro sia www.miodominio.com/galleria#page=4 Utilizzando tale segnalibro, viene compiuta una richiesta http per www.miodominio.com/galleria e l'utente viene portato esattamente su tale pagina; successivamente, attraverso uno script JS che analizza l'URL, viene effettuata una chiamata AJAX per ricevere esattamente la risorsa presso l'indirizzo www.miodominio.com/galleria#page=4

A questo punto l'utente può continuare la navigazione dalla pagina in cui aveva salvato il segnalibro.



5.14 Funzionalità del bottone "indietro" del browser

Se non si salva lo stato di navigazione dell'utente, un click sul bottone "indietro" porta l'utente ad uscire dal sito che sta navigando e a tornare su siti precedenti nella cronologia degli URL visitati.

Utilizzando invece il meccanismo di "anchor path" differenti, ad ogni click sul bottone "indetro" corrisponde uno scroll verso l'alto del browser, per posizionarsi esattamente nel punto in cui è stata impostato l' "anchor path" precedente.

5.15 Navigazione tramite parametro URL "page"

Siccome è possibile spostarsi tra le pagine di una ricerca cambiando il valore dell'attributo "page" dell'anchor path, si è deciso di replicare questa funzionalità anche per la nuova soluzione client-side.

Procediamo prima a parlare di questa funzionalità.
Poniamo che un utente si trovi in posizione "page=4": se viene impostato manualmente "page=9", verrà catturato l'evento `window.onhashchange` e inviata una richiesta web esattamente per quell'URL con parametro "page=9".

Sostanzialmente è come forzare l'Infinite Scrolling cambiando manualmente il parametro "page" dell'URL. E in più si possono caricare pagine non necessariamente consecutive (ad esempio far susseguire richieste per "page=5", "page=10", "page=15"): ovviamente il rischio è di capitare su una pagina non esistente.

5.15.1 Navigazione e funzionalità “segnalibro”

Un problema, per quanto riguarda la soluzione server-side, deriva dal fatto che cliccando su un segnalibro per “page=4”, viene sì scaricata “page=4” ma viene posta sotto “page=1”, ossia la pagina portata in cache in maniera statica.

Per di più non si viene posizionati su “page=4”: la navigazione parte esattamente da inizio pagina web, ossia da “page=1”.
Sebbene i tempi di risposta per la pagina in cache della CDN siano minimi, non è da trascurare il fatto che l’utente possa non accorgersi di questo comportamento e abbandonare preventivamente la navigazione.

Per questo motivo, questo comportamento è stato cambiato. L’utente cliccando sul segnalibro per “page=4” riceverà e vedrà esattamente solo tale pagine web.

5.16 SEO e rendering client-side di pagine web

Un nodo fondamentale del rendering client-side è la SEO: esso infatti è incompatibile con la modalità attuale che i motori di ricerca usano per indicizzare le pagine web.

Le pagine web create dinamicamente a livello di client sono difficili da processare per i motori di ricerca: il crawler di un motore di ricerca non riesce a simulare il comportamento di tali applicazioni, in cui il contenuto viene creato dinamicamente dal browser sulla base dei dati ricevuti dalle chiamate AJAX.

Un crawler compiendo una richiesta web per una pagina costruita dinamicamente a livello del client, ottiene in risposta solo il template, senza contenuto.
Quindi non vedendo il contenuto di tali pagine, il crawler non può indicizzarle.

Per indicizzare la pagina creata dinamicamente, il crawler ha bisogno di un HTML snapshot, una preview di tale pagina: questo snapshot deve essere disponibile come file statico presso un server e deve rappresentare la pagina creata dinamicamente.

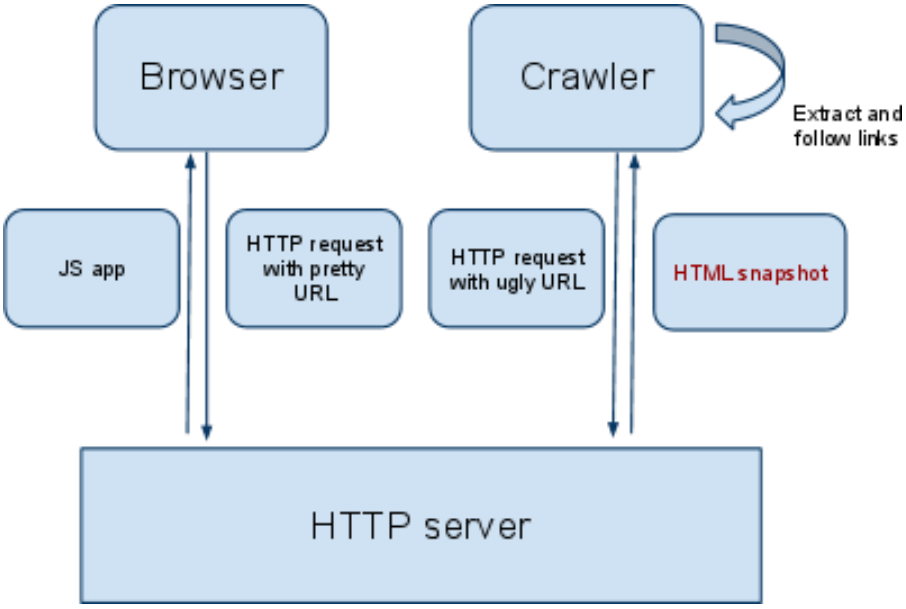
Occorre quindi creare una versione parallela a quella dinamica, lato JS. Gli utenti accedono alla versione dinamica, mentre il crawler accede a quella statica.

Occorre quindi adottare il seguente schema, detto AJAX crawling.

1. per ogni URL che identifica una risorsa creata dinamicamente, il server deve fornire un HTML snapshot;

2. il motore di ricerca indicizza gli HTML snapshot e mostra nei risultati di ricerca l'URL originale alla pagina create dinamicamente.

Perché ciò accada l'applicazione deve usare una sintassi specifica per gli URL AJAX ("pretty URLs"). Il crawler modifica temporaneamente questi "pretty URLs" in "ugly URLs", che fanno riferimento agli HTML snapshot e fa una richiesta per questi ultimi al server. Quando il crawler ha ottenuto l'HTML snapshot, lo analizza, ne indicizza il contenuto e nei risultati della ricerca mostra il "pretty URL".



1.

Segnala al crawler che il tuo sito supporta lo schema di AJAX crawling

Per riuscire in questo intento occorre utilizzare un token speciale per gli "hash fragments" (ossia, tutto ciò oltre il simbolo # dell'URL): gli hash fragments devono cominciare con un punto esclamativo.

```
www.example.com/ajax.html#key=value
```

L'URL qua sopra dovrebbe essere trasformato nell' URL seguente:

```
www.example.com/ajax.html#!key=value
```

Il crawler vedrà il contenuto del sito se vengono implementati gli HTML snapshots.

2.

Configura il server per la gestione di richieste con URL contenenti la stringa `_escaped_fragment_`

Questa richiesta, che può essere formulata solo dal crawler, deve avere in risposta l'HTML snapshot. Il crawler infatti compirà una trasformazione sull'AJAX URL.

Questo è l'URL che identifica una risorsa dinamica:

```
www.example.com/ajax.html#!key=value
```

Questo è l'URL dello snapshot HTML per tale risorsa:

```
www.example.com/ajax.html?_escaped_fragment_=key=value
```

Abbiamo appena visto quindi che una soluzione client-side, che crea dinamicamente le pagine web, necessita di una serie di modifiche, soprattutto lato server, per motivi di indicizzazione delle pagine web.

Tuttavia, prendiamo in considerazione l'indicizzazione delle pagine di yoox.com. Il portale prevede 2 versioni, una per l'ambito desktop e una per la parte mobile. Le pagine web differiscono solo per il numero di oggetti.

Dal momento che su yoox.com è in vigore una soluzione server-side, che crea le pagine dinamicamente lato server, tutte le pagine sono visibili ad un crawler, poiché quando richieste vengono servite con il tenuto già incorporato. Il crawler, una volta arrivato sulla root del dominio è in grado di percorrere ricorsivamente tutti i path possibili dei link presenti in quella pagina e così riesce ad indicizzare tutte le risorse.

Ricordiamo che l'URL alla pagina seguente si trova in pagina (`#infiniteScrollBlock`) e a sua volta la pagina successiva contiene l'URL per la pagina web ancora oltre, e così via.

Nel nostro caso, avendo implementato una soluzione client-side solo per la parte mobile, avevamo la parte desktop che si basava ancora su architettura server-side.

Quindi non c'è stato motivo di metter mano allo schema di AJAX crawling.
Ci siamo trovati esattamente nella situazione in cui al crawler sono servite pagine statiche e al client pagine dinamiche, come per lo schema di AJAX crawling.

5.17 I filtri di ricerca

Per quanto riguarda l'uso dei filtri, c'erano 2 possibilità: spostare tutta la logica client-side, oppure mantenerla lato-server.
L'implementazione client-side avrebbe richiesto molto, troppo tempo e quindi li si è lasciati server-side.

Una volta impostati i filtri, l'URL riflette l'utilizzo di tali filtri e viene inviata un richiesta web per tale URL: mentre nella soluzione server-side vengono ricaricati solo la galleria di oggetti, i filtri e le categorie utilizzate, nella soluzione client-side viene fatta ricaricata tutta la pagina e fatta partire la chiamata web per quell'URL. In questo caso viene re-istanziata tutta l'applicazione AngularJS.

Capitolo 5

Implementazione della soluzione client-side con AngularJS

Introduzione

In questo capitolo parleremo di come è stata implemetato AngularJS per replicare il funzionamento dell'item gallery.

In particolare partiremo dal template Razor e ne analizzeremo il codice per blocchi; successivamente verrà fatta una trasposizione di tali blocchi per il template AngularJS.

Proseguiremo con la View logic, cioè la logica legata al template, e la Business Logic, legata all'Infinite Scrolling, all'URL e allo stato di navigazione.

In seguito vedremo la trasformazione del template AngularJS tramite l'utilizzo delel Direttive e infine parleremo di performance della soluzione client-side e possibili miglioramenti.

6.1 Il template Razor

Cominciamo nell'analizzare il template Razor rappresentante la gallery item di una ricerca su mobile.yoox.com. Per semplificare e avere un'idea generale di come è organizzato il template, osserviamo il codice seguente.

```
@model SearchResultViewModel

<div id="itemsGrid>
    @foreach (var item in Model.Items){
        // <a href='objectpageURL'>
        //     <div class="itemImg"> </div> (immagine oggetto)
        //     <div class="itemData">
        //         <span class="itemInfos"></div> (descrizione)
        //         <div class="itemInfos"></div> (prezzo)
        //         colori
        //         taglie
        //     </div>
        // </a>
    }
    <div id="infiniteScrollBlock"></div>
</div>
```

Vediamo che `<div id="itemsGrid>` è il contenitore della nostra item gallery: per questo motivo vengono ciclati, uno a uno, gli oggetti dell'item gallery. Ad ogni passo verrà preso in considerazione un oggetto, il quale verrà aggiunto in fondo a `<div id="itemsGrid>`.

Le informazioni da visualizzare in pagina, relative ad un oggetto, sono state suddivise in parti che fanno rispettivamente riferimento a: l'immagine dell'oggetto in item gallery, la descrizione, il prezzo, i colori e le taglie disponibili.

Infine, fuori dal ciclo, troviamo a fine pagina `<div id="infiniteScrollBlock">`, l'elemento HTML sui cui deve posizonarsi l'utente per attivare l'Infinite Scrolling. Questo elemento contiene il collegamento alla pagina successiva.

`@model SearchResultViewModel`: viene definita una variabile `model` di tipo `SearchResultViewModel`

`@foreach (var item in Model.Items)`: si tratta di un ciclo che viene compiuto su tutti gli oggetti presenti in `Model.Items`. Ad ogni passo viene preso in considerazione un elemento, `var item` dell'array `Model.Items`, il quale rappresenta la collezione di oggetti di una sola pagina di navigazione.

Procediamo per gradi a vedere ciò che si trova internamente a `@foreach`; tutti i blocchi di codice a seguire sono interni a `@foreach`.

BLOCCO 1

In questo blocco viene definito, insieme a una serie di attributi, il collegamento di un oggetto dell'item gallery alla relativa pagina del prodotto.

```
var microCategory = (!String.IsNullOrEmpty(item.AlternativeMicro))
? item.AlternativeMicro :
(!String.IsNullOrEmpty(item.MicroCategoryPlural)) ?
item.MicroCategoryPlural : item.MicroCategory;

        <a href="@Url.GetItemUrl(item.Cod10, Model.Department,
Model.Section)" id="item_@(item.Cod10)"

           @Html.Attribute("class", "itemContainer",
item.IsSoldout ? "soldout" : "")
           @Html.Attribute("data-cod8", item.Cod10.Substring(0,
8))
           @Html.Attribute("data-category", microCategory)
           @Html.Attribute("data-category_id",
item.MicroCategoryCode)
           @Html.Attribute("data-brand", item.Brand)
           @Html.Attribute("data-discountprice_eur",
item.DiscountedPriceEur.ToString())>
```

`var microCategory`: questa variabile viene definita in base a un controllo dei valori `item.AlternativeMicro` e `item.MicroCategoryPlural`. Se `item.AlternativeMicro` non è `''` o `null`, viene assegnato `item.AlternativeMicro`. Altrimenti se `item.AlternativeMicroPlural` non è `''` o `null`, viene assegnato `item.MicroCategoryPlural`, altrimenti `item.MicroCategory`.

`<a href="@Url.GetItemUrl(item.Cod10, Model.Department, Model.Section)"`: viene creato dinamicamente l'attributo `href` tramite la funzione `@Url.GetItemUrl` e relativi parametri `item.Cod10, Model.Department, Model.Section.`

`id=item_@(item.Cod10)`: viene creato dinamicamente l'attributo `id` utilizzando `item.Cod10.`

`@Html.Attribute("class", "itemContainer", item.IsSoldout ? "soldout" : "")`: viene creato dinamicamente l'attributo `class=ItemContainer` in base al valore di `item.IsSoldout.`

`@Html.Attribute("data-cod8", item.Cod10.Substring(0, 8))`: viene creato dinamicamente l'attributo `data-cod8` in base a una sottostringa di `item.Cod10.`

`@Html.Attribute("data-category", microCategory)`:viene creato dinamicamente l'attributo `data-category.`

`@Html.Attribute("data-category_id", item.MicroCategoryCode)`: viene creato dinamicamente l'attributo `data-category-id.`

`@Html.Attribute("data-brand", item.Brand)`: viene creato dinamicamente l'attributo `data-brand.`

`@Html.Attribute("data-discountprice_eur", item.DiscountedPriceEur.ToString())`: viene creato dinamicamente l'attributo `data-discountprice_eur` come tipo di dato `String`.

BLOCCO 2

Procediamo con un altro blocco di codice, che si pone internamente a BLOCCO 1. Il seguente blocco HTML, `<div class="itemImg">`, rappresenta l'immagine dell'oggetto all'interno della galleria della pagina web. Inoltre si controlla se aggiungere un'immagine badge, riservata agli oggetti in promozione.

```
<div class="itemImg">

    @{var code10 = (item.Colors != null && item.Colors.Any()) ?
item.Colors.First().Cod10 : item.Cod10; }

    <img class="front" src="@Url.GetImageUrl(code10, "10_f")"
alt="@item.Brand - @item.MicroCategory" />

    @if (item.BadgeSearchResult != null &&
item.BadgeSearchResult.IsActive &&
!string.IsNullOrEmpty(item.BadgeSearchResult.Img))
    {
        <div class="itemBadge">
            <img src="@item.BadgeSearchResult.Img" alt="" />
        </div>
    }
</div>
```

`var code10`: questa variabile viene definita in base a un controllo del valore di `item.Colors`. Se `item.Colors` non è `null` e `item.Colors` è una sequenza con almeno un elemento, allora viene assegnato il valore `item.Colors.First().Cod10`, altrimenti viene assegnato il valore `item.Cod10`.

`src="@Url.GetImageUrl(code10, "10_f")"`: l'attributo `src` viene bindato dinamicamente al valore restituito dalla funzione `@Url.GetImageUrl` con parametri `code10, "10_f"`.

`@if (item.BadgeSearchResult != null && item.BadgeSearchResult.IsActive && !string.IsNullOrEmpty(item.BadgeSearchResult.Img))`: viene restituito `true` se `item.BadgeSearchResult` è non nulla, `item.BadgeSearchResult.IsActive` è diversa dai valori `null, undefined, '', 0 , false, NaN`) e `item.BadgeSearchResult.Img` è non nulla e diversa da ''. Se ciò non accade viene restituito `false`.

`alt="@item.Brand - @item.MicroCategory"`: viene bindato dinamicamente l'attributo `alt` con la concatenazione delle proprietà `@item.Brand` e `@item.MicroCategory`.

BLOCCO 3

Proseguiamo con il prossimo blocco di codice, interno prima a BLOCCO 1 e poi a `<div class="itemData">`. Riguarda la descrizione del prodotto: vengono riportati la marca, la categoria e un'eventuale micro descrizione.

```
<span class="itemInfos">
    <span class="brand infoRow fontBold">
        @Html.Raw((string.IsNullOrEmpty(item.Title)) ? item.Brand
: item.Title)
    </span>

    @if (!string.IsNullOrEmpty(item.Author) && (Model.Department
== "books" || Model.Area == "design" ||
Model.Department.EndsWith("art")))
    {
        <span class="auth infoRow">@item.Author</span>
    }

    @if (!string.IsNullOrEmpty(item.Title))
    {
        <span class="infoRow">@item.Brand</span>
    }

    <span class="microCategory infoRow">
        @Html.Raw(microCategory)
    </span>
</span>
```

@Html.Raw((string.IsNullOrEmpty(item.Title)) ? item.Brand : item.Title): Restituisce il markup non codificato in formato HTML di `item.Brand`, se `item.Title` è '' o `null`, altrimenti restituisce il markup non codificato in formato HTML `item.Title`.

@if (!string.IsNullOrEmpty(item.Author) && (Model.Department == "books" || Model.Area == "design" || Model.Department.EndsWith("art"))): Restituisce `true` se `item.Author` è `null` o '' e viene soddisfatta una delle seguenti condizioni: `Model.Department` uguale a 'books', `Model.Area` uguale a 'design', il valore di `Model.Department` finisce con 'art'. Altrimenti viene restituito `false`.

@item.Author: viene bindata dinamicamente la proprietà `item.Author`

@if (!string.IsNullOrEmpty(item.Title)): viene restituito true se `item.Title` non è né ‘’ né `null`.

@item.Brand: viene bindata dinamicamente la proprietà `item.brand`.

@Html.Raw(microCategory): Restituisce il markup non codificato in formato HTML di `microCategory`.

BLOCCO 4

Viene posto internamente a BLOCCO 1 e `<div class="itemData">`, sotto a BLOCCO 3. Questo blocco riguarda invece il prezzo dell’oggetto: se l’oggetto è esaurito il `<div>` entro cui si trova l’oggetto diventa opaco e non cliccabile; se è in sconto ha invece 2 prezzi, di cui uno barrato; altrimenti viene riportato un solo prezzo.

```
<div class="itemInfos">
    @if (item.IsSoldout)
    {
        <div class="currentprice infoRow fontBold">
            @Html.CmsLabel(CmsLabelFiles.SearchResult, "soldOut")
        </div>
    }

    else if (item.FullPrice == item.DiscountedPrice)
    {
        <div class="currentprice infoRow fontBold">
            @Html.Currency(item.FullPrice)
        </div>
    }

    else
    {

        <div class="oldprice infoRow">
            @Html.Currency(item.FullPrice)
        </div>
        <div class="currentprice infoRow fontBold">
            @Html.Currency(item.DiscountedPrice)
        </div>
    }
</div>
```

`@if (item.IsSoldout)`: viene restituito true se il valore di `item.isSoldout` è diverso da `null, undefined, '', 0 , false, NaN`.

`@Html.CmsLabel(CmsLabelFiles.SearchResult, "soldOut")`:

`@Html.Currency(item.FullPrice)`: viene restituito `item.FullPrice` nella valuta corretta

`@Html.Currency(item.DiscountedPrice)`: viene restituito `item.DiscountedlPrice` nella valuta corretta

BLOCCO 5

Viene posto internamente a BLOCCO 1 e `<div class="itemData">`, sotto a BLOCCO 4. Vengono aggiunti fino a 4 colori nella descrizione dell'oggetto; se l'oggetto è disponibile in più di 4 tonalità di colore verrà aggiunto il simbolo '+' in coda al quarto colore.

```
@if (item.Colors != null && item.Colors.Any() &&
!(item.Colors.Any(c => c.Description == "(-)") &&
item.Colors.Count() == 1))
{
    <div class="itemInfos colorInfo">
    @foreach (var col in item.Colors.Select((x, i) => new { Data =
x, Index = i }))
    {
        if (col.Index == 4 && item.Colors.Count() > 5)
        {
            <span class="moreColorSize fontBold">+</span>
            break;
        }
        <span class="aColor" style="background-color:
#@(col.Data.Rgb)"></span>
    }
    </div>
}
```

`@if (item.Colors != null && item.Colors.Any() && !(item.Colors.Any(c => c.Description == "(-)") && item.Colors.Count() == 1))`: ritorna `true` se `item.Colors` non è `null`; e la sequenza è composta almeno di un elemento; e quando c'è un solo elemento nella sequenza questo ha valore diverso da '(-)'.

`@foreach (var col in item.Colors.Select((x, i) => new { Data = x, Index = i }))`: viene compiuto un ciclo su `item.Colors` dove il valore degli elementi viene assegnato a `Data` e l'indice degli elementi a `Index`.

`<span class="aColor" style="background-color: #@(col.Data.Rgb)"></span>`: viene bindato dinamicamente il valore di `col.Data.Rgb` dove `col.Data` è l'elemento `col` di indice `Index` e contenuto `Data`. `Data` è un dizionario.

BLOCCO 6

Viene posto internamente a BLOCCO 1 e `<div class="itemData">`, sotto a BLOCCO 4. Questo blocco riguarda le taglie di un vestito: vengono utilizzati fino a 16 caratteri per la visualizzazione delle taglie disponibili; oltre i 16 caratteri, se esistono ancora taglie da visualizzare, viene aggiunto il simbolo '+'.

```
@if (item.Sizes.Any())
{
    <div class="itemInfos sizeInfo">
       @{ var strCnt = 0; }

       @foreach (var size in item.Sizes.Select((x, i) => new {
Data = x, Index = i }))
        {
            strCnt += size.Data.Text.Length + 1;

            if (strCnt > 16)
            {
                <span class="moreColorSize fontBold">+</span>
                break;
            }

            <span class="aSize">@Html.Raw(size.Data.Text)</span>
        }
    </div>
}
```

`@if (item.Sizes.Any())`: ritorna `true` se la sequenza `item.Sizes` è composta almeno di un elemento.

`@foreach (var size in item.Sizes.Select((x, i) => new { Data = x, Index = i }))`: viene compiuto un ciclo su `item.Sizes` dove il valore degli elementi viene assegnato a `Data` e l'indice degli elementi a `Index`.

`<span class="aSize">@Html.Raw(size.Data.Text)</span>`: Restituisce il markup non codificato in formato HTML di `size.Data.Text`

BLOCCO 7

Infine il blocco di codice che abbiamo chiamato, all'inizio, semplicemente `<div id="infiniteScrollBlock">`. Si trova sotto il blocco `@foreach`. Se esiste una pagina successiva a quella su cui si trova l'utente, allora viene bindato dinamicamente il collegamento alla pagina successiva.

In ogni caso viene `@Html.Partial("_SearchResultJsInitUpdate", Model)`

```
@if (Model.Pagination.NextPage != null)
{
    <div id="infiniteScrollBlock">
        <a id="infiniteScrollLink"
href="@Model.Pagination.NextPage.Href"
name="&lid=browse_top_numbers&lpos=browse_top"
rel="@(Model.Pagination.NextPage.OnClick)&infiniteScroll=true
"> 
        </a>
        @Html.Partial("_SearchResultJsInitUpdate", Model)
    </div>
}
else
{
    @Html.Partial("_SearchResultJsInitUpdate", Model)
}
```

`href="@Model.Pagination.NextPage.Href"`: viene bindato dinamicamente l'attributo `Model.Pagination.NextPage.Href`.

`rel="@(Model.Pagination.NextPage.OnClick)&infiniteScroll=true"`: viene bindato dinamicamente l'attributo `Model.Pagination.NextPage.OnClick` a cui viene aggiunto in coda la stringa `'&infiniteScroll=true'`. In questo modo, aggiungendo un attributo all'URL, la richiesta web è autoesplicativa della piattaforma da cui proviene: in questo caso da mobile.

`@Html.Partial("_SearchResultJsInitUpdate", Model)`:

Riportiamo brevemente un esempio dei valori delle proprietà `Href` e `OnClick`:

```
"Href":"http://mobile.yoox.lcl/it/donna/shoponline/maglie e
felpe_mc/3#/dept=clothingwomen\u0026gender=D\u0026page=3\u0026attr
ibutes=%7b%27ctgr%27%3a%5b%27mglr%27%5d%7d\u0026season=X"

"OnClick":"dept=clothingwomen\u0026gender=D\u0026page=3\u0026attri
butes=%7b%27ctgr%27%3a%5b%27mglr%27%5d%7d\u0026season=X"
```

Come si può notare, la proprietà OnClick contiene la parte della proprietà `Href` oltre `"http://mobile.yoox.lcl/it/donna/shoponline/maglie e felpe_mc/3#/".`

Inoltre come abbiamo visto in fase di progettazione ogni pagina web contiene un link alla pagina successiva: è in questo modo che il crawler dei motori di ricerca riesce ad indicizzare tutti i prodotti.

Non entriamo nel merito dello script JS che implementa l'Infinite Scrolling. L'unica cosa importante è il comportamento da replicare.

Osservando il CSS notiamo che:

```
#infiniteScrollBlock {
    clear: both;
    background: url("http://cdn2.yoox.biz/yoox14/loading.gif?_2")
no-repeat center center transparent;
    height: 50px;
    width: 100%;
}
```

Quindi, quando vediamo una GIF con puntini di caricamento (http://cdn2.yoox.biz/yoox14/loading.gif?_2mentre si è in attesa per il caricamento della prossima pagina di oggetti, si tratta proprio di `<div id="infiniteScrollBlock">`.

A questo punto, avendo visto l'implementazione di Razor per la soluzione server-side, dobbiamo tradurre questo template nella rispettiva versione in AngularJS.

6.2 AngularJS Template

Ricordiamo, dal capitolo “Conoscere AngularJS”, che per avviare un’applicazione AngularJS occorre inizializzare il framework. Occorre applicare la direttiva `ng-app` ad un elemento HTML contenitore del template AngularJS.

Quindi semplicemente creiamo un `<div>` presso cui utilizziamo la direttiva `ng-app`. Il valore di ng-app è del tutto arbitrario, rappresenta il nome dell’applicazione.

Secondariamente abbiamo bisogno di un Controller per gestire la business logic di tutti i blocchi condizionali visti nel template Razor: essa infatti deve essere gestita a livello di client.

Decidiamo di creare un Controller sullo stesso elemento di `ng-app`.

```
<div ng-app="itemGalleryApp" ng-
controller="itemGalleryController">
```

Procediamo oltre. A questo punto ci troviamo su `<div id="itemsGrid">`, al cui interno troviamo subito `@foreach`. Per implementare l’Infinite Scrolling l’approccio del plugin ngInfiniteScroll è differente: occorre infatti utilizzare la Direttiva `infinite-scroll` ed eventuali parametri sul `<div>` padre del `@foreach` visto nel template Razor. Quindi, dato che `<div id="itemsGrid">` contiene tale `@foreach`, utilizziamo tale Direttiva su questo elemento HTML.

```
<div id="itemsGrid" infinite-scroll='item_gallery.nextPage()'
infinite-scroll-disabled='loading_json'>
```

`nextPage` sarà il nome della funzione che definiremo nella parte JS, all’interno del Controller, definita internamente all’oggetto `item_gallery` di tipo `ItemGallery`.
Aggiungiamo anche un parametro relativo al plugin ngInfiniteScroll, ovvero 'loading_json'. Questo parametro ci permette di bloccare l’Infinite Scrolling fintanto che il file JSON e il rendering degli oggetti per la prossima pagina non sono avvenuti.

BLOCCO 1

AngularJS permette l'utilizzo delle Direttive solo se definite presso elementi HTML. Quindi per replicare `@foreach` applichiamo la Direttiva `ng-repeat` all'elemento `<a>`, che corrisponde al nostro BLOCCO 1.

```
<a  ng-repeat="item in items">
```

Dove items è un oggetto dello `$scope` e item un oggetto creato localmente al ciclo.
Ad ogni passo del ciclo viene preso in considerazione un nuovo oggetto item e viene clonato un elemento HTML `<a>` insieme a tutto ciò che si trova internamente ad esso; alla fine saranno stati creati tanti elementi `<a>` quanti gli oggetti in items.

Secondariamente occorre creare dinamicamente gli attributi di `<a>`.

```
<a  ng-repeat="item in items" ng-init="department =
item_gallery.Department; area = item_gallery.Area" ng-
class="[itemContainer, {soldout : item.IsSoldout}]" ng-
href={{item.Href}} id={{item.id}} class="itemContainer isViewed"
data-cod8={{item.Cod8}} data-category={{item.MicroCategory}} data-
category_id={{item.MicroCategoryCode}} data-brand={{item.Brand}}
data-discountprice_eur={{item.DiscountedPriceEur}}>
```

Si noti l'utilizzo della Direttiva `ng-href`, per rimandare la valutazione del valore di `ng-href` (che sarebbe `href`) da parte del browser fino quando non è avvenuta l'interpolazione con il dato dinamico. La Direttiva `ng-class` invece ci permette di aggiungere dinamicamente la classe `itemContainer` e la classe `soldout` se `item.IsSoldout` è diverso dai valori `null, undefined, '', 0 , false, NaN`.

La Direttiva `ng-init` ci permette di dichiarare delle variaibili locali e inizializzarne i valori. Sono accessibili internamente all'elemento, discendenti compresi, presso cui è stata dichiarata la Direttiva.

BLOCCO 2

```
<div class="itemImg">
    <img class="front" ng-src="{{item.Src}}"
alt="{{ALT_divClass_itemBadge(item)}}" />
    <div ng-if="IF_divClass_itemBadge(item)" class="itemBadge">
        <img ng-src={{item.BadgeSearchResult.Img}} alt="" />
    </div>
</div>
```

La Direttiva `ng-src`, come `ng-href`, viene utilizzata per rimandare la valutazione dell'attributo `src` da parte del browser fino quando non è avvenuta l'interpolazione con il dato dinamico.

L'attributo alt viene definito dinamicamente dalla funzione `ALT_divClass_itemBadge(item)` definita nello `$scope`.

La Direttiva `ng-if="IF_divClass_itemBadge(item)"` viene utilizzata per stabilire se debba essere eliminato dal DOM l'elemento stesso (e discendenti) su cui si trova la Direttiva nel caso in cui la funzione `IF_divClass_itemBadge(item)` restituisca `false`.

Abbiamo tradotto il blocco condizionale `@if` nella direttiva `ng-if` e non in `ng-show` poiché mentre `ng-if` rimuove l'elemento dal DOM, `ng-show` si limita a nascondere l'elemento con la proprietà `CSS display: none`. Non è pratica consigliata tenere in pagina codice non utilizzato, anche dal punto di vista della leggibilità del codice, quindi `ng-if` faceva al caso nostro.

Come nel template Razor anche qua BLOCCO 3, BLOCCO 4, BLOCCO 5, BLOCCO 6 interni a `<div class="itemData">`.

BLOCCO 3

```
<span class="itemInfos">
    <span class="brand infoRow
fontBold">{{spanClass_brand(item)}}</span>
    <span ng-if="IF_spanClass_auth(item, area, department)"
class="auth infoRow">{{item.Author}}</span>
    <span ng-if="IF_spanClass_infoRow(item)"
class="infoRow">{{item.Brand}}</span>
    <span class="microCategory
infoRow">{{spanClass_microCategory(item)}}</span>
</span>
```

`spanClass_brand`, `IF_spanClass_auth`, `spanClass_microCategory` sono funzioni definite nello `$scope`.

Ancora una volta abbiamo tradotto 2 blocchi condizionali `@if` in ngDirettive `ng-if`.

BLOCCO 4

```
<div ng-init="IF1 = IF_divClass_currentprice(item); IF2 =
IF_div2Class_currentprice(item)" class="itemInfos">
    <div ng-if="IF1" class="currentprice infoRow
fontBold">{{item.SoldoutLabel}}</div>
    <div ng-if="!IF1 && IF2" class="currentprice infoRow
fontBold">{{item.FullPriceStr}}</div>
    <div ng-if="!IF1 && !IF2" class="oldprice
infoRow">{{item.FullPriceStr}}</div>
    <div ng-if="!IF1 && !IF2" class="currentprice infoRow
fontBold">{{item.DiscountedPriceStr}}</div>
</div>
```

Abbiamo utilizzato nuovamente la Direttiva `ng-init`: in questo caso per non dover invocare per ogni elemento le stesse funzioni più volte, ed eseguire gli stessi calcoli.

`IF_divClass_currentprice`, `IF_div2Class_currentprice` sono funzioni definite nello `$scope`.

Da un altro punto di vista è migliore anche per leggibilità del codice.

BLOCCO 5

```
<div ng-if="IF_divClass_colorInfo(item)" class="itemInfos
colorInfo" ng-init="styles=colorsFilter(item.styles)">
    <span ng-repeat="style in styles" class="aColor"
style={{style}}></span>
    <span ng-if="IF_spanClass_moreColorSize(item)"
class="moreColorSize fontBold">+</span>
</div>
```

Utilizzando ng-init e la funzione colorsFilter definita nello `$scope` vengono salvati i colori da visualizzare. Se i colori sono oltre il numero consentito, allora viene aggiunto il simbolo '+'.

`IF_divClass_colorInfo`, `IF_spanClass_moreColorSize` sono funzioni definite nello `$scope`.

BLOCCO 6

```
<div ng-if="IF_divClass_sizeInfo(item)" class="itemInfos sizeInfo"
ng-init="sizes=sizesFilter(item.Sizes)">
    <span ng-repeat="size in sizes"
class="aSize">{{size.Text}}</span>
    <span ng-if="IF_span2Class_moreColorSize(item)"
class="moreColorSize fontBold">+</span>
</div>
```

E' identico a BLOCCO 5, a parte le funzioni associate alle Direttive: `IF_divClass_sizeInfo, sizesFilter`, `IF_span2Class_moreColorSize` tutte definite nello `$scope`.

BLOCCO 7

```
<div id="infiniteScrollBlock" ng-show='item_gallery.busy'></div>
```

Come si può osservare, è diventato assai minimale.
Non c'è alcun bisogno di salvare le informazioni relative all'URL del prossimo file JSON. E' già tutto contenuto nel JSON attuale, già salvato nella memoria dell'applicazione.

Per quanto riguarda la Direttiva ng-show invece, viene utilizzata per mostrare `<div id="infiniteScrollBlock">`, ossia la GIF di caricamento, quando si è in attesa della prossima pagina.

Ora che abbiamo tradotto Razor -> AngularJS occorre fare una precisazione. Dalla fase di progettazione, avevamo detto per motivi SEO e di performance, la prima pagina di una ricerca deve sempre sessere servita dal server.

Quindi occorre che la pagina web ricevuta in risposta dal server (o più probabilmente da un POP della CDN) contenga sia il template di Razor, sia il template di AngularJS.

In questo modo possiamo ricevere, da un nodo della CDN, una pagina web contenente la prima pagina di navigazione relativa alla ricerca compiuta dall'utente.
In più, nella pagina web ricevuta, è anche presente il template AngularJS, che entra in funzione a partire dal primo Infinite Scrolling.

Nello specifico, il template Razor viene utilizzato quando l'utente compie una ricerca senza "anchor path", la parte di URL preceduta da '#'.

Dal momento che entrambi i template convivono nello stesso file, occorre un meccanismo per nascondere l'uno o l'altro a seconda della necessità.

CASO 1

L'utente si trova sulla prima pagina di una ricerca, il template AngularJS deve essere nascosto e il template Razor, bindato dinamicamente, deve essere visibile. Nel caso in cui l'utente faccia uno scroll occorre mostrare anche il template AngularJS.

CASO 2

Invece nel caso in cui l'utente compia un ricerca con un URL avente "anchor path", occorre mostrare solo il template AngularJS.
Per questo occorrono un `ng-if` per omettere /costruire il DOM del template Razor e un `ng-show` per mostrare/ nascondere a livello di CSS il template AngularJS.

La scelta delle Direttive ha uno scopo preciso: ng-show, per il caso 1, ci permette di nascondere prima il template AngularJS, quando l'utente si trova sulla prima pagina di una ricerca, poi di mostrarlo al primo Infinite Scrolling.

D’altro canto per il template Razor bisogna utilizzare `ng-if`: se non viene utilizzato per la prima pagina di una ricerca, non verrà mai più utilizzato per la ricerca corrente. Non c’è alcun vantaggio a tenere il template Razor in pagina se non utilizzato e quindi si elimina dal DOM.

Quindi la situazione che si presenta è la seguente, con `ng-show=”show_angular()”` e `ng-if=”server_page()”`:

```
@using YOOX.Web.Mobile.SearchResults
@model SearchResultViewModel

<div ng-app="itemGalleryApp" ng-
controller="itemGalleryController">
    <div id="itemsGrid" infinite-scroll='item_gallery.nextPage()'
infinite-scroll-disabled='item_gallery.loading_json'>

        <div ng-if="server_page()">
            @foreach (var item in Model.Items){
            // <a href=’objectpageURL’>
            //     <div class="itemImg"> </div> (immagine oggetto)
            //     <div class="itemData">
            //         <span class="itemInfos"></div> (descriz)
            //         <div class="itemInfos"></div> (prezzo)
            //         colori
            //         taglie
            //     </div>
            // </a>
            }
            <div id="infiniteScrollBlock"></div>
        </div>

        <div ng-show="show_angular()">
            // <a href=’objectpageURL’>
            //     <div class="itemImg"> </div> (immagine oggetto)
            //     <div class="itemData">
            //         <span class="itemInfos"></div> (descriz)
            //         <div class="itemInfos"></div> (prezzo)
            //         colori
            //         taglie
            //     </div>
            // </a>
        </div>
        <div id="infiniteScrollBlock" ng-
show='item_gallery.loading_img'></div>
</div>
```

A questo punto, avendo concluso con la View, ci inoltriamo nella business logic e nella view logic dell'applicazione, preso il lato JS.

6.3 AngularJS View Logic

La logica legata alla View, cio è il template, è costituita da quelle funzioni che creano/ eliminano nodi del DOM, insieme a quelle che regolano l'applicazioni di stili diversi, cioè il CSS. Riassumendo, è la logica legata alla rappresentazione del contenuto.

BLOCCO 1

```
$scope.VAR_microCategory = function(item) {
    if (!(item.AlternativeMicro === null || item.AlternativeMicro
=== "")) return item.AlternativeMicro;
    else if (!(item.MicroCategoryPlural === null ||
item.MicroCategoryPlural === "")) return item.MicroCategoryPlural;
    else return item.MicroCategory;
}
```

BLOCCO 2

```
$scope.IF_divClass_itemBadge = function(item) {
    if (item.BadgeSearchResult !== null &&
item.BadgeSearchResult.IsActive && !(item.BadgeSearchResult.Img
=== null || item.BadgeSearchResult.Img === "")) return true;
    return false;
}

$scope.ALT_divClass_itemBadge = function(item) {
    return item.Brand + " - " + item.MicroCategory;
}
```

BLOCCO 3

```
$scope.spanClass_brand = function(item) {
    if (item.Title === null || item.Title === "") return
item.Brand;
    return item.Title;
}

$scope.IF_spanClass_auth = function(item, Area, Department) {
    if (!(item.Author === null || item.Author === "") &&
(Department === "books" || Area === "design" ||
Department.endsWith("art"))) return true;
    return false;
}

$scope.IF_spanClass_infoRow = function(item) {
    if (!(item.Title === null || item.Title === "")) return true;
    return false;
}

$scope.spanClass_microCategory = function(item) {
    return $scope.VAR_microCategory(item);
}
```

BLOCCO 4

```
$scope.IF_divClass_currentprice = function(item) {
    if (item.IsSoldout) return true;
    return false;
}

$scope.IF_div2Class_currentprice = function(item) {
    if (item.FullPrice === item.DiscountedPrice) return true;
    return false;
}
```

BLOCCO 5

```
$scope.any = function(array) {
    if (typeof array !== "undefined" && array !== null &&
array.length > 0) return true;
    return false;
}

$scope.IF_divClass_colorInfo = function(item) {
    var nocolor = false;
    for (var i = 0; i < item.Colors; i++) {
        if (item.Colors[i] === '(-)') {
            var nocolor = true;
            break;
        }
    }
    if (item.Colors !== null && $scope.any(item.Colors) &&
!(nocolor && item.Colors.length === 1)) return true;
    return false;
}

$scope.count = function(array) {
    var char = 0;

    for (i in array) {
        if (typeof array[i] !== "undefined")
            char++;
    }
    return char;
}

$scope.IF_spanClass_moreColorSize = function(item) {
    var colors = item.Colors;

    for (var i = 0; i < colors.length; i++) {
        if (colors.indexOf(colors[i]) === 4 &&
$scope.count(colors) > 5) return true;

        return false;
    }
}
```

```
$scope.colorsFilter = function(styles) {
    var selectStyles = [];
    for (var i = 0; i < styles.length; i++) {
        if (styles.indexOf(styles[i]) === 4 &&
$scope.count(styles) > 5) break;
        selectStyles.push(styles[i]);
    }
    return selectStyles;
}
```

BLOCCO 6

```
$scope.IF_divClass_sizeInfo = function(item) {
    if ($scope.any(item.Sizes)) return true;
    return false;
}

$scope.IF_span2Class_moreColorSize = function(item) {
    var sizes = item.Sizes;
    var strCnt = 0;
    for (var i = 0; i < sizes.length; i++) {
        strCnt += sizes[i]["Text"].length + 1;

        if (strCnt > 16) return true;
    }
    return false;
}

$scope.sizesFilter = function(sizes) {
    var strCnt = 0;
    var selectSizes = [];
    for (var i = 0; i < sizes.length; i++) {
        strCnt += sizes[i]["Text"].length + 1;

        if (strCnt > 16) break;
        selectSizes.push(sizes[i]);
    }
    return selectSizes;
}
```

6.4 AngularJS Business Logic

```
myApp.controller('itemGalleryLogicController', function($scope,
ItemGallery) {
```

Creiamo il Controller `itemGalleryLogicController` che fungerà da contenitore della business logic dell'applicazione. `ItemGallery` è una funzione Factory, che vedremo successivamente, da cui dipende questo Controller.

Ci inoltriamo ora in `itemGalleryLogicController.`

```
$scope.item_gallery = new ItemGallery();
var server_page;

$scope.server_page = function() {
    if (server_page) return server_page;

    if (!window.location.hash) server_page = true;
    else server_page = false;

    return server_page;
}
```

Viene definita la proprietà `item_gallery` dello `$scope` e viene dichiarata la variabile `server_page`.

La funzione `$scope.server_page` imposta la variabile server_page solo una volta, quando viene ricevuta la pagina web contenenente i 2 template.

Se l'URL contiene un simbolo '#' viene impostato `server_page=false`, restituito `false`, e quindi viene omesso dal DOM il template Razor (`ng-if`).

Se l'URL non contiene il simbolo '#', come *http://mobile.yoox.com/it/uomo/shoponline*, viene impostato `server_page=true`, restituito `true`, e viene visualizzato il template Razor perché la prima pagina arriva dalla cache della CDN.

```
$scope.show_angular = function() {
    if (window.location.hash) return true;
    return false;
}
```

Contrariamente a `$scope.server_page`, in `$scope.show_angular` se l'URL contiene il simbolo '#', allora deve essere restituito true per mostrare il template AngularJS; altrimenti si restituisce false per nasconderlo (`ng-show`).

6.4.1 L'evento del browser *onhashchange*

```
$(window).on('hashchange', function(event) {

    var strip_page_param = function(url) {
        return url.replace(/&page=[0-9a-zA-Z]*/g, "");
    }

    var next_json_url = $scope.item_gallery.domain_url +
window.location.hash.replace("#/", "");
    if (next_json_url.indexOf("&infiniteScroll=true") == -1)
        next_json_url += "&infiniteScroll=true";
    if (next_json_url.indexOf("&json=true") == -1)
        next_json_url += "&json=true";

    var current_json_url = $scope.item_gallery.current_json_url;

    if (current_json_url != next_json_url) {
        var last_href =
strip_page_param($scope.item_gallery.data_search[0].Pagination.Cur
rentPage.Href);
        var new_href = strip_page_param(location.href);

        if (last_href != new_href) {
            $scope.item_gallery.data_items = [];
            $scope.item_gallery.data_search = [];
        }

        $scope.item_gallery.json_url = next_json_url;
        window.scroll(0, 50000);
    }
});
```

Questa è una funzione associata all'evento del browser `onhashchange`, ed è stata implementata per consentire di effettuare ricerche tramite la barra degli indirizzi.

Si cerca di confrontare 2 URL, il vecchio e quello appena cambiato di valore '#', per capire se si tratta di una ricerca diversa o solo un cambio di pagina all'interno della stessa ricerca. L'idea è che togliendo l'attributo per la pagina, se i 2 URL differiscono, allora si stratta di una nuova ricerca e occorre resettare l'item gallery, cioè eliminare i vecchi oggetti e ripartire coi nuovi.

Innanzi tutto viene assegnata una funzione alla variabile `strip_page_param;` questa funzione elimina da una stringa in ingresso, url, le stringhe composte da '&page=' seguita da qualsiasi quantità di numeri e/o lettere.

Successivamente viene creato un link per `next_json_url,` costituito dalla parte dell'URL antecedente al simbolo '#' `(http://mobile.yoox.com/it/uomo/shoponline)` quindi fino a 'shoponline'; poi gli viene aggiunto in coda prima il simbolo '?' e poi l'anchor path attuale, escluso '#/'.

A questo punto vengono aggiunte in coda a `next_json_url` le stringhe "&infiniteScroll=true" e "&json=true", se non sono già presenti.

Si salva l'URL precedente nella variabile current_json_url e si confronta con next_json_url: se risultano diversi significa che è stata cambiata ricerca o pagina di navigazione.

Quindi si procede ad utilizzare la funzione assegnata a `strip_page_param`, per entrambi gli URL.
A questo punto vengono riconfrontati gli URL: se sono diversi significa che si è compiuta un'altra ricerca, se sono uguali significa che è stata cambiata pagina.

In caso URL differenti prima verrà re-inizializzato l'array contenente gli oggetti dell'item gallery.

Successivamente, in entrambi i casi verrà effettuata uno scroll verso il basso fino a raggiungere la fine della pagina web, il qual comportamento attiverà l'Infinite Scrolling e sarà inviata una chiamata web per il JSON relativo.

Per fare un esempio: ecco una serie di link.

URL

```
http://mobile.yoox.com/it/uomo/shoponline/infradito_c#/dept=shoesm
en&gender=U&page=2&attributes=%7b%27ctgr%27%3a%5b%27nfrdt%27%5d%7d
&season=X&infiniteScroll=true&json=true
```

chiamata JSON

```
http://mobile.yoox.com/it/shoponline?dept=shoesmen&gender=U&page=2
&attributes=%7B%27ctgr%27%3A%5B%27nfrdt%27%5D%7D&season=X&infinite
Scroll=true&json=true
```

URL differente

```
http://mobile.yoox.com/it/uomo/shoponline/mocassini_c#/dept=shoesm
en&gender=U&page=2&attributes=%7b%27ctgr%27%3a%5b%27mcssn%27%5d%7d
&season=X&infiniteScroll=true&json=true
```

6.4.2 Il Servizio di tipo Factory *ItemGallery*

Procediamo oltre ed analizziamo il Servizio di tipo Factory `ItemGallery`

```
myApp.factory('ItemGallery', function($http) {
                var ItemGallery = function() {

                    this.data_search = [];
                    this.busy = false;
                    this.data_items = [];
                    this.domain_url =
window.jsInit.links.sr.ajaxUrl + "?";
                    this.json_url;
                    this.current_json_url;
                    this.Area;
                    this.Department;
                    this.response;

                    if (!window.location.hash) {
                        window.onload = function() {
                            this.nextPageQueryString =
document.getElementById("infiniteScrollLink").rel;
                            this.json_url = this.domain_url +
this.nextPageQueryString;
                    };

                        this.json_url = this.domain_url +
window.location.hash.replace("#/", "");

                        if
(this.json_url.indexOf("&infiniteScroll=true") == -1)
this.json_url += "&infiniteScroll=true";
```

```
                        if (this.json_url.indexOf("&json=true") ==
-1) this.json_url += "&json=true";
                    }
                };

                return ItemGallery;
});
```

Una funzione Factory è una funzione all'interno della quale viene definito uno oggetto e viene restituito l'oggetto stesso: in questo caso, `ItemGallery`.

Analizziamo un po' di variabili:

```
this.data_search = [];
```

Contiene tutti i file JSON

```
this.busy = false;
```

Si riferisce allo stato dell'Infinite Scrolling

```
this.data_items = [];
```

Contiene tutti gli oggetti della galleria

```
this.domain_url = window.jsInit.links.sr.ajaxUrl + "?";
```

Contiene il path dell'URL + '?'

```
this.json_url;
```

rappresenta l'URL al JSON della pagina successiva

```
this.current_json_url;
```

rappresenta l'URL al JSON della pagina corrente

```
this.Area;
```

la valore della chiave 'Area' del file JSON

```
this.Department;
```

il valore della chiave 'Department' del file JSON

```
this.response;
```

la risposta ricevuta dal server (contiene il file JSON)

Quelli a seguire sono blocchi condizionali che vengono eseguit solo una volta, quando viene ricevuta la pagina web dal client.

Se l'URL corrente non ha alcun simbolo '#' allora significa che abbiamo ricevuta la prima pagina dalal cache della CDN. La pagina web è stata creata con Razor e il link alla prossima pagina si ricava dall'attributo `rel` dell'elemento #infiniteScrollLink, che viene salvato in `this.nextPageQueryString`.

A questo punto si prepara l'URL per la pagina successiva `this.json_url`, unendo `this.domain_url` e `this.nextPageQueryString`.

Altrimenti se l'URL contiene un anchor path, l'URL per la pagina successiva `this.json_url` è dato da this.domain_url più l'anchor path senza '#/'.
Se non sono presenti vengono aggiunti "&infiniteScroll=true" e "&json=true" in coda a `this.json_url`.

6.4.3 La funzione *ItemGallery.nextPage*

In ultima analisi troviamo la funzione nextPage, relativa all'oggetto `ItemGallery`, definita internamente alla funzione Factory `ItemGallery`.

Questa funzione viene attivata dall'Infinite Scrolling, in particolare da ngInfiniteScroll.

```
ItemGallery.nextPage = function($scope) {
    if (this.busy) return;

    if ((this.data_search.length == 0) || !(this.json_url == ""))
{

        this.busy = true;
        this.current_json_url = this.json_url;

        console.log(this.json_url);
        $http.get(this.json_url)
            .then(function(response) {

                console.log("ajax to " + this.json_url);
                console.log(response.data);

                this.data_search.push(response.data);
                var new_items = response.data.Items;

                new_items.forEach(function(item) {
```

```
                    item["id"] = 'item_'.concat(item['Cod10']);
                    item["Cod8"] = item.Cod10.substring(0, 8);
                    item["DiscountedPriceEur"] =
item.DiscountedPriceEur.toString();
                    item["alt"] = item.Brand + " - " +
item.MicroCategory;

                    item["styles"] = new Array();
                    for (var i = 0; i < item["Colors"].length;
i++) {
                        item["styles"][i] = "background-color:
#".concat(item.Colors[i].Rgb);
                    }
                });

                for (var i = 0; i < new_items.length; i++) {
                    this.data_items.push(new_items[i]);
                }

                this.Area = response.data.Area;
                this.Department = response.data.Department;


                if (!(response.data.Pagination.NextPage == null))
{
                    this.json_url = this.domain_url +
response.data.Pagination.NextPage.OnClick +
"&infiniteScroll=true&json=true";
                } else {
                    this.json_url = "";
                }

                var response_hash = "/" +
response.data.Pagination.CurrentPage.OnClick +
"&infiniteScroll=true&json=true";
                if (location.hash != response_hash &&
!(this.current_json_url.indexOf("&page=") == -1)) {
                    location.hash = response_hash;
                }

                this.busy = false;

            }.bind(this));
    };
};
```

1) Se l'Infinite Scrolling è attivo, esci dalla funzione

2) Se il link alla prossima pagina è non nullo o il numero dei file JSON ricevuto è pari a 0, si può entrare nella fase di Infinite Scrolling

3) Viene settatta `this.busy= true` per bloccare altri tentativi di Infinite Scrolling e viene settata la variaibile che rappresentava l'URL al JSON prossimo, con l'URL al JSON corrente.
   `this.current_json_url = this.json_url`;

4) Viene salvato il file JSON appena ricevuto

5) Vengono estratti gli oggetti, della galleria, dal file JSON

6) Vengono creati per ogni oggetto i valori per le chiavi `'id'`, `'Cod8'`, `'DiscountedPriceEur'`, `'alt'`, `'CurrentPrice'`,`'styles'`

7) Gli oggetti vengono aggiunti all'array `this.data_items`

8) Vengono salvate le proprietà `Area` e `Department` che serviranno nel template

9) Se l'URL alla prossima pagina è non nullo viene settato l'URL per la prossima chiamata web al file JSON relativo, altrimenti viene settato come stringa vuota

10) Occorre salvare lo stato di navigazione dell'utente: se l'URL corrente e quello nuovo situato nel JSON sono differenti e se l'URL del JSON contiene la stringa '&page=' allora viene salvato il nuovo anchor path nella barra degli indirizzi.

11) Infine viene settato `this.busy=false`, l'Infinite Scrolling non è più bloccato.

12) Viene utilizzato il metodo `.bind(this)` per racchiudere il corpo del servizio `$http`: essendo esso asincrono, all'interno di esso si perde il contesto e gli oggetti riferiti con `this` risultano 'undefined'.
    Possiamo bindare il contesto della funzione al contesto del servizio, per risolvere questo problema; in questo modo, internamente al servizio, possiamo accedere al valore reale degli oggetti riferiti con `this`.

6.5 Il Template AngularJS e le Direttive

Sono state utilizzate le Direttive per organizzare il codice del template in parti relative ai dettagli di un oggetto: l'immagine, la descrizione, il prezzo, i colori e le taglie disponibili.
La parte di template AngularJS interna all'elemento HTML `<a>` assume così la seguente forma.

```
<div class="itemImg" itemimg></div>

<div class="itemData">
    <div iteminfos></div>
    <div price></div>
    <div colorinfo></div>
    <div sizeinfo></div>
</div>
```

Come si può notare le Direttive sono state utilizzate come attributo di elementi HTML. Il template assume decisamente un aspetto più pulito, oltre che minimale e conciso.

Procediamo dunque ad esplorarne l'implementazione lato JS.

6.5.1 Direttiva *itemimg*

```
itemGallery.directive('itemimg', function() {

    return {
        templateUrl: 'itemimgDirectiveFile.html',
        restrict: 'A'
    };
});
```

La Direttiva `itemimg`, come le altre Direttive a seguire, è stata implementata attraverso l'utilizzo di 2 opzioni di configurazione:

- `templateUrl`: indica il percorso del file contenente la parte di testo o HTML da inserire come nodo discendente del nodo HTML presso cui è stata dichiarata la Direttiva `itemimg`.

- `restrict`: indica che la Direttiva è limitata ad essere utilizzata solo come attributo di nodi HTML

itemimgDirectiveFile.html

```
<img class="front" ng-src="{{item.Src}}"
alt="{{ALT_divClass_itemBadge(item)}}" />
<div ng-if="IF_divClass_itemBadge(item)" class="itemBadge">
        <img ng-src={{item.BadgeSearchResult.Img}} alt="" />
</div>
```

6.5.2 Direttiva *iteminfos*

```
itemGallery.directive('iteminfos', function() {

    return {
        templateUrl: 'iteminfosDirectiveFile.html’,
        restrict: 'A'
    };
});
```

iteminfosDirectiveFile.html

```
<span class="itemInfos">

        <span class="brand infoRow
fontBold">{{spanClass_brand(item)}}</span>
        <span ng-if="IF_spanClass_auth(item, area, department)"
class="auth infoRow">{{item.Author}}</span>
        <span ng-if="IF_spanClass_infoRow(item)"
class="infoRow">{{item.Brand}}</span>
        <span class="microCategory
infoRow">{{spanClass_microCategory(item)}}</span>

</span>
```

6.5.3 Direttiva *price*

```
itemGallery.directive('price', function() {

    return {
        template: 'priceDirectiveFile.html’,
        restrict: 'A'
    };
});
```

priceDirectiveFile.html

```
<div ng-init="IF1 = IF_divClass_currentprice(item); IF2 =
IF_div2Class_currentprice(item)" class="itemInfos">

    <div ng-if="IF1" class="currentprice infoRow
fontBold">{{item.SoldoutLabel}}</div>
    <div ng-if="!IF1 && IF2" class="currentprice infoRow
fontBold">{{item.FullPriceStr}}</div>
    <div ng-if="!IF1 && !IF2" class="oldprice
infoRow">{{item.FullPriceStr}}</div>
    <div ng-if="!IF1 && !IF2" class="currentprice infoRow
fontBold">{{item.DiscountedPriceStr}}</div>

</div>
```

6.5.6 Direttiva *colorinfo*

```
itemGallery.directive('colorinfo', function() {

    return {
        template: 'colorinfoDirectiveFile.html',
        restrict: 'A'
    };
});
```

colorinfoDirectiveFile.html

```
<div ng-if="IF_divClass_colorInfo(item)" class="itemInfos
colorInfo" ng-init="styles=colorsFilter(item.styles)">

    <span ng-repeat="style in styles" class="aColor"
style={{style}}></span>
    <span ng-if="IF_spanClass_moreColorSize(item)"
class="moreColorSize fontBold">+</span>

</div>
```

6.5.7 Direttiva *sizeinfo*

```
itemGallery.directive('sizeinfo', function() {

    return {
        template: 'sizeinfoDirectiveFile.html',
        restrict: 'A'
    };
});
```

sizeinfoDirectiveFile.html

```
<div ng-if="IF_divClass_sizeInfo(item)" class="itemInfos sizeInfo"
ng-init="sizes=sizesFilter(item.Sizes)">

    <span ng-repeat="size in sizes"
class="aSize">{{size.Text}}</span>
    <span ng-if="IF_span2Class_moreColorSize(item)"
class="moreColorSize fontBold">+</span>

</div>
```

Capitolo 7

Performance e miglioramenti

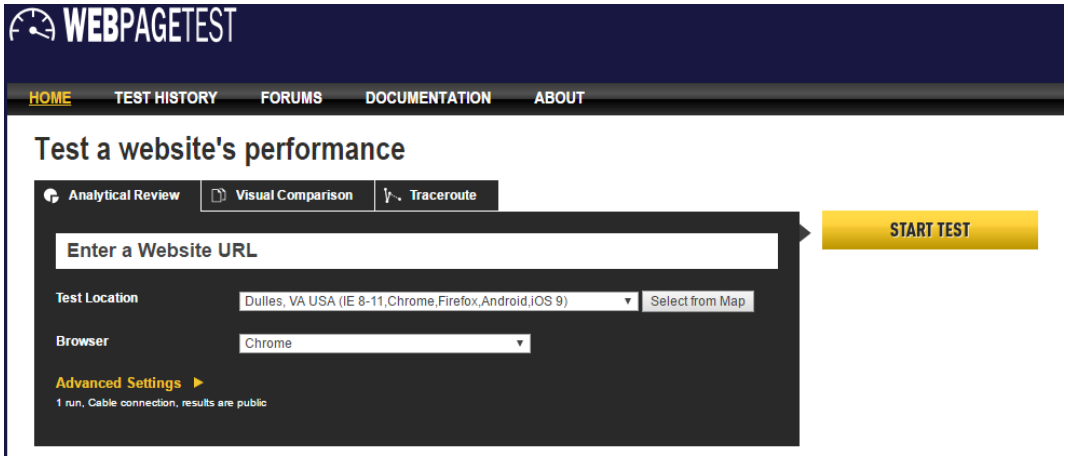
Introduzione

Una parte importante del percorso di tirocinio, una volta portata a termine l'implementazione, ha coinvolto l'analisi delle performance della soluzione client-side: sia in termini di tempi di risposta per quanto riguarda l'Infinite Scrolling, sia come quantità di dati in download.

Per quanto riguarda gli ambienti di test delle performance sono stati presi in considerazione sia l'ambiente locale che uno strumento online, WebPagetest.

Faremo un'analisi delle differenze tra le performance delle 2 soluzioni e proveremo, laddove possibile, a migliorare le prestazioni della soluzione client-side.

7.1 WebPagetest



E' uno strumento open source, disponibile come applicazione web e desktop, in grado di compiere una o più richieste di pagine web e dare in risposta all'utente un'analisi grafica, dettagliata, per tali richieste.

Tra le opzioni di esecuzione di questo strumento c'è la possibilità di selezionare diverse zone geografiche e diversi tipi di dispositivi mobile.

Abbiamo proceduto a mettere online entrambi gli ambienti di produzione, sia la soluzione server-side che quella client-side. Sì anche quella server-side: poichè per le pagine servite da yoox.com è attiva la CDN, avremmo potetuto avere risultati sfasati.

Lo strumento purtroppo, per mia personale esperienza, è di assoluta inaffidabilità. I test per le pagine web sono stati eseguiti sempre dalla stessa posizione geografica, Dulless VA USA, tuttavia i tempi di risposta andavano da 2 secondi fino a 25 secondi.

Secondariamente, riprovando a testare la stessa pagina, questa si trovava ancora in cache del nodo remoto: solo in questo caso i tempi di risposta erano inferiori a 1 secondo.

Un altro problema riguarda i tempi biblici in cui si rimaneva in attesa che il nodo prendesse in carico la nostra chiamata web: già solo con 4 richieste web in coda, provenienti da altri utenti, occorreva aspettare almeno 30 minuti.

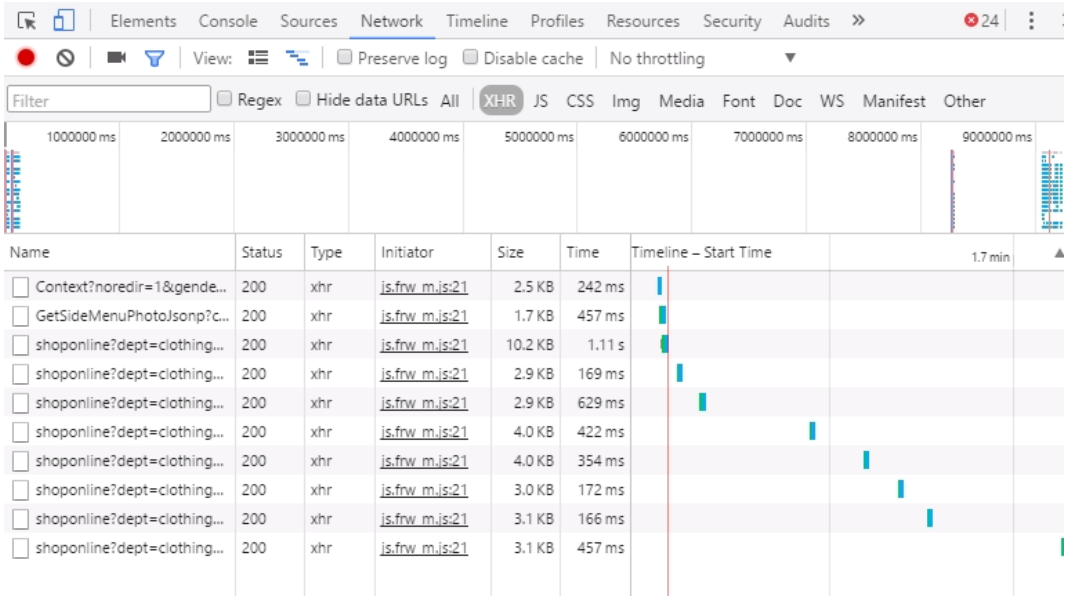
Infine, Webpagetest permette di misurare le performance di sole pagine web.
La soluzione client-side appena sviluppata prevede il download della pagina web una sola volta, all'inizio, quando viene compiuta la ricerca: successivamente vengono scaricati solo file JSON. Ciò non è possibile con WebPagetest.

7.2 Ambiente locale

Dal momento che non sono stati trovati strumenti disponibili in rete per misurare le performance di una SPA con Infinite Scrolling, si è deciso di procede in ambiente locale.

Dunque per ogni sessione di test, per le soluzioni software server-side e client-side, si è proceduto prima a pulire la cache del browser, poi ad effettuare i test tenenedo sotto controllo la situazione attraverso il pannello “Strumenti per sviluppatori” di Chrome.

Di seguito viene riportata una serie di chiamate web per la versione online di yoox.com, solo a titolo figurativo del pannello “Strumenti per sviluppatori” di Chrome:



Quindi, come si può vedere, ci si è posizionati sul pannello XHR (XMLHttpRequest), sono state prelevate le richieste completate con successo (Status 200) e di queste sono state analizzate il tempo di risposta e la dimensione in termini di KB.

Procediamo quindi con i test per la soluzione server-side. L’URL per una richiesta web si presenta come:

http://mobile.yoox.com/it/shoponline?dept=clothingwomen&gender=D&page=22&attributes=%7B%27ctgr%27%3A%5B%27vstt%27%5D%7D&season=X&infiniteScroll=true

Per una leggibilità migliore di seguito indicheremo una richiesta web con il solo attributo “page” e relativo valore.

Name	Status	Type	Initiator	Size	Time
page=22	200	xhr	jquery-1,8,2,js:8416	31,9 KB	1,09 s
page=23	200	xhr	jquery-1,8,2,js:8416	30,4 KB	1,42 s
page=24	200	xhr	jquery-1,8,2,js:8416	30,8 KB	1,18 s
page=25	200	xhr	jquery-1,8,2,js:8416	31,7 KB	1,38 s
page=26	200	xhr	jquery-1,8,2,js:8416	29,1 KB	1,36 s
page=27	200	xhr	jquery-1,8,2,js:8416	30,5 KB	1,61 s
page=28	200	xhr	jquery-1,8,2,js:8416	30,1 KB	1,31 s
page=29	200	xhr	jquery-1,8,2,js:8416	29,7 KB	0,936 s
page=30	200	xhr	jquery-1,8,2,js:8416	31,0 KB	0,957 s
page=31	200	xhr	jquery-1,8,2,js:8416	30,4 KB	1,57 s
page=32	200	xhr	jquery-1,8,2,js:8416	30,7 KB	1,74 s
page=33	200	xhr	jquery-1,8,2,js:8416	31,4 KB	1,31 s
page=34	200	xhr	jquery-1,8,2,js:8416	30,1 KB	1,21 s
page=35	200	xhr	jquery-1,8,2,js:8416	30,5 KB	1,56 s
page=36	200	xhr	jquery-1,8,2,js:8416	31,0 KB	1,4 s
page=37	200	xhr	jquery-1,8,2,js:8416	31,9 KB	1,15 s
page=38	200	xhr	jquery-1,8,2,js:8416	31,6 KB	1,34 s
page=39	200	xhr	jquery-1,8,2,js:8416	30,1 KB	1,08 s
page=40	200	xhr	jquery-1,8,2,js:8416	31,2 KB	1,39 s
page=41	200	xhr	jquery-1,8,2,js:8416	30,1 KB	0,889 s

Di tutti questi valori relativi ai parametri “Size” (dimensione) e “Time” (tempo di risposta), è stata calcolata la somma e la media.

	SOMMA	MEDIA
Time	25,9 s	1,3 s
Size	614,2 KB	30,7 KB

In totale, per 20 pagine web della stessa ricerca, di cui 20 caricate per Infinite Scrolling sono stati impiegati: 25,9 secondi in download, mediamente 1,3 secondi per pagina; sono stati scaricati 614,2 KB di dati (solo relative alle pagine, tralasciando JS, CSS e immagini), mediamente 30,7 KB per pagina.

Vediamo ora le performance per la soluzione client-side. L’URL per una richiesta web si presenta come:

http://mobile.yoox.com/it/shoponline?dept=clothingwomen&gender=D&page=22&attributes=%7B%27ctgr%27%3A%5B%27vstt%27%5D%7D&season=X&infiniteScroll=true&json=true

Name	Status	Type	Initiator	Size	Time
page=22	200	xhr	angular,min,js:99	13,6 KB	1,09 s
page=23	200	xhr	angular,min,js:99	15,1 KB	1,42 s
page=24	200	xhr	angular,min,js:99	14,6 KB	1,18 s
page=25	200	xhr	angular,min,js:99	15 KB	1,38 s
page=26	200	xhr	angular,min,js:99	14,1 KB	1,36 s
page=27	200	xhr	angular,min,js:99	13,9 KB	1,61 s
page=28	200	xhr	angular,min,js:99	13,8 KB	1,31 s
page=29	200	xhr	angular,min,js:99	13,8 KB	0,936 s
page=30	200	xhr	angular,min,js:99	12,8 KB	0,957 s
page=31	200	xhr	angular,min,js:99	13,7 KB	1,57 s
page=32	200	xhr	angular,min,js:99	13,5 KB	1,74 s
page=33	200	xhr	angular,min,js:99	13,4 KB	1,31 s
page=34	200	xhr	angular,min,js:99	13,6 KB	1,21 s
page=35	200	xhr	angular,min,js:99	13,2 KB	1,56 s
page=36	200	xhr	angular,min,js:99	13,1 KB	1,4 s
page=37	200	xhr	angular,min,js:99	13,6 KB	1,15 s
page=38	200	xhr	angular,min,js:99	13,2 KB	1,34 s
page=39	200	xhr	angular,min,js:99	13,7 KB	1,08 s
page=40	200	xhr	angular,min,js:99	13,5 KB	1,39 s
page=41	200	xhr	angular,min,js:99	13,1 KB	0,889 s

	SOMMA	MEDIA
Time	20,2 s	0,92 s
Size	287,5 KB	13,7 KB

Per le stesse pagine web testate per la soluzione server-side, sono stati impiegati in totale 20,2 secondi in download, mediamente 0,92 secondi per pagina; sono stati scaricati 287,5 KB di dati (solo relative alle pagine, tralasciando JS, CSS e immagini), mediamente 13,7 KB per pagina.

Differenza Time

Somma: 25,9 s (server-side) – 20,2 s (client-side) = 5,7 secondi

Media: 1,3 s (server -side) – 0,92 s (client-side) = 0,38 secondi

In totale, utilizzando la soluzione client-side, durante la navigazione di 20 pagine di risultati di una ricerca vengono impiegati 5,7 secondi in meno, mediamente 0,38 secondi in meno per pagina.

Differenza Size

Somma: 614,2 KB (server-side) – 287,5 KB (client-side) = 326,7 KB

Media: 30,7 KB (server-side) – 13,7 KB (client-side) = 17 KB

In totale, utilizzando la soluzione client-side, durante la navigazione di 20 pagine di risultati di una ricerca vengono scaricati 326,7 KB in meno, mediamente 17 KB in meno per pagina.

Osservazioni

Occorre riflettere su queste differenze tenendo a mente che mobile.yoox.com riceve migliaia di richieste web da tutte le parti del mondo.

Giusto per provare a dare qualche numero: se sono collegati allo store di mobile.yoox.com un numero pari a 4000 utenti che si inoltrano fino alla 20esima pagina di una ricerca, vediamo che:

Server side

30,7 KB (peso medio di una pagina web) x 4000 (utenti) = 123 MB x 20 (pagine di risultati) = 2,46 GB

Vengono preparati e inviati dai server, nonchè ricevuti dai client, 2,46 GB di dati per una navigazione pari a 20 pagine di risultati.

Client side

30,7 KB (peso medio di una pagina web) x 4000 (utenti) + 13,7KB (peso medio di un file JSON) x 4000 (utenti) x 19 (pagine di risultati) = 1,04 GB

Vengono preparati e inviati dai server, nonchè ricevuti dai client, 1,04 GB di dati.

Purtroppo non avendo accesso ai dati di yoox.com non posso fare previsioni accurate di alcun tipo: ricordiamoci poi che la CDN è in grado di inviare enormi quantità di dati a velocità tali che i dati sembrano arrivare dalla cache del computer.

Comunque, essendo proibitivo archiviare in cache le oltre 7000 pagine di risultati di ricerca di mobile.yoox.com, in continuo cambiamento per l'arrivo di nuovi prodotti, questa potrebbe essere una soluzione che si adatta bene alle pagine servite direttamente dai server interni a yoox.com.

Secondariamente, ma di grande importanza, è la reattività di mobile.yoox.com: un utente risparmia circa 0,4 secondi, cioè quasi mezzo secondo, ogni volta che scrolla in basso e attiva l'Infinite Scrolling per spostarsi nella navigazione alla pagina successiva. Ovviamente per i soli risultati di ricerca serviti dai server di yoox.com, e non dalla CDN.

7.3 Infinite Scrolling: il collo di bottiglia

Osservando la modalità con cui avviene l'Infinite Scrolling, su mobile.yoox.com e così anche per la soluzione client-side, ci si accorge che la pagina di navigazione successiva viene caricata soltanto una volta che si arriva a fine pagina web.

Questo è esattamente il comportamento di default dell'Infinite Scrolling, ossia l'applicazione del princpio di Lazy Loading.

Tutte le volte che si raggiunge la fine della pagina c'è una micro-attesa di caricamento: come abbiamo visto questa varia da 1,3 secondi (soluzione server-side) a 0,92 secondi (soluzione client-side).
Se moltiplichiamo questa attesa anche solo per 5 pagine, l'utente ha già perso dai 4,5 a 6,5 secondi di navigazione. Non è il massimo dell'esperienza utente per un sito web che pone al centro del metodo di navigazione dell'item gallery proprio queste attese.

Quindi, siamo decisi ad eliminare questo collo di bottiglia, ma si vuol comunque aderire al principio di Lazy Loading. E' stato quindi deciso di effettuare un pre-caricamento (pre-fetching) solo della pagina successiva a quella su cui si trova l'utente.

Lo standard HTML5 mette a disposizione gli attributi `pre-fetch` e `pre-load`, ma entrambi devono essere associati ad un elemento HTML `<img>`. Non è il nostro caso, dal momento che vogliamo pre-caricare solo il file JSON (che non contiene alcuna immagine) e che non è presente in pagina.

La soluzione era quindi gestire il pre-caricamento manualmente presso il lato JS.

Esploriamo ora tale soluzione.

7.4 Pre-caricamento del file JSON

Innanzi tutto creiamo un Servizio AngularJS per gestire la richiesta web per il prossimo file JSON.

```
itemGallery.service('ajaxcall', function($http) {

    this.getJson = function(url) {

        return $http.get(url)
            .then(function(response) {
                return response;
            });
    }
});
```

Viene creato il Servizio AngularJS `ajaxcall`, dipendente a sua volta dal servizio integrato di AngularJS, `$http`.

Al suo interno viene definito un oggetto `getJson` a cui è assegnata una funzione che riceve in ingresso un URL, la quale ritorna una JavaScript Promise.

Questa JavaScript Promise, una volta risolta (`.then()`), restituisce il contenuto del file JSON, `response.data`.

Analizziamo ora la nuova funzione `ItemGallery.prototype.nextPage` insieme alle modifiche apportate alla funzione Factory `ItemGalleryFactory`

```
myApp.factory('itemGalleryFactory', function($http, $q, ajaxcall)
{

        var ItemGallery = function() {

                this.loading_json = false;
                this.loading_img = false;
                var prefetch = false;
                var get_items;
```

Ad `ItemGallery` sono state aggiunte 2 dipendenze: il servizio che abbiamo appena creato, `ajaxcall`, e il servizio integrato di AngularJS, `$q`.

La proprietà `busy` dello `$scope` è stata disivsa in 2 roprietà: `loading_json` e `loading_img`. Questo perché nel momento in cui avvien il pre-fetch per la pagina successiva non vogliamo che si veda la GIF di caricamento. Il comportamento dell'applicazione deve essere trasparente all'utente, egli non si dove accorgere che è in atto il pre-caricamento.

Quando invece l'utente scrolla a fine pagina, viene attivato l'Infinite Scrolling e nell'attesa viene mostrata la GIF di caricamento.

La variabile prefetch viene utilizzata per controllare se occorre effettuare il pre-fetch del JSON relativo alla pagina successiva, nel caso non sia già stato effettuato.

La variabile `get_items` invece viene utilizzata per invocare il servizio ajaxcall. Procediamo ad analizzare le modifiche apportate ad `ItemGallery.nextPage`.

```
ItemGallery.prototype.nextPage = function($scope) {
    if (this.loading_json) return;
    this.loading_json = true;

    if ((this.data_search.length == 0) || !(this.json_url == ""))
    {

        if (!prefetch) {
            prefetch = true;
            get_items = ajaxcall.getjson(this.json_url);
        }

        this.current_json_url = // come prima
        this.loading_img = true;

        interval = $q.when(get_items).then(function(response)
                   {

                        this.data_search.push(response.data);
                        var new_items = // come prima

                        new_items.forEach(function(item) {
                               // come prima
                        }

                        for(var i=0; i<new_items.length; i++) {
                               // come prima
                        }
```

```
                        this.Area = // come prima
                        this.Department = // come prima

                        if (!(response.data.Pagination.NextPage ==
null)) {
                               // come prima
                        } else {
                               // come prima
                        }

                        var response_hash = // come prima
                        if (location.hash != response_hash &&
!(this.current_json_url.indexOf("&page=") == -1)) {
                           // come prima
                       }

                       this.loading_img = false;

                   }.bind(this)); // come prima

         if (!(this.json_url == "")) {
             get_items = ajaxcall.getjson(this.json_url);
         }

         this.loading_json = false;
    }
};
```

Cerchiamo ora di capire come funzionano i cambiamenti apportati.

```
if (this.loading_json) return;
```

se esiste già una istanza di questa funzione, poichè l'utente ha già scrollato a fine pagina, esci subito dalla funzione.
Replica il comportamento di quello che era `this.busy`.

```
this.loading_json = true;
```

in questo modo, qualsiasi altra chiamata alla funzione `ItemGallery.nextPage` viene rigettata dal controllo `if (this.loading_json)return;`

Replica il comportamento di quello che era `this.busy`.

```
if (!prefetch) {
        prefetch = true;
        get_items = ajaxcall.getjson(this.json_url);
}
```

se non è stato compiuto il prefetch procedi e imposta `prefetch=true` in modo tale che non venga ripetuto il pre-fetch.

```
get_items = ajaxcall.getjson(this.json_url);
```

Viene utilizzata la funzione getjson del servizio ajaxcall, con il parametro relativo all'URL del prossimo file JSON.

La variabile pre-fetch non verrà mai più impostata e questo blocco condizionale verrà valutato sempre false, dopo la prima volta cui viene acceduto.

```
this.loading_img = true;
```

a questo punto deve essere mostrata la GIF di caricamento. Infatti successivamente a questa istruzione si attenderà il file JSON e verranno aggiunti gli oggetti all'item gallery per la pagina successiva.

```
interval = $q.when(get_items).then(function(response) {
```

Viene attesa (`$q`) la Promise restituita da `get_items`: quando la Promise viene risolta (`.then()`) , essa restituisce la risposta per la richiesta al file JSON (insieme al file JSON).

```
$q

.then()
```

```
this.loading_img = false;
```

a questo punto deve essere nascosta la GIF di caricamento. Il file JSON per la pagina successiva è stato utilizzato e sono stati aggiunti gli oggetti all'item gallery per la pagina successiva.

```
if (!(this.json_url == "")) {
        get_items = ajaxcall.getjson(this.json_url);
}
```

Se L'URL al prossimo file JSON non è nullo alora viene fatto partire il pre-caricamento del JSON per la prossima pagina.

```
this.loading_json = false;
```

A questo punto, in prossimità di uscita dal blocco della funzione di Infinite Scrolling, viene sbloccata la funzione di Infinite Scrolling. La chiamata successiva, a tale funzione, non verrà rigettata.

Ricapitolando, in questo modo verrà invocata la funzione per scaricare un file JSON, getjson, ben 2 volte quando l'utente si trova alla sua prima pagina (che sia la numero 2, 10 o altro) di ricerca. Una volta cambiata pagina, poniamo che sia la 2, il file JSON per tale pagina sarà già in memoria e verrà invocata la funzione getjson per pre-caricare la pagina successiva, la pagina 3, se esistente.

In questo modo vengono scaricati 2 file JSON solo quando l'utente si trova sulla sua prima pagina di ricerca, uno per la pagina corrente e uno pre-caricato per la pagina successiva.

Successivamente verrà scaricato un solo file JSON per pagina, quello pre-caricato per la pagina successiva: quando l'utente si inoltra a pagina 2 verrà invocata la funzione getitems per pre-caricare la pagina 3, e così via.

In questo modo, prima di aver scrollato un'intera pagina, costituita di 20 oggetti, il file JSON viene sempre pre-caricato e non si verificano attese di caricamento: abbiamo così implementato un Infinite Scrolling continuo.

Di fatto, nei test compiuti, non ho mai più visto la GIF di caricamento. I nuovi oggetti erano già stati pre-caricati in memoria e l'inserimento in pagina è stato quasi istantaneo.

Parte 3

Conclusioni

Giudizio sul framework AngularJS

AngularJS a mio parere, ma anche tramite una veloce ricerca Internet, sembra essere il framework più difficile da imparare.
Il fatto che esista una documentazione ampia e completa stupisce ancora di più.

Il problema è che tale documentazione è scritta davvero male: è come se fosse scritta per sviluppatori che conoscono già AngularJS.

Il malcontento riguardo questo problema è diffuso, tant'è che sono incappato in una domanda su StackOverflow (più di 500 "up") che suggerisce di utilizzare un plugin per eliminare dai risultati di ricerca di Google la documentazione ufficiale di AngularJS.

Un esempio fra tutti: le opzioni di configurazione per una Direttiva non sono elencate nel capitolo "Direttive" della documentazione, ma invece vengono spiegate nel capitolo relativo al Servizio "$compile", di cui non si fa voce precedentemente.

Per quanto riguarda la documentazione, quindi, ho rimediato con libri e tutorial di AngularJS.

Una cosa mal concepita di AngularJS 1 è il two-way data binding: sebbene sia una gran funzionalità, è di poca utilità e soprattutto non ha senso che sia attivata di default per tutte le variaibli dello $scope. Occorre utilizzare la clausola di one-time data binding per ogni variabile dello $scope collegata alla View.

Il problema di AngularJS purtroppo è che è stato concepito male, troppo complicato. Lo sviluppatore passa più tempo sulla documentazione e nel capire come implementare una funzionalità, che invece nel realizzarla.

Secondariamente, sebbene io abbia utilizzato la versione stabile di AngularJS, questo è un capitolo già chiuso di AngularJS. La versione 2 infatti non sarà retro-compatibile con le applicazioni scritte con AngularJS 1.
Il framework è stato completamente riscritto: nella versione 2 viene eliminato di default il 2 way data binding ed anche il one-time data binding.

Un altro problema deriva dalla compatabilità dei browser: IE8 non è più supportato, a meno che non sia voglia rimanere sulle versioni precedente alla 1.3.

Infine parliamo di questo fattore popolarità di AngularJS tra le ricerche Google: ci si rende conto che deriva per buona parte dai problemi di comprensione di AngularJS. Molte volte ci si accorge di qualcosa che sarebbe dovuto funzionare, che invece non funziona e non si capisce il perché: per questo si arriva a StackOverflow.

Giudizio riguardo la soluzione client-side con AngularJS

Abbiamo visto nel capitolo Performance la bontà dei risultati raggiunti.

Tuttavia sono test che andrebbero effettuati per l'ambiente online.
La versione online di mobile.yoox.com registra chiamate di Infinite Scrolling con un peso delle pagine pari a 3KB e tempi di risposta che variano da 400ms ai 700ms.

Occorre però tenere a mente che le pagine del dominio yoox.com fanno utilizzo della CDN (tempi di risposta), sono compresse (3 KB), e chissà quale altra tecnologia è stata implementata per il sito web yoox.com.

Comunque, opininone del tutto personale, sarebbe fuori luogo da parte mia pensare di poter raggiungere le performance di mobile.yoox.com, per cui solo nella sede di Zola Predosa lavorano oltre 600 persone, nel periodo di 2 mesi.

Sviluppi Futuri

Di sicuro sarebbe interessante fare l'upgrade ad AngularJS 2 e magari esplorare qualche forma di tecnologia HTML5.

Si è pensato ad una sorta di navigazione offline, in cui l'utente scarica preventivamente i file JSON e immagini per un certo numero X di pagine. Una volta offline, l'utente riuscirebbe comunque a navigare i risultati di ricerca di X pagine.

Per questa funzionalità occorre esplorare il Web Storage di HTML5 (localstorage e sessionstorage), IndexedDB, Web SQL DB, il tipo di dato blob e le JS Promise .fetch() e Promise.all().

Lo sviluppo è quasi al termine, occorre implementare un finestra di dialogo gestita da AngularJS e aggiustare la funzione di ricorsione per scaricare come file blob tutte le immagini dei prodotti di ogni pagina.

Bibliografia

https://docs.angularjs.org

https://developer.mozilla.org

http://www.w3schools.com/js/default.asp

http://www.w3schools.com/angular/

https://sroze.github.io/ngInfiniteScroll/index.html

http://www.json.org/json-it.html

http://www.asp.net/web-pages/overview/getting-started/introducing-razor-syntax-c