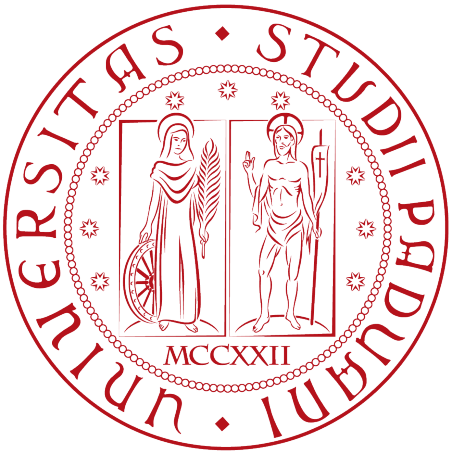
Università degli Studi di Padova

Dipartimento di Matematica

Corso di Laurea in Informatica



Sviluppo test driven di un front-end con AngularJS

Tesi di laurea triennale

Relatore

Prof. Mauro Conti

Laureando

Mattia Sorgato

Anno Accademico 2014-2015

Lorem ipsum dolor sit amet, consectetuer adipiscing elit.

— Oscar Wilde

Dedicato a ...

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di Stage, della durata di circa trecento ore, dal laureando Mattia Sorgato presso l'azienda IKS s.r.l. Gli obbiettivi da raggiungere erano molteplici.
In primo luogo, l'ideazione e l'implementazione del Front-end di un software per la gestione del personale e dei progetti aziendali.
In secondo luogo, la scrittura delle Application Program Interface (API) necessarie alla stesura del Back-end.
Per garantire la qualità del codice, durante l'intero progetto l'approccio utilizzato è stato Test Driven.

"Life is really simple, but we insist on making it complicated"

— Confucius

Ringraziamenti

Innanzitutto, vorrei esprimere la mia gratitudine al Prof. Mauro Conti, relatore della mia tesi, per l'aiuto e il sostegno fornitomi durante la stesura del lavoro.

Desidero ringraziare con affetto i miei genitori per il sostegno, il grande aiuto e per essermi stati vicini in ogni momento durante gli anni di studio.

Ho desiderio di ringraziare poi i miei amici per tutti i bellissimi anni passati insieme e le mille avventure vissute.

Padova, Jun 2015 Mattia Sorgato

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Introduzione al contesto applicativo.

Esempio di utilizzo di un termine nel glossario

Esempio di citazione in linea
Manifesto Agile. URL: http://agilemanifesto.org/iso/it/.

Esempio di citazione nel pie' di pagina
citazione[1]

1.1 L'azienda

Dal 1999 IKS focalizza il proprio agire per fornire ai Clienti soluzioni abilitanti per la realizzazione di servizi di e-business, nell'ambito della sicurezza e della Governance Information and Communication Technology (ICT), il tutto con un approccio rivolto all'ottimizzazione.
Sin dall'inizio, l'azienda investe su temi strategici per l'ICT: sicurezza informatica, automazione e Governance di infrastrutture, sviluppo di soluzioni software complesse e tecnologicamente innovative, ottimizzazione di architetture informatiche di alto livello.
La propensione all'innovazione ed il costante investimento nella formazione permettono alla società a metà degli anni Duemila di affermarsi a livello nazionale con sedi a Padova, Milano e Roma e di diventare il riferimento tecnologico di aziende di alto livello che affidano all'infrastruttura ICT applicazioni e servizi business critical.
Grazie alla propria vocazione innovativa ed al continuo sviluppo, IKS oggi si presenta come un gruppo aziendale in grado di fornire supporto su temi che spaziano dalla Compliance all'ottimizzazione di processi di filiera, dal WEB 2.0 all'energy management.

[1] Daniel T. Jones James P. Womack. *Lean Thinking, Second Editon*. Simon & Schuster, Inc., 2010.

figura 1.1: Logo di IKS

1.1.1 Aree di intervento

Sicurezza La priorità è garantire che le risorse e i servizi di un sistema informativo siano sempre accessibili, fruibili, garantendone però l'integrità e la conformità; proteggere l'azienda da violazioni e attacchi informatici accidentali e fraudolenti.

Ottimizzazione dell'infrastruttura In tempi di virtualizzazione, consolidamento e risparmio energetico ed in attesa del cloud computing è fondamentale ripensare al modello di infrastruttura con criteri legati alle modalità e tempistiche di provisioning dei sistemi, alla semplificazione dell'infrastruttura, al nuovo paradigma di sicurezza, al cost effective.

Governance La Governance è la risposta per l'azienda che oggi è chiamata a confrontarsi con la sempre maggiore complessità dei data center e delle applicazioni, la necessità di gestire i fenomeni dirompenti della virtualizzazione e dell'accesso a servizi in the cloud, l'obbligo stringente e vincolante di mantenere adeguati livelli di performance, nonché di ottimizzare i costi IT.

Ricerca e innovazione Progettare e costruire un'applicazione a valore aggiunto significa conoscere le possibilità e potenzialità della tecnologia. Il valore aggiunto è dato dal poter garantire competenze sempre aggiornate, continue attività di ricerca e di prototipizzazione su temi innovativi, capacità consolidate nel disegno di architetture complesse e di integrazione, forti partnership tecnologiche con i principali leader di mercato. Tutto ciò sostenuto da una forte, determinata quanto ragionevole propensione all'innovazione.

1.2 L'idea

Il progetto Sviluppo test driven di un front-end con AngularJS nasce come strumento di gestione interna delle competenze dei dipendenti e dei progetti presenti in azienda. L'obiettivo di tale progetto è una migliore organizzazione del personale dedicato ai vari progetti, con una conseguente maggiore efficienza ed efficacia nell'assegnazione dei ruoli nei processi ed attività. Fino ad ora, all'interno di IKS, tale compito veniva svolto dalla consultazione e modifica di un foglio elettronico *Excel*. Questa soluzione, oltre ad essere poco pratica e difficilmente manutenibile, ha portato all'esigenza di uno

strumento più specifico ed espandibile. Da qui la creazione del progetto Sviluppo test driven di un front-end con AngularJS.
Lo strumento Sviluppo test driven di un front-end con AngularJS è pensato per essere utilizzato da diverse tipologie di utenti, dal singolo sviluppatore al project manager.
L'applicazione sarà composta da un Front-end realizzato in *AngularJS*, un Back-end realizzato tramite Spring e un database di persistenza. L'attività di Stage ha riguardato la realizzazione del Front-end e la definizione delle API Representational State Transfer (REST) del Back-end.
L'utente finale sviluppatore può vedere Sviluppo test driven di un front-end con AngularJS come una sorta di curriculum digitale ampliato, in cui poter inserire le proprie competenze riconosciute, le esperienze acquisite ed i progetti a cui ha preso parte. Nella propria *dashboard*, l'utente finale può gestire le proprie informazioni e richiedere l'inserimento di nuove entità, come nuovi progetti a cui ha preso parte, nuove skill acquisite, certificazioni esterne o interne o altri titoli formativi e le sue esperienze lavorative pregresse.
Chi ha compiti di gestione (ad esempio la figura del Responsabile di progetto), utilizza lo strumento per vedere quali risorse può utilizzare per un dato progetto. In base alle competenze richieste può quindi consultare quali risorse può richiedere per svolgere il progetto con maggior efficienza ed efficacia.
Si prefissa quindi come obiettivo la creazione di un portale aziendale unificato per una migliore gestione del personale ed il suo dislocamento all'interno dei progetti aziendali, oltre che alla gestione dei curricula vitae del personale aziendale.
Per quanto riguarda l'organizzazione del progetto, è stato deciso che Sviluppo test driven di un front-end con AngularJS avrebbe avuto due layer Front-end e Back-end comunicanti tramite chiamate REST. Questo per rendere le due componenti più separate possibile, dovendo soltanto definire le API di comunicazione tra il client ed il server. Inoltre, essendo che questo progetto di Stage ha coperto la parte Front-end, in un progetto futuro di implementazione del Back-end, il server dovrà solamente esporre le corrette funzionalità alla chiamata delle API REST concordate.

1.3 Vincoli

Questa sezione illustra i vincoli che sono stati stipulati all'inizio dell'attività di Stage, sia tecnici che organizzativi.

1.3.1 Tecnologici

I vincoli sullo stack tecnologico da utilizzare nell'implementazione di Sviluppo test driven di un front-end con AngularJS sono stati fissati sin dai primi contatti. Per questo, la prima settimana di Stage è stata rivolta all'apprendimento di tali strumenti e all'integrazione con gli strumenti e le procedure aziendali.
Tali strumenti sono:

* *AngularJS*, framework JavaScript per la scrittura di applicazioni web Front-end;
* *Bootstrap*, libreria Cascading Style Sheet (CSS) grafica per gestire la presentazione di pagine web HyperText Markup Language (HTML);
* *JasmineJS*, framework JavaScript utilizzato per la scrittura di test, soprattutto di unità ed End-To-End (E2E).

La scelta di queste tecnologie ovviamente non è casuale. Nello sviluppo di applicazioni web con *AngularJS*, lo stack tecnologico utilizzato più diffusamente è proprio questo. L'utilizzo di *Jasmine* è quasi obbligatorio, data la sua immediatezza di configurazione con *AngularJS* e la filosofia stessa del framework Front-end, che è stato pensato appunto per rendere il testing più semplice.
Bootstrap, d'altro canto, è estremamente facile da integrare con qualsiasi strumento che si basi su HTML per fornire le sue viste.

1.3.2 Temporali

Per la realizzazione di questo progetto sono state impiegate le 8 settimane di lavoro di 40 ore/settimana. Il periodo in cui si è svolto ha compreso le settimane che sono intercorse tra il 20 Aprile e il 13 Giugno, per un totale di ore compreso tra le 300 e 320 stabilite all'inizio della pianificazione.
Di queste ore, le prime 40 sono state dedicate alla mia formazione sulle tecnologie adottate nel progetto e nell'istruzione sull'utilizzo degli strumenti aziendali, per poi iniziare la progettazione e lo sviluppo vero e proprio del progetto.

1.3.3 Organizzativi

La realizzazione del progetto prevedeva un ciclo di vita basato sullo stile Agile. Questo è l'approccio maggiormente utilizzato all'interno dell'azienda, caratterizzato dall'applicazione dei principi dell'*Agile programming* con consultazione di una *kanban*.
Inoltre, i progetti aziendali di IKS sono gestiti da un sistema di versionamento di Atlassian, compreso il progetto oggetto del mio stage.
I vincoli organizzativi si sono quindi concretizzati attraverso la comprensione e l'utilizzo degli strumenti JIRA e STASH utilizzati all'interno dell'azienda, i quali assolvono appunto i compiti di gestione, rispettivamente, di una *Agile kanban* e di un repository aziendale.

1.4 Organizzazione del testo

Riguardo la stesura del testo, relativamente al documento sono state adottate le seguenti convenzioni tipografiche:

- gli acronimi, le abbreviazioni e i termini ambigui o di uso non comune menzionati vengono definiti nel glossario, situato alla fine del presente documento;
- per la prima occorrenza dei termini riportati nel glossario viene utilizzata la seguente nomenclatura: *parola*[g];
- i termini in lingua straniera o facenti parti del gergo tecnico sono evidenziati con il carattere *corsivo*.

Capitolo 2

Processi e metodologie

Breve introduzione al capitolo
Questo capitolo tratta delle metodologie di sviluppo adottate nello sviluppo del progetto e dello stack tecnologico che ha permesso il suo completamento.

2.1 Metodologie

2.1.1 Agile

Agile è una metodologia di sviluppo nata in contrapposizione ad altri modelli più stringenti e formali, quali ad esempio il modello a Cascata o a Spirale.
I principi generali dell'Agile Programming sono descritti nell'Agile Manifesto[1], e possono essere riassunti in quattro punti cardine:

- *individui e interazioni:* organizzazione e motivazione autonoma sono importanti, come lo sono le interazioni personali come la condivisione dello stesso luogo di sviluppo;
- *software funzionante:* un prodotto che funziona è più utile e meglio accettato di documenti cartacei presentati agli acquirenti durante i meeting;
- *collaborazione con gli acquirenti:* i requisiti non possono essere pienamente individuati all'inizio del ciclo di sviluppo del software. Perciò l'interazione con i clienti e gli stakeholder è estremamente importante;
- *responsività al cambiamento:* i metodi agile sono focalizzati sul fornire risposte veloci al cambiamento e allo sviluppo continuo.

Lo sviluppo Agile permette di valutare ed eventualmente correggere la direzione durante il processo stesso. Questo risultato è ottenuto attraverso brevi e regolari iterazioni di lavoro, alla fine delle quali ogni team deve presentare un incremento del prodotto, considerandolo come una nuova feature applicabile e funzionante. Concentrandosi sulla ripetizione di cicli di lavoro brevi e definiti, proporzionati alla funzionalità da consegnare, le metodologie Agile si definiscono "iterative" o "incrementali". Nel modello a cascata, i team di sviluppo hanno una sola opportunità di realizzare un aspetto del progetto nel modo giusto. Nel paradigma Agile, ogni aspetto dello sviluppo - requisiti,

[1] *Manifesto Agile.* URL: http://agilemanifesto.org/iso/it/.

progettazione, ecc. - è continuamente rivisitato. Quando un team si ferma e rivaluta la direzione di un progetto ogni due settimane, è possibile cambiare tale direzione con facilità.
Questo approccio "ispeziona-e-adatta" riduce i costi e il tempo di consegna dello sviluppo. Dato che i vari team possono sviluppare software nello stesso tempo in cui individuano i requisiti, la "paralisi dell'analisi" ha meno probabilità di bloccare i progressi di un team. E siccome il ciclo di lavoro di un team è limitato a due settimane, gli stakeholder hanno opportunità ricorrenti di analizzare i rilasci del software e il loro feedback dal mercato.
La metodologia Agile si basa sul concetto di *User Story*, ovvero un compito significativo che l'utente vuole poter svolgere attraverso il software realizzato. Le User Stories catturano il 'chi', 'cosa' e 'perché' di un requisito in maniera semplice e concisa.

Kanban

Kanban è un framework usato per implementare la metodologia agile. Negli anni '40, Toyota ottimizzò i suoi processi modellandoli come se fossero degli scaffali in un supermercato. I supermercati offrono una quantità di prodotti atti a soddisfare con il minimo spreco la domanda dei consumatori. Siccome i livelli di inventario sono conseguenti ai pattern di consumazione, il supermercato ottiene una significante efficienza ed ottimizzazione nella gestione dell'inventario.
Quando Toyota portò questa idea ai suoi piani di lavoro, i team (come ad esempio il team che aggiunge le portiere al telaio dell'auto) consegnavano una carta, o "kanban", agli altri dipendenti (ad esempio, ai team che assemblano le portiere) per segnalare di aver ecceduto la capacità e di essere pronti a ritirare più materiale. Anche se la tecnologia di segnalazione si sia evoluta, questo sistema è ancora al centro della produzione "just in time".
Kanban presenta lo stesso comportamento per i team software. Tramite la comparazione dell'ammontare del lavoro in progresso rispetto alla capacità del team, kanban fornisce agli stessi opzioni di pianificazione più flessibili, output più veloci, miglior concentrazione sui singoli compiti e trasparenza durante il ciclo di sviluppo.
Kanban è un metodo di gestione della consapevolezza del lavoro con un'enfasi particolare sulla consegna "just in time", assicurandosi nel mentre che i membri del team non abbiano troppo lavoro rispetto alle loro capacità di carico. In questo approccio, il processo, dalla definizione dei task alla consegna al cliente, è mostrato visivamente ai partecipanti. I membri del team estraggono ogni unità di lavoro da una coda. Kanban nel contesto dello sviluppo software può essere inteso come un sistema di gestione di processo visuale, che dice cosa produrre, quando e in quale quantità produrlo - ispirato dal sistema di produzione Toyota.
Nella sua forma più basilare, un sistema di kanban consiste di una grande lavagna appesa ad un muro con delle carte o dei postit organizzati in colonne con dei numeri su ogni colonna[2].
Le carte rappresentano le unità di lavoro mentre attraversano il processo di sviluppo, rappresentato dalle colonne.
I limiti sono la differenza sostanziale tra una lavagna Kanban e un'altra storyboard qualsiasi. Limitando l'ammontare di Work-In-Progress (WIP) ad ogni passo del processo, si previene la sovrapproduzione di risorse e si rivelano dinamicamente i colli di bottiglia, così che possano essere presi dei provvedimenti adattativi il prima possibile. Ogni team può quindi avere una determinata quantità di lavoro in esecuzione per unità

[2] *Kanban Agile.* URL: http://kanbanblog.com/explained/.

di tempo. Così facendo si limitano gli sprechi di tempo dovuti al cambio di contesto chiesto se si implementa un certo numero di User Stories parallelamente. Ogni membro del team, ogniqualvolta finisce uno step del processo di realizzazione di una User Story, sposta la scheda corrispondente nella colonna successiva della kanban, la quale proseguirà nel suo processo, mentre lo sviluppatore potrà passare all'implementazione di una nuova User Story.

Procedure Le procedure seguite nell'applicazione della metodologia Kanban si appoggiano ai software aziendali rilasciati da Atlassian, ovvero JIRA e STASH.
In particolare, l'implementazione di una singola User Story passa attraverso diversi passi definiti.
Dal lato organizzativo della stesura e realizzazione delle User Stories e della Kanban vera e propria, i passi da seguire sono stati i seguenti, applicati al framework JIRA:

1. il Responsabile di progetto scrive la User Story;
2. il Responsabile decide a chi assegnare l'implementazione della User Story;
3. lo sviluppatore designato prende in carico il ticket aperto, contrassegnandolo come "In Progress";
4. lo sviluppatore realizza la funzionalità descritta nella User Story della comanda;
5. lo sviluppatore notifica l'avvenuta implementazione della funzionalità contrassegnando il ticket come "Done";
6. il Responsabile viene notificato dell'avvenuta realizzazione e provvede all'inserimento della nuova funzionalità nel progetto.

A questa procedura si associa anche la gestione vera e propria dei sorgenti aziendali, memorizzati in repository acceduti tramite il framework STASH. L'implementazione di ogni feature prevede il passaggio per vari punti, necessari alla stabilità e all'organizzazione del sistema.
La procedura da seguire all'interno del framework STASH è la seguente:

1. prendere in carico una User Story dal backlog;
2. creare un nuovo branch dall'attuale branch *develop* del repository;
3. implementare la funzionalità presa in carico, ricordandosi di effettuare commit periodici con messaggi esplicativi del lavoro effettuato;
4. al termine dell'implementazione, assicurarsi che i test disegnati per le unità implementate passino, così da non introdurre errori di unità nel sistema;
5. una volta appurato che i test sono positivi, proseguire con la richiesta di pull nel branch *develop* da notificare al Responsabile del progetto.

A questo punto si può verificare un imprevisto, ovvero che nel tempo in cui viene implementata una User Story, il branch *develop* subisca delle modifiche. Questo fatto comporta una pull del *develop* nel branch attuale ed una conseguente risoluzione de conflitti che potrebbero venirsi a creare. Dopodiché si prosegue con la procedura:

6. esaminare la risposta del Responsabile; in caso di approvazione, la procedura termina;

7. in caso di mancata approvazione della pull request, esaminare i punti in cui vengono alzate obiezioni da parte del Responsabile apportare le dovute modifiche al codice;

8. ritornare al punto 4 e iterare fino ad avvenuta approvazione della pull request.

2.1.2 Test Driven Developement

Il modello di sviluppo Test Driven (guidato dai test) prevede che lo sviluppo di ogni unità software avvenga soltanto dopo aver scritto il corrispondente test sulla base delle specifiche dell'unità stessa.
Il dovere di ogni programmatore che segue questo modello è quello di scrivere il minimo quantitativo di codice necessario a far validare un test precedentemente scritto, minimizzando gli sprechi e provando la correttezza del prodotto, delegando il refactoring stilistico ad un momento successivo, in cui si ha già un codice testato e funzionante.
Il motto dello sviluppo Test Driven è appunto "Red, Green, Refactor"; la procedura da seguire prevista da questa metodologia è la seguente:

1. scrivere un "singolo" test di unità che descrive una funzionalità del programma;

2. eseguire il test, il quale dovrebbe fallire, dato che il programma non presenta ancora la feature descritta (da qui, la fase Rossa);

3. scrivere la quantità minima di codice necessaria far passare il test (fase Verde);

4. eseguire il Refactoring del codice, in modo da eliminare eventuali ridondanze e superficialità;

5. ripetere, accumulando test di unità con il passare dello sviluppo.

È stato scelto di utilizzare questo approccio in quanto presenta dei vantaggi significativi. Questi test permettono di individuare con precisione le specifiche del codice, e quindi il suo comportamento in base alle situazioni a cui sarà sottoposto. Ciò facilita la scrittura di un codice funzionante, più pulito, più affidabile e manutenibile.
La stesura di un test aiuta molto nella comprensione delle funzionalità di un modulo, evitando così errori concettuali che si potrebbero avere e priori nell'implementazione. La scrittura di un singolo test implica la piena comprensione dei metodi e delle funzionalità esposte, quindi permette di stabilire già prima della stesura di un modulo gli output attesi.
Il framework AngularJS è stato creato appositamente per rendere il testing più efficiente e semplice possibile, grazie a molte funzionalità focalizzate al testing già comprese all'interno del framework. Questa scelta è stata compiuta perché JavaScript ha una grandissima potenza espressiva, ma quasi nessun controllo da parte del compilatore, il che rende necessaria la costante presenza dei test nello sviluppo con questo linguaggio.

Test di Unità

La tipologia di test utilizzata maggiormente con questo approccio è l'insieme dei test di unità. Questa tipologia di test si prefissa di verificare il corretto funzionamento di una singola unità software. Per unità si intende normalmente il minimo componente di un programma dotato di funzionamento autonomo; a seconda del paradigma di programmazione o linguaggio di programmazione, questo può corrispondere per esempio a una singola funzione nella programmazione procedurale, o una singola classe o un

singolo metodo nella programmazione a oggetti.
Applicando questo concetto al progetto in analisi, i test di unità scritti verificano la correttezza di ogni funzione scritta nei vari componenti del pattern Model View Controller (MVC) di Angular.

Test di scenario

I test di scenario, conosciuti anche come E2E, sono una metodologia di test utilizzata per verificare il corretto flusso di un'applicazione, se questa si comporta come pianificato dall'inizio alla fine del flusso. L'obiettivo che sta alla base della creazione di test di scenario è di identificare le dipendenze di sistema e di assicurarsi che le informazioni (e il loro formato) siano passate correttamente tra le varie componenti del sistema.

2.2 Tecnologie utilizzate

2.2.1 Tecnologie di sviluppo

AngularJS

figura 2.1: Logo di AngularJS

AngularJS è un framework open source JavaScript mantenuto da Google, utilizzato prevalentemente nello sviluppo di Single Page Application (SPA). Attraverso questo framework è possibile aumentare le capacità del classico linguaggio HTML (o simili, quali Jade), per poter realizzare Web Application responsive e solide. L'obiettivo di questo framework, infatti, è quello di semplificare lo sviluppo ed il testing di applicazioni scritte in JavaScript, fornendo una base di librerie e direttive utili ad implementare un applicazione di architettura MVC.
Come altri framework utilizzati con lo stesso fine, una delle feature più utili ed interessanti di Angular è il cosiddetto "two-way data binding". Tramite l'utilizzo di direttive proprie del framework, uno sviluppatore può "legare" una variabile della vista ad una particolare entità di un modello, facendo sì che i dati visualizzati rappresentino costantemente il dato aggiornato; dualmente, tramite Angular, se si utilizza questo legame è possibile aggiornare i dati del modello direttamente dalla vista.
Angular inoltre fornisce degli strumenti per astrarre e realizzare la comunicazione con un back-end. Tramite Angular, infatti, è possibile creare una web app totalmente RESTful, tramite l'utilizzo delle **$resource** e dei servizi **$http** messi a disposizione dal framework.

Bootstrap

figura 2.2: Logo di Bootstrap

Bootstrap è una raccolta di strumenti liberi per la creazione di siti e applicazioni per il Web. Essa contiene modelli di progettazione basati su HTML e CSS, sia per la tipografia, che per le varie componenti dell'interfaccia, come moduli, bottoni e navigazione, e altri componenti dell'interfaccia, così come alcune estensioni opzionali di JavaScript. Includendo Bootstrap nella propria pagina web (sia da Content Delivery Network (CDN) che da codice sorgente), è possibile usufruire di tutta una serie di agevolazioni nello sviluppo di una pagina, dalla stilistica di base della pagina stessa, agli effetti di stransizione CSS avanzati, al layout stesso. Bootstrap includ una libreria CSS di base con una grande varietà di classi da applicare agli elementi di HTML statico, più un file *JavaScript* con le funzioni *jQuery* basilari di comportamento, entrambi facilmente estendibili dallo sviluppatore con file personalizzati. è una raccolta di strumenti liberi per la creazione di siti e applicazioni per il Web. Essa contiene modelli di progettazione basati su HTML e CSS, sia per la tipografia, che per le varie componenti dell'interfaccia, come moduli, bottoni e navigazione, e altri componenti dell'interfaccia, così come alcune estensioni opzionali di *JavaScript*. Includendo Bootstrap nella propria pagina web (sia da CDN che da codice sorgente), è possibile usufruire di tutta una serie di agevolazioni nello sviluppo di una pagina, dalla stilistica di base della pagina stessa, agli effetti di stransizione CSS avanzati, al layout stesso. Bootstrap includ una libreria CSS di base con una grande varietà di classi da applicare agli elementi di *html* statico, più un file *JavaScript* con le funzioni *jQuery* basilari di comportamento, entrambi facilmente estendibili dallo sviluppatore con file personalizzati.

2.2.2 Strumenti di supporto ai processi

Git

Git è un sistema software di controllo di versione distribuito. Le funzionalità principali di *Git* sono le seguenti:

* **distribuzione:** *Git* dà a ogni sviluppatore una copia locale dell'intera cronologia di sviluppo, e le modifiche vengono copiate da un tale repository a un altro. Queste modifiche vengono importate come diramazioni aggiuntive di sviluppo, e possono essere fuse allo stesso modo di una diramazione sviluppata localmente;
* **grande community:** considerata l'estesa community di sviluppatori che usa Git è possibile trovare diversi tool e plugin che spesso semplificano il lavoro, risparmiando così tempo e risorse;

figura 2.3: Logo di Git

* **sviluppo non lineare:** *Git*, attraverso il suo sistema di *branching* e *merging* consente lo sviluppo di codice parallelamente, su più linee che possono dividersi o unirsi.

JIRA

figura 2.4: Logo di JIRA

Jira è un prodotto proprietario finalizzato al tracciamento dei compiti, sviluppato da Atlassian dal 2002[3]. Esso fornisce il tracciamento di bug, compiti e funzionalità di project management. JIRA non è un acronimo, bensì il troncamento della parola giapponese *Gojira*.
JIRA è scritto in Java e si basa sul sistema di inversione di controllo sviluppato da Pico, su Apache OFBiz e lo stack tecnologico di WebWork 1. JIRA supporta diverse chiamate procedurali remote, quali REST, Simple Object Access Protocol (SOAP) e altre. JIRA si integra con i programmi di versionamento Clearcase, CVS, Git, Mercurial, Perforce, Subversion e Team Foundation Server.
Secondo i dati di Atlassian, JIRA viene usato nel tracciamento delle attività e nel project management da oltre 25,000 utenti in 122 paesi nel mondo. Alcune delle organizzazioni che usano JIRA sono Fedora Commons, Hibernate, Honeywell Aerospace, JBoss, Linden Lab, Skype, Spring Framework e The Apache Software Fundation.

Stash

Stash è un software aziendale di controllo di versionamento distribuito basato su Git. Esso offre un sistema di gestione dei repository a progetto, il che predispone un naturale accompagnamento con il sistema di project management JIRA.

[3] *About Atlassian – customers, life, community, FedEx days.* Atlassian, 2012.

figura 2.5: Logo di Stash

Branching L'implementazione di una nuova funzionalità, secondo le strategie aziendali, viene effettuata su un nuovo branch. Questa scelta permette lo sviluppo indipendente di più feature, partendo da un punto comune (solitamente il branch "develop" del repository) e diversificando in base ai task assegnati agli sviluppatori del team.

Pull request Ogniqualvolta una nuova funzionalità viene implementata, con il corrispondente completamento del ticket dell'attività di JIRA, viene creata una "pull request" per integrare il nuovo contenuto. Questa pull request viene effettuata in un branch più generale, solitamente nel branch "develop" se il prodotto è ancora in via di sviluppo, oppure direttamente nel "master" se si sta per rilasciare una nuova release del prodotto.
Ogni pull request non viene integrata finché non viene approvata dal responsabile, il quale visiona i cambiamenti introdotti nella nuova funzionalità; egli può in ogni caso segnalare eventuali cambiamenti indesiderati allo sviluppatore e richiedere una modifica del codice.

2.2.3 Ambiente di sviluppo

PhPStorm

figura 2.6: Logo di PhPStorm

PhpStorm è un Integrated Development Environment (IDE) commerciale sviluppato e distribuito da JetBrains e si basa su IntelliJ IDEA. PhpStorm fornisce un editor per PHP, HTML e JavaScript, aiutando lo sviluppatore con correzioni e segnalazioni in real time e molte feature di refactoring del codice.
PhpStorm è basato su IntelliJ IDEA, che è scritto in Java. Oltre alle funzioni core dell'IDE, l'utente può estendere le funzionalità installando i numerosi plugin disponibili o scrivendone alcuni propri.
Oltre ad offrire un editor responsivo ed efficiente, PhpStorm si presta molto bene per lo sviluppo di applicazioni in JavaScript, supportando i maggiori framework del linguaggio, come soprattutto AngularJS e NodeJS. Inoltre è possibile configurare l'IDE per l'esecuzione automatica di test scritti con Jasmine, mostrando direttamente i

risultati di tali esecuzioni all'interno dell'IDE stesso, senza bisogno di un cambio di contesto e strumenti, oltre alla classica funzionalità di debugging.

2.2.4 Tecnologie di Testing

Jasmine

figura 2.7: Logo di Jasmine

Jasmine è un framework di testing open source per il linguaggio JavaScript. Jasmine, all'interno di un progetto in AngularJS o più in generale, può essere utilizzato sia per i test di unità dei componenti, sia per i test e2e.
Jasmine si basa sulla sua facilità di comprensione dei test. Infatti, ogni test di Jasmine è diviso in blocchi di più suite. Ogni blocco riguarda una particolare componente del codice, ed ogni funzionalità testata viene correlata da una specifica descrizione del comportamento atteso.
Idealmente ogni statement di controllo di Jasmine dovrebbe testare non più di una funzione del codice, così da rendere i controlli maggiormente atomici. Per aiutare lo sviluppatore nella scrittura dei test, Jasmine presenta al suo interno una grande quantità di costrutti di confronto, che spaziano sui diversi tipi di dato che si possono incontrare, lasciando comunque allo sviluppatore il modo di creare confronti personalizzati.
Usato insieme ad AngularJS, Jasmine arricchisce ulteriormente le sue potenzialità, in quanto Angular comprende un modulo apposito scritto per il testing, ovvero *angular-mocks*. In questo modulo, da includere in Jasmine, troviamo i costruttori di ogni componente di Angular, con cui iniettare le varie componenti mockate quando ne abbiamo l'esigenza, come ad esempio nel test di un controller che dipende da uno o più servizi, più altre componenti utili allo sviluppatore.
Per quanto riguarda i test E2E, Jasmine contiene al suo interno vari strumenti di simulazione di input utente, quali ad esempio il click su un elemento o l'inserimento di testo in una casella. Attraverso l'uso di questi strumenti, è possibile simulare vari scenari di interazione con le varie funzionalità del sistema, andando quindi a testare il corretto funzionamento delle varie unità. Questo si ottiene sfruttando di browser driver come *Selenium*, i quali automatizzano l'interazione con i vari browser.

2.2.5 Strumenti di Automazione

Grunt

Grunt è un gestore ed un esecutore di attività JavaScript. Utilizzato soprattutto per l'automazione di attività come minificazione, compilazione, test di unità, di scenario, ecc., aiuta lo sviluppatore soprattutto nella creazione di applicazioni *client side*, aumentando l'efficienza nello sviluppo. A seconda delle esigenze e delle possibilità hardware, infatti, Grunt può essere configurato in modo che vengano eseguiti certi script ad ogni modifica di un file, o solo di alcuni. Si può quindi ricaricare un webserver alla modifica di un file html o eseguire un certo test di unità alla modifica di un file JavaScript, tutto assolutamente automatizzato.

figura 2.8: Logo di Grunt

Inoltre Grunt e la sua community sono in continuo sviluppo; ci sono molti sviluppatori che scrivono attorno a questa tecnologia, creando plugin da utilizzare per automatizzare anche il sistema di configurazione di Grunt, per adattarsi ai vari framework utilizzati durante lo sviluppo.

Karma

figura 2.9: Logo di Karma

Karma, precedentemente conosciuto come *Testacular*, è un ambiente creato per il lancio e la gestione dei test in ambiente JavaScript.
Karma supporta diversi framework di testing, inoltre è compatibile con i maggiori browser per assicurare che l'ambiente utilizzato per il testing sia funzionante in tutti i casi. L'unico compito che ha lo sviluppatore, per poter utilizzare Karma, è (oltre all'installazione del client e dei plugin necessari) creare e personalizzare il file di configurazione di Karma, la cui struttura viene creata in automatico con un comando del client da terminale. All'interno del file di configurazione, lo sviluppatore può configurare tutti gli aspetti dell'esecuzione dei test, dai browser utilizzati alla selezione di quali test eseguire, in una esecuzione singola o ad aggiornamento dei file. Oltre ad eseguire i test che vengono specificati nei path del file di configurazione, Karma offre anche altre funzionalità interessanti che si appoggiano sui test scritti. Una di queste funzionalità è la generazione automatica di un file di *html* statico in cui vengono mostrate le singole metriche di coverage di ogni file e in media per il progetto. Le metriche di copertura generate automaticamente con il plugin *coverage* di Karma sono:

* line coverage;
* statement coverage;
* branch coverage;
* function coverage.

Nella "vista" che viene creata dal plugin, è possibile visualizzare quali linee non sono eventualmente coperte dai test, quali branch e quali funzioni, su cui quindi andare ad intervenire o meno per aumentare la copertura, o per verificare che ogni funzionalità sia effettivamente implementata e testata.

NPM

figura 2.10: Logo di NPM

Acronimo di *Node Package Manager*. Come da definizione, npm è un gestore di pacchetti e moduli JavaScript, basato su Node.js.
Tramite npm è possibile cercare ed installare localmente ad un progetto o globalmente una grande serie di utilità pubblicate nel registro stesso di Node Package Manager (NPM). Oltre a questo, è possibile gestire ed aggiornare facilmente ogni pacchetto installato tramite semplici comandi da terminale. La gestione dei pacchetti di npm consente di suddividere i moduli in funzionalità, per garantire così il riutilizzo atomico e una maggiore manutenibilità dei singoli file, così come la condivisione con altri sviluppatori attraverso il registro NPM.

Protractor

figura 2.11: Logo di Protractor

Protractor è un framework utilizzato per realizzare test *e2e*, pensato soprattutto per la compatibilità con Angular.js. Protractor è un programma scritto in Node.js, mentre la sua base e l'interazione con i vari browser è mediata da WebDriverJS.
Questo framework permette di eseguire i comandi pensati e scritti nei test end-to-end. Ovvero, permette di simulare il comportamento di un utente con una certa Web Application, verificando che gli output attesi di una certa applicazione ad un certo comportamento utente, siano quelli previsti. Tramite l'utilizzo di Selenium WebDriverJS, questo procedimento di testing viene completamente automatizzato; si riescono a replicare le tipiche azioni di un utente e Protractor analizza ogni risposta dell'applicazione e la confronta con le attese.
Alla fine di ogni test suite, ci vengono presentati i risultati dell'esecuzione dei test e, se ci sono, quali statement hanno provocato un errore.

Capitolo 3

Descrizione dello stage

Breve introduzione al capitolo
Questo capitolo tratterà l'organizzazione temporale del progetto, ovvero la sua divisione in attività e l'estensione temporale di ognuna. Il progetto è stato diviso in quattro fasi distinte, con diverse attività.

3.1 Analisi preventiva dei rischi

Durante la fase di analisi iniziale sono stati individuati alcuni possibili rischi a cui si potrà andare incontro. Si è quindi proceduto a elaborare delle possibili soluzioni per far fronte a tali rischi.

1. Conflitti nell'implementazione parallela di User Stories
Descrizione: durante lo svolgimento di un progetto, ogni implementazione di una User Story deve trovarsi su un branch separato, derivato dal branch *develop*. Può quindi accadere che, in determinati casi, l'esecuzione di ogni ticket e il suo assorbimento nel sistema non siano sequenziali, in quanto il Responsabile di progetto può non essere sempre presente. Quindi, nel caso in cui si realizzino più feature senza avere una pull request alla fine di ognuna, il codice che poi dovrà essere inserito nel sistema darà quasi certamente conflitto nella fase di *merge*.
Soluzione: sollecitare il Responsabile di progetto all'analisi dei cambiamenti apportati ad ogni feature, notificandolo periodicamente sullo stato di esecuzione del ticket, in modo che il Responsabile sia pronto all'esaminazione della User Story implementata in modo più repentino.

2. Mancata competenza in strumenti e tecnologie aziendali
Descrizione: al momento del mio inserimento in azienda, la mia competenza sull'applicazione di metodologie *Agile* è stata pressoché nulla, così anche per quanto riguarda gli strumenti aziendali utilizzati in IKS e per determinate tecnologie da applicare durante il progetto. Questo fattore potrebbe portare ovviamente ad un rallentamento nel processo produttivo e, soprattutto, l'introduzione di errori.
Soluzione: la prima attività che verrà svolta appena dopo l'inizio dello stage sarà appunto la formazione su tali tematiche. In particolare, come da Piano di Progetto, sono state stimate 40 ore di formazione per colmare le lacune possedute.

3. Reperibilità del Responsabile di Progetto
Descrizione: data la varietà di progetti che il Responsabile di Progetto a me assegnato

deve seguire, può presentarsi la possibilità che questi non sia sempre reperibile in sede per un confronto diretto. Il rischio si applica anche a quelle giornate in cui il Responsabile è impegnato ad altri progetti aziendali per l'intera durata della giornata lavorativa..
Soluzione: la gestione delle attività tramite Kanban serve proprio a mitigare questo tipo di rischi, in quanto uno sviluppatore può implementare più User Stories senza dover consultare direttamente il Responsabile. Questo ovviamente se il Responsabile ha stilato un numero adeguato di User Stories da implementare. Inoltre, l'utilizzo degli strumenti JIRA e STASH consente a due o più collaboratori di discutere il codice prodotto direttamente dall'interfaccia web, quindi anche in maniera telematica, ignorando il confronto diretto in sede..

4. Cambiamento dei requisiti in corso d'opera
Descrizione: data anche la natura mutevole di un ciclo di sviluppo software come la metodologia Agile, è possibile che alcuni requisiti varino in corso d'opera, causando così una riorganizzazione di tempo e risorse non prevista.
Soluzione: la presenza di uno strumento aziendale che fornisce lo stesso servizio consente una precisa individuazione dei requisiti ad inizio opera. Nel caso in cui vengano individuati requisiti aggiuntivi, il loro impatto sarà minimo e controllabile, aggiungendo le user stories corrispondenti nel backlog.

3.2 Requisiti e obiettivi

L'analisi dei Requisiti è stata svolta grazie alla suddivisione i storie intrinseca nella metodologia *Agile*. Infatti, la stesura di una singola storia porta molte volte all'individuazione di uno o più requisiti, per la maggior parte funzionali.
La scelta delle storie da implementare è avvenuta tramite specificazione dei vari *epic* noti all'inizio della pianificazione.

3.3 Pianificazione

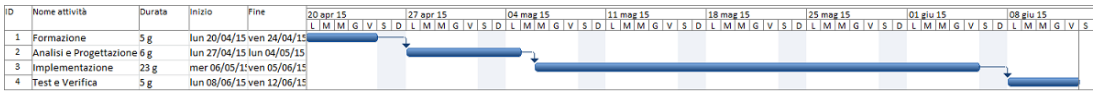


figura 3.1: Diagramma Gantt della pianificazione iniziale

3.3.1 Fase 1: Formazione (40 ore)

Scopo: formazione sulle problematiche e tecnologie che si incontreranno durante lo stage. In dettaglio:

- angularjs, rest angular, jasmine, istanbul;
- ambiente di sviluppo IKS (IDE per Javascript PhpStorm, Git);
- linee guida per lo sviluppo e concetti di programmazione sicura;
- verifica competenze acquisite.

Output (oggetto di verifica per il passaggio alla fase successiva):

- predisposizione ambiente di sviluppo;
- realizzazione di una web application di esempio che utilizzi le tecnologie e gli approcci indicati.

3.3.2 Fase 2: Analisi e Progettazione (40 ore)

Scopo: analisi delle specifiche funzionali, definizione del piano di test e progettazione tecnica della soluzione da realizzare.
In dettaglio:

- analisi specifiche funzionali;
- progettazione di dettaglio;
- documentazione.

Output (oggetto di verifica per il passaggio alla fase successiva):

- documento di Specifica Tecnica;
- User Stories.

3.3.3 Fase 3: Implementazione (180 ore)

Scopo: preparazione dell'ambiente di sviluppo e implementazione della soluzione.
In dettaglio:

- implementazione tecnologica dei requisiti definiti in fase di analisi;
- documentazione.

Milestone settimanali:

1. realizzazione delle funzionalità di dialogo REST con il Back-end;
2. creazione dell'interfaccia del profilo utente singolo;
3. visione della gerarchia aziendale e dell'organizzazione per progetti dell'organico;
4. gestione operazioni Create Read Update Delete (CRUD) sui vari progetti aziendali e sui membri partecipanti.

Output (oggetto di verifica per il passaggio alla fase successiva):

- prodotto finale completo in tutte le sue parti;
- sorgenti commentati e pubblicati nel repository dei sorgenti aziendale.

3.3.4 Fase 4: Test e Verifica (40 ore)

Scopo: Verifica funzionale, dei requisiti e stesura della documentazione.
In dettaglio:

- analisi dei risultati;
- verifica funzionale;

- verifica dei requisiti di progetto;
- revisione e correzione di eventuali bug;
- documentazione;
- verifica finale.

Output (oggetto di verifica per la conclusione del progetto):

- documento finale (conclusioni sull'attività di Stage svolta, con obiettivi raggiunti e conoscenze acquisite);
- eliminazione di bug riscontrati e pubblicazione delle modifiche nel repository dei sorgenti aziendale.

3.4 Progettazione Architetturale

3.4.1 Front-end REST

La struttura base dell'applicativo SkillMatrix è un Front-end REST. Un Front-end di quest tipo ha diversi vantaggi, come la modularità e portabilità che si può ottenere da client a server.
Con un tipo di architettura come questo, infatti, il client ed il server rimangono fortemente separati, limitando le chiamate a dei singoli punti di accesso del server. Questa modularità consente una maggiore manutenibilità di entrambe le parti, garantendo allo stesso modo delle interfacce comuni per i client che vogliono comunicare con il server REST.
Per creare un client REST, si ha bisogno di:

- un client HyperText Transfer Protocol (HTTP);
- le Uniform Resource Identifier (URI) delle risorse a cui vogliamo accedere;
- la capacità di interpretare il formato delle rappresentazione delle risorse.

In definitiva, il server mette a disposizione delle risorse e le operazioni con cui è possibile manipolare tali risorse. Solitamente, e per quanto riguarda SkillMatrix, le operazioni che vengono "pubblicizzate" da un server con questa architettura sono le classiche CRUD.
Trattandosi di un client che dialoga con il server attraverso protocollo HTTP, le richieste da utilizzare per gestire le risorse tramite operazioni CRUD saranno i verbi HTTP standard, ovvero:

- GET;
- POST;
- PUT;
- DELETE.

Rispettivamente, questi verbi consentono di visualizzare i dati della risorsa, di modificarli, di inserire una nuova risorsa o di cancellarla. Ovviamente, a seconda della risorsa e dei permessi del client, certe operazioni possono essere possibili o no (ad esempio, un

utente senza privilegi di amministrazione può non avere il permesso di eliminare una risorsa).
In questo tipo di architettura, il client deve utilizzare questi verbi HTTP per comunicare con il server. Le risposte del server sono riassunte in uno specifico codice di stato, il quale esprime se l'operazione è andata a buon fine, se ci sono stati degli errori di trasmissione o semplicemente trasmette ulteriori informazioni al client. Tali codici di errore sono fondamentali per il client, il quale deve modificare il suo stato interno in base al codice ritornato in risposta dal server. Per fare un esempio, il tipico comportamento di un client che ha richiesto la visualizzazione di una risorsa è, in caso di successo, l'effettivo caricamento delle informazioni della risorsa in una pagina; in caso di errore, ad esempio una risorsa non presente, il client deve sapersi adattare a tale errore e mostrare visivamente quale errore si è presentato, per informare l'utente.

3.4.2 Architettura Generale

3.4.3 AngularJS best practice

Capitolo 4

Analisi dei requisiti

Breve introduzione al capitolo

4.1 Casi d'uso

Per lo studio dei casi di utilizzo del prodotto sono stati creati dei diagrammi. I diagrammi dei casi d'uso (in inglese *Use Case Diagram*) sono diagrammi di tipo Unified Modeling Language (UML) dedicati alla descrizione delle funzioni o servizi offerti da un sistema, così come sono percepiti e utilizzati dagli attori che interagiscono col sistema stesso.
Essendo il progetto finalizzato alla creazione di un tool per l'automazione di un processo, le interazioni da parte dell'utilizzatore devono essere ovviamente ridotte allo stretto necessario. Per questo motivo i diagrammi d'uso risultano semplici e in numero ridotto.

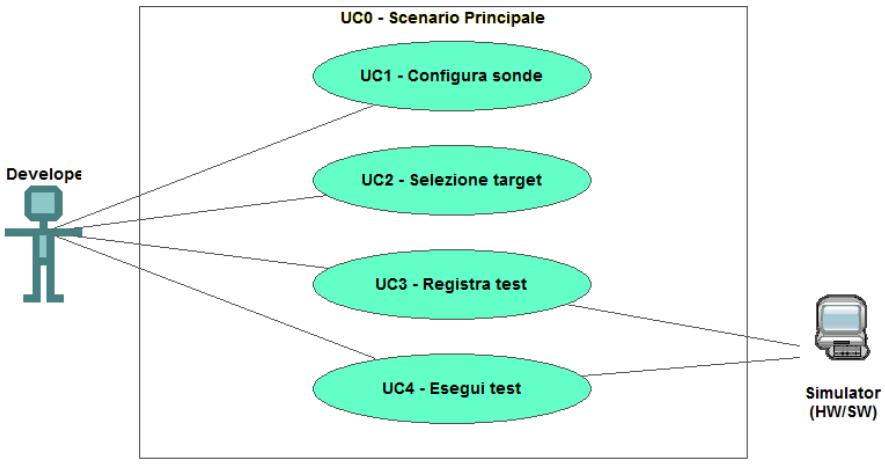


figura 4.1: Use Case - UC0: Scenario principale

UC0: Scenario principale

Attori Principali: Sviluppatore applicativi.

Precondizioni: Lo sviluppatore è entrato nel plug-in di simulazione all'interno dell'IDE.

Descrizione: La finestra di simulazione mette a disposizione i comandi per configurare, registrare o eseguire un test.

Postcondizioni: Il sistema è pronto per permettere una nuova interazione.

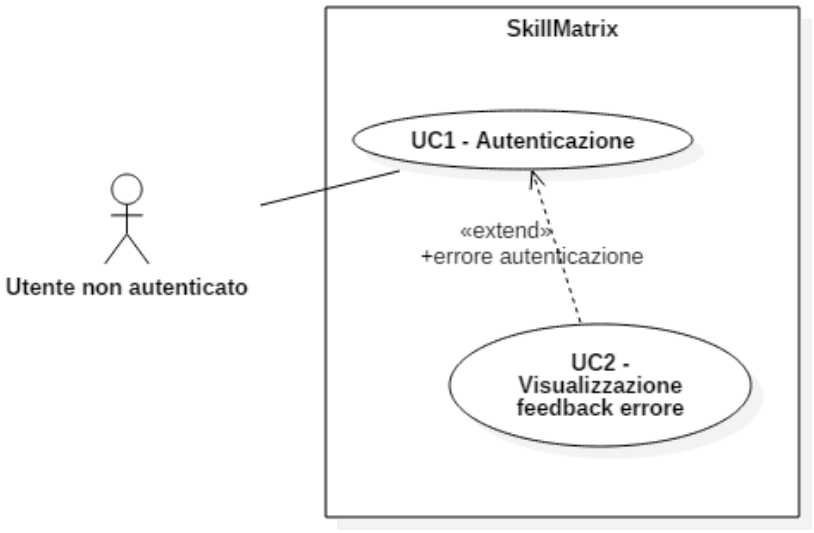


figura 4.2: Use Case - Login

UC1: Autenticazione

Attori Principali: Utente non autenticato.

Precondizioni: l'utente non ha effettuato il login in SkillMatrix e deve possedere credenziali valide.

Descrizione: l'utente fornisce dei dati validi di accesso nella pagina di Login e viene reindirizzato alla pagina di gestione del profilo.

Postcondizioni: le credenziali di accesso sono state verificate e l'utente è stato reindirizzato alla DASHBOARD.

UC2: Visualizzazione feedback errore

Attori Principali: Utente non autenticato.

Precondizioni: l'utente non ha effettuato il login a SkillMatrix.

Descrizione: l'utente visualizza un messaggio d'errore nel caso in cui avvenga una delle seguenti situazioni:

* le credenziali sono errate;
* la comunicazione con il server è fallita.

.

Postcondizioni: viene visualizzato un messaggio d'errore e il contatore che esprime la quantità di volte che l'errore si è manifestato, per poi far ritornare l'utente alla schermata di login.

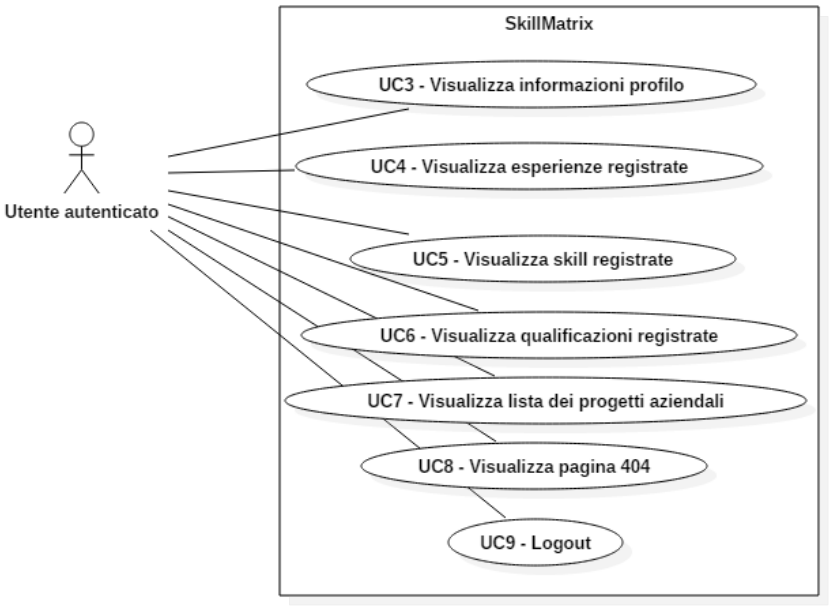


figura 4.3: Use Case - Dashboard

UC3: Visualizza informazioni profilo

Attori Principali: Utente autenticato.

Precondizioni: l'utente deve aver eseguito con successo il login al sistema.

Descrizione: l'utente seleziona il menù di visualizzazione delle informazioni del proprio profilo.

Postcondizioni: l'utente viene reindirizzato alla pagina di presentazione delle proprie informazioni.

UC4: Visualizza esperienze registrate

Attori Principali: Utente autenticato.

Precondizioni: l'utente deve aver eseguito con successo il login al sistema.

Descrizione: l'utente seleziona il menù di visualizzazione delle esperienze registrate nel sistema.

Postcondizioni: l'utente viene reindirizzato alla pagina di presentazione delle proprie esperienze professionali registrate.

UC5: Visualizza skill registrate

Attori Principali: Utente autenticato.

Precondizioni: l'utente deve aver eseguito con successo il login al sistema.

Descrizione: l'utente seleziona il menù di visualizzazione delle skill registrate nel sistema.

Postcondizioni: l'utente viene reindirizzato alla pagina di presentazione delle proprie skill registrate.

UC6: Visualizza qualificazioni registrate

Attori Principali: Utente autenticato.

Precondizioni: l'utente deve aver eseguito con successo il login al sistema.

Descrizione: l'utente seleziona il menù di visualizzazione delle qualificazioni registrate nel sistema.

Postcondizioni: l'utente viene reindirizzato alla pagina di presentazione delle proprie qualificazioni registrate.

UC7: Visualizza lista dei progetti aziendali

Attori Principali: Utente autenticato.

Precondizioni: l'utente deve aver eseguito con successo il login al sistema.

Descrizione: l'utente seleziona il menù di visualizzazione dei progetti aziendali.

Postcondizioni: l'utente viene reindirizzato alla pagina di visualizzazione della lista di progetti aziendali.

UC8: Visualizza pagina 404

Attori Principali: Utente autenticato.

Precondizioni: l'utente deve aver effettuato il login presso SkillMatrix ed avere una sessione attiva.

Descrizione: se l'utente, all'interno del dominio, effettua la ricerca di una risorsa non presente, il sistema informa l'utente di questa mancanza e fornisce istruzioni per la segnalazione.

Postcondizioni: l'utente visualizza la pagina di segnalazione di risorsa non presente e decide se segnalare la mancanza o tornare alla DASHBOARD.

UC9: Logout

Attori Principali: Utente autenticato.

Precondizioni: l'utente deve aver effettuato il login presso SkillMatrix ed avere una sessione attiva.

Descrizione: l'utente seleziona il pulsante di logout presente nella dashboard.

Postcondizioni: l'utente viene reindirizzato alla pagina di login e la sua sessiona viene cancellata.

4.2 Tracciamento dei requisiti

Da un'attenta analisi dei requisiti e degli use case effettuata sul progetto è stata stilata la tabella che traccia i requisiti in rapporto agli use case.
Sono stati individuati diversi tipi di requisiti e si è quindi fatto utilizzo di un codice identificativo per distinguerli.
Il codice dei requisiti è così strutturato R(F/Q/V)(N/D/O) dove:

R = requisito

F = funzionale

Q = qualitativo

V = di vincolo

N = obbligatorio (necessario)

D = desiderabile

O = opzionale

Nelle tabelle 4.1, 4.2 e 4.3 sono riassunti i requisiti e il loro tracciamento con gli use case delineati in fase di analisi.

tabella 4.1: Tabella del tracciamento dei requisti funzionali

Requisito	Descrizione	Use Case
RFN-1	L'applicazione deve reagire ad un'errata autenticazione tramite reindirizzamento all'interfaccia di login	UC2
RFN-2	L'applicazione deve reagire alla scadenza di una sessione tramite reindirizzamento all'interfaccia di login	UC2
RFN-3	L'applicazione non deve permettere l'accesso anonimo al sistema, tranne che per l'interfaccia di login	UC1
RFN-4	All'avvio, l'applicazione deve controllare se esiste una sessione attiva	-
RFN-5	In caso di sessione attiva, l'utente deve essere reindirizzato alla Dashboard se la sessione è valida	-
RFN-6	In caso di sessione non attiva o non valida, l'utente deve essere riportato alla schermata di login	UC2
RFN-7	L'applicazione deve poter consentire ad un utente registrato di effettuare l'autenticazione	UC1
RFN-8	L'utente autenticato deve poter visionare i propri dati dopo il login nel sistema	UC3
RFN-9	L'utente autenticato deve poter effettuare il logout dal sistema	UC9
RFN-10	L'utente deve poter visualizzare i propri titoli di studio e/o abilitazioni professionali una volta effettuato il login	UC6
RFN-11	L'utente deve poter inserire un nuovo titolo di studio o un'abilitazione professionale nel sistema	UCN
RFN-12	L'utente deve poter visualizzare le proprie skill una volta effettuato il login	UC5
RFN-12.1	L'utente deve poter filtrare le skill visualizzate per livello di competenza	UCN
RFN-12.2	L'utente deve poter inserire una nuova skill nel sistema	UCN
RFN-13	L'utente deve poter visualizzare i progetti ad esso associati una volta effettuato il login	UC7
RFN-13.1	L'utente deve poter visualizzare quale progetto necessita di registrazione nel sistema	UCN
RFN-14	L'utente deve poter visualizzare le informazioni dettagliate di un singolo progetto	UCN
RFN-15	L'utente deve poter visualizzare le proprie esperienze professionali una volta effettuato il login al sistema	UC4
RFN-16	L'utente deve poter inserire una nuova esperienza professionale nel sistema	UCN
RFD-1	Il sistema deve implementare una struttura gerarchica	UCN
RFD-1.1	L'interfaccia mostrata in base al tipo di ogni utente deve essere adattata al tipo dello stesso	UCN
RFD-1.1.1	L'amministratore di un progetto deve poter vedere le persone assegnate a quel progetto	UCN
RFD-2	L'utente deve ricevere un feedback nell'interfaccia ogni volta che avvenga un errore di comunicazione col server	UCN
RFO-1	Per tutti i dati presentati tramite lista va implementato l'infinite scroll	UCN
RFO-2	L'utente deve poter visualizzare la quantità di progetti che richiedono registrazione tramite notifica nella dashboard	UCN

tabella 4.2: Tabella del tracciamento dei requisiti qualitativi

Requisito	Descrizione	Use Case
RQD-1	Il codice deve seguire le linee stilistiche aziendali	-
RQD-2	Deve venire prodotta la documentazione tecnica di dettaglio	-
RQD-3	La grafica dell'interfaccia deve essere conforme agli standard aziendali	-
RQD-4	L'utente deve visualizzare una pagina di supporto se naviga verso una pagina non presente	UC8

tabella 4.3: Tabella del tracciamento dei requisiti di vincolo

Requisito	Descrizione	Use Case
RVN-1	Il framework di sviluppo per l'applicazione Front-end dev'essere AngularJS	Piano di Progetto
RVN-2	La libreria CSS da utilizzare dev'essere Bootstrap	Piano di Progetto
RVN-3	Il framework di testing da utilizzare deve essere Jasmine	Piano di Progetto
RVN-4	L'approccio di sviluppo deve essere Test Driven	Piano di Progetto
RVN-5	Devono essere realizzate una o più componenti che simulino il Back-end in assenza di un server funzionante	-
RVN-6	Le API definite devono rispettare il paradigma REST/HATEOAS	Piano di Progetto
RVN-6.1	L'interfaccia deve poter manipolare correttamente gli header HTTP	-
RVN-7	L'assegnamento dei ticket deve essere eseguito tramite il sistema kanban (JIRA)	-
RVN-8	Il repository da utilizzare deve essere la configurazione aziendale di Atlassian STASH	-
RVN-9	Lo schema dei dati associati ad un utente deve essere preso da schema.org	-
RVO-1	I dati tabellari devono essere richiesti al server in maniera paginata	-

Capitolo 5

Progettazione e codifica

Breve introduzione al capitolo

5.1 Architettura di AngularJS

AngularJS è stato il framework maggiormente utilizzato in questo stage e mi ha consentito di implementare l'intero progetto agilmente.
Alla base del framework, è collocato il design pattern MVC, leggermente modificato per adattarsi alle funzionalità di AngularJS. Il design pattern che ne risulta è qualcosa di più flessibile del classico MVC, consentendo agli sviluppatori una maggior libertà di utilizzo.
Ovviamente ci sono delle direttive e delle *best practice* consigliate, soprattutto se si intende creare un Front-end davvero REST-ful.

5.1.1 Model

In AngularJS, il "'*model*"' è realizzato con precise strutture dati. Innanzitutto, la logica di business dell'applicazione si colloca nei cosiddetti *servizi*. I servizi sono dei *singleton* istanziati con tecnica *lazy*, ovvero alla prima invocazione. In queste strutture è collocata tutta la logica che è indipendente dalla *view*, puntando al suo massimo riutilizzo.
Differentemente dai controller, le *Factory* dei servizi vengono richiamate dal sistema di *Dependendcy Injection*. Questo avviene passando come argomento al construttore di una struttura il nome del servizio da richiamare; sarà poi compito dell'*injector* di AngularJS richiamare la corretta *factory* del servizio richiesto.
I servizi vegono utilizzati inoltre (e soprattutto) per gestire i dati sensibili dell'utenza e dei vari oggetti presenti nel Back-end. Sono loro, infatti, gli esecutori materiali delle richieste REST al Back-end, grazie alle direttive apposite, come **$http** e **$resource**; i dati risultanti verranno poi forniti alle viste, se necessario, passando per gli adeguati controller.

5.1.2 View

Le viste realizzate con AngularJS sono essenzialmente pagine web con delle funzionalità di *markup* aggiuntive. All'interno di una pagina scritta in codice HTML, si possono

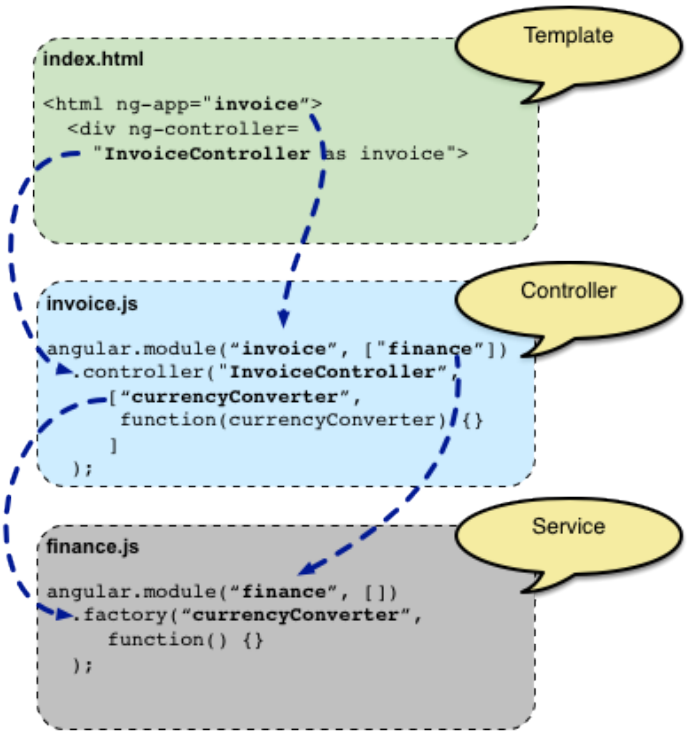


figura 5.1: Esempio di legame tra vista-controller-servizio

utilizzare delle direttive di AngularJS che verranno interpretate prima di renderizzare la pagina. Tramite queste direttive è possibile visualizzare dati del modello, modificarli in *real time* grazie al *Two-way data binding* richiamare funzioni di uno specifico controller e molto altro.
La classica dicitura di AngularJS per identificare un'espressione da interpretare si ottiene racchiudendo l'espressione tra due parentesi graffe:

```
{{ espressione }}
```

All'interno di questa dicitura, ogni espressione viene valutata e il suo risultato viene renderizzato nella pagina web. Se, ad esempio, nel *template* HTML si inserisce un'operazione matematica all'interno delle doppie graffe, una volta renderizzata la pagina il risultato dell'operazione (se possibile) verrà mostrato al posto dell'espressione. Oltre a questa funzionalità, AngularJS mette a disposizione una grande varietà di direttive per manipolare il Document Object Model (DOM), come iteratori per scorrere insiemi di dati o filtri da applicare ad una collezione per limitarne la visualizzazione.

5.1.3 Controller

I controller in AngularJS sono ciò che implementa la logica applicativa. Vengono legati al *template* di una pagina tramite la direttiva **ng-controller** posta all'interno di un elemento HTML. Così facendo, si invoca il costruttore del controller designato; lo *scope* del controller appena creato è l'elemento in cui è stato dichiarato e gli eventuali nodi figli dell'elemento stesso.
Il principale scopo dei controller è di esporre funzionalità alle espressioni ed alle direttive

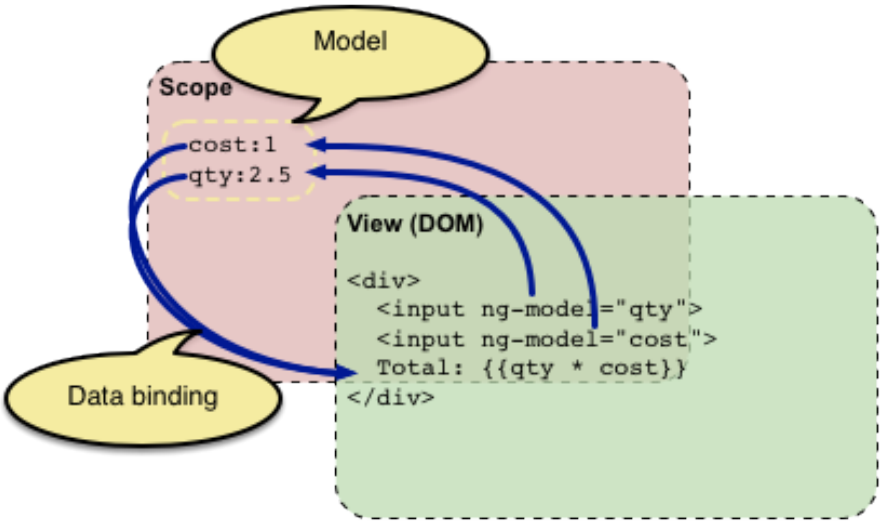


figura 5.2: Esempio di Template di una vista e Data Binding

utilizzate all'interno delle pagine web. Questo si ottiene aggiungendo metodi e proprietà all'oggetto **$scope** che verrà condiviso con la vista. Questo oggetto in particolare permette di realizzare il cosiddetto *view-model*, ovvero quella parte del *model* che verrà presentata alla *view*. Inoltre, tutte le proprietà associate al controller saranno rese disponibili al *template*, nello *scope* che compete al controller.

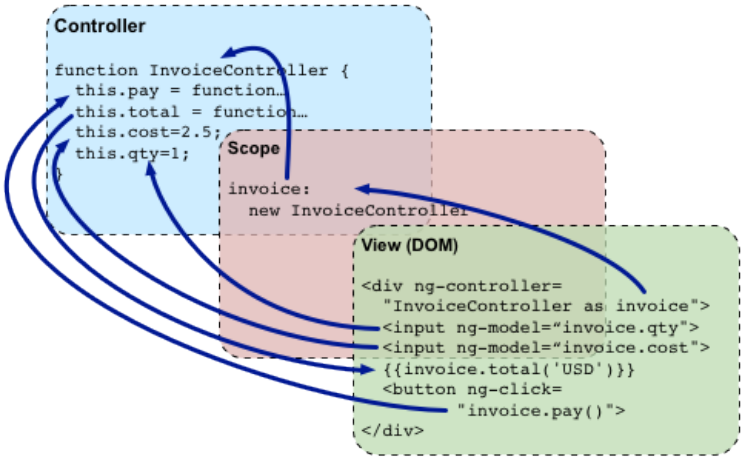


figura 5.3: Esempio di legame tra template e Controller

5.1.4 Two-way Data Binding

Una funzionalità importante che AngularJS espone è il cosiddetto *Two Way Data Binding*. Si parla di *legame doppio tra dati* quando una variabile del modello è legata ad un elemento che può cambiare ed al contempo mostrare il contenuto della variabile stessa. In una vista di AngularJS, ogniqualvolta un elemento che applica il *Two Way*

Data Binding viene modificato, il corrispondente campo nel modello viene notificato e aggiornato correttamente.
In AngularJS, si usa la direttiva **ng-model** per legare una variabile del modello ad un elemento HTML che può sia mostrare il suo valore, che modificarlo.

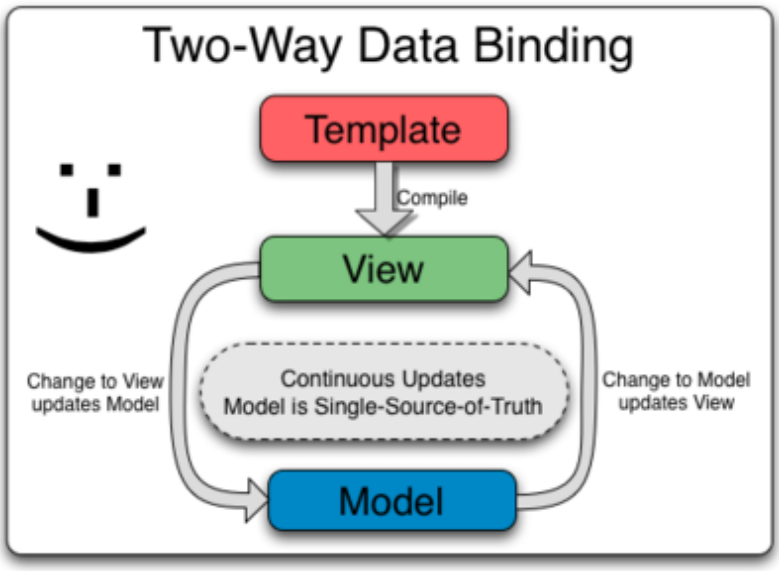


figura 5.4: Doppio legame tra vista e modello di AngularJS

5.1.5 Dependency Injection

Dependency Injection è un design pattern creato per gestire e risolvere le varie dipendenze tra più moduli software, e viene utilizzato all'interno di questo particale *framework* JavaScript.
La responsabilità di fornire le varie componenti richieste dalle dipendenze è demandata al sottosistema di gestione delle iniziezioni di AngularJS. Ogni volta che si dichiara una componente, viene registrata al modulo corrispondente; in questo modo, il sistema di gestione delle dipendenze può riconoscere la componente ed usarla dove è richiesto. Quando è richiesta una dipendenza, l'iniettore controlla il nome con cui è stata richiesta e la cerca nelle dipendenze registrate nel modulo, per iniettarla dove richiesto, se presente.

5.2 Definizione delle API REST

Questa sezione presenta la lista delle API concordate per le richieste REST al Back-end. Ogni definizione comprende l'URI di riferimento, il tipo di richiesta e i valori di ritorno possibili.
Nelle URI delle richieste, la dicitura **vx** sta ad indicare il numero di versione della richiesta, mentre l'espressione **:variabile** sta ad indicare una porzione variabile dell'URI, utilizzata da AngularJS.

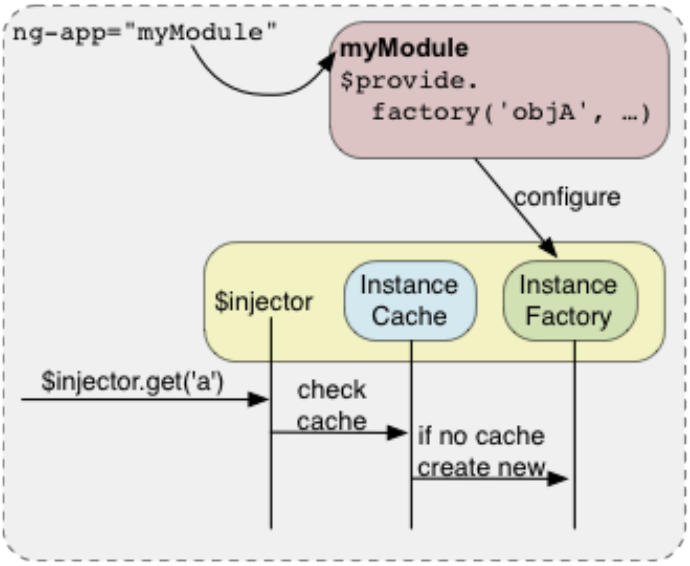


figura 5.5: Esempio di configurazione di una factory per l'iniettore

5.2.1 Informazioni utente

Lettura

URL /api/vx/users/:username

Metodo GET

Contenuto della richiesta nessuno

Risposte

200 la risposta contiene il JavaScript Object Notation (JSON) compatto con le proprietà utente;

404 se l'utente non è presente;

403 se l'utente non è autorizzato ad accedere all'informazione.

Invio di dati per l'autenticazione

URL /api/vx/login

Metodo POST

Contenuto della richiesta file JSON con la seguente formattazione:

```
{
  'username': 'some_username',
  'password': 'some_password'
}
```

Risposte

200 la risposta contiene il JSON compatto con le proprietà utente, più un *token* utilizzato per comporre l'*header* di autenticazione;

400 se gli *header* sono invalidi oppure se il contenuto della richiesta non è corretto;

403 se l'autenticazione non è andata a buon fine.

Creazione di un nuovo utente

URL /api/vx/users

Metodo PUT

Contenuto della richiesta file JSON compatto contenente username ed email

Risposte

201 la risposta contiene l'*id* della proprietà creata;

400 se gli *header* sono invalidi oppure se il contenuto della richiesta non è corretto;

403 se l'utente non è autorizzato alla creazione di un nuovo utente.

Modifica dei dati utente

URL /api/vx/users/:username

Metodo PATCH

Contenuto della richiesta file JSON contenente le modifiche da applicare all'utente designato

Risposte

204 corpo vuoto;

400 se gli *header* sono invalidi oppure se il contenuto della richiesta non è corretto;

403 se l'utente non è autorizzato a modificare l'entità.

5.2.2 Esperienze professionali

5.2.3 Titoli di studio ed abilitazioni professionali

5.2.4 Skill

5.2.5 Progetti associati

5.3 Stub Back-end

Il progetto di stage ha avuto come oggetto la realizzazione di un Front-end gestionale, senza però avere al contempo un Back-end funzionante. Per le verifiche funzionali,

quindi, ho avuto bisogno di realizzare un Back-end fittizio, il quale mi consentisse di effettuare chiamate alle API concordate con il Responsabile di Progetto per il Back-end vero e proprio.
Per realizzare questo Back-end sono ricorso a delle feature fornite dalle librerie di AngularJS adibite al testing di unità. Durante il testing delle funzionalità di una componente, le richieste ad un server funzionante vengono sostituite con una chiamata ad hoc che deve rispettare certe precondizioni e che restituisce dei dati prestabiliti. Il concetto del Back-end di Stub che ho utilizzato è lo stesso. In particolare, grazie all'utilizzo di espressioni regolari per filtrare le chiamate REST, sono stato in grado di intercettare tali chiamate e di farle gestire al mio Back-end ad hoc, con risposte prestabilite.
Tutto ciò è stato possibile gran parte grazie al servizio **$httpBackend** di AngularJS, il quale contiene metodi di intercettazione di chiamate REST e di gestione di richieste e risposte (e relativi *header*) HTTP. Il frammento sottostante è un esempio di una chiamata HTTP gestita dal Back-end di Stub:

```
$httpBackend.whenGET(/\/api\/\d.\d\/users\/\w*\/qualification/).respond(function(method, url, d
  console.log('Received these data:', method, url, data, headers);
  var getStatus,
   getData = {},
   getHeaders = headers;
  if(headers.Authorization == "Bearer XXX") {
    getStatus = 200;
    getData = {
      qualifications: [
        {
          type: 'Workshop',
          authority: 'W3C',
          address: 'Silicon Valley, 42',
          grade: 'Senior',
          earnDate: new Date(2015, 4, 20).toISOString(),
          vendor: 'RFC',
          product: 'HTML6',
          expireDate: new Date(2050, 4, 20).toISOString()
        },
        {
          type: 'Certificazione',
          authority: 'Mongo',
          address: 'MongoDB avenue, 8080',
          grade: 'Senior',
          earnDate: new Date(2014, 7, 8).toISOString()
        },
        {
          type: 'Corso',
          authority: 'AngularJS',
          address: '1600 Amphitheatre Parkway Mountain View, CA 94043',
          grade: 'Junior',
          earnDate: new Date(2015, 5, 20).toISOString()
        }
      ]
```

```
    };
  }
  else
    getStatus = 401;
  return [getStatus, getData, getHeaders];
});
```

5.4 Tecnologie e strumenti

Di seguito viene data una panoramica delle tecnologie e strumenti utilizzati.

Tecnologia 1

Descrizione Tecnologia 1.

Tecnologia 2

Descrizione Tecnologia 2

5.5 Ciclo di vita del software

5.6 Progettazione

Namespace 1

Descrizione namespace 1.

Classe 1: Descrizione classe 1

Classe 2: Descrizione classe 2

5.7 Design Pattern utilizzati

5.8 Codifica

Capitolo 6

Verifica e validazione

6.1 Test di unità

L'approccio Test Driven garantisce intrinsecamente che ogni unità software sia testata ancora prima della sua implementazione. Ogni unità dovrebbe contenere il quantitativo minimo di codice necessario a far rendere positivo il test ideato e scritto in precedenza. Questo approccio, sebbene all'inizio sia piuttosto insolito ed a prima vista tedioso, risulta invece essere imprescindibile nello sviluppo di applicazioni web, soprattutto utilizzando framework JavaScript. AngularJS stesso è stato pensato per scrivere applicazioni facilmente testabili, includendo al suo interno delle librerie create appositamente per il testing.
Mi è stato quindi possibile testare ogni componente della logica dell'applicazione, controllando il comportamento delle unità e dei loro metodi. Tramite l'utilizzo di sistemi di automazione, nel mio caso *Karma*, sono riuscito a rendere il processo di testing quanto più automatico possibile ed a calcolare con facilità le metriche base per verificare l'utilità dei test scritti.
Tramite il plugin di *coverage* di Karma, ho generato automaticamente tali cifre, suddividendole per:

* line coverage;
* statement coverage;
* branch coverage;
* function coverage.

Nella figura sottostante, presento i dati in forma tabellare, come vengono visualizzati dall'output del plugin *karma-coverage*, installabile facilmente da NPM.

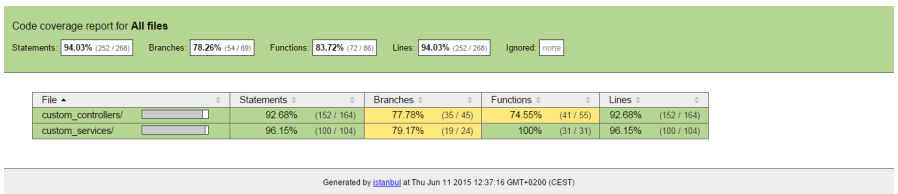


figura 6.1: Metriche di copertura generali nel progetto SkillMatrix

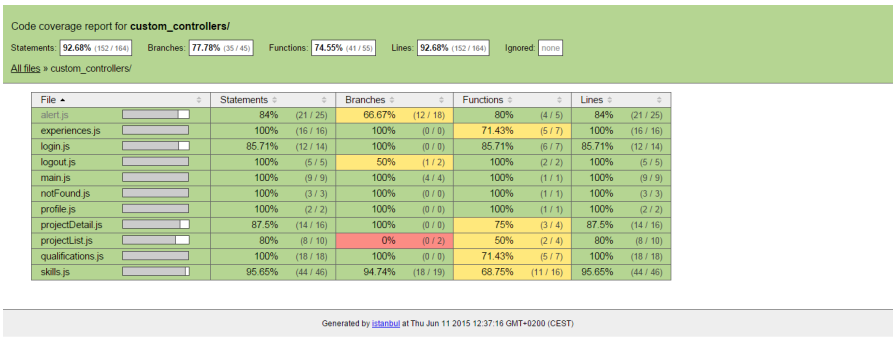


figura 6.2: Metriche di copertura dei controller nel progetto SkillMatrix

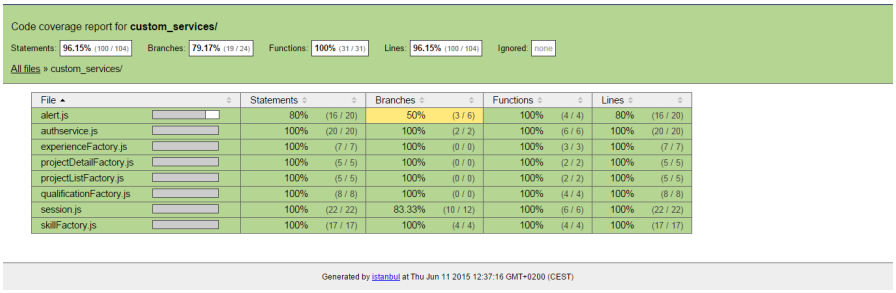


figura 6.3: Metriche di copertura dei servizi nel progetto SkillMatrix

6.2 Test End-To-End

I test di scenario (o E2E) servono a verificare la corretta interazione di uno o più unità software al fine di perseguire una certa azione dell'utente. AngularJS consiglia di utilizzare lo stack fornito da *Protractor* e da *Selenium* per automatizzare il processo di testing. Ogni test di scenario automatizza gli input utente in un browser ed analizza gli output ad ogni modifica.
Questi test sono molto più laboriosi di un test di unità eseguito con *Karma*, ma sono molto più vicini ad essere dei test di validazione.
In questo progetto ho creato un test per ogni tipo di azione utente, dalla semplice consultazione di informazioni del profilo all'inserimento di una nuova *skill* nel sistema. Ogni test avviene automatizzando l'input attraverso la selezione dei vari elementi HTML della pagina a cui fa riferimento il test e gli elementi vengono individuati tramite semplici selettori CSS o *jQuery*.

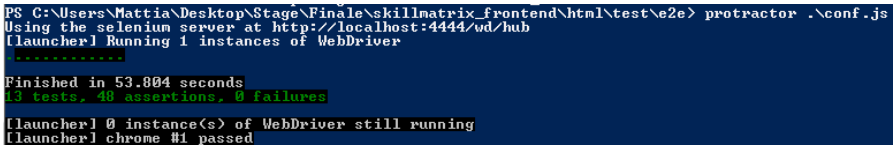


figura 6.4: Grafica da console dell'esecuzione di test e2e di Protractor

Ogni test è composto di passi ben precisi, ovvero le stesse azioni che compongono l'input utente. Il test quindi deve effettuare gli stessi passi, compresi i singoli click su

dei link e l'input di testo quando richiesto. La prossima porzione di codice è il test E2E del login, sia errato che corretto, con le proprie *expectations*, ovvero i valori previsti in output.

```
'use strict';

describe('SkillMatrix login', function() {

  beforeEach(function() {
    browser.get('base.html');
    browser.waitForAngular();
  });

  it('should have a title', function() {
    expect(browser.getTitle()).toEqual('SkillMatrix');
  });

  it('should appear an error message on wrong login', function() {
    element(by.model('credentials.name')).sendKeys('IKS');
    element(by.model('credentials.password')).sendKeys(2);

    element(by.css('button[type="submit"]')).click();

    expect(element(by.binding('alert.message')).getText()).
        toEqual('I dati di autenticazione sono errati(1)');
  });

  it('should login correctly with right credentials', function() {
    element(by.model('credentials.name')).sendKeys('IKS');
    element(by.model('credentials.password')).sendKeys('pippo');

    element(by.css('button[type="submit"]')).click();

    browser.getLocationAbsUrl().then(function(url) {
        expect(url.split('#')[1]).toBe('/profile');
      });
  });

});
```

Capitolo 7

Conclusioni

7.1 Consuntivo finale

7.2 Raggiungimento degli obiettivi

7.3 Conoscenze acquisite

7.4 Valutazione personale

Appendice A

Appendice A

Citazione

Autore della citazione

Glossario

API in informatica con il termine *Application Programming Interface API* (ing. interfaccia di programmazione di un'applicazione) si indica ogni insieme di procedure disponibili al programmatore, di solito raggruppate a formare un set di strumenti specifici per l'espletamento di un determinato compito all'interno di un certo programma. La finalità è ottenere un'astrazione, di solito tra l'hardware e il programmatore o tra software a basso e quello ad alto livello semplificando così il lavoro di programmazione.. 49

Back-end un'applicazione o un programma back-end serve indirettamente come supporto ai servizi del front-end, solitamente essendo vicino alla risorsa richiesta o avendo la capacità di comunicare con data risorsa. L'applicazione back-end può interagire direttamente con il front-end oppure, forse più tipicamente, è un programma chiamato da un programma intermediario che si inserisce tra le attività del front-end e del back-end.. v, 3, 19, 29, 31, 34, 36, 37, 47

CDN sistema di computer collegati in rete attraverso Internet, che collaborano in maniera trasparente, sotto forma di sistema distribuito, per distribuire contenuti (specialmente contenuti multimediali di grandi dimensioni in termini di banda) agli utenti finali ed erogare servizi e file di generi diversi.. 49

Compliance atto di essere in allineamento con linee guida, regolamentazioni e/o legislazioni.. 1, 47

CRUD con la sigla *CRUD* si rappresenta l'insieme delle quattro operazioni basilari su dei dati persistenti, ovvero creazione, lettura, modifica ed eliminazione.. 49

CSS linguaggio usato per definire la formattazione di documenti HTML, XHTML e XML e relative pagine web. Le regole per comporre il CSS sono contenute in un insieme di direttive (Recommendations) emanate a partire dal 1996 dal W3C. L'introduzione del CSS si è resa necessaria per separare i contenuti dalla formattazione e permettere una programmazione più chiara e facile da utilizzare, sia per gli autori delle pagine HTML che per gli utenti, garantendo contemporaneamente anche il riuso di codice ed una sua più facile manutenibilità.. 49

DOM letteralmente *modello a oggetti del documento*, è una forma di rappresentazione dei documenti strutturati come modello orientato agli oggetti.. 49

E2E acronimo di *End-To-End*, rappresenta quella categoria di test che si prefigge l'obiettivo di testare il comportamento di un utente su un sistema, usualmente automatizzando gli input utente previsti ed esaminando che i risultati attesi siano rispettati.. 49

Front-end nel campo della progettazione software il front end è la parte di un sistema software che gestisce l'interazione con l'utente o con sistemi esterni che producono dati di ingresso (es. interfaccia utente con un form).. v, 3, 4, 20, 29, 31, 36, 47

Governance L'IT Governance è responsabilità diretta del consiglio di amministrazione e del management esecutivo. È parte integrante della governance aziendale ed è costituita dalla direzione, dalla struttura organizzativa e dai processi in grado di assicurare che l'IT sostenga ed estenda gli obiettivi e le strategie dell'organizzazione.. 1, 2, 47

HTML linguaggio di markup solitamente usato per la formattazione e impaginazione di documenti ipertestuali disponibili nel World Wide Web sotto forma di pagine web.. 49

HTTP tradotto in *protocollo di trasferimento di un ipertesto*, è usato come principale protocollo per la trasmissione d'informazioni sul web.. 49

ICT acronimo di *Information and Communication Technology*, tradotto in Tecnologie dell'informazione e della comunicazione. Sono l'insieme dei metodi e delle tecnologie che realizzano i sistemi di trasmissione, ricezione ed elaborazione di informazioni.. 49

IDE si traduce in italiano come Ambiente di Sviluppo Integrato. Un IDE è un software che, in fase di programmazione, aiuta i programmatori nello sviluppo del codice sorgente di un programma. Spesso l'IDE aiuta lo sviluppatore segnalando errori di sintassi del codice direttamente in fase di scrittura, oltre a tutta una serie di strumenti e funzionalità di supporto alla fase di sviluppo e debugging.. 49

JSON formato adatto all'interscambio di dati fra applicazioni client-server.. 49

MVC pattern architetturale software utilizzato nell'implementazione di interfacce utente. Divide l'applicazione software in tre parti interconnesse, così da separare la rappresentazione interna delle informazioni dal modo in cui tali informazioni sono presentate all'utente.. 49

NPM package manager di JavaScript, usato di default da Node.js. Una volta installato Node.js, è possibile utilizzare anche NPM, dato che viene distribuito assieme a suddetto framework. Al suo interno si trovano tutti i maggiori framework ed utilities che si basano sul linguaggio JavaScript.. 49

REST REST si riferisce ad un insieme di principi di architetture di rete, i quali delineano come le risorse sono definite e indirizzate. Il termine è spesso usato nel senso di descrivere ogni semplice interfaccia che trasmette dati su HTTP senza un livello opzionale.. 49

SOAP acronimo di *Simple Object Access Protocol*, è un protocollo leggero per lo scambio di messaggi tra componenti software, tipicamente nella forma di componentistica software. La parola object manifesta che l'uso del protocollo dovrebbe effettuarsi secondo il paradigma della programmazione orientata agli oggetti.. 49

SPA una Single Page (Web) Application è un'applicazione web o semplicemente un sito che si propone di realizzare un comportamento più espressivo rispetto ad un'applicazione desktop. In una *SPA*, il codice necessario viene caricato al caricamento della pagina e tutti i contenuti vengono visualizzati dinamicamente, mentre tutte le comunicazioni con il server vengono nascoste all'utente.. 49

Spring framework open source per lo sviluppo di applicazioni su piattaforma Java. Spring fornisce un modello comprensivo di programmazione e configurazione per applicazioni *enterprise* basate sulla piattaforma Java.. 3, 47

Stub porzione di codice utilizzata in sostituzione di altre funzionalità software. Uno stub può simulare il comportamento di codice esistente e temporaneo sostituto di codice ancora da sviluppare. Gli stub sono perciò molto utili durante il porting di software, l'elaborazione distribuita e in generale durante lo sviluppo di software e il software testing.. 37, 47

UML in ingegneria del software *UML, Unified Modeling Language* (ing. linguaggio di modellazione unificato) è un linguaggio di modellazione e specifica basato sul paradigma object-oriented. L'*UML* svolge un'importantissima funzione di "lingua franca" nella comunità della progettazione e programmazione a oggetti. Gran parte della letteratura di settore usa tale linguaggio per descrivere soluzioni analitiche e progettuali in modo sintetico e comprensibile a un vasto pubblico.. 49

URI stringa che identifica univocamente una risorsa generica che può essere un indirizzo Web, un documento, un'immagine, un file, un servizio, un indirizzo di posta elettronica, ecc.. 49

Acronimi e abbreviazioni

API Application Program Interface. v, 3, 34, 37

CDN Content Delivery Network. 10

CRUD Create Read Update Delete. 19, 20

CSS Cascading Style Sheet. 3, 10, 29, 40

DOM Document Object Model. 32

E2E End-To-End. 3, 9, 13, 40, 41

HTML HyperText Markup Language. 3, 4, 9, 10, 31, 32, 34, 40

HTTP HyperText Transfer Protocol. 20, 21, 37

ICT Information and Communication Technology. 1

IDE Integrated Development Environment. 12, 13, 18

JSON JavaScript Object Notation. 35, 36

MVC Model View Controller. 9, 31

NPM Node Package Manager. 15, 39

REST Representational State Transfer. 3, 11, 19, 20, 31, 34, 37

SOAP Simple Object Access Protocol. 11

SPA Single Page Application. 9

UML Unified Modeling Language. 23

URI Uniform Resource Identifier. 20, 34

Bibliografia

Riferimenti bibliografici

James P. Womack, Daniel T. Jones. *Lean Thinking, Second Editon*. Simon & Schuster, Inc., 2010 (cit. a p. 1).

Siti Web consultati

Manifesto Agile. URL: http://agilemanifesto.org/iso/it/ (cit. alle pp. 1, 5).