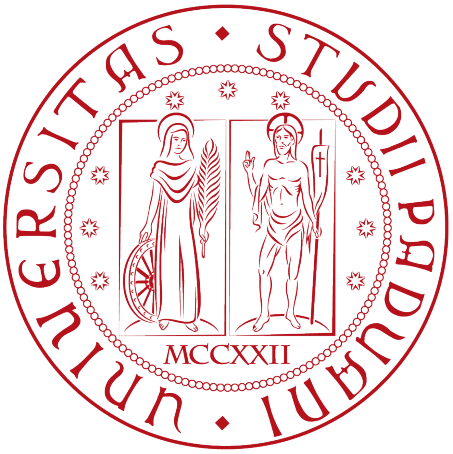
Università degli Studi di Padova

Dipartimento di Matematica

Corso di Laurea in Informatica



Editor visuale per la manipolazione di template HTML

Tesi di laurea triennale

Relatore

Prof. Claudio Enrico Palazzi

Laureando

Daniele Marin

Anno Accademico 2016-2017

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage, della durata di trecentoventi ore, dal laureando Daniele Marin presso l'azienda Zucchetti S.p.a.. L'obiettivo di tale attività di stage è l'analisi di varie librerie Javascript per la realizzazione di template HTML, al fine di poter realizzare un editor grafico che permetta la selezione e la modifica dei template per un loro sucessivo inserimento all'interno di pagine HTML. Inoltre è stato effettuato uno studio sul comportamento dei template in ambito responsive, sulla possibilità di inserire plug-in jQuery all'interno dei template e su di un metodo di caricamento delle librerie controllato in modo di non avere più istanze della stessa libreria se utilizzata da diversi template.

Ringraziamenti

Indice

Elenco delle figure

Elenco delle tabelle

Elenco dei frammenti di codice

Capitolo 1

Introduzione

1.1 L'azienda

Figura 1.1: Logo di Zucchetti S.p.a.

La Zucchetti S.p.a. è una software house con sede a Lodi, che si occupa di soluzioni complete per le aziende, professionisti(commercialisti, consulenti del lavoro, avvocati, curatori fallimentari, notai ecc.) e pubbliche amministrazioni(Comuni, Province, Regioni, Ministeri, società pubbliche ecc.).
Il gruppo Zucchetti è la prima azzienda italiana in Europa con oltre 3300 addetti, 1100 partner e oltre 105000 clienti.
Le soluzioni principali proposte dall'azienda sono :

- Softwere: gestionali, per la sicurezza sul lavoro, analisi business ecc.
- Hardware: per la rilevazione presenze, controllo accessi e controllo produzione.
- Servizi: di outsourcing, cloud computing e data center.

1.1.1 Portal Studio

Lo stage si è svolto nella sede distaccata di Padova che si occupa di ricerca e sviluppo. Tra i software che vengono sviluppati in questa sede è presente Portal Studio che consiste in una WEB application per la creazione di siti web.
L'applicazione offre all'utente un set completo di strumenti per la creazione di pagine web, permette la creazione e modifica in modo grafico della struttura HTML, la gestione degli stili tramite editor grafico per il CSS ed inoltre permette di gestire i dati provenienti da diversi tipi di database, il loro filtraggio e il binding con varie strutture HTML come liste e tabelle.
Il software risulta essere molto maturo e oltre alle funzionalità sopracitate permette

anche la creazione di portlet e pagelet e altri elementi riutilizzabili e la gestione di risorse come dati in formato JSON.

1.2 Il progetto

Il progetto proposto dall'azienda per lo stage, nasce dal desiderio di aggiungere all'applicazione Portal Studio una nuova funzionalità che consiste nell'offrire all'utente la possibilità di inserire nelle proprie pagine HTML dei template già pronti e selezionabili da un insieme prestabilito.
Questo desiderio ha portato l'azienda ad interessarsi ai template engine come Mustache.js, HandlebarJS ecc.
Lo stage si divide in due parti.
La prima consisteva nello studio dei template e degli aspetti ad essi correlati, la seconda nella realizzazione di un editor che ne permettesse la visualizzazione e la modifica.

1.2.1 Prima parte

La prima parte del progetto inizia con la realizzazione di qualche template prototipo, utile sia per studiare le possibilità della libreria scelta sia per avere un insieme di template da inserire nell'editor che è stato realizzato in seguito.
Durante questa parte del progetto l'attenzione è stata rivolta alla possibilità di realizzare template statici, dinamici, template come composizione di altri template (es. lista di contatti) e template che utilizzano SVG.
In seguito alla realizzazione dei template prototipo è stato effettuato uno studio sulla possibilità di rendere i template responsive cioè permetterne la visualizzazione sia su dispositivi desktop che mobile.
La prima parte si è conclusa con uno studio sulla possibilità di realizzare template che contenessero al loro interno plug-in JQuery e sulla gestione del caricamento delle librerie necessarie al funzionamento dei template all'interno della pagina HTML.

1.2.2 Seconda parte

La seconda parte del progetto consisteva nella realizzazione di un editor grafico che permette all'utente la selezione di un template da una lista prestabilita. In seguito alla selezione del template desiderato quest'ultimo verrà visualizzato in un box dedicato e tramite un editor, che viene costruito sulla base dei dati editabili del template (i dati sono contenuti in un oggetto JSON che fa parte del template), è possibile vedere il comportamento del template durante la modifica dei suoi dati.
Per determinati template, come quelli considerati composti, l'editor deve dare la possibilità di visualizzare direttamente l'oggetto JSON contenente i dati e permetterne la modifica.
Non essendo presente una struttura di beck-end proposta dall'applicazione Portal Studio, perché ancora in fase di valutazione, l'editor è stato sviluppato separatamente, l'insieme dei template consiste in un gerarchia di directory contenenti i vari elementi che compongono i template (file HTML, JSON, immagini e librerie) suddivise per categoria.
Il caricamento delle risorse viene eseguito tramite chiamate http-get request, non essendo presenti delle API fornite dall'azienda.

Capitolo 2

Librerie analizzate

In questo capitolo vengono messe a confronto varie librerie *JavaScript* che permettono la realizzazione di template HTML, ne vengono analizzati i pregi e i difetti per arrivare a descrivere i motivi che hanno portato alla scelta della libreria utilizzata nel progetto.

2.1 Considerazioni generali

Negli ultimi anni sono nate molte librerie che permettono la creazione di template HTML che hanno portato notevoli vantaggi agli sviluppatori, offrendo loro un nuovo strumento che permette di creare modelli HTML per la rappresentazione dei dati e riutilizzarli all'interno di pagine differenti con una considerevole diminuzione del codice JavaScript e HTML che normalmente viene utilizzato per la modifica de DOM. Queste librerie si sono evolute velocemente fino ad arrivare a permettere agli sviluppatori di creare intere User interface per applicazioni web, creare componenti riutilizzabili ed in qualche caso offrire funzionalità avanzate come il two-way binding.

2.1.1 I template con sintassi mustache

Le librerie studiate durante lo stage utilizzano tutte questa particolare sintassi, che permette di rappresentare variabili, sezioni, parziali ed altri elementi utili alla creazione del template, tramite l'inserimento di **tag**.
Questi particolari **tag** sono caratterizzati dall'utilizzo delle parentesi graffe come delimitatori e questo è il motivo per cui vengono definiti mustaches (baffi in inglese).
I tag si presentano nella forma "{{I P}}" dove I è un simbolo o una stringa ed identifica il tipo di tag, mentre P è un parametro o una chiave appartenente all'oggetto JSON correlato al template.
Per l'inserimento di variabili o parziali il **tag** è singolo, mentre per l'inserimento di sezioni, controlli del tipo not-exist ed altri, sono presenti un **tag** di apertura ed uno di chiusura.
Per capire meglio il funzionamento che sta alla base di questi template engine è utile fare degli esempi.

Questo esempio mostra il rendering di due variabili.

```
// oggetto JSON contenente i dati
var dati = { name: "Jon", age: 35};
// template HTML con l'aggiunta del tag mustache
var template = "<h1>Il mio nome è {{name}} e ho {{age}} anni.</h1>";

// il risultato del rendering sarà:

Il mio nome è Jon e ho 35 anni.
```

Codice 2.1: Esempio di template rappresentante una variabile.

In questo esempio viene definito il template per rappresentare una lista di prodotti.

```
// oggetto JSON contenente i dati
var dati = prodotti: [
                { name: "pane" },
                { name: "pasta" },
                { name: "biscotti" }
            ];

// template HTML con l'aggiunta del tag mustache
var template = "<p>Lista della spesa:
          <ul>
            {{#prodotti}}
              <li>{{name}}</li>
            {{/prodotti}}
          </ul>
        </p>";

// il risultato del rendering sarà:

Lista della spesa:
- pane
- pasta
- biscotti
```

Codice 2.2: Esempio di template rappresentante una sezione.

2.2 Mustache.js

Figura 2.1: Logo Mustache

Mustache può essere considerato come il padre dei template system, è open-souce e logic-less e presenta implementazioni per i più famosi linguaggi di programmaione, come Java, Phyton, Ruby, PHP, JavaScript e molti altri.
Mustache.js è un implementazione per JavaScript del template system Mustache.
La libreria è molto leggera e versatile visto che permette il rendering sia lato server che lato client.
Le funzioni offerte sono *render* e *parse*, la prima si occupano di creare la stringa HTML contenente il template renderizzato partendo dai dati JSON e dal template HTML e la seconda è opzionale e permette di preparare il template in modo da velocizzare l'operazione di render.
Mustache.js viene definita logic-less perché non presenta nessun tipo di costrutto *if-then-else* e loop come *for* o *do-while*

2.2.1 Come funziona

Il suo funzionamento è molto semplice dato che la libreria offre solamente due metodi.
Il metodo principale è *Mustache.render(data, template)* che prendendo in ingresso il template HTML, con gli opportuni **tag** mustache, e i dati tramite oggetto JSON, restituisce una stringa risultato dall'interpolazione dei due elementi.
La stringa risultante deve essere inserita all'interno della pagina tramite manipolazione del DOM.
Sfortunatamente Mustache.js non offre strumenti per la manipolazione del DOM per cui l'inserimento dovrà essere fatto dal programmatore utilizzando funzioni offerte dallo standard JavaScript o da altre librerie come JQuery.

2.2.2 Pregi e difetti

Uno dei pregi principali è sicuramente la semplicità e la leggerezza della libreria.
Inoltre la stesura dei template risulta intuitiva e semplificata dall'assenza di costrutti *if-else* o cicli *for* .
Come contro si può citare l'impossibilità di creare funzioni aggiuntive per la gestione dei template, possibilità offerta da altre librerie e la totale mancanza di strumenti per la manipolazione del DOM e dei dati del template che costringe il programmatore ad appoggiarsi ad altre librerie.

I template una volta renderizzati sono statici e un cambiamento nei dati non ha effetto sulla loro rappresentazione.

2.3 HandlebarsJS

Figura 2.2: Logo HandlebarsJS

HandlebarsJS è una libreria costruita sopra a Mustache quindi offre tutte le funzionalità di quest'ultimo e aggiunge al normale set di tag anche costrutti di controllo come l'espressione *if* e iteratori come *each*.
La libreria offre anche un set di metodi globali che permettono al programmatore di effettuare varie operazioni sul template e i suoi dati, in più la possibilità di creare delle funzioni personalizzate che possono essere inserite in uno spazio globale e riutilizzate a piacere su diversi template.
Handlebare permette di precompilare il template e offre prestazioni migliori rispetto a Mustache.

2.3.1 Come funziona

Il funzionamento di HandlebarsJS è simile a quello di Mustache, avendo a disposizione l'oggetto JSON contenente i dati e il template, prima si compila il template tramite il metodo *Handlebars.compile(template)*, il risultato della compilazione è una funzione che richiamata passandole come parametro l'oggetto JSON provvede ad interpolare i dati con il template e restituisce una stringa HTML che dovrà essere inserita nella pagina.
Anche in questo caso l'inserimento del template nella pagina deve essere fatto utilizzando strumenti esterni alla libreria perché essa non offre funzioni adeguate.

2.3.2 Pregi e difetti

Anche per Handlebars la leggerezza della libreria e la velocità nel rendering è da considerare un pregio.
Inoltre la possibilità di sfruttare un set di metodi e di poterne creare di propri risulta un vantaggio rispetto a Mustache.
Come Mustache i template una volta creati sono statici e una modifica dei dati non causa un aggiornamento del template che deve essere nuovamente ricompilato.

2.4 Ractive.js

Figura 2.3: Logo Ractive.js

Ractive.js è una libreria sviluppata al theguardian.com per creare **reactive interface** per il web, in modo semplice e efficente.
Questa libreria utilizza la sintassi mustache al pari delle librerie viste in precedenza, quindi un template scritto per funzionare con Mustache.JS o HandlebarsJS può essere utilizzato anche con Ractive.js.
Ractive.js arricchisce la sintassi mustache con nuovi strumenti come:

- Array index;
- Object iterator;
- Special e restricted reference;
- Espressioni;
- Alias.

Oltre all'estensione della sintassi mustache, Ractive.js offre un set molto variegato di opzioni e metodi che possono essere utilizzati per modificare sia il DOM del template che i suoi dati, per l'emissione e la ricezione di segnali e la gestione di animazioni.
Tutti questi strumenti permettono la realizzazione di template interattivi e di una certa complessità, cosa che non è possibile fare utilizzando le librerie precedentemente descritte.
La libreria è compatibile con tutti i browsers e offre un'ottima compatibilità con SVG e lo standard ES15.
I template ottenuti con Ractive.js offrono un *binding* tra la rappresentazione del template e il data model, quando i dati vengono modificati la libreria modifica in modo intelligente ed efficente il DOM del tempalte.

2.4.1 Come funziona

Ractive.js utilizza un approccio differente rispetto alle librerie viste in precedenza.
In particolare viene istanziato un oggetto di tipo Ractive che può essere inizializzato implementando varie opzioni tra quelle offerte dalla libreria.
Le opzioni principali sono l'oggetto JSON contenente i dati, il template e l'elemento HTML a cui dovrà essere agganciato il template all'interno del DOM.

Una volta istanziato l'oggetto viene creata nella **RAM** una rappresentazione virtuale del DOM (virtual DOM) e viene popolato il model con i dati contenuti nel JSON.
In seguito viene renderizzata nel browser la rappresentazione del template interpolato con i dati ad esso relativi.
Ia libreria si occupa di creare un legame tra il model e il virtual DOM.
Nel momento in cui il model subisce variazioni viene effettuato un confronto tra DOM virtuali e nel caso in cui venga rilevato un cambiamento la libreria si occupa di modificare il DOM reale in modo intelligente, cioè andando a modificare solamente l'elemento interessato e non tutta la pagina.
Questa operazione viene effettuata nella RAM confrontando la rappresentazione virtuale precedente con quella successiva alla modifica e quindi risulta molto veloce.

2.4.2 Pregi e difetti

I pregi di questa libreria risiedono nel fatto che offre un insieme completo di strumenti che permettono di gestire l'intero ciclo di vita di un template.
L'arricchimento degli strumenti relativi alla sintassi mustache permette di inserire nel template campi che possono essere risultato di espressioni ed andare a variare sia i dati rappresentati che elementi del CSS, questo permette di creare template che cambiano dinamicamente in base al variare dei dati anche nell'aspetto grafico.
Inoltre la possibilità di creare template che racchiudano HTML, CSS, e comportamento all'interno di un unico file, risulta molto comodo nello sviluppo di quest'ultimi.
Come contro, in relazione alle librerie viste in precedenza si può citare il peso maggiore in termini di risorse e la quantità di tempo neccessaria ad apprendere il funzionamento della libreria.

2.5 Confronto finale

Mettendo a confronto le varie librerie si nota che *Mustache.js* e *HandlebarsJS* sono molto simili sia come supporto alla sintassi mustache che come approccio previsto per la creazione dei template.
Nel confronto fra le due vince sicuramente *HandlebarsJS* per il fatto che offre qualche strumento in più rispetto a *Mustache.js* e la possibilità di poter usufruire degli helpers lo rendono più completo e versatile.
Ractive.js può essere considerata di una categoria superiore rispetto alle due librerie precedenti sia per il suo approccio nella gestione dei template che per il grande numero di funzionalità offerte, che permette la gestione del template in ogni suo aspetto.
Questa libreria può essere utilizzata sia per la creazione di template semplici che molto complessi, inoltre è l'unica delle tre librerie ad offrire la possibilità di rendere interattivi i template tramite il data-binding e la gestione degli eventi.

2.6 Libreria scelta

Non essendo stati stabiliti vincoli sulla creazione dei template, questi possono essere di diversa complessità, possono essere costruiti con linguaggio HTML o SVG e presentare elementi interattivi.
Queste condizioni hanno fatto ricadere la scelta sulla libreria Ractive.js, per le sue caratteristiche che la rendono uno strumento potente e versatile e che non limita le possibilità di sviluppo dei template.

Capitolo 3

Strumenti e tecnologie utilizzati

3.1 Linguaggi utilizzati

I template e l'editor creati durante il periodo di stage sono stati sviluppati per poter essere utilizzati all'interno di un browser, quindi i linguaggi utilizzati sono strettamente legati a questo ambiente.
Questi linguaggi sono i seguenti:

- Il linguaggio HTML è stato utilizzato per creare la struttura dei template e dell'editor;
- Il linguaggio SVG è stato utilizzato nella creazione di template contenenti immagini vettoriali.
- Il linguaggio CSS è stato utilizzato sia per definire la rappresentazione grafica dei template sia dell'editor;
- Il linguaggio JavaScript è stato utilizzato per definire il comportamento dell'editor e per implementare varie funzionalità dei template;
- La sintassi mustache è stata utilizzata all'interno dei template, sia HTML che SVG.

3.2 JavaScript ES5

Il codice JavaScript prodotto durante lo sviluppo del progetto segue gli standard ES5 trattandosi della versione supportata dal JavaScriptCore e compatibile con tutti i browser recenti sia in versione desktop che mobile.
La libreria *Ractive.js* offre il supporto ad ES2015 permettendo di utilizzare le **promise** ed altri elementi che sono stati introdotti nello standard ES6 anche se non supportati dal browser.
Nel caso in cui il browser supporti le **promise**, vengono utilizzate quelle definite nel ES6, altrimenti vengono utilizzate quelle definite dalla libreria che per il momento non supportano *Promise.race* e *Promise.cast*.

3.3 Editors

Per lo svolgimento del progetto poteva essere utilizzato qualsiasi editor di testo ma la scelta è ricaduta su *SublimeText* data la sua versatilità e la sua licenza free.
Questo editor riconosce di default i linguaggi HTML, CSS, e Javascript e offre varie funzionalità come l'auto-completamento del codice, e un set di *bundle* che permettono la stesura del codice in modo automatico tramite l'inserimento di *keyword*.
Inoltre *SublimeText* permette l'estensione delle sue funzionalità tramite l'aggiunta di moduli offerti dalla comunità che lo supporta.

3.4 Inkscape

Inkscape è un software per la grafica vettoriale libero e open, disponibile per varie piattaforme.
Questo software permette la creazione di immagini vettoriali nel formato SVG (Sclable Vector Graphics) e la loro esportazione in più varianti di questo formato.
L'utilizzo di questo strumento è risultato molto utile per la creazione della struttura base di vari template, che in seguito sono stati modificati tramite la sintassi mustache.

3.5 Google Chrome Dev Tools

Sono un insieme di strumenti offerti da Google agli sviluppatori accessibili all'interno del browser Google Chrome.
Questi strumenti risultano molto utili sia nella fase di sviluppo che di testing dell'applicazione perché permettono allo sviluppatore di vedere tutti gli elementi che compongono la pagina, ottenere informazioni sul network, tempi di caricamento delle risorse ed altro.
Inoltre è possibile eseguire il codice passo passo tramite il debuger integrato e utilizzare la console per stampare messaggi (tramite *console.log()*) o eseguire metodi.

3.6 QUnit

QUnit è un framework per il test su script JavaScript.
Normalmente viene utilizzato per effettuare test di unità sulle funzioni o metodi JavaScript a livello di dati ricevuti e restituiti.
Nelle ultime versioni del framework è stata aggiunta la possibilità di effettuare test anche sulla manipolazione del DOM, questa funzione risulta utile per testare script JavaScript che non restituiscono dati ma vanno a manipolare il browser DOM.

Capitolo 4

I template

In questo capitolo viene descritto il lavoro svolto nella prima parte del progetto, che consiste nella realizzazione dei template, nel spiegare come sono stati resi responsive ed in fine come sia stato possibile inserire *plug-in* jQuery al loro interno.
Viene anche trattato l'argomento relativo al loro caricamento all'interno della pagina HTML e la gestione delle librerie.

4.1 Utilizzo di Ractive.js

In questa sezione viene descritta più in dettaglio la libreria scelta per svolgere il progetto.

4.1.1 L'oggetto Ractive

Per iniziare ad usare Ractive bisogna innanzitutto creare un istanza dell'oggetto Ractive e passarle le opzioni desiderate tra quelle offerte dalla libreria.
Una volta creato l'oggetto questo si occuperà di creare e popolare il proprio registro dati e di creare in memoria una rappresentazione virtuale del template.

```
1 var ractive = new Ractive({
2   el: '#container', // id dell'elemento target nella pagina
3   template: '<p>{{greeting}}, {{recipient}}!</p>', // esempio di template
4   data: { greeting: 'Hello', recipient: 'world' } // oggetto JavaScript
      contenente i dati
5 });
```

Codice 4.1: Creazione di un oggetto Ractive.

4.1.2 Le opzioni principali

La libreria fornisce un insieme consistente di opzioni che possono essere inizializzate alla creazione dell'oggetto Ractive.
Queste si dividono in sei categorie che sono le seguenti:

- data;
- templating;
- transitions;
- binding;
- parse;
- lifecycle event.

Tutte queste opzioni opportunamente inizializzate vanno a definire le proprietà ed il comportamento del template nel suo intero ciclo di vita.
Le opzioni principali sono:

- **el**: identifica l'elemento target all'interno del browser DOM dove verrà renderizzato il template;
- **template**: rappresenta il template da renderizzare;
- **data**: rappresenta i dati che dovranno essere interpolati con il template;
- **compute**: oggetto che può contenere espressioni o funzioni che verranno ricalcolate al variare del data model;

4.1.3 Il two-way data biding

La libreria fornisce la funzionalità di one-way-binding tra i dati contenuti nel data registry e il virtual DOM, cioè al variare del model il virtual DOM e di conseguenza anche la sua rappresentazione nel browser vengono modificate.
Oltre a questo tipo di legame, la libreria offre, per elementi di input , il two-way-binding, che permette di modificare il model al variare dei dati nella view e viceversa.

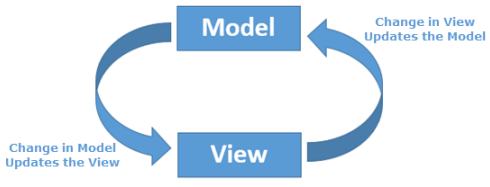


Figura 4.1: Rappresentazione two way data binding.

4.1.4 Gli eventi

La libreria Ractive implementa il pattern architetturale *publish/subscribe*, che permette di rispondere o innescare particolari eventi, che attualmente vengono gestiti su due livelli.
Il primo è un interazione a basso livello con gli eventi del DOM che viene specificata tramite *directives* del template che specificano anche come l'evento deve essere gestito, tramite *proxy event* o chiamate a metodi.
Il secondo è gestito dalle API *publish/subscribe* e dal sistema di eventi all'interno di Ractive e tra i componenti.
I *proxy events* collegano gli eventi del DOM con gli eventi di Ractive, mentre le chiamate a metodi direttamente sull'istanza ractive non utilizzano l'infrastruttura *publish/subscribe*.

4.1.5 Il virtual DOM

Ractive utilizza un sistema differente dagli altri per tracciare le modifiche, ricorrendo al cosiddetto Virtual DOM, ossia a una rappresentazione virtuale della struttura del template immagazzinata in memoria e del tutto simile al DOM originale, del quale può essere vista come una astrazione.
Nel momento in cui si verifica un evento ed è necessario reagire ad esso modificando gli elementi della pagina, Ractive applica prima tali interventi al Virtual DOM.
Attraverso l'analisi delle differenze tra lo stato del Virtual DOM precedente al verificarsi dell'evento e quello nuovo ottenuto dall'applicazione delle modifiche, Ractive determina i cambiamenti effettivi da apportare al DOM vero e proprio.
Il calcolo delle differenze tra i due stati del DOM virtuale è estremamente veloce, e grazie a esso si limitano al minimo indispensabile gli interventi sul DOM reale, tendenzialmente più lento, garantendo quindi ottime performance.

4.1.6 Plug-in di terze parti

I *plug-in* permettono di aumentare le funzionalità offerte dalla libreria Ractive.
Agli sviluppatori è data la possibilità di creare i propri *plug-in* o di scegliere quelli più adatti alle proprie esigenze da una lista presente sul sito di Ractive.js.
Durante lo sviluppo del progetto sono stati utilizzati i *plug-in* ractive-events-tap, per gestire il click/tap sui dispositivi mobili , e ractive-load, per il caricamento tramite protocollo *http* dei template.

4.2 Sviluppo dei template

La creazione dei template, per il progetto, non presentava nessuna restrizione né per la forma né per i contenuti, quindi la decisione di quali template sviluppare era a discrezione del programmatore.
L'unica richiesta, per questa fase del progetto, è stata quella di avere template sia HTML che SVG.

4.2.1 Struttura dei template

I template HTML sono strutturati come una pagina web, cioè sono formati dal codice HTML per quanto riguarda i contenuti, il codice CSS per la loro rappresentazione grafica e JavaScript per definire il loro comportamento.
Il codice HTML rappresentante il template deve essere arricchito tramite la sintassi *mustache* contenente le variabili, le espressioni e tutti gli elementi utili al funzionamento del template.
Inoltre possono essere presenti le direttive necessarie nel caso il template presenti la possibilità di interazione con esso.
Il comportamento del template viene sviluppato tramite gli strumenti offerti dalla libreria Ractive.
La libreria permette di creare un singolo file che contiene al suo interno l'intero template cioè HTML+mustache, CSS e Javascript.
Questo risulta molto vantaggioso per diminuire la quantità di file e migliorare l'organizzazione dei vari template.
Ogni template dovrà inoltre avere un file con estensione *.json* contenente i dati da visualizzare ed eventuali immagini o librerie necessarie per il suo corretto funzionamento.
I vari template sono stati suddivisi in cartelle nel seguente modo:

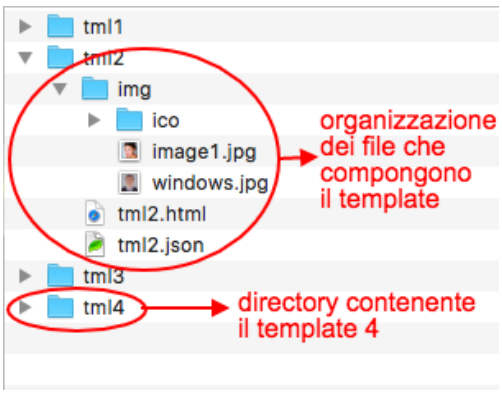


Figura 4.2: Organizzazione dei template nel file system.

4.2.2 Prototipo di template

Di seguito viene riportato una parte del codice utilizzato per la creazione di un template HTML.

```
<div id="tml1" style="background-color: {{bgColor}}; color: {{textColor}};">
  <img class="image" src= {{ img ? img : 'templates/tml1/img/no-image.png'}}>
  <div class="prod-data">
    <h2 class="center">{{nome}}</h2>
    prezzo: <strong>{{prezzoCent}} euro</strong>
    <p class={{ quantita ? 'disponibile' : 'non-disponibile'}}>Disponibilità:
    {{#if quantita}}
      <strong>disponibili {{quantita}}</strong>
    {{else}}
      <strong>Non disponibile</strong>
    {{/if}}
    </p>
    <h3>Descrizione:</h3>
    <p class="desc">{{descrizione}}</p>
  </div>
</div>

<!-- stili -->
<style>
  <!-- parte relativa alle regole CSS -->
  ...
</style>

<script>
  <!-- parte relativa al comportamento del template -->
  component.exports = {
    computed: {
      prezzoCent: function(){
        return this.get('prezzo').toFixed(2);
      }
    }
    ...
  }
</script>
```

Codice 4.2: Esempio di template.

Come si può notare il file è suddiviso in tre sezioni.
La prima contiene il codice HTML con l'aggiunta di variabili ed espressioni inserite tramite mustache.

```
<p class={{ quantita ? 'disponibile' : 'non-disponibile'}} >
```

Codice 4.3: Espressione con mustache.

Nell'esempio sopra riportato il codice permette di modificare l'attributo "classe" del tag *p* in base al valore della variabile *quantita*.
La porzione di codice seguente rappresenta una condizione *if* che visualizza nel template la stringa *"disponibili: numero prodotti"* se *quantita* è maggiore di 0 e *"non disponibile"* altrimenti.

```
  {{#if quantita}}
    <strong>disponibili {{quantita}}</strong>
  {{else}}
    <strong>Non disponibile</strong>
  {{/if}}
```

Codice 4.4: Condizione if-else con mustache.

Il file contiene anche l'implementazione dell'opzione *computed* fornita dalla libreria Ractive.js che verrà utilizzata durante la creazione dell'oggetto Ractive.
Tramite il metodo *export* di component è possibile definire la logica del template utilizzando le opzioni e i metodi forniti dalla libreria Ractive.js.
In questo caso viene indicato che ogni volta che il prezzo subisce modifiche questo deve essere convertito in una stringa e rappresentato con due decimali.

```
component.exports = {
  computed: {
    prezzoCent: function(){
      return this.get('prezzo').toFixed(2);
    }
  }
  ...
}
```

Codice 4.5: Implementazione ed esportazione opzione computed.

Al template viene fornito un oggetto JSON contenente dei dati di default, utilizzati per ottenere una rappresentazione del template completa.
Questi dati sono contenuti in un file con estensione *.json* presente all'interno della directory del template.
L'oggetto JSON è definito come segue.

```
{
  "bgColor":"#ededed",
  "textColor":"#000000",
  "nome": "Cuffie rosse WiFi",
  "img": "templates/tml1/img/cuffie.jpg",
  "prezzo": 200,
  "quantita": 6,
  "descrizione": " ... "
}
```

Codice 4.6: Rappresentazione dell'oggetto JSON.

In seguito vengono proposte due immagini del template descritto in precedenza dopo il rendering da parte della libreria Ractive.js.
Queste immagini fanno vedere come cambia la rappresentazione del template quando la variabile *quantita* è maggiore di 0 e quando è uguale a 0.

Figura 4.3: Rappresentazione del template con *quantità* maggiore di 0.

Figura 4.4: Rappresentazione del template con *quantità* uguale a 0.

Nel caso in cui il template utilizzi SVG, la struttura del template è la stessa, con l'unica differenza che la sintassi mustache viene aggiunta al codice SVG.

4.3 Rendere il template responsive

Uno dei requisiti richiesti per quanto riguarda lo sviluppo dei template è quello di renderli *responsive*, in modo che sia possibile visualizzarli su dispositivi desktop e mobile.
La soluzione individuata per soddisfare questo requisito è stata quella di inserire delle *media-query* all'interno del CSS del template, definendo le regole per la corretta rappresentazione a diverse risoluzioni.
La pagina che andrà ad ospitare i template dovrà contenere il meta-tag *viewport* per garantire la corretta visualizzazione.
Utilizzando questo metodo si riesce a mantenere il CSS all'interno del file che contiene il template, senza ricorrere ad uno o più file esterni contenenti le regole relative a vari dispositivi.

Di seguito viene proposto un esempio di media query.

```
@media only screen and (min-width: 320px) and (max-width: 639px){

  .image{
    width: 90%;
    ...
  }

  .prod-data{
    float: none;
    ..
  }

  ...
}
```

Codice 4.7: Esempio di media query nel CSS del template.

4.4 Inserimento plug-in jQuery nei template

Oltre a rendere i template responsive, un'altra richiesta da soddisfare durante questa fase del progetto consisteva nello studiare la possibilità di sviluppare template che contenessero plug-in jQuery.
L'utilizzo di plug-in jQuery implica:

- il caricamento della libreria del plug-in;
- la stesura del codice HTML seguendo le regole fornite dallo sviluppatore della libreria;
- la stesura di uno script JavaScript che va a richiamare le funzionalità desiderate tra quelle offerte dal plug-in.

Questo particolare script deve essere eseguito in seguito al caricamento del template all'interno della pagina.
La soluzione individuata per avere template che contengano plag-in jQuery consiste nell'inserire lo script per il funzionamento del plug-in come implementazione dell'opzione *oncomplete* dell'oggetto Ractive.
L'opzione *oncomplete* fa parte dei *lifecycle events* di Ractive e viene richiamata nel momento in cui il template è stato completamente renderizzato.
Questa soluzione oltre a permettere l'inserimento dello script all'interno del file del template, assicura che la sua esecuzione avvenga in maniera corretta.
In seguito viene riportato un esempio di implementazione dell'opzione *oncomplete* per l'utilizzo di un plug-in jQuery.

```
<script>
  component.exports = {
    oncomplete: function() {
      jQuery(this.find('#img1')).actuate('wobble');//find metodo di Ractive
      jQuery(this.find('#img2')).actuate('pulse');//actuate metodo plug-in
    }
  }
</script>
```

Codice 4.8: Implementazione dell'opzione *oncomplete*

Ogni template che utilizza plug-in jQuery contiene nella propria directory, oltre alle librerie necessarie anche un file *.json* che contiene un array con le *path* delle librerie da caricare.

4.5 Caricamento dei template nelle pagine HTML

Per effettuare il caricamento dei template all'interno della pagina, sono state sviluppate varie funzioni.
Queste funzioni si distinguono per il tipo di template che devono caricare, cioè:

- caricamento di template senza plug-in jQuery;
- caricamento di template con plug-in jQuery e script di attivazione del plug-in all'interno del file del template;
- caricamento di template con plug-in jQuery e script di attivazione del plug-in come file esterno al file del template;

Quest'ultimo caso è stato inserito per dare una possibilità in più nello sviluppo dei template, anche se non risulta essere la scelta migliore.
Queste funzioni ,una volta ottenuto il nome del template da caricare, si occupano di effettuare una richiesta *http-get* per ottenere i vari componenti del template.
In seguito, utilizzando le risorse ottenute, viene istanziato un oggetto Ractive per ogni template.
Quest'ultimo si occupa di renderizzare il template nella pagina.
Per quanto riguarda il caricamento dei template che utilizzano plug-in jQuery, prima di effettuare la richiesta del template e dei suoi dati, viene effettuato il caricamento delle librerie necessarie al funzionamento del plug-in.
Questo viene fatto leggendo il file *libs.json* relativo al template e inserendo all'interno della pagina la richiesta di caricamento delle librerie.
Su richiesta del tutor aziendale è stata sviluppata una funzione che controlla le librerie che dovranno essere caricate dal browser e nel caso in cui la libreria che deve essere caricata con il template sia già presente non viene effettuata un'ulteriore richiesta di caricamento.
Questo comportamento evita di effettuare richieste superflue nel caso venissero caricati più template che utilizzano la medesima libreria.

In seguito viene proposto il codice delle funzioni che si occupano di gestire il caricamento delle librerie.

```
//funzione che controlla se la libreria fa parte di quelle gia caricate
var confrontaScript = function confrontaScript(library) {
  var scriptArray = document.scripts;
  var trovato = false;
  for (var i = 0; i < scriptArray.length && !trovato; i++) {
    var scriptUrl = scriptArray[i].attributes.src.value;
    var scriptName = scriptUrl.slice( scriptUrl.lastIndexOf('/')+1, scriptUrl
    .length);
    if (scriptName === library) {
      trovato = true;
      console.log('lo script '+library+' è già presente!');
    }
  }
  return trovato;
}

this.confrontaScriptByUrl = function(url) {
  var libraryUrl = url;
  var libraryName = libraryUrl.slice( libraryUrl.lastIndexOf('/')+1,
    libraryUrl.length );
  return confrontaScript(libraryName);
}
// funzione che si occupa se necessario di aggiungere la richiesta di
    caricamento alla pagina
this.addLibraryFromUrl = function(url) {
  var libraryUrl = url;
  var libraryName = libraryUrl.slice( libraryUrl.lastIndexOf('/')+1,
    libraryUrl.length );
  if (!confrontaScript(libraryName)) {
    var node = document.createElement('script');
    node.setAttribute('src', libraryUrl);
    document.head.appendChild(node);
    console.log('libreria '+libraryName+' aggiunta al file HTML');
  }
  else {
    console.log('libreria '+libraryName+' gia presente nel file HTML');
  }

}
```

Codice 4.9: Funzioni che si occupano del caricamento delle librerie

4.5.1 Problemi riscontrati

Un problema che è stato rilevato, a riguardo del caricamento di template che utilizzano jQuery consiste nel fatto che il browser effettua il caricamento delle librerie in modo asincrono, quindi non gestibile da parte dello sviluppatore.
Nel caso in cui la libreria non sia ancora stata caricata prima dell'avvenuto rendering del template, si verifica un errore perché le funzioni richiamate non vengono riconosciute.
Il problema riscontrato sarebbe risolvibile se fosse possibile rilevare l'avvenuto caricamento delle risorse richieste, ma momentaneamente i browser non offrono strumenti adeguati.
Una delle possibili soluzioni è quella di effettuare il *reload* della pagina una volta che questa è stata completamente caricata.

Capitolo 5

Analisi dei Requisiti

5.1 Applicazione per la modifica dei template

5.1.1 Visualizzazione lista dei template

5.1.2 Visualizzazione template selezionato

5.1.3 Editor per la modifica del template

5.2 Requisiti individuati

5.3 Riepilogo requisiti

Capitolo 6

Progettazione

6.1 Suddivisione template

6.2 Caricamento template

6.3 Creazione lista template

6.4 Visualizzazione template selezionato

6.5 Editor per la modifica del template

Capitolo 7

Realizzazione

7.1 Il caricamento dei template

7.2 Controllo delle librerie caricate

7.3 Visualizzatore lista template

7.4 Visualizzatore template selezionato

7.5 Editor per la manipolazione del template

Capitolo 8

Conclusioni

In questo capitolo finale vengono tratte le conclusioni riguardo alle attività svolte durante il periodo di stage.

8.1 Valutazione del risultato e di Ractive.js

8.1.1 Requisiti soddisfatti

8.2 Criticità

8.3 Conoscenze acquisite