ALMA MATER STUDIORUM – UNIVERSITA' DI BOLOGNA

CAMPUS DI CESENA

SCUOLA DI SCIENZE

CORSO DI LAUREA IN SCIENZE E TECNOLOGIE INFORMATICHE

TITOLO DELLA RELAZIONE FINALE

PROGETTAZIONE DI UNA WEB APPLICATION INTERATTIVA PER L'ELABORAZIONE E LA VISUALIZZAZIONE DI DATI COMMERCIALI

Relazione finale in

PARADIGMI DI PROGRAMMAZIONE

Relatore

VITTORIO MANIEZZO

Presentata da

PASQUINELLI ANDREA

SESSIONE 10/12/2015

ANNO ACCADEMICO 2015/2016

INDICE

Capitolo 01

Capitolo 02

Capitolo 03

Capitolo 01

INTRODUZIONE

La programmazione ormai si è fortemente spostata sul web, non basta più creare semplici siti web statici o con semplici script.

Lo scopo ora è cercare di spostare interi programmi direttamente su siti internet, senza avere bisogno di installare direttamente nulla sul proprio pc, per avere tutto a portata di mano in ogni momento e in qualsiasi luogo. Lasciando però la pagina leggera e minimizzando il traffico generato, per questo si punta sempre di più sullo studio e l'utilizzo di nuove tecnologie.

In questo progetto si cercherà proprio di venire incontro a queste esigenze cercando di sviluppare un applicazione web generata completamente da script.

L'IDEA

L'idea è creare una web application ovvero una applicazione multipiattaforma dove, da qualsiasi dispositivo dotato di una semplice connessione internet, si possono osservare e controllare i dati riguardanti la propria azienda o l'azienda per cui si lavora.

Incentrata sulla fase di login e sulle possibili opzioni che questa applicazione ci può offrire.

WEB2

Il Web 2.0 è un'espressione utilizzata spesso per indicare uno stato dell'evoluzione del World Wide Web, rispetto a una condizione precedente. Si indica come Web 2.0 l'insieme di tutte quelle applicazioni online che permettono un elevato livello di interazione tra il sito web e l'utente come i blog, i forum, le chat, i wiki, le piattaforme di condivisione di media come Flickr, YouTube, Vimeo, i social network come Facebook, Myspace, Twitter, Google+, Linkedin, Foursquare, ottenute tipicamente attraverso opportune tecniche di programmazione Web e relative applicazioni web afferenti al paradigma del Web dinamico in contrapposizione al cosiddetto Web statico o Web 1.0. Il termine 2.0 è mutuato direttamente dallo sviluppo software nel quale la notazione puntata indica l'indice di sviluppo e successivo rilascio di un particolare software. In questo caso la locuzione pone l'accento sulle differenze rispetto al cosiddetto Web 1.0, diffuso fino agli anni novanta, e composto prevalentemente da siti web statici, senza alcuna possibilità di interazione con l'utente eccetto la normale navigazione ipertestuale tra le pagine, l'uso delle e-mail e dei motori di ricerca. Per le applicazioni Web 2.0, spesso vengono usate tecnologie di programmazione particolari, come AJAX (Gmail usa largamente questa tecnica) o Adobe Flex. Tale possibilità di creazione e condivisione di contenuti su Web, tipica del Web 2.0, è data da una serie di strumenti (tool) on-line che permettono di utilizzare il web come se si trattasse di una normale applicazione. In pratica il Web di seconda generazione è un Web dove poter trovare quei servizi che finora erano offerti da pacchetti da installare sui singoli computer. Esempi di Web 2.0 sono Ckeditor e Writely, veri e propri elaboratori di testi e convertitori di formato, oppure NumSum, una sorta di foglio elettronico. Anche Google ha recentemente lanciato la sua suite di editor, chiamata Google Docs & Spreadsheet, e Microsoft ha rilasciato una versione online della suite Office, Office 365. Oltre alla creazione condivisa di contenuto on-line, il Web 2.0 è caratterizzato dalla pubblicazione immediata del contenuto e alla sua classificazione e indicizzazione nei motori di ricerca, in modo che l'informazione sia subito disponibile a beneficio dalla comunità, realizzando in maniera veloce il ciclo di vita del content management. Per la pubblicazione dei contenuti fanno da padrone sul Web i provider di blog come Blogger, Wordpress e Splinder, ma anche piattaforme commerciali come Microsoft Sharepoint Services che nella versione 3.0 accentua le sue caratteristiche di collaborazione diventando la parte server di Office 12. L'utente da semplice fruitore di informazioni è diventato un vero cittadino della Rete: produce contenuti per se stesso e li condivide con gli altri. Si

tratta di uno strumento di sviluppo tecnologico e di un fenomeno sociale, ma anche di un efficace strategia di marketing per rilanciare la Rete a livello globale. Il web 2.0 è per l'individuo un'opportunità per l'autoaggiornamento, per la creazione e per la gestione di network, per migliorare la velocità e la qualità delle informazioni; per il team sono invece disponibili diversi strumenti per facilitare la collaborazione a distanza e la condivisione delle informazioni; per l'impresa che punta ad essere più competitiva e più efficiente sono disponibili applicazioni per migliorare il knowledge management, per arricchire la comunicazione esterna e per ricevere facilmente positive contaminazioni di idee e contributi esterni all'organizzazione.

JAVASCRIPT

JavaScript è un linguaggio di scripting lato-client, che viene interpretato dal browser. Il web infatti funziona a due livelli:

1. Le pagine web vengono inviate all'utente da un web server, cioè da un programma che si trova su un computer remoto, e che per lo più non fa nient'altro che inviare le pagine a chi ne fa richiesta (in realtà è in grado di fare un sacco di altre cose, ma in questo contesto non è necessario specificare oltre)

2. L'utente da casa visualizza sul proprio browser le pagine che gli sono state inviate. Un browser è un programma che permette di leggere ed interpretare le pagine scritte in linguaggio HTML: si tratta di Internet Explorer, Netscape Navigator, Opera e altri.

Quando visualizziamo le nostre pagine web da casa ci sono dunque due computer che si parlano: il server e il client. Alcuni linguaggi di scripting (asp, php, perl) vengono eseguiti dal web server (si chiamano appunto linguaggi server side o lato server). JavaScript, invece, viene eseguito sul nostro computer di casa dal browser è quindi un linguaggio client side o lato client. Dire che JavaScript è un linguaggio lato client, significa anche che i vostri script avranno validità all'interno delle singole pagine web, e non da una pagina all'altra: con JavaScript è possibile infatti passare una piccola quantità di dati da una pagina all'altra, ma è un'operazione che può essere effettuata con una certa difficoltà a differenza dei linguaggi server side dove si esegue invece in maniera intuitiva; non è possibile invece trasmettere quantità di dati elevate. Come detto prima inoltre JavaScript è un linguaggio di scripting: questo significa che la sintassi JavaScript potete scriverla direttamente dentro la pagina HTML, senza bisogno di produrre alcun file compilato. Con i linguaggi di programmazione invece (come il C, il C++) si scrive la sintassi, e poi la si passa a un compilatore, che produce un file "compilato", in cui la sintassi è scomparsa. Tutti i programmi di windows ad esempio sono dei file compilati, in cui non c'è più traccia della sintassi originaria. Dire che è un linguaggio di scripting sottintende dunque il fatto che sia un linguaggio interpretato: come abbiamo visto non esiste nessun compilatore, ma è direttamente il browser, tramite un apposito motore di scripting (cioè di visualizzazione), che legge le parti di codice JavaScript. Javascript è stato il primo linguaggio di scripting web, molto diffuso agli inizi della sua vita, ma anche adesso. E' stato sviluppato da Netscape Communications Corporation che non è altro che l'ideatore del vecchio browser Web Netscape Navigator. Il nome iniziale non era questo,

bensì Mocha, poi divenne LiveScript ma è stato cambiato nuovamente e in maniera definitiva in JavaScript perché la sintassi si basa su quella di Java; comunque non bisogna confonderli perché sono due linguaggi differenti. Javascript è un linguaggio interpretato perché esegue il codice script riga per riga così come gli arriva mentre altri linguaggi devono essere compilati, ovvero tradotti nel codice macchina e sicuramente per questo motivo Javascript è considerato uno dei linguaggi più semplici e un punto di partenza per chi vuole iniziare a programmare. Javascript in linea generale può essere utilizzato per abbellire un sito web o per creare programmi per far interagire gli utenti. Tra gli usi più frequenti possiamo elencare:

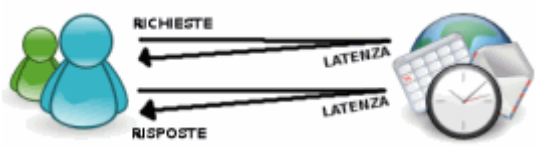
1. L'inserimento di messaggi a scorrimento che cambiano sulla barra di stato del browser.
2. Eseguimento di calcoli matematici e di conteggi di lettere o parole (ed usi simili)
3. Inserimento di messaggi di avviso all'interno di una pagina di un sito o all'interno di un frame.
4. Rendere le immagini animate e cambiare la loro visualizzazione quando si passa il mouse sopra di esse.
5. Identificare il tipo di browser in utilizzo e visualizzare un contenuto adatto e diverso per ogni browser.
6. Ricercare i plugin installati e avvisare l'utente se è richiesto un ulteriore plugin per continuare.

AJAX

In questo paragrafo spiegherò dettagliatamente l'acronimo e le sue potenzialità al fine di fornire ai meno aggiornati o meno preparati su Javascript, le conoscenze per usare questa tecnica di sviluppo. AJAX è l'acronimo di Asynchronous Javascript And XML e indica una tecnica per permettere di costruire applicazioni web paragonabili a quelle stand-alone che utilizziamo normalmente. Se parliamo di AJAX parliamo di un oggetto specifico: XMLHttpRequest. A seconda del browser usato prende nomi differenti o viene richiamato in maniera differente. Nel caso di Internet Explorer, ad esempio, questo oggetto è restituito da un ActiveXObject mentre nei browsers alternativi più diffusi (Mozilla, Safari, FireFox, Netscape, Opera ed altri) XMLHttpRequest è supportato nativamente. Questo oggetto permette di effettuare la richiesta di una risorsa (con HTTP) ad un server web in modo indipendente dal browser. Nella richiesta è possibile inviare informazioni, ove opportuno, sotto forma di variabili di tipo GET o di tipo POST in maniera simile all'invio dati di un form. AJAX è stato creato per consentire alle applicazioni web un dialogo asincrono tra client e server minimizzando così il tempo di risposta percepito dall'utente. In pratica si tratta di interporre tra il browser del client e la rete nella quale si trova il server un motore AJAX, che consente all'utente di interagire con l'applicazione web direttamente senza aspettare la risposta del server. La richiesta quindi è asincrona, il che significa che non bisogna necessariamente attendere che sia stata ultimata per effettuare altre operazioni, stravolgendo sotto diversi punti di vista il flusso dati tipico di una pagina web. Generalmente infatti il flusso è racchiuso in due passaggi alla volta, richiesta dell'utente (link, form o refresh) e risposta da parte del server per poi passare, eventualmente, alla nuova richiesta da parte dell' utente.

Il terzo ed inevitabile incomodo di questo ciclo è l'attesa che trascorre tra una richiesta dell' utente e la risposta del server. Con l'aggiunta di AJAX si perde questa linearità e mentre l'utente è all'interno della stessa pagina le richieste sul server possono essere numerose e completamente

indipendenti. Nulla infatti vieta, teoricamente, di effettuare decine di richieste simultanee al server per operazioni differenti con o senza controllo da parte del navigatore. E' questo in sostanza che consente al sistema di fornire all'utente un tempo di risposta inferiore, simile a quello di un'applicazione desktop. Qui però la velocità di un applicativo non è più determinata principalmente dalla velocità del sistema, ma dalla velocità della rete.



Ciò che resta del vecchio flusso, il tempo di attesa, passa spesso in secondo piano ed in molti tipi di interazione è praticamente impercettibile. Ma attenzione poiché questo tempo è anche uno dei maggiori problemi dell'utilizzo di AJAX, sia per gli sviluppatori sia per i navigatori. I primi potrebbero trovarsi in difficoltà qualora l'operazione asincrona dovesse attendere assolutamente una risposta al fine di completare una serie di operazioni più o meno sensibili mentre i secondi potrebbero non avere idea di cosa stia accadendo alla pagina chiudendola ignari di aver richiesto un'informazione. A completare questa breve introduzione tecnica e teorica è il tipo di risposta che l'oggetto si aspetta dopo una chiamata che non deve essere necessariamente di tipo XML o letta come tale ma che può essere semplicemente testuale, in contro tendenza con l'acronimo stesso ma non per questo inusuale. Un'applicazione web tradizionale solitamente costruisce una pagina web che invia al client. Questo compie delle azioni (riempie form, clicca dei link, ecc.) che vengono inviate al server come richieste http, il quale le elabora e costruisce una nuova pagina web che ritrasmette al client. Questo modello funziona, ma ha l'inconveniente di essere sincrono: da quando l'utente invia una richiesta a quando il server risponde con una nuova pagina, l'utente non può far altro che aspettare. Da cosa è costituito:

- XHTML e CSS: per la presentazione
- DOM: per il layout dinamico e l'interazione con la pagina
- XML e XSLT: scambio e manipolazione dati
- XMLHttpRequest: recupero asincrono dei dati
- Javascript: collante per tutte le tecnologie precedenti

Come per le applicazioni DHTML, anche le applicazioni AJAX devono essere testate su più browser per verificarne la compatibilità. Inoltre è richiesto che nel client sia attivato Javascript. Il vantaggio di usare AJAX è la grande velocità alla quale un'applicazione risponde agli input dell'utente. Un problema abbastanza degno di nota è che, senza l'adozione di adeguate contromisure, le applicazioni AJAX possono rendere non utilizzabile il tasto "indietro" del browser: con questo tipo di applicazioni, infatti, non si naviga da una pagina all'altra, ma si aggiorna di volta in volta una singola parte del medesimo documento. Proprio per questo i browser, che sono programmi orientati alla pagina, non hanno possibilità di risalire ad alcuna di tali versioni "intermedie". Le ultime tecnologie HTML5 che permettono di manipolare la cronologia del browser, hanno permesso di ovviare al problema dell'orientamento a pagina. In ogni modo, un attento design delle applicazioni AJAX permette di risolvere in parte (talvolta tutti) questi aspetti negativi.

MVC

Il Model-View-Controller (MVC, talvolta tradotto in italiano Modello-Vista-Controllo), in informatica, è un pattern architetturale molto diffuso nello sviluppo di sistemi software, in particolare nell'ambito della programmazione orientata agli oggetti, in grado di separare la logica di presentazione dei dati dalla logica di business. Questo pattern si posiziona nel livello di presentazione in una Architettura multi-tier. Storicamente il pattern MVC è stato implementato lato server. Recentemente, con lo sviluppo e la parziale standardizzazione di JavaScript sono nate le prime implementazioni lato client.

LATO SERVER

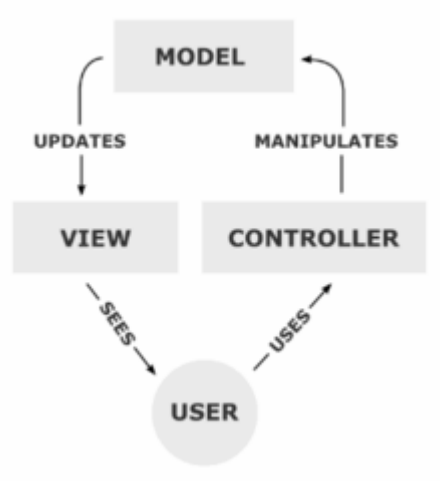
Originariamente impiegato dal linguaggio Smalltalk, il pattern è stato esplicitamente o implicitamente sposato da numerose tecnologie moderne, come framework basati su PHP, Ruby, Python, Java, Objective C o .NET. A causa della crescente diffusione di tecnologie basate su MVC nel contesto di framework o piattaforma middleware per applicazioni Web, l'espressione framework MVC o sistema MVC sta entrando nell'uso anche per indicare specificamente questa categoria di sistemi.

LATO CLIENT

Recentemente è aumentata la richiesta di Rich Internet application che facciano chiamate asincrone al server (AJAX), senza fare redirect per visualizzare i risultati delle elaborazioni. Col crescere del codice JavaScript che veniva eseguito sul client, si è sentita l'esigenza di creare i primi framework che implementino MVC in puro JavaScript. Uno dei primi è stato Backbone.js, seguito da una serie interminabile di altri framework, tra cui JavascriptMVC ed AngularJS.

STRUTTURA

Il pattern è basato sulla separazione dei compiti fra i componenti software che interpretano tre ruoli principali:



Il model fornisce i metodi per accedere ai dati utili all'applicazione;

Il view visualizza i dati contenuti nel model e si occupa dell'interazione con utenti e agenti;

Il controller riceve i comandi dell'utente (in genere attraverso il view) e li attua modificando lo stato degli altri due componenti.

Questo schema, fra l'altro, implica anche la tradizionale separazione fra la logica applicativa (in questo contesto spesso chiamata "logica di business"), a carico del controller e del model, e l'interfaccia utente a carico del view. I dettagli delle interazioni fra questi tre oggetti software dipendono molto dalle tecnologie usate (ad esempio linguaggio di programmazione, eventuali librerie, middleware) e dal tipo di applicazione (per esempio se si tratta di un'applicazione web, o di un'applicazione desktop). Quasi sempre la relazione fra view e model è descrivibile anche come istanza del pattern Observer. A volte, quando è necessario cambiare il comportamento standard dell'applicazione a seconda delle circostanze, il controller implementa anche il pattern Strategy. Model-View-Controller (MVC) quindi è un pattern utilizzato in programmazione per dividere il codice in blocchi dalle funzionalità ben distinte. Per capire come questo approccio si possa adattare allo sviluppo Web, pensiamo al classico funzionamento di una applicazione internet. Un client, tipicamente un browser, inoltra la richiesta ad un server per una pagina HTML. Il server ospita un’applicazione scritta in un linguaggio di programmazione lato server (come C# o VB.NET) che preleva i dati da un database, li elabora e li restituisce al client in formato HTML.

La parte più attiva in questo procedimento è l'applicazione Web che ha il compito di reperire ed inviare le informazioni.

Lo schema che identifica il pattern MVC è in particolare:

Model: contiene i metodi di accesso ai dati.

View: si occupa di visualizzare i dati all'utente e gestisce l'interazione fra quest'ultimo e l'infrastruttura sottostante.

Controller: riceve i comandi dell'utente attraverso il View e reagisce eseguendo delle operazioni che possono interessare il Model e che portano generalmente ad un cambiamento di stato del View.

Inoltre riguardo i framework citati prima, basati sul paradigma Model-View-Controller (MVC), posso aggiungere che essi offrono allo sviluppatore diverse funzionalità tra cui:

maggiore organizzazione del codice

scaffolding dei dati

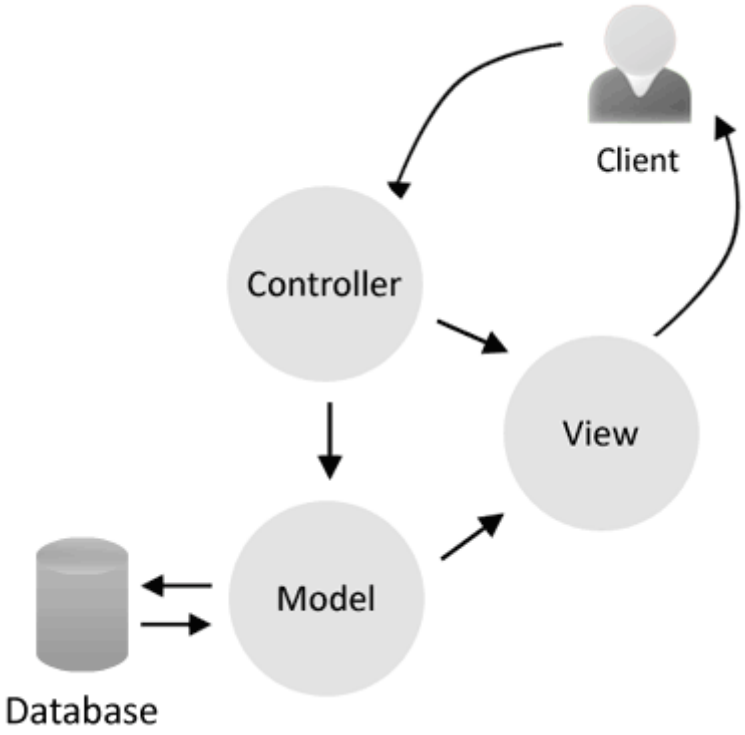
URL "puliti", adatti ai motori di ricerca

maggiore rapidità di sviluppo

ASP.Net MVC è la risposta di Microsoft a questa tendenza. Sviluppato sotto la guida di alcuni fra i più apprezzati sviluppatori della famiglia di prodotti web di casa Microsoft, tra cui Scott Guthrie e Phil Haack, la nuova tecnologia si inserisce nell'ecosistema .NET al fianco del classico motore ASP.Net basato su WebForms. ASP.Net MVC rappresenta anche uno dei primi esperimenti Open Source di Microsoft. Il nuovo componente del framework .NET è stato infatti sviluppato in collaborazione con la community attraverso il rilascio su CodePlex di diverse versioni di prova durante tutto il ciclo di sviluppo, iniziato nel 2007, e del codice sorgente, disponibile sotto licenza Ms-PL anche per la versione 1.0 di ASP.Net MVC. Il

diverso approccio utilizzato da Microsoft per ASP.Net MVC trova conferma anche nell'inserimento all'interno del download ufficiale della libreria Javascript JQuery, in una versione con supporto all'Intellisense di Visual Studio. È bene fare fin da subito una precisazione molto importante: ASP.NET MVC non va a sostituire ASP.NET classico (WebForms). Le due tecnologie sono infatti parallele e adatte a compiti differenti e chi non si trovasse bene con MVC potrà continuare a costruire pagine Web utilizzando WebForms. Lo sviluppo di quest'ultimo infatti continua e per la versione quattro del framework .Net sono attese diverse migliorie. L'approccio introdotto da ASP.NET MVC presenta diverse differenze rispetto al metodo di lavoro a cui ASP.NET classico ci ha abituati, in particolare per quanto riguarda la scrittura e l'organizzazione del codice. MVC obbliga infatti lo sviluppatore a suddividere il codice in blocchi dalle funzionalità e compiti ben distinti tra loro. Sebbene questo tipo di organizzazione fosse possibile anche con ASP.NET "tradizionale", ad esempio dividendo il codice nei livelli di Data Access Layer, Business Layer e Presentation Layer, la scelta di applicare o meno questa suddivisione non era dettata da motivazioni tecniche ma dalla scelta dello sviluppatore. MVC richiede inoltre di scrivere manualmente il codice HTML presente all'interno delle viste, diversamente da quanto accade con ASP.NET classico in cui il codice viene generato automaticamente dai controlli lato server. Per scegliere al meglio cosa utilizzare, oltre a una scelta dettata prettamente dal gusto dello sviluppatore, MVC si adatta particolarmente bene a scenari di uso quali blog, cataloghi o gallerie di immagini; in generale potremmo dire per siti che hanno bisogno principalmente di visualizzare informazioni. Per sviluppare pagine che richiedono molte interazioni con l'utente e che hanno bisogno estensivo di form, ASP.NET classico è sicuramente di uso più immediato in quanto propone strumenti (come i controlli lato server o la recente tecnologia Dynamic Data) tesi a semplificare lo sviluppo in questi scenari. Sebbene anche MVC possa essere utilizzato per questi scopi, presenta l'indubbio svantaggio di dover scrivere manualmente il codice HTML e di non gestire in automatico il postback tra le pagine. Le pagine scritte utilizzando MVC, tuttavia, proprio perchè questa tecnologia richiede la scrittura manuale del markup, avranno un codice HTML più leggero e attinente agli standard rispetto a quelle generate dalla controparte WebForms.

È comunque possibile utilizzare WebForms ed MVC all'interno dello stesso sito, scegliendo di volta in volta quale delle due tecnologie utilizzare per scrivere le pagine. Questa funzionalità permette inoltre di migrare facilmente applicazioni scritte in ASP.NET classico verso il framework MVC e viceversa: potremo infatti effettuare una transizione graduale di alcune parti senza intaccare il funzionamento complessivo del sito internet.


Client
Controller
View
Model
Database

JSON

JSON, acronimo di JavaScript Object Notation, è un formato adatto all'interscambio di dati fra applicazioni client-server come API o Mashup. È basato su JavaScript ma il suo sviluppo è specifico per lo scambio di dati ed è indipendente dallo sviluppo del linguaggio di scripting dal quale nasce e con il quale è perfettamente integrato e semplice da utilizzare. Json è una valida alternativa al formato XML-XSLT e sempre più servizi di Web Services mettono a disposizione entrambe le possibilità di integrazione. Leggere ed interpretare uno stream in Json è semplice in tutti i linguaggi, non solo in JavaScript, con cui è completamente integrato ma anche con PHP, Java ed altri linguaggi server-side, per mostrare i dati da fonti esterne ed impaginarli secondo le proprie soluzioni personalizzate. Json si è fatto largo tra i vari protocolli e formati per lo scambio dati per la semplicità di implementazione, possibile con la sola funzione, comune a molti linguaggi, chiamata eval(). 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 di Programmazione JavaScript. JSON è un formato di testo completamente indipendente dal linguaggio di programmazione, ma utilizza convenzioni conosciute dai programmatori di linguaggi della famiglia del C, come C, C++, C#, Java, JavaScript, Perl, Python, e molti altri. Questa caratteristica fa di JSON un linguaggio ideale per lo scambio di dati. Esso infatti è basato su due strutture:

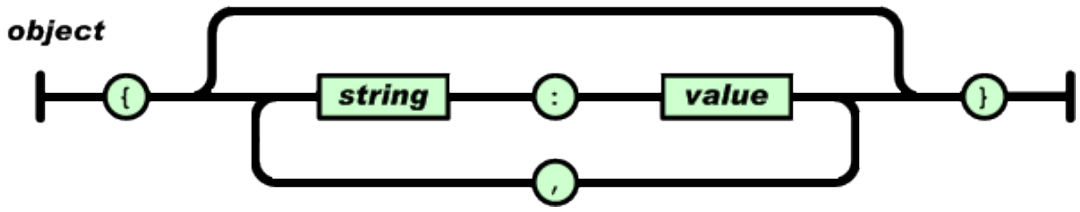
1. Un insieme di coppie nome/valore. In diversi linguaggi, questo è realizzato come un oggetto, un record, uno struct, un dizionario, una tabella hash, un elenco di chiavi o un array associativo.

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

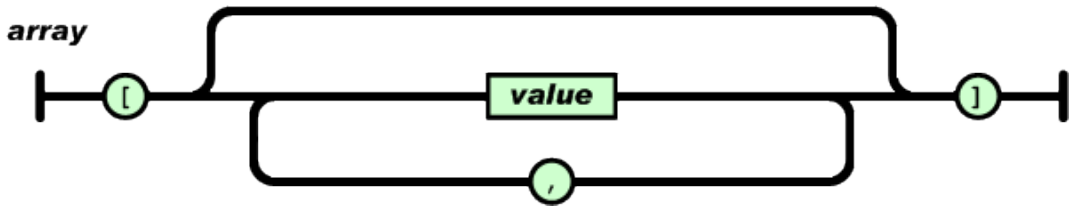
Queste sono strutture di dati universali. Virtualmente tutti i linguaggi di programmazione moderni li supportano in entrambe le forme. E' sensato che un formato di dati che è interscambiabile con linguaggi di programmazione debba essere basato su queste strutture.

In JSON, assumono queste forme:

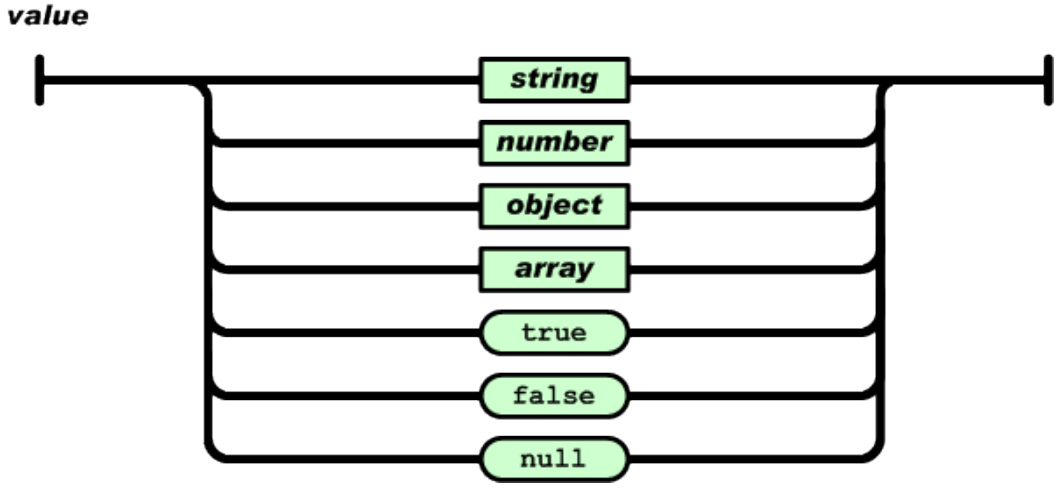
- Un oggetto è una serie non ordinata di nomi/valori. Un oggetto inizia con { (parentesi graffa sinistra) e finisce con } (parentesi graffa destra). Ogni nome è seguito da : (due punti) e la coppia di nome/valore sono separati da una virgola.



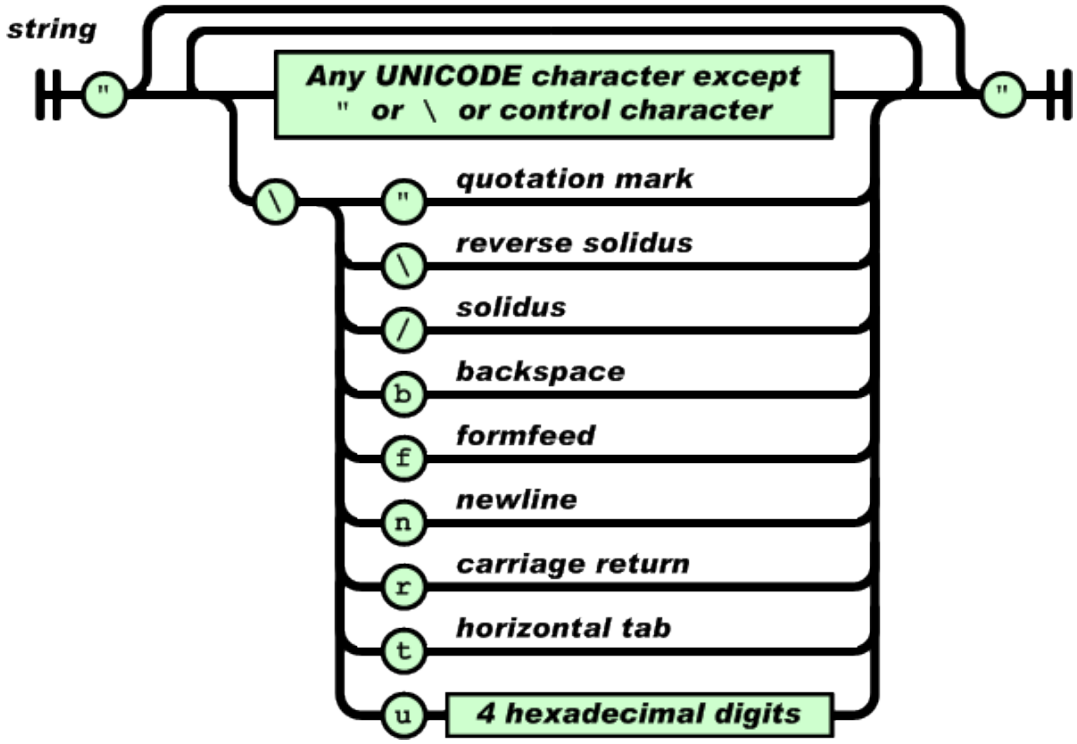
Un array è una raccolta ordinata di valori. Un array comincia con [(parentesi quadra sinistra) e finisce con] (parentesi quadra destra). I valori sono separati da , (virgola).



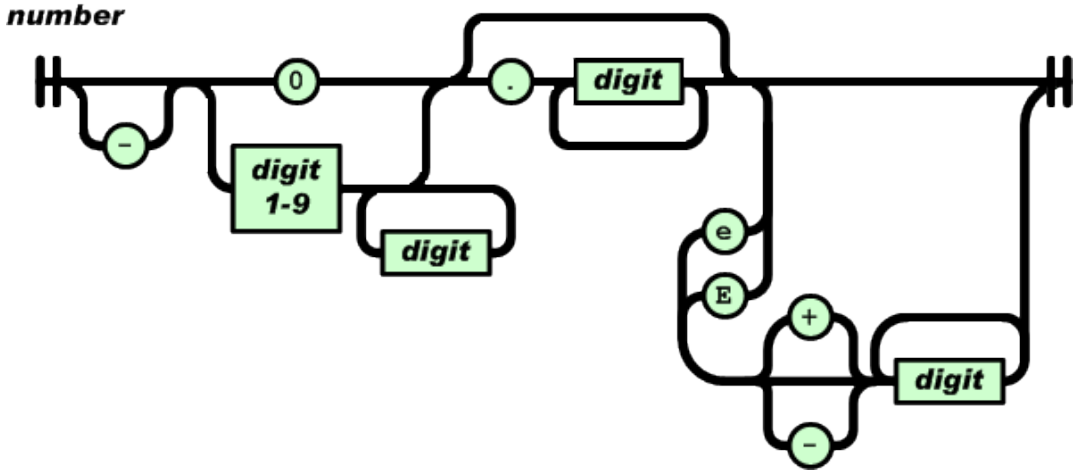
Un valore può essere una stringa tra virgolette, o un numero, o vero o falso o nullo, o un oggetto o un array. Queste strutture possono essere annidate.



Una stringa è una raccolta di zero o più caratteri Unicode, tra virgolette; per le sequenze di escape utilizza la barra rovesciata. Un singolo carattere è rappresentato come una stringa di caratteri di lunghezza uno. Una stringa è molto simile ad una stringa C o Java.



Un numero è molto simile ad un numero C o Java, a parte il fatto che i formati ottali e esadecimali non sono utilizzati.



I caratteri di spaziatura possono essere inseriti in mezzo a qualsiasi coppia di token. A parte alcuni dettagli di codifica, questo descrive totalmente il linguaggio. Inoltre la maggior parte dei linguaggi di programmazione possiede un typesystem molto simile a quello definito da JSON per cui sono nati molti progetti che permettono l'utilizzo di JSON con altri linguaggi quali, per esempio: ActionScript, C, C#, Adobe ColdFusion, Common LISP, Delphi, E, Erlang, Java, JavaScript, Lua, ML, Objective Caml, Perl, PHP, Python, REBOL e Ruby.

IONIC

In questo paragrafo analizzerò Ionic Framework, un insieme di API e tool pensati per lo sviluppo di app ibride: in altre parole app scritte in HTML+CSS+JavaScript che però hanno accesso a features tipiche dei dispositivi come fotocamera, accesso ai file, ecc. Ionic mette semplicemente insieme Apache Cordova e AngularJS. Inoltre come potete notare dalla sua documentazione, le API sono composte anche da una serie di servizi e direttive che ci aiutano ad ottenere in maniera veloce un look and feel da app nativa, tra le quali:

- Action Sheet
- Side Menu
- Slide Box
- Tabs

Essendo basato su Cordova, le app svilluppate con Ionic sono compatibili non solo con iOS e Android, ma anche con Windows Phone, Amazon Fire OS, Firefox OS. La premessa è doverosa in quanto Ionic è un framework HTML, CSS e JS che consente di realizzare App ibride, ovvero adatte sia alla fruizione mezzo browser che alla "pacchettizzazione" per la distribuzione negli App Store. Come gran parte di questi framework utilizza Apache Cordova (PhoneGap) come contenitore qualora si voglia ottenere un'App per smartphone che interagisca con le API della piattaforma specifica. Ionic promette performance mai viste, sfruttando la leggerezza del framework, minima manipolazione del DOM e accelerazione Hardware. Un altro aspetto che lo contraddistingue è che si tratta di un framework "Angular friendly", ovvero studiato appositamente per essere utilizzato insieme ad AngularJS, il potentissimo frontend framework di Google che sta guadagnando sempre più consensi fra gli sviluppatori. Purtroppo Ionic è ancora in fase beta, quindi non è adatto a progetti che devono andare subito in produzione. Intanto ricordiamo che un framework si può definire una struttura che facilita lo sviluppo, sia in termini di velocità che in termini di ordine e mantenibilità del codice. Un framework è quindi uno strumento che aiuta chi scrive il codice e soprattutto aiuta i team di sviluppo ed eventuali terzi che riprenderanno in mano il codice. Alcuni dei più utilizzati o famosi framework a disposizione nel mondo web:

PHP:

- Zend
- Symfony
- YII
- CodeIgniter
- CakePHP

JS:

- AngularJS
- Backbone
- Knockout
- Emberjs

ANGULARJS

AngularJS è il framework javascript creato direttamente da Google: è scritto secondo pattern MVVM, ma permette un'implementazione del codice secondo il pattern MVC. AngularJS è utile a semplificare la realizzazione di applicazioni Web single page: favorisce un approccio dichiarativo allo sviluppo client-side, migliore per la creazione di interfacce utente, laddove l'approccio imperativo è ideale per realizzare la logica applicativa. Si tratta di design pattern, che sono degli schemi logici con cui si definiscono le relazioni tra le classi, questo favorisce la progettazione dell'applicativo. MVC è molto utilizzato perchè favorisce la mantenibilità del codice, la riusabilità e la divisione logica. Uno dei vantaggi più grandi e che caratterizzano questo framework è la possibilità di integrare e utilizzare molte funzioni utilizzando quasi esclusivamente l'HTML grazie all'approccio dichiaritivo. Questo rende molto facile l'interazione con il framework anche ai meno esperti; un altro punto a favore di Angular è il fatto che grazie a questa semplicità di utilizzo si ottengono già grandi risultati e si ha la possibilità, in pochissimo tempo, di rendere il contenuto dinamico via variabili definite in Javascript. Le direttive sono ciò che permette ad Angular di lavorare anche tramite un approccio dichiaritivo. Si presentano tramite forma di attributi HTML o veri e propri custom tags, ma sono delle vere e proprie entità funzionali e strutturali. Ne abbiamo già a disposizione molte di default. Essendo queste direttive presenti nelle API di Angular presentano la particolarità di iniziare con il prefisso ng-. Il bello è che possiamo definire anche noi queste direttive, ed essendo un approccio dichiarativo, una volta creata una direttiva questa è estremamente riutilizzabile. Una nota particolare è che queste direttive sono estremamente legate all'elemento del DOM a cui sono associate. Il data-binding è un processo che sincronizza un elemento della User Interface con un elemento della logica applicativa. In Angular questo processo è bidirezionale, in quanto la modifica può essere effettuata sia dalla logica applicativa ed essere sincronizzato nella view, sia essere modificato nella view ed essere sincronizzato nella logica applicativa.La View quindi corrisponde al codice HTML, mentre la logica applicativa, meglio definita come Controller secondo il modello MVC, sarà scritta in codice javascript; il Model sarà quello che nell'esempio porta la variabile nome e sarà utilizzabile sia dalla view che dal controller grazie al data-binding. AngularJS infine quindi si ispira al pattern MVC, come altri framework analoghi quali Knockout o Ember.js. Ma rispetto ai diretti concorrenti, questo framework è in grado di ridurre in maniera considerevole il codice necessario a realizzare applicazioni HTML/JavaScript.

Capitolo 02

IL PROGETTO

Il mio progetto, come già detto, è un'applicazione web interattiva. In particolare è una spa ovvero una single page application; quello che differenzia questo tipo di applicazioni web, rispetto le altre, è che si svolge tutto all'interno della stessa pagina che quindi non viene ricaricata ogni volta che si effettua un operazione.

Mi sono soffermato sull'uso aziendale e quindi il progetto riguarda l'elaborazione di tutti i dati di un ipotetico negozio e mostra dei grafici riferiti alle vendite e agli stock in magazzino in diversi archi temporali.

A principio in questo capitolo spiegherò le due parti principali, Login e App.

Login come vedremo è la pagina iniziale che ci permette, grazie a specifiche credenziali di entrare nell'App, cuore vero e proprio di questo progetto.

A seguire poi ci saranno altri paragrafi, app.js, services.js, controller.js che riguardano invece il funzionamento dell'applicazione e come sono implementati Angular e Ionic di cui abbiamo parlato in precedenza.

INDEX

L'index.html è il file che viene caricato al momento dell'avvio della nostra applicazione e contiene tutti i riferimenti esterni che andremo ad usare.

Vediamo al suo interno i riferimenti a due fogli di stile:

style.css, il classico foglio di stile che contiene le informazioni generali e di default sulla grafica e la posizione dei vari oggetti che saranno presenti

ionic.css, il foglio di stile che invece viene aggiornato e modificato dinamicamente dai vari javascript di ionic, che come visto in precedenza si occupa della visualizzazione delle pagine totalmente attraverso script.

Il corpo vero e proprio di questo file dato dal caricamento a tutti i file di script di ionic che vengono così inizializzati e caricati all'interno della pagina e saranno poi il motore che renderà possibile alla nostra app di fare tutto quello per la quale è stata pensata.

Di questi andremo a vedere i tre script principali: app.js, controllers.js e servicies.js.

Infine visualizza finalmente il body vero e proprio che in questo momento risulta un operazione molto trasparente, c'è infatti solo un semplice richiamo ad un variabile app "AWG" e nient'altro, vedremo in seguito bene come viene gestita, per ora ci basta sapere che grazie a questo andremo a visualizzare finalmente la nostra prima pagina, quella dove andremo ad effettuare il login.

LOGIN

Qui ci troviamo nella pagina di login, formata dai classici elementi che siamo abituati a vedere, un form che contiene due input di type text che sono username e password ed il pulsante di submit che all'evento onclick ha associato la funzione login().

La prima cosa che notiamo è che è tutto racchiuso in una vista di ionic che delineerà quindi come i nostri elementi saranno visualizzati tramite l'utilizzo dei model all'interno dei nostri input, ed a cascata dei div con associate classi per il corretto posizionamento del loro contenuto.

La funzione login() si trova nel file di script servicies.js, che contiene molte delle funzioni che utilizzo all'interno del programma e al quale sarà dedicato un intero paragrafo, intanto guardiamo questa specifica funzione.

CODICE factory.login()

La prima cosa che si nota è che tra i parametri passati, oltre all'username ed alla password, c'è anche la variabile cb (di CallBack) che ci permetterà di gestire i dati che la funzione di login dovrà restituire a seconda dell'esito dello stesso.

Nello specifico vediamo che in caso di successo la session della nostra applicazione viene creata nella fase di login, e viene inoltre caricata in essa i dati ad essa associati e le viste dei vari elementi, a questo punto viene utilizzata la funzione di cb per restituire il risultato del login che nel nostro caso si trova tutto all'interno della variabile data.

E' proprio questo che ci permette di avere in un unica app la possibilità di cambiare con un semplice controllo sull'utente, il risultato di questa operazione, permettendo di personalizzare l'aspetto dell'applicazione a seconda di chi effettua il login, di visualizzare o no determinati dati o funzioni dell'applicazione a seconda delle autorizzazioni che il determinato utente possiede o addirittura cambiare completamente le funzionalità dell'applicazione, avendo così due o più applicazioni completamente differenti e personalizzate all'interno di un unica applicazione.

La funzione di login ci permette comunque di gestire nello stesso modo anche quando non viene ritrovato un abbinamento utente e password nel database corrispondenti a quelli forniti, in caso di errore si possono dare determinati segnali all'utente ed informarlo che ci sono problemi nell'username o password. Infine in ogni caso viene comunque richiamata la funzione di cb senza passargli però alcun dato, lasciando quindi la pagina così com'è.

APP

In caso di avvenuto login la pagina caricata che è visibile sullo schermo è app.html, l'applicazione vera e propria.

Sulla sinistra si trova il menù di navigazione, che ci permette di scegliere quali tipi di dati vogliamo visualizzare. Si trova tutto racchiuso nell'Ion-side-menu che specifica prima il comportamento degli elementi all'interno del menù e poi del menù stesso.

Ai singoli elementi associa la funzione toggleLeft(), che permette oltre ad una breve animazione di modificare il MenuCtrl, contenente una lista dei dati da visualizzare. Tutto questo avviene all'interno del controllers.js ed anche a questo dedicherò un intero paragrafo. Per ora ci basta sapere che al suo interno si possono modificare dinamicamente i dati da visualizzare, così da poter gestire sia i dati delle diverse viste, sia eventuali modifiche in tempo reale di dati che stiamo visualizzando.

A questo punto nel menù vero e proprio, come suo contenuto si trova appunto il MenuCtrl che contiene a sua volta una lista con i vari elementi da visualizzare. Ora per ogni elemento della lista del menu si crea un nuovo elemento associato ad esso e viene usato come collegamento esterno per ricavare il grafico appropriato, assegnandogli il giusto nome e tipo di grafico. I grafici vengono generati tutti attraverso i chart di angular di cui si è già parlato in precedenza.

Quello che viene visualizzato all'interno di app.html, alla destra del menù, è a sua volta una vista: Chart.html.

In questo file semplicemente sono richiamati i grafici che saranno poi visualizzati, si definisce prima la posizione e il metodo di visualizzazione sempre associandoli al foglio di stile di ionic, divisi in "macro-contenitori" quali sono i chart-container.

Diversamente da come sono generati gli elementi finora in questo caso per i grafici si fa ricorso ad un pratico espediente, essi vengono infatti "filtrati" mentre la pagina viene strutturata, e vengono caricati solo come passo finale, rendendoli visibili e animandoli solo come ultimo passo. Questo è possibile attraverso l'attributo iniziale on-finish-render-filters associato al controller chartCtrl.

In questo modo visualizziamo l'intera pagina con alla destra del menù tutti i grafici rappresentanti i dati che vogliamo visualizzare, completamente generata attraverso script; a questo punto ogni volta che scegliamo un nuovo elemento del menù possiamo vedere come la pagina cambia immediatamente stato, cambiando i dati all'interno dei grafici con le varie animazioni senza alcun bisogno di ricaricare l'intera pagina, ma solo chiedendo i dati realmente necessari al server con semplici chiamate AJAX in background, senza aver bisogno di chiedere al server di generare l'intera pagina e di doverla inviare per intero risparmiando così le risorse del server e la banda in generale.

APP.JS

Iniziamo ora a vedere il cuore di questa applicazione che si trova nei file .js, per primo vediamo il file app.js.

Per prima cosa troviamo questo frammento di codice:

```
var app = angular.module('AWD', [

'ionic',

'AWD.controllers',

'AWD.services'

]);
```

In questa variabile app andiamo a mettere il modulo di angular AWD, includendoci in notazione json anche gli altri moduli ionic,

AWD.controllers e AWD.services, in questo modo possiamo dire che praticamente in app risiede l'intera nostra applicazione.

La prima funzione che andiamo a vedere è quella di config, che imposta la variabile di stato $stateProvider per conoscere da qui in avanti lo stato della nostra applicazione e la variabile di routing $urlRouterProvider che gestisce l'indirizzo di questa applicazione.

Le parti interessanti di questa funzione che andiamo a studiare sono i due state che l'applicazione può assumere:

L'index iniziale alla quale viene subito integrato il file login.html e d associato il controller AppCtrl che gestisce la prima parte dell'applicazione

L'app, alla quale viene assegnato il file app.html e lo stesso controller, ha però una particolarità: è di tipo abstract e verrà caricata di volta in volta uno state differente a seconda dei dati che vorremo visualizzare, in seguito si nota infatti che l'url dello state app.home è /chart/:type che cambia di

volta in volta ed al suo interno come comando va ad insesire nel menuContent dell'app.html il file chart.html completando così la pagina.

Un accenno alla funzione di onFinishRenderFilters che si trova sempre in questo file, è una directive che permette il caricamento dei dati contrassegnati da questa proprietà alla fine del caricamento del resto della pagina, è stato inserito principalmente per ovviare al ritardo di caricamento dei grafici che sono dotati di animazioni e quindi non sono istantanei come il resto, un altro modo per risolvere questo problema sarebbe stato utilizzando un timer ma questa sembrava la soluzione più funzionante ed a livello di codice elegante.

SERVICES.JS

In questo file troviamo il modulo di angular AWD.services che contiene il dispatcherFactory, un insieme di funzioni che vengono gestite dall'app.

Viene tutto creato all'interno della variabile factory e permette di gestire tutto attraverso essa.

Troviamo infatti che le funzioni all'interno di questo file provengono tutte dalla stessa variabile, come la funzione di login che per esteso si chiamerà factory.login.

Come per il login anche la funzione logout si basa su una chiamata al server, questa volta di tipo Get e non Post, che ci permette sempre attraverso il passaggio di tipo data e di altri parametri di gestire sia il

successo sia un eventuale errore di questa chiamata.

A questo punto in caso di avvenuto logout non facciamo altro che azzerare la session e ricaricare il file di index per poter permettere un nuovo login e di ricominciare tutto dall'inizio.

Infine vediamo un'ultima direttiva che sarà poi utilizzata dal file controllers.js che permette di generare un grafico a torta creando direttamente i dati, ora generando direttamente i dati e i parametri dai chart richiesti, ma in seguito recuperandoli direttamente dal server.

In questa vediamo per la prima volta comparire la variabile scope, una variabile che vedremo essere “globale” che verrà largamente usata nel file controllers.

CONTROLLERS.JS

Per finire vediamo cosa contiene il modulo AWD.controllers, inizializzato nella variabile ctrlModule.

Qui si trova la variabile $scope della quale abbiamo accennato poco fa, è una variabile creata una sola volta che è poi possibile richiamare all'interno dell'intero progetto.

In questo file analizzeremo tutti i vari controller che abbiamo trovato fino ad ora lungo i vari file del progetto.

Per primo il controller AppCtrl che andrà a gestire il menu a scomparsa a sinistra dell'applicazione, passerà i dati alla variabile scope attraverso una funzione Date.getDate() per poi completare l'azione eseguendo il bind di questo evento alla funzione delegata toggleLeft, della quale abbiamo visto il funzionamento in precedenza.

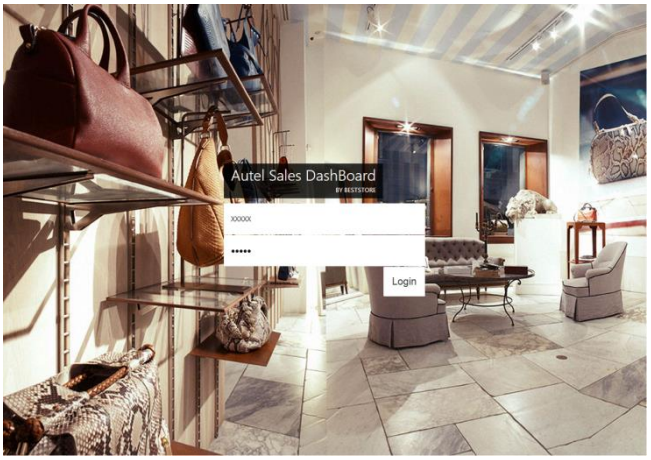
L'UserCtrl è il controller che si occupa in fase di login di recuperare e passare i dati relativi all'utente, sempre attraverso la variabile scope. Per prima cosa creiamo un nuovo dispatcherFactory nella variabile uf (UserFactory), poi inizializzeremo innanzitutto i loginUser con username e password, a questo punto si dichiara la funzione di login vera e propria all'interno della variabile scope. La dichiareremo richiamandola dalla variabile uf creata in precedenza e gli passeremo come parametri username e password recuperati in precedenza.In caso di errore di quest'ultima chiamata lancerà un segnale di Login ko; altrimenti in caso di Login ok prepariamo il LoginUser per un futuro utilizzo azzerandolo ed imposteremo il nuovo stato della nostra applicazione facendola procedere all'app.

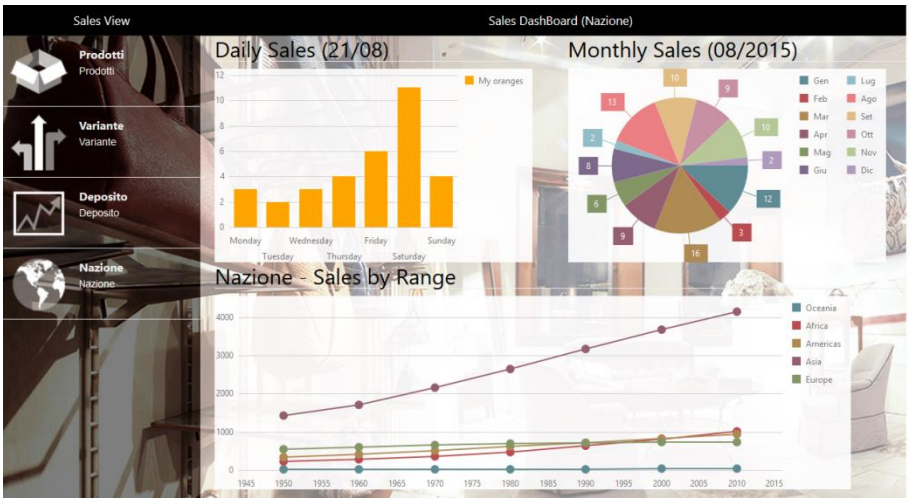
Per finire il controller MenuCtrl che in poche righe riesce a gestire cosa sarà visualizzato all'interno della nostra pagina. Per farlo si limiterà ad usare nuovamente la variabile uf per assegnargli sempre la UserFactory, a questo punto nello scope andremo ad inserire gli item appropriati ricavandoli dalla vista della sessione corrente di uf, in questo modo quando nell'app principale andremo a visualizzare gli items uno ad uno lo faremo sempre nello stesso modo, scorrendo la stessa lista, ma che di volta in volta conterrà i dati appropriati.

RISULTATI

Il risultato ottenuto è un'applicazione web funzionante generata interamente da Script con il framework Ionic in grado di gestire un login iniziale e a seconda di questo mostrare ed elaborare determinati dati con grafici, di vendite o stock, giornalieri e mensili. Questa web app utilizza Json per scambiare i dati con un server; la particolarità di questo tipo di linguaggio è che riesce ad essere universalmente interpretabile e quindi permette di aver un qualsiasi server che può inviarci dati senza aver alcun tipo di problema e senza dover apporre alcuna modifica al client.

Inoltre ricordo che questo tipo di applicazione è in grado di adattarsi ad ogni tipo di dispositivo, mobile o fisso che sia, senza aver problemi di alcun genere e riuscendo quindi a mantenere le stesse performance e la stessa visibilità ovunque.





CONCLUSIONI

Pur avendo un'app web completa e funzionante, una futura implementazione di questa applicazione può essere l'aggiunta di ulteriori funzionalità anche estendendo il login. Tutto questo sarebbe possibile creando altre interfacce e riuscendo quindi a differenziare a seconda delle credenziali il tipo di visualizzazione, i dati visualizzati e il tipo di elaborazione.

Inoltre si potranno attivare e disattivare delle specifiche funzioni a seconda dei privilegi dell'utente attualmente loggato, in modo da mostrare ad esempio dei dati sensibili solo a chi ne possiede davvero l'autorizzazione.

Oppure è possibile personalizzare l'aspetto dell'applicazione a seconda dell'azienda che la sta utilizzando, avremo una login neutra per il primo utilizzo, ma che da li in poi sarà anch'essa sarà dotata di un tema e uno stile fatto appositamente per l'utente.