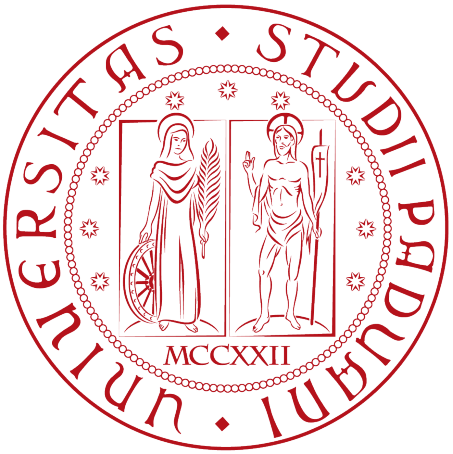
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



Revisione dell'interfaccia grafica di un software CRM con l'uso di AngularJS

Tesi di laurea triennale

Relatore

Prof.Paolo Baldan

Laureando

Davide Stocco

Anno Accademico 2017-2018

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 Pinco Pallino presso l'azienda Azienda S.p.A. Gli obbiettivi da raggiungere erano molteplici.
In primo luogo era richiesto lo sviluppo di ... In secondo luogo era richiesta l'implementazione di un ... Tale framework permette di registrare gli eventi di un controllore programmabile, quali segnali applicati Terzo ed ultimo obbiettivo era l'integrazione ...

Organizzazione del testo

Il primo capitolo descrive l'azienda ospitante ed il contesto in cui si è svolto lo stage.

Il secondo capitolo enuncia e tecnologie e gli strumenti con cui ho dovuto relazionarmi durante lo sviluppo del progetto;

Il terzo capitolo approfondisce il dominio applicativo ed i casi d'uso del progetto.

Il quarto capitolo approfondisce l'architettura del software JGalileo CRM in generale, ed in particolare la portlet relativa alla form.

Il quinto capitolo descrive la logica adottata per la realizzazione del prototipo oggetto dell'esperienza di stage ed i risultati ottenuti

Nel capitolo conclusivo vengono tratte le conclusioni, sia oggettive che personali, dell'esperienza di stage.

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*.

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

— Confucius

Ringraziamenti

Innanzitutto, vorrei esprimere la mia gratitudine al Prof. NomeDelProfessore, 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, Settembre 2018 Davide Stocco

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Introduzione al contesto applicativo.

Esempio di utilizzo di un termine nel glossario
Application Program Interface (API).

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

Figura 1.1: Logo di Sanmarco Informatica

1.1 L'azienda

Sanmarco Informatica Spa è un'azienda italiana leader nella progettazione e realizzazione di soluzioni a supporto della riorganizzazione di tutti i processi aziendali e professionali.
Ha sede a Grisignano di Zocco (VI), è attiva da 35 anni, può contare su più di 400 dipendenti e collaboratori, per seguire quotidianamente più di 1000 aziende clienti.
Il punto di forza dell'azienda sta nella formazione del personale e nella ricerca, campi nei quali viene investito in media il 20% del fatturato annuo.
Il prodotto di punta di Sanmarco Informatica è *JGalileo*, un software gestionale *ERP* utilizzato, oltre che da aziende europee, anche in USA, Russia e Cina.
Un software *ERP, Enterprise Resource Planning*, pianificazione delle risorse d'impresa, è un sistema di gestione che integra tutti i processi di business rilevanti di un'azienda (vendite, acquisti, gestione magazzino, contabilità ecc.).
Esso si pone l'obiettivo di rendere fluido il passaggio di informazioni tra le varie divisioni di un'azienda cliente, ad esempio allo scopo di ottimizzare la produzione in base alla richiesta, senza produrre e poi stockare in magazzino.
Oltre a JGalileo, Sanmarco si occupa di svariati prodotti software, tutti potenzialmente interconnessi e che occupano variegati settori di mercato: dal commercio B2B e B2C, alla Business Intelligence al CRM, quest'ultimo oggetto del mio progetto di stage.

1.1.1 Sede di lavoro

Sanmarco Informatica, in Italia, può contare su 5 sedi: la sede principale ed il centro Ricerca e Sviluppo si trovano a Grisignano di Zocco (VI), ci sono poi una filiale a Vimercate (MB), una a Tavagnacco (UD), ed una a Reggio Emilia.
La sede del mio stage è stato il centro di Ricerca e Sviluppo.
Qui viene adottato il metodo di lavoro Agile: esso nasce nel 2001 e definisce i dodici principi fondamentali per sviluppare con tempistiche e procedure ottimizzate un software che sia per il cliente una soluzione di successo. La metodologia "Agile" privilegia

* *gli individui e le interazioni più che i processi e gli strumenti*;
* *il software funzionante più che la documentazione esaustiva*;

- *la collaborazione col cliente più che la negoziazione dei contratti*;
- *rispondere al cambiamento più che seguire un piano.*

L'obiettivo è quello di garantire all'azienda tutta la flessibilità e la versatilità necessaria alla realizzazione di progetti efficaci. Questo metodo di lavoro si concentra su una collaborazione più face-to-face fra colleghi con cadenza regolare e ravvicinata, permettendo di mantenere e perfezionare man mano il focus sui progetti, valorizzando i nuovi punti di vista e le nuove risposte alle problematiche da affrontare.
In particolare, il metodo di gestione progetti "SCRUM" prevede di dividere il progetto in blocchi (Sprint), all'interno dei quali vengono sviluppate delle funzioni/moduli/applicazioni complete (denominate storie) pronte per la potenziale installazione al cliente. Il termine Scrum è mutuato dal termine del Rugby che indica il pacchetto di mischia ed è una metafora del team di sviluppo che deve lavorare insieme in modo che tutti gli attori del progetto spingano nella stessa direzione, agendo come un'unica entità coordinata.

1.2 L'offerta di Stage

Da anni l'azienda collabora con l'università di Padova alla ricerca di neolaureati da formare ed inserire nel proprio organico e, grazie ad iniziative come StageIt le possibilità di incontro tra studenti ed aziende sono molto più concrete e produttive. StageIT è un'iniziativa che mira ad agevolare l'incontro tra le imprese e gli studenti che entreranno a breve in stage nel mondo del lavoro con specifico riferimento al settore ICT, favorendo un'occasione di conoscenza reciproca mediante colloqui individuali. Gli studenti hanno

Figura 1.2: Logo di StageIT

infatti l' occasione di conoscere svariate realtà, anche piccole, che operano nel settore nei più disparati rami applicativi mentre per le aziende, oltre ad essere un'importante vetrina ed un momento di confronto con potenziali partner e concorrenti, costituisce anche la possibilità di conoscere tanti giovani studenti in un breve arco temporale.

1.2.1 Il progetto

Tra le varie proposte di stage che quest'azienda offriva, mi è stato proposto di collaborare a quello riguardante il software CRM da loro sviluppato.
Un software CRM, acronimo di Costumer Relationship Management, è un prodotto che permette all'utilizzatore di mantenere una rete di relazioni con clienti e potenziali

clienti, non soltanto al fine di stabilire una relazione commerciale, ma di fiducia reciproca che si possa protrarre nel tempo, anche attraverso l'impiego di strumenti di fidelizzazione quali newsletters, campagne ed eventi.
In questo contesto si inserisce lo stage cui ho preso parte e che aveva come obiettivo la ridefinizione della form principale di tale software.
Una form è l'interfaccia di un'applicazione che consente ad un utente di compilare dati e di inviarli ad un server, tipicamente per l'immissione degli stessi nel sistema.
Questa form, dapprima realizzata utilizzando il linguaggio *Java* e successivamente tradotta in *JavaScript* grazie all'utilizzo della libreria *GWT*, è ora richiesta in *JavaScript* nativo. Una fase del progetto prevede infatti la ricerca e l'analisi di framework e librerie JavaScript *open-source* disponibili sul mercato[1], per poi utilizzare la libreria scelta allo scopo di sviluppare un prototipo di form che abbia le stesse funzionalità di quella attualmente in uso, e possibilmente vada ad aggiungere nuove funzionalità come ad esempio la possibilità per un utente di inserire, in completa autonomia, nuovi campi dati personalizzati all'interno delle form.

Le principali difficoltà nell'affrontare questo progetto sono legate principalmente al dover studiare e capire un software molto grande e già in uso da molti anni e che implementa diverse tecnologie che non avevo mai affrontato durante il corso di studi, come Liferay, GWT ed Hibernate.
Oltre a ciò conoscere ed imparare ad utilizzare abbastanza velocemente una libreria JavaScript può portare a perdere molto tempo in prove ed esercitazioni prima di poter maneggiare con sufficiente sicurezza gli strumenti messi a disponizione da essa.
In terzo luogo è necessario che il codice da me prodotto non influenzi le altre funzionalità presenti all'interno del software: questo da un lato pone delle semplificazioni nello sviluppo, come ad esempio il riutilizzo di servizi REST oppure dei parametri che vengono passati in fase di chiamata al costruttore, dall'altro limita però la creatività e costringe l'adozione di un flusso di creazione e gestione abbastanza similare a quello già in uso.

La principale soluzione alle difficoltà appena descritte consiste nel dedicare buona parte dell'esperienza allo studio, in particolare allo studio del software e della form già esistente al fine di capire al meglio le modalità di interazione tra quest'ultima e il resto del sistema, in special modo riguardo i dati che vengono trasmessi ed inviati. In questo modo diventa semplice capire di che risorse si dispone e di come ottimizzarne l'utilizzo.

1.2.2 Pianificazione del lavoro

In accordo con l'azienda, è stato redatto un piano di lavoro a granularità settimanale. Il piano, mostrato in tabella, descrive per ogni settimana di stage il lavoro di approfondimento e sviluppo necessari al raggiungimento degli obiettivi.

Settimana 1:
La prima settimana è stata volta all'introduzione nell'azienda, al concetto di CRM, all'installazione dell'ambiente di sviluppo ed ad una prima fase di formazione sul funzionamento del software JGalileo CRM.
Settimane 2 e 3:
Nella seconda settimane di stage ho approfondito la conoscenza del software JGalileo

[1] **queste ed altre tecnologie verranno discusse nel dettaglio nei capitoli seguenti**.

CRM e attuato l'analisi dei requisiti tecnici ed applicativi, che si è protratta fino alla fine della terza settimana.
Settimane 4:
La quarta settimana è stata dedicata all'analisi di varie alternative open-source per il successivo sviluppo del prototipo.
Settimane 5-8:
La seconda metà dello stage è stata interamente dedicata allo sviluppo del prototipo, utilizzando la tecnologia scelta durante il precedente periodo.

Tabella 1.1: Pianificazione del lavoro

Periodo	Dal	Al	Descrizione
Prima settimana	21-05	25-05	Ricerca, studio e documentazione per inquadramento del progetto. Introduzione al concetto di CRM ed al software JGalileo CRM
Seconda settimana	28-05	01-06	Analisi dei requisiti applicativi e tecnici
Terza settiamana	04-06	08-06	Analisi dei requisiti applicativi e tecnici
Quarta settiamana	11-06	15-06	Ricerca degli ambienti di sviluppo open source soddisfacenti i requisiti richiesti e scelta dell'ambiente di sviluppo
Quinta settimana	18-06	22-06	Sviluppo prototipo
Sesta settimana	25-06	29-06	Sviluppo prototipo
Settima settimana	02-07	06-07	Sviluppo prototipo
Ottava settimana	09-07	13-07	Sviluppo prototipo

1.3 Struttura del testo

I capitoli seguenti descriveranno in maniera approfondita strumenti e metodologie con le quali è stato affrontato il periodo di stage.
In particolare, il **secondo capitolo** enuncia le tecnologie con le quali ho dovuto interfacciarmi durante lo studio del progetto o durante la fase di sviluppo.
Il **terzo capitolo** elenca i requisiti ed i casi d'uso del progetto.
Il **quarto capitolo** descrive lo stato dell'applicazione esistente, con particolare riguardo alla parte di mio interesse.
Il **quinto capitolo** illustra lo sviluppo del protoptipo oggetto del mio progetto, iniziando dallo studio delle tecnologie utilizzate.
Nel **sesto capitolo** sono infine tratte le conclusioni sull'esperienza, dagli obiettivi formativi e produttivi a quelli personali.

Capitolo 2

Tecnologie e strumenti

In questo capitolo verranno descritte le principali tecnologie che ho approfondito durante l'esperienza di stage.

2.1 Tecnologie utilizzate

Durante la mia esperienza di stage ho potuto prendere contatto con molte tecnologie differenti e che non avevo incontrato durante il corso di studi. Esse spaziano dall'ambiente server a quello client. L'approfondimento di tali tecnologie, specialmente nelle prime settimane di stage, è stato fondamentale per capire il funzionamento del software da modificare

2.1.1 Java

Figura 2.1: Logo di Java

Java è uno dei linguaggi di programmazione general-purpose più popolari al mondo. Rilasciato nel 1995, riprende molta della sintassi dal linguaggio C e C++, ma spostando parte delle responsabilità prima lasciate ai programmatori, come la gestione della memoria, alla *Java Virtual Machine (JVM)*: una macchina virtuale su chi viene eseguito tutto il codice Java.
La presenza di una macchina virtuale crea un livello aggiuntivo tra il sistema operativo e l' *IDE* di sviluppo: Questo permette al linguaggio Java di aderire al principio *Write*

Once Run Anywhere (WORA): Il codice compilato (bytecode) può essere eseguito da qualsisasi computer provvisto di una JVM, senza necessitare di ricompilazioni e perfino adattamenti del codice sorgente in base al sistema operativo presente sul computer. Java è inoltre molto utilizzato per eseguire applicazioni client-server, cioè applicazioni (Client) che per ottentere i dati necessari si appoggiano ad un fornitore (Server), che recupera per loro i dati necessari, in maniera del tutto trasparente al client e quindi all'utente.
Java dispone di una libreria proprietaria specializzata, ma in JGalileoCRM tale libreria è sostituita da *Liferay*.

2.1.2 Liferay

Figura 2.2: Logo di Liferay

Liferay è una tecnologia *enterprise portal* open source, realizzato in Java.
Per enterprise portal si intende un sistema informatico evoluto, in grado di integrare informazioni, processi e persone allo scopo di fornire valore aggiunto in termini di:

* Gestione del *Single Sign On*;
* Semplice personalizzazioni ad hoc per ogni cliente;
* Analisi delle pagine, in termini di accessi, click, download molto semplice;
* Integrazione tra funzionalità e dati di diversi sistemi in nuove componenti definite portlet.

Le portlets sono il cuore del sistema di Liferay. Esse infatti permettono di concentrare lo sviluppo solamente sulla gestione della funzionalità principale, lasciando a liferay la gestione degli accessi, dei menù di navigazione e degli altri componenti globali dell'applicazione.

2.1.3 GWT

GWT, acronimo di *Google Web Toolkit* è un insieme di tool open source che permette la creazione ed il mantenimento di complesse applicazioni front-end JavaScript scritte in Java.
GWT infatti si occupa di tradurre il codice scritto in Java in codice JavaScript, interpretabile da tutti i moderni browsers Tutto il codice Java può essere compilato ed eseguito grazie ai file Ant inclusi. Ant è un progetto open source della Apache foundation, volto ad automatizzare il processo di build. è simile al comando make di

Figura 2.3: Logo di Google Web Toolkit

Unix, ma scritto in Java.
Il plug-in di Google per Eclipse (IDE in uso presso Sanmarco Informatica) è molto comleto, offrendo la possibilità di creare progetti, farne il debug, il testing, meccanismi di validazione e di controllo della sintassi.

2.1.4 Apache Tomcat

Figura 2.4: Logo di Apache Tomcat

Apache Tomcat è un web server, cioè un' applicazione web che, in esecuzione su di un server, gestisce le richieste di trasferimento tra le varie pagine web di un client.
In particolare, Tomcat opera utilizzando servlet, cioè oggetti scritti in Java molto usati nella generazione di pagine web dinamiche. Queste servlet, in JGalileoCRM non vengono direttamente scritte in Java, ma sono scritte con *JavaServer Pages (JSP)*:
JSP è una tecnologia di programmazione web, scritta in Java, che si basa sull'uso di speciali tag all'interno di una pagina HTML, con cui possono essere chiamate funzioni scritte in linguaggio Java o JavaScript. A runtime, le pagine JSP vengono tradotte automaticamente in servlet utilizzabili da Tomcat.
Il vantaggio principale di questa tecnologia rispetto, ad esempio, a PHP, consiste nel poter scrivere tutto il codice, frontend e backend in un solo linguaggio di programmazione: Java.
Inoltre permette la creazioni di applicazioni web dinamiche, rispetto all'utilizzo di codice HTML statico.

2.1.5 Databases

Un database, o base di dati, è un insieme di dati omogeneo memorizzato in un elaboratore, che può essere interrogato attraverso uno dei linguaggi di interrogazione esistenti, con lo scopo di ottenere una parte dei dati memorizzati. JGalileoCRM utilizza due database differenti per gestire i propri dati:

2.1.5.1 Database SQL

Il primo database utilizzato dal software sopracitato è un comune database SQL: esso è un database relazionale, cioè opera creando tabelle che vengono messe in relazione tra loro tramite gli attributi inseriti nellle tabelle stesse.
Questo database è utilizzato per la gestione di tutti i dati diversi dai dati personali dei clienti e dei contatti.

2.1.5.2 Database AS400

AS/400 (Application System 400)è un è un minicomputer sviluppato a partire dal 1988 dall'IBM per usi prevalentemente aziendali, come supporto del sistema informativo gestionale.
Il punto di forza sta nel database integrato con il sistema operativo, che permette la gestione dei dati attraverso una suite di programmi e librerie altamente specializzati. Attualmente consente di eseguire tutte le operazioni disponibili su di un comune server web, grazie ai continui aggiornamenti che hanno portato, ad esempio, all'installazione di PHP direttamente a livello di sistema operativo. Questo database viene usato, in Sanmarco Informatica, per contenere tutti i dati degli utenti, sia interni che esterni, nonchè i dati relativi ai contatti.

2.1.6 Hibernate

Hibernate è una piattaforma middleware open-source per gestire la persistenza dei dati in un database.
è largamente utilizzato per la gestione dei dati di applicazioni web scritte in Java, in quanto i dati vengono rappresentati attraverso degli oggetti Java chiamati *entità*. Un'entità hibernate si sostituisce logicamente alla tabella reale del database. In questo modo il programmatore può operare sul database come fosse un oggetto Java, semplificando di molto le operazioni di lettura/scrittura e di modifica delle tabelle stesse.

2.1.7 DBeaver

DBeaver è un client SQL, con anche la funzione di tool di amministrazione.
La sua applicazione desktop è scritta in Java e si basa sulla piattaforma Eclipse.
Dbeaver consente la visualizzazione grafica delle tabelle del database SQL sopra citato ed inoltre su di esse è possibile eseguire queries scritte in appositi file di script.

2.1.8 JavaScript

JavaScript è un linguaggio di scripting orientato agli oggetti ed agli eventi.
Viene utilizzato principalmente come linguaggio per la *logica di presentazione* di applicazioni web. Ciò consiste nella creazione di script, scatenati dall'utente per mezzo

Figura 2.5: Logo di DBeaver

di strumenti quali mouse e tastiera, che producono effetti dinamici ed interattivi lato client, cioè nell'interfaccia utente. Questo linguaggio eredita la sintassi dal sopracitato Java (derivato comunque dal linguaggio C), ma differisce sostanzialmente da esso per alcuni motivi:

* JavaScript è un linguaggio interpretato: cio significa che non è necessaria la compilazione perchè esso venga eseguito, bensì sarà il browser che, a runtime, interpreta il codice ed esegue i calcoli necessari;

* a differenza di Java, javascript è debolmente tipizzato: ciò significa che una variabile può assumere diversi tipi, a seconda del suo utilizzo;

* Javascript è un linguaggio debolmente orientato all'ereditarietà tra oggetti, altro aspetto dai cui differisce in maniera sostanziale da Java.

2.1.8.1 Analisi delle alternative

Figura 2.6: I loghi delle alternative prese in esame

Durante la mia esperienza di stage era richiesto, come illustrerò dettagliatamente nel capitolo successivo, l'analisi e la successiva scelta di librerie JavaScript open-source da impiegare nella reimplementazione della componente form di JGalileo CRM, con campi che fossero creati dinamicamente a partire da file JSON.
Ho impiegato circa 40 ore di lavoro per documentarmi sulle varie librerie oggi esistenti, e mi sono focalizzato sulle librerie che seguono:

2.1.8.2 React

React è una libreria JavaScript open-source sviluppata da Facebook a partire dal 2013. è utilizzata, oltre che per Facebook stesso, per una moltitudine di applicazioni web come WhatsAppWeb, Netflix, Aribnb, BBC,...
Le principali caratteristiche di React sono:

* **One way data binding**: Singifica che l'HTML, quindi la pagina visibile all'utente, non è in grado di modificare il componente stesso. L'unico modo per modificare un componente è scatenare un evento che modifichi il componente, che a sua volta renderizzerà la modifica sullo schermo.
* **VirtualDOM**: React opera su di una rappresentazione del DOM, ciò vuol dire che il programmatore può sviluppare pensando che ad ogni cambiamento la pagina venga interamente renderizzata nuovamente: in realtà React valuta le differenze tra il DOM reale e quello virtuale, renderizzando in maniera efficiente solamente le parti interessate al cambiamento.

Alcuni aspetti negativi comprendono invece:

* **Consumo di risorse**: a causa del virtualDOM, sono necessarie molte risorse per l'esecuzione, specialente in termini di RAM da parte del browser;
* **Difficoltà con le form**: I valori dei campi dati delle form sono passati utilizzando dei riferimenti (a causa del one way data flow descritto in precedenza), ciò va contro le *best practices* ed, in form molto grandi, causa sensibili peggioramenti in termini di prestazioni.

2.1.8.3 Angular

Angular è un framework, cioè un'infrastruttura per la creazione di applicazioni composta da un'insieme di funzionalità, sviluppato da Google e disponibile in due versioni: AngularJS ed Angular.
AngularJS viene rilasciato nel 2012 e si dichiara, citando la documentazione ufficiale

> quello che HTML avrebbe dovuto essere se fosse stato progettato per sviluppare applicazioni.

AngularJS ha quindi l'obiettivo di esaltare l'aspetto dichiarativo dell'HTML da un lato, e fornire degli strumenti per la creazioni di componenti per la gestione della logica applicativa di un'applicazione.
Le principali caratteristiche comprendono:

* **Supporto al pattern MVC**;
* **Two ways data binding**: La pagina HTML modifica direttamente lo stato del componente, che viene realizzato mediante l'uso di controller;
* **Dependency injection**.

Nonostante fosse ormai uno dei framework più adottati a livello mondiale per costruire SPA, Google ha deciso di rilasciare, nel 2016, una versione (chiamata genericamente Angular oppure Angular 2+) che modifica radicalmente il framework, tanto da non essere compatibile con AngularJS.
Questo cambiamento, dapprima molto criticato dalla comunità di sviluppatori, non ha

inficiato sulla popolarità di Angular, che rimane anche nella nuova versione uno dei framework più utilizzati.
Angular 2+: La seconda versione di Angular si differenzia con la precedente per queste caratteristiche:

* **typeScript**: Angular2+ è stato scritto in typeScript, una sorta di super-set di JavaScript a cui vengono aggiunti costrutti come classi, interfacce e moduli.

* **Sparisce il two ways data binding**: Quello che era stato il punto di forza di AngularJS si è rivelato una debolezza in termini di prestazioni e soprattutto di memory leak. Resta comunque attivabile quando necessario.

* **Componenti**: spariscono l'idea di scope e controllers, tutta la logica di un componente un è racchiusa all'interno di un componente, esportato poi da un modulo.

2.1.8.4 Vue.js

Vue.js è un framework rilasciato nel 2014 da Evan You, ex dipendente Google, con l'intento di prendere solo le parti migliori di AngularJS per creare qualcosa di molto più leggero.
è stato sviluppato soprattutto per la costruzione di interfacce utente, ma comunque è possibile creare intere SPA.
Vue.js prende spunto sia da AngularJS (two way binding ottimizzato) che da React(VirtualDOM, componenti).
La diffusione relativamente ridotta rende difficoltosa la ricerca della soluzione a problemi che si presenteranno durante lo sviluppo.

2.1.8.5 Scelta finale

Nella scelta finale del framework sono stati presi in considerazione i pregi e difetti illustrati sopra, naturalmente riportati all'obiettivo finale dello stage, ma anche i vincoli personali che mi sono imposto ed i vincoli aziendali che mi sono stati imposti.

* **Angular**: La prima scelta ricadeva su Angular2+, per la grande diffusione e per le API specifiche per le form, ma il vincolo aziendale che mi è stato imposto è stato quello di non appesantire troppo il software, riutilizzando se possibile i framework già presenti.
  Essendo AngularJS già utilizzato in alcune portlet, la mia scelta è alla fine ricaduta su quello, anche a causa dell'impossibilità che hanno i due framework a coesistere;

* **React**: Ho scartato l'utilizzo di questa libreria a causa della già citata mancanza di soluzioni ad-hoc per le form e perché ho già utilizzato questa tecnologia durante lo sviluppo del progetto di ingegneria del software;

* **Vue**: Ho scartato Vue.js soprattutto a causa del vincolo che mi è stato imposto, consideravo affascinante l'idea di cimentarmi con un linguaggio nuovo e di crescente popolarità.

2.1.9 IDE e Sistema Operativo

Sanmarco informatica utilizza l'IDE Eclipse nella versione 4.4 (Luna), mentre per quanto riguarda il sistema operativo ho sviluppato con Windows 7 Professional, ma alcuni colleghi utilizzano indifferentemente macOS.

2.2 Vincoli implementativi

Come già anticipato, mi sono stati imposti alcuni vincoli da rispettare durante l'esperienza di stage:

* Tutte le tecnologie eventualmente utilizzate avrebbero necessariamente dovuto essere *open-source*;
* Evitare il più possibile di appesantire ultieriormente il software con nuove librerie o framework, ma possibilmente riutilizzare quelli già in uso.

Capitolo 3

Descrizione dello stage

In questo capitolo descriverò in maniera approfondita lo scopo dello stage, focalizzandomi sugli obiettivi da raggiungere e le metodologie per farlo.
Seguirà poi una descrizione dettagliata dei casi d'uso di interesse.

3.1 Introduzione al progetto

3.2 Dominio applicativo

3.2.1 Tipi di utenti

JGalileo CRM è stato pensato per essere utilizzato da svariate tipologie di utenti: i casi d'uso seguiranno solamente i due utenti più importanti:

* **amministratore**: che ha il potere di creare nuovi utenti e configurare l'interfaccia finale aggiungendo e togliendo portlet. Inoltre può accedere al pannello di controllo e monitorare la lista di utenti, modificarne i permessi e visualizzare svariate statistiche di utilizzo del prodotto;

* utente semplice, cui è permesso solamente utilizzare il software, senza aggiungere o togliere portlet e naturalmente senza avere la possibilità di creare altri utenti.

Quest'ultima è la tipologia di utenti più comune, infatti la clientela di JGalileo CRM è composta solamente da utenti semplici, mentre solamente i membri del team di sviluppo possiedono i permessi di amministratore.

3.2.2 Funzionalità

Le funzionalità prese in esame si sviluppano soprattutto sull'interazione tra l'utente semplice e la form la cui creazione era l'obiettivo del mio stage. Quindi i casi d'uso riguarderanno principalmente:

* **Inserimento di un nuova categoria di clienti**;

* **Inserimento di attività ed opportunità**;

- **visualizzazione dettagliata dei clienti già inseriti**;
- **visualizzazione dettagliata delle attività ed opportunità già inserite**;
- **modifica e salvataggio dei dati**;
- **invio di email, fax e newsletter**.

In particolare, le categorie di clienti inseribili tramite form sono:

- **Leads**: Sono persone ed aziende che possono avere interesse ai prodotti e servizi dell'utente. Anche un primo contatto, come ad esempio un biglietto da visita, è considerato un lead;
- **Accounts**: Contengono i dati di un'azienda con cui è già presente un qualche tipo di relazione commerciale, anche solo una roposta d'acquisto;
- **Contatti**: I contatti consistono in persone fisiche con cui è già presente un qualche tipo di relazione commerciale. Spesso account e contatto sono in relazione tra di loro, ma è anche possibile che un contatto non abbia nessun account associato (ad esempio un privato).

mentre per attività ed opportunità si intende:

- **attività**: un attività riguarda qualsiasi tipo di operazione ed evento con lo scopo di ottenere un ritorno in termini di visibilità ai clienti ed alle aziende partner. Tra le attività si inseriscono fiere, esposizioni, eventi, cene,...;
- **opportunità**: Le opportunità costituiscono, per il cliente che acquista JGalileo CRM, di tenere traccia delle possibilità di concludere accordi commerciali con i clienti.

Per quanto riguarda la parte di amministrazione, verrà illustrata solamente la procedura di inserimento di un nuovo utente.

3.3 Casi d'uso

I casi d'uso (use case) sono una tecnica dell'ingegneria del software per ottenere un'analisi precisa e senza ambiguità dei requisiti di un sistema, con l'obiettivo di perseguire la creazione di software di qualità.
I casi d'uso sono solitamente rappresentati in forma testuale, attraverso la descrizione dettagliata del caso d'uso stesso in termini di attori coinvolti, pre e post condizioni, e da una rappresentazione grafica, definita Use Case Diagram, con l'ausislio di un altro linguaggio, UML.
Tutti i casi d'uso saranno qui presentati attraverso descrizione testuale, mentre la rappresentazione grafica verrà utilizzata solamente per gli scenari più generali.
I casi d'uso qui riportati non si riferiscono all'intero software a causa della sua grande complessità ed ampiezza d'utilizzo, ma solamente agli scenari principali che, direttamente od indirettamente, sono influenzati dal codice da me scritto durante l'esperienza di stage.

3.3.1 Classificazione dei casi d'uso

Ogni caso d'uso è classificato secondo la seguente convenzione:

UC[codice]

Dove [codice] è un codice numerico che identifica univocamente il caso d'uso.
Esso è sequenziale e gerarchico, dunque se ad esempio il caso d'uso "X" necessita di un ulteriore livello di dettaglio si procede individuando i sotto-casi d'uso "X.Y", "X.Y.Z" etc.. dove:

* **X**: il codice identificativo del caso d'uso padre, solitamente generico e difficilemte descrivibile nel dettaglio;
* **Y**: codice figlio, identifica un caso d'uso più particolare rispetto al padre (che lo contiene).
* **Z**: codice nipote, individua lo scenario più particolare possibile.

X parte con indice 1 e valore crescente, mentre Y e Z parto dall'indice 0 con valore crescente.
Indicherò con il suffisso "_G" tutti quei casi d'uso di alto livello, a segnalare che si tratta di una visione generale del contesto.

3.3.2 Struttura dei casi d'uso

Ogni caso d'uso verrà descritto in maniera testuale, utilizzando la seguente struttura:

* **Attori**: Gli attori comprendono le entità umane che interagiscono con il sistema. Nella fattispecie, vengono distinti tre tipologie di attori:
  - **Amministratore**: Questa tipologia di attore è quella riservata solamente agli sviluppatori del prodotto;
  - **Utente non autenticato**: indica un attore non ancora autenticato nel sistema;
  - **Utente autenticato**, indicato anche semplicemente come **Utente**, indica un utente non amministratore che viene riconosciuto dal sistema e può quindi usufruirne.
* **Pre-condizione**: Indica le condizioni che devono necessariamente essere soddisfatte affinchè sia possibile l'interazione dell'attore con lo specifico caso d'uso;
* **Post-condizione**: Indica le condizioni in cui l'attore si verrà a trovare dopo la fine dell'interazione con lo specifico caso d'uso;
* **Descrizione**: Riporta una breve descrizione testuale del caso d'uso.

3.4 UC 1_G: Funzionalità amministratore

3.4.1 UC 1 - inserimento nuovo utente

* **Descrizione**: l'amministratore può aggiungere un nuovo utente inserendo i seguenti campi nell'apposita form:

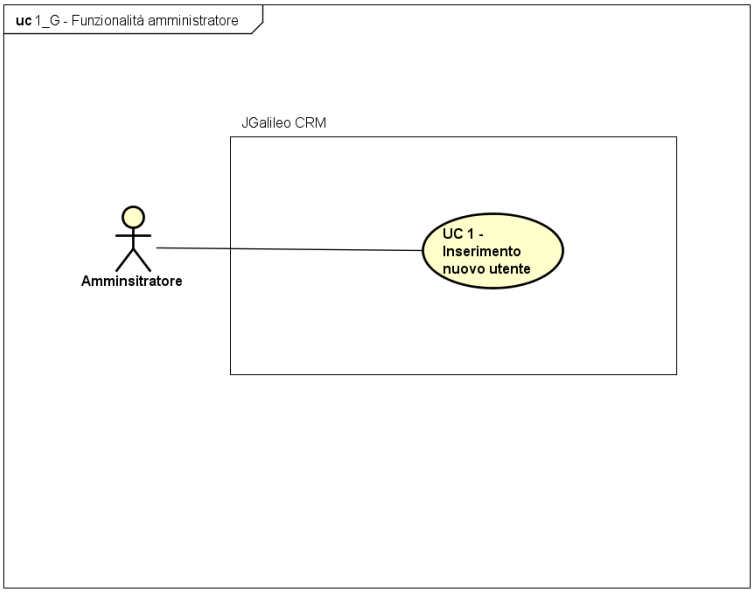


Figura 3.1: Caso d'uso generale-funzionalità amministratore

  - nome;
  - cognome;
  - titolo;
  - data di nascita;
  - sesso;
  - occupazione.

* **Attore**: Utente amministratore;
* **Pre-condizione**: L'amministratore ha già effettuato l'accesso tramite inserimento di username e password, ed ha raggiunto il pannello di amministrazione;
* **Post-condizione**: è stato inserito un nuovo utente.

3.5 UC 2_G: Funzionalità utente - inserimento

I casi d'uso qui riportati si riferiscono all'inserimento di nuovi dati nel sistema

Negli UC 3, 4, 5, 6, 7 non vengono riportati tutti i campi inseribili, in quanto, per ogni singola form, si hanno almeno 40 differenti campi.

3.5.1 UC 2 - Autenticazione

* **Descrizione**: per l'autenticazione è necessario inserire nel sistema:
  - **nome utente**;
  - **password**.

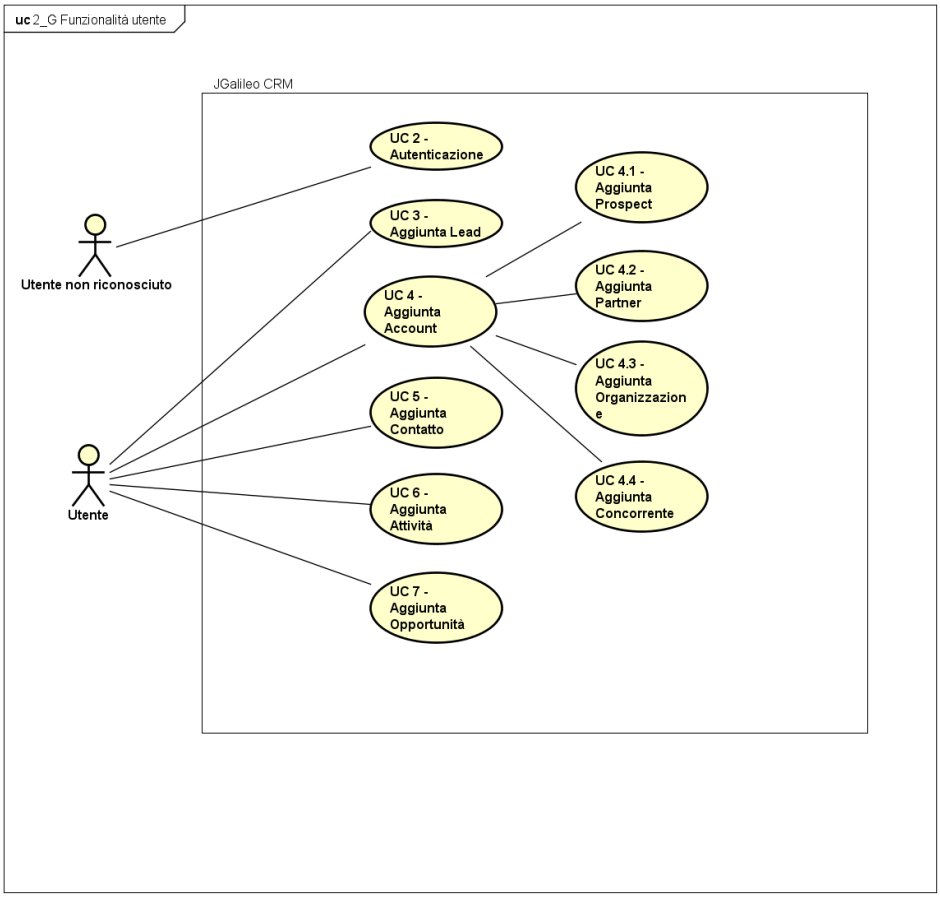


Figura 3.2: Caso d'uso generale-funzionalità utente

* **Attore**: Utente non autenticato;
* **Pre-condizione**: Un utente amministratore deve aver aggiunto l'utente che sta per effettuare l'accesso affinchè l'autenticazione vada a buon fine;
* **Post-condizione**:l'utente viene autenticato e può utilizzarne le funzionalità del prodotto in base ai suoi permessi.

3.5.2 UC 3 - Aggiunta lead

* **Descrizione**: un utente può aggiungere un lead al suo insieme di potenziali clienti. Per farlo deve inserire alcune informazioni obbligatorie, come:
  - **cognome**;
  - **nome della società**.

  e molte altre facoltative, ma che sono molto importanti al fine di avere una rapida e completa panoramica del potenziale cliente. Tra queste ci sono:
  - **email**;

  - **indirizzo dell'ufficio**;
  - **descrizione del lead**;
  - **numero di cellulare**;
  - **provenienza del lead**.

* **Attore**: Utente autenticato;

* **Pre-condizione**: Un utente deve aver eseguito l'accesso ed essersi portato alla pagina di inserimento lead;

* **Post-condizione**:il nuovo lead contenente le informazioni inserite viene salvato [UC 12] nel sistema ed è disponibile all'utente.

3.5.3 UC 4 - Aggiunta account

* **Descrizione**: un utente può aggiungere un account alla lista di clienti. Un account può essere di vari tipi, come:

  - **Prospect** (UC 4.1): un prospect rappresenta una potenziale azienda cliente, con la quale non si è ancora aperto alcun tipo di rapporto commerciale;
  - **Partner** (UC 4.2):un partner consiste in un'azienda con cui si è stabilito un duraturo rapporto di scambi, eventualmente anche non solo commerciali;
  - **Organizzazione** (UC 4.3):un organizzazione consiste in un'insieme di persone e mezz, volti ad uno stesso obiettivo comune;
  - **Concorrente** (UC 4.4): questo tipo di account si riferisce ad un'azienda che opera nello stesso settore del cliente, in concorrenza con esso.

  Questi tipi di account hanno form che utilizzano campi dati anche molto diversi tra loro, a causa della differente e talvolta profonda natura che li connota, ma anche taluni campi in comune, come:

  - **Ragione sociale**;
  - **indirizzo**;
  - **numero di telefono**;
  - **città**;
  - **partita IVA**.

* **Attore**: Utente autenticato;

* **Pre-condizione**: Un utente deve aver eseguito l'accesso ed essersi portato alla pagina di inserimento account;

* **Post-condizione**:il nuovo account contenente le informazioni inserite viene salvato nel sistema [UC 12] ed è disponibile all'utente.

3.5.4 UC 5 - Aggiunta contatto

* **Descrizione**: un utente può aggiungere un contatto al sistema. Per farlo deve inserire alcune informazioni obbligatorie, come:
  - **cognome**;
  - **account da associare al contatto**.

  e molte altre facoltative, tra le quali:
  - **email**;
  - **indirizzo dell'ufficio**;
  - **qualifica del contatto**;
  - **stabilimento di lavoro del contatto**.
* **Attore**: Utente autenticato;
* **Pre-condizione**: Un utente deve aver eseguito l'accesso ed essersi portato alla pagina di inserimento contatto;
* **Post-condizione**:il nuovo contatto contenente le informazioni inserite viene salvato nel sistema [UC 12] ed è disponibile all'utente.

3.5.5 UC 6 - Aggiunta attività

* **Descrizione**: un utente può aggiungere un'attività alla lista di attività già inserite in precedenza. Per farlo deve inserire alcune informazioni obbligatorie, come:
  - **descrizione**: una piccola descrizione testuale dell'attività;

  e molte altre facoltative, tra le quali:
  - **sede della visita**;
  - **account e contatti associati**;
  - **data d'inizio**;
  - **data di fine**.
* **Attore**: Utente autenticato;
* **Pre-condizione**: Un utente deve aver eseguito l'accesso ed essersi portato alla pagina di inserimento attività;
* **Post-condizione**:la nuova attività contenente le informazioni inserite viene salvata nel sistema [UC 12] ed è disponibile all'utente.

3.5.6 UC 7 - Aggiunta opportunità

* **Descrizione**: un utente può aggiungere un'opportunità alla lista di opportunità create in precedenza. Per farlo deve inserire alcune informazioni obbligatorie, come:
  - **nome utente**;
  - **password**.

  e molte altre facoltative, tra le quali: s
  - **nome utente**;
  - **password**.
* **Attore**: Utente autenticato;
* **Pre-condizione**: Un utente deve aver eseguito l'accesso ed essersi portato alla pagina di inserimento opportunità;
* **Post-condizione**:la nuova opportunità contenente le informazioni inserite viene salvata nel sistema [UC 12] ed è disponibile all'utente.

3.6 UC 3_G - Visualizzazione dettagli

Questi casi d'uso si riferiscono alla visualizzazione ed eventuale modifica di schede di leads, account, contatti ed opportunità già inserite.

3.6.1 UC 8 - Visualizzazione dettaglio lead

* **Descrizione**: un utente può vedere nel dettaglio le informazioni inserite a proposito di un cliente lead. Può eventualmente aggiornarne le informazioni, eliminarlo oppure mandare una mail od un fax, come evidenziato negli [UC 12], [UC 13], [UC 14], [UC 15], [UC 16] ed [UC 17].
* **Attore**: Utente autenticato;
* **Pre-condizione**: Un utente deve aver eseguito l'accesso, essersi portato alla pagina riguardante i lead ed averne selezionato uno;
* **Post-condizione**:il lead eventualmente modificato viene salvato nel sistema per una prossima interazione.

3.6.1.1 UC 8.1 - Conversione lead

* **Descrizione**: un utente può convertire il lead: In fase di inserimento, infatti, un lead viene di default salvato come "non convertito".
* **Attore**: Utente autenticato;
* **Pre-condizione**: Un utente deve aver eseguito l'accesso, essersi portato alla pagina riguardante i lead, averne selezionato uno ed essere quindi entrato nella pagina del dettaglio;
* **Post-condizione**:il lead viene convertito e le informazioni riguardo ad esso non sono più modificabili.

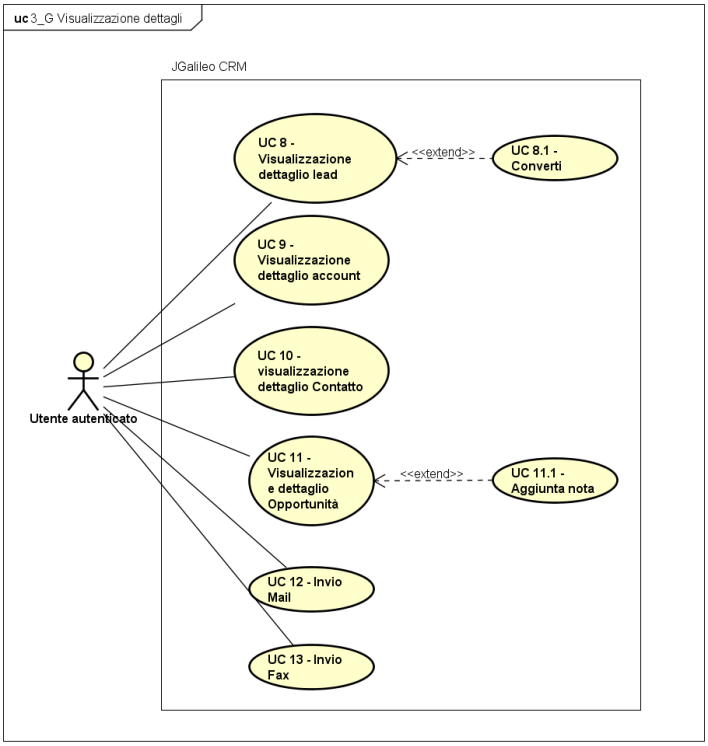


Figura 3.3: Caso d'uso generale 3 - visualizzazione dettagli dei dati già inseriti

3.6.2 UC 9 - Visualizzazione dettaglio account

* **Descrizione**: un utente può vedere nel dettaglio le informazioni relative ad un account in particolare. Può eventualmente aggiornarne le informazioni, eliminarlo oppure mandare una mail od un fax, come evidenziato negli [UC 12], [UC 13], [UC 14], [UC 15], [UC 16] ed [UC 17].

* **Attore**: Utente autenticato;

* **Pre-condizione**: Un utente deve aver eseguito l'accesso, essersi portato alla pagina relativa agli account ed averne selezionato uno;

* **Post-condizione**:l'account eventualmente modificato viene salvato nel sistema per una prossima interazione.

3.6.3 UC 10 - Visualizzazione dettaglio contatto

* **Descrizione**: un utente può vedere le informazioni relative ad un contatto precedentemente inserito nel sistema e può eventualmente modificarlo, inviare mail o fax, come evidenziato in [UC 12], [UC 13], [UC 14], [UC 15], [UC 16] ed [UC 17].

* **Attore**: Utente autenticato;

* **Pre-condizione**: Un utente deve aver eseguito l'accesso, essersi portato alla pagina di inserimento contatto ed averne selezionato uno;
* **Post-condizione**:il contatto eventualmente modificato viene salvato nel sistema per una prossima interazione.

3.6.4 UC 11 - Visualizzazione dettaglio opportunità

* **Descrizione**: un utente può vedere ed eventualemnte modificare le informazioni riguardanti una specifica opportunità. Le azioni disponibili per le opportunità sono descritte nei casi d'uso [UC 12], [UC 13], [UC 14], [UC 15], [UC 16] ed [UC 17].

* **Attore**: Utente autenticato;
* **Pre-condizione**: Un utente deve aver eseguito l'accesso, essersi portato nella pagina relativa alle opportunità ed averne selezionata una;
* **Post-condizione**:l'opportunità eventualmente aggiornata viene salvata nel sistema per una successiva interazione.

3.6.4.1 UC 11.1 - Aggiunta nota

* **Descrizione**:
* **Attore**:
* **Pre-condizione**:
* **Post-condizione**:

3.7 UC 4_G - Azioni comuni

Questo paragrafo si riferisce alle azioni che è possibile compiere sulle categorie di clienti come lead, account e contatti e sulle schede di attività ed opportunità.// La scelta di riportarle come caso d'uso indipendenti deriva dalla volontà di evitare ripetizioni verbose e migliorare la leggibilità dei diagrammi UML. Per facilitare il riferimento ad un qualsiasi lead, account, contatto, attività ed opportunità precedentemente inserito nel sistema oppure in fase di inserimento verrà utilizzato il termine **scheda**.

3.7.1 UC 12 - Salva

* **Descrizione**: un utente può confermare di voler inserire i dati appena immessi nel sistema o salvare le modifiche fatte ad una scheda.
in particolare i dati vengono salvati nel database AS400.
* **Attore**: Utente autenticato;
* **Pre-condizione**: Un utente deve aver eseguito l'accesso, essersi portato ad una pagina di inserimento o di visualizzazione dettaglio ed avere inserito le informazioni obbligatorie oppure modificato almeno un campo rispetto ad una scheda preesistente;
* **Post-condizione**:viene notificato il corretto inserimento nel sistema oppure le corretta modifica di dati precedentemente inseriti.

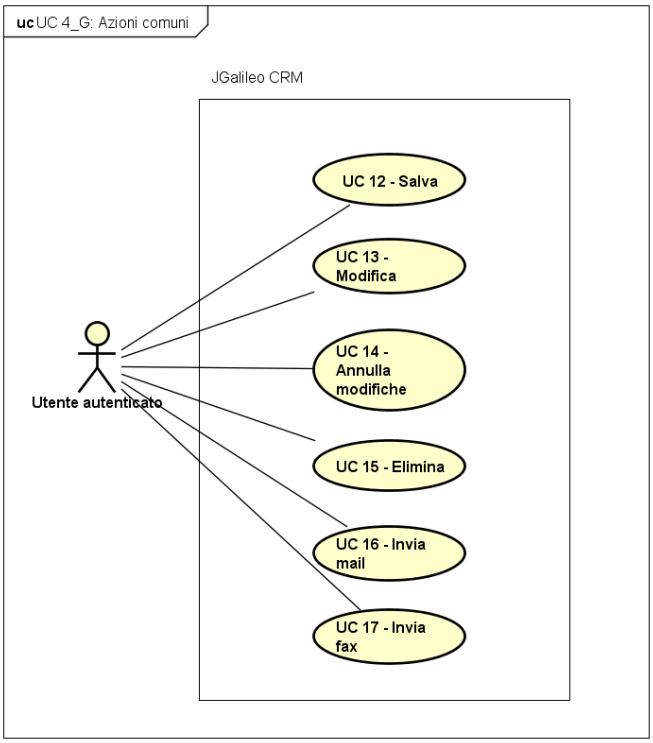


Figura 3.4: Caso d'uso generale 4 - azioni comuni alle categorie sopradescritte

3.7.2 UC 13 - Modifica

* **Descrizione**: un utente può modificare le informazioni riguardanti una specifica scheda.
* **Attore**: Utente autenticato;
* **Pre-condizione**: Un utente deve aver eseguito l'accesso ed avere aperto una specifica scheda di suo interesse;
* **Post-condizione**:l'utente è ora abilitato alla modifica dei campi su cui è premessa la modifica.

3.7.3 UC 14 - Annulla modifiche

* **Descrizione**: un utente può annullare le modifiche eventualmente apportate ad una scheda, riportandola allo stato in cui è salvata sul database;
* **Attore**: Utente autenticato;
* **Pre-condizione**: Un utente deve aver eseguito l'accesso, avere aperto una specifica scheda di suo interesse ed averne modificato almento un campo;
* **Post-condizione**:la scheda viene ripulita da tutte le modifiche e ritorna quindi allo stato in cui è stata recuperata dal database.

3.7.4 UC 15 - Elimina

* **Descrizione**: un utente può eliminare una scheda precedentemente inserita;
* **Attore**: Utente autenticato;
* **Pre-condizione**: Un utente deve aver eseguito l'accesso ed avere aperto una specifica scheda di suo interesse;
* **Post-condizione**:I dati relativi alla scheda in esame vengono eliminati dal database e non sono quindi più disponibili.

3.7.5 UC 16 - Invia mail

* **Descrizione**: un utente può inviare una email all'indirizzo email presente nella scheda, se questo è stato inserito;
* **Attore**: Utente autenticato;
* **Pre-condizione**: Un utente deve aver eseguito l'accesso ed avere aperto una specifica scheda di suo interesse;
* **Post-condizione**:L'utente viene reindirizzato ad una pagina in cui può scrivere la mail e può, tramite una form, inserire ulteriori informazioni.

3.7.6 UC 17 - Invia fax

* **Descrizione**: un utente può inviare un fax al numero di fax presente nella scheda, se questo è stato inserito;
* **Attore**: Utente autenticato;
* **Pre-condizione**: Un utente deve aver eseguito l'accesso ed avere aperto una specifica scheda di suo interesse;
* **Post-condizione**:L'utente viene reindirizzato ad una pagina in cui può scrivere il fax e può, tramite una form, inserire ulteriori informazioni.

3.8 Tracciamento dei requisiti

3.8.1 Struttura di un requisito

Ogni requisito identificato è stato classificato come segue:

R[Importanza][Classificazione][Identificativo]

* **Importanza:** Specifica il grado di necessità che il requisito possiede, si articola in:
  - **O (Obbligatorio)**: Il suo soddisfacimento è essenziale alla buona riuscita del progetto;
  - **D (Desiderabile)**: Il suo soddisfacimento non è necessario ma offre funzionalità tali da renderlo uno dei primi requisiti da soffisfare in sviluppi successivi;

  - **F (Funzionale)**: Il valore aggiunto dato dal soddisfacimento di questo requisito non è così importante, quindi prima di soddisfare il requisito è consigliata un'analisi di tempi e costi per evitare ritardi nella consegna e/o costi superiori a quelli preventivati.

* **Classificazione**: Indica il tipo del requisito, in termini di finalità dello stesso:
  - **F (Funzionali)**: indica una funzionalità che il prodotto deve avere;
  - **Q (Qualità)**: indica un requisito che riguarda l'aspetto qualitativo del prodotto;
  - **V (Vincolo)**: indica un vincolo che deve essere rispettato durante lo sviluppo del prodotto.

* **Identificativo**: Un codice numerico crescente a partire da 1 e che identifica un requisito in modo univoco.

3.8.2 Tracciamento dei requisiti

Le tabelle XXX e seguenti illustrano il codice del requistito, una breve descrizione testuale e lo stato di completamento alla fine dell'esperienza di stage.

Requisito	**Descrizione**	**Stato**
ROF01	Un utente può compilare la form per l'inserimento di un nuovo lead	Completato
ROF02	Un utente può salvare un nuovo lead	Completato
ROF03	Il sistema, in fase di salvataggio, deve validare il nuovo lead	Completato
ROF04	Un utente può compilare la form per l'inserimento di un nuovo prospect	Completato
ROF05	Un utente può salvare un nuovo prospect	Completato
ROF07	Il sistema, in fase di salvataggio, deve validare il nuovo prospect	Completato
ROF07	Un utente può compilare la form per l'inserimento di un nuovo partner	Completato
ROF08	Un utente può salvare un nuovo partner	Completato
ROF09	Il sistema, in fase di salvataggio, deve validare il nuovo partner	Completato
ROF10	Un utente può compilare la form per l'inserimento di una nuova organizzazione	Completato
ROF11	Un utente può salvare un nuova organizzazione	Completato
ROF12	Il sistema, in fase di salvataggio, deve validare la nuovoa organizzazione	Completato
ROF13	Un utente può compilare la form per l'inserimento di un nuovo concorrente	Completato
ROF14	Un utente può salvare un nuovo concorrente	Completato
ROF15	Il sistema, in fase di salvataggio, deve validare il nuovo concorrente	Completato
ROF16	Un utente può compilare la form per l'inserimento di una nuova attività	Completato

ROF17	Un utente può salvare un nuova attività	Completato
ROF18	Il sistema, in fase di salvataggio, deve validare la nuovoa attività	Completato
ROF19	Un utente può compilare la form per l'inserimento di un nuovo contatto	Completato
ROF20	Un utente può salvare un nuovo contatto	Completato
ROF21	Il sistema, in fase di salvataggio, deve validare il nuovo contatto	Completato
ROF22	Un utente può compilare la form per l'inserimento di una nuova attività	Completato
ROF23	Un utente può salvare una nuova attività	Completato
ROF24	Il sistema, in fase di salvataggio, deve validare la nuova attività	Completato
ROF25	Un utente può compilare la form per l'inserimento di una nuova opportunità	Completato
ROF26	Un utente può salvare una nuova opportunità	Completato
ROF27	Il sistema, in fase di salvataggio, deve validare la nuova opportunità	Completato
ROF28	Un utente può vedere nel dettaglio un lead precedentemente inserito	Completato
ROF29	Un utente può modificare un lead precedentemente inserito	Completato
ROF30	Un utente può annullare le modifiche apportate ad un lead precedentemente inserito	Completato
ROF31	Un utente può eliminare un lead precedemente inserito	Completato
ROF32	Un utente può convertire un lead precedemente inserito	Completato
ROF33	Un utente può inviare una mail ad un lead precedentemente inserito	Completato
ROF34	Un utente può inviare un fax ad un lead precedentemente inserito	Completato
ROF35	Un utente può vedere nel dettaglio un prospect precedentemente inserito	Completato
ROF36	Un utente può modificare un prospect precedentemente inserito	Completato
ROF37	Un utente può annullare le modifiche apportate ad un prospect precedentemente inserito	Completato
ROF38	Un utente può eliminare un prospect precedemente inserito	Completato
ROF39	Un utente può inviare una mail ad un prospect precedentemente inserito	Completato
ROF40	Un utente può inviare un fax ad un prospect precedentemente inserito	Completato
ROF41	Un utente può vedere nel dettaglio un prospect precedentemente inserito	Completato
ROF42	Un utente può modificare un partner precedentemente inserito	Completato

Tabella 3.2: Requisiti di qualità

Requisito	Descrizione	Stato
ROQ01	Il prototipo sviluppato deve avere le stesse funzionalità del precedente	Completato
RDQ02	Il prototipo deve essere completato utilizzando il design Material	Non completato

ROF43	Un utente può annullare le modifiche apportate ad un partner precedentemente inserito	Completato
ROF44	Un utente può eliminare un partner precedemente inserito	Completato
ROF45	Un utente può inviare una mail ad un partner precedentemente inserito	Completato
ROF46	Un utente può inviare un fax ad un partner precedentemente inserito	Completato
ROF47	Un utente può vedere nel dettaglio un contatto precedentemente inserito	Completato
ROF48	Un utente può modificare un contatto precedentemente inserito	Completato
ROF49	Un utente può annullare le modifiche apportate ad un contatto precedentemente inserito	Completato
ROF50	Un utente può eliminare un contatto precedemente inserito	Completato
ROF51	Un utente può inviare una mail ad un contatto precedentemente inserito	Completato
ROF52	Un utente può inviare un fax ad un contatto precedentemente inserito	Completato
ROF53	Un utente può vedere nel dettaglio un' opportunità precedentemente inserita	Completato
ROF54	Un utente può modificare un'opportunità precedentemente inserita	Completato
ROF55	Un utente può annullare le modifiche apportate ad un' opportunità precedentemente inserita	Completato
ROF56	Un utente può eliminare un' opportunità precedemente inserita	Completato
ROF57	Un utente può inviare una mail ad un' opportunità precedentemente inserita	Completato
ROF58	Un utente può inviare un fax ad un' opportunità precedentemente inserita	Completato
ROF59	Un utente può inviare una nota ad un'opportunità precedentemente inserita	Completato
RDF1	Un utente può aggiungere un nuovo campo ad una form	Non completato
RDF2	Un utente può eliminare un campo non obbligatorio da una form	Non completato

Tabella 3.3: Requisiti di vincolo

Requisito	Descrizione	Stato
ROV01	Il sistema deve usare solamente solamente tecnologie open source	Completato
ROD02	Il prodotto deve essere sviluppato cercando di appesantire il meno possibile il sistema già in uso	Completato

3.9 Gli obiettivi

Gli obiettivi del progetto di stage sono elencati nella tabella 1.
Essi sono classificati da:

* un ID, che rappresenta univocamente l'obiettivo;
* un aggettivo di importanza dell'obiettivo;
* una breve descrizione testuale.

L'ID è formato da una lettera iniziale (P o F), ad indicare se il requisito da soddisfare sia di tipo *Produttivo* o *Formativo*, e da un numero sequenziale.
L'agettivo di importanza può essere: *Obbligatorio*, *Desiderabile* oppure *Facoltativo*, in base alla necessità che ha lo lo stesso di essere soddisfatto.
La breve descrizione testuale descrive nel modo più concreto e riassuntivo possibile l'obiettivo.

Tabella 3.4: Obiettivi dello stage

ID	Importanza	Descrizione
F1	Obbligatorio	Acquisizione delle competenze di base sulla famiglia di software denominata CRM.
F2	Obbligatorio	Acquisizione delle competenze di base sul software di sviluppo GWT.
F3	Desiderabile	Acquisizione di competenze avanzate sul linguaggio di programmazione utilizzato per lo sviluppo del prototipo.
P1	Obbligatorio	Analisi dei requisiti tecnici ed applicativi.
P2	Obbligatorio	Analisi dell'User Interface.
P3	Obbligatorio	Analisi degli Use Case.
P4	Obbligatorio	Sviluppo di un prototipo che implementi le stesse funzionalità del componente esistente.
P5	Facoltativo	Implementazione di nuove funzionalità di interazione con l'utente sul prototipo sviluppato.

3.10 Analisi dei rischi

Ho voluto analizzare una serie di rischi che sarebbero potuti incorrere durante il periodo di stage. Sono riassunti nella tabella

Tabella 3.5: Analisi preventiva dei rischi

Descrizione	Trattamento	Rischio
La scelta di una libreria JavaScript che non soddisfacesse gli scopi dello stage avrebbe significato una grande perdita di tempo e il dover ripartire da capo con lo sviluppo	Analisi approfondita delle alternative disponibili e dialogo con il Tutor aziendale per trovare la soluzione più adeguata	Alto
La mancanza di documentazione dovuta all'adozione della metosologia agile potrebbe portare ad allungare i tempi di comprensione delle componenti	Collaborazione da parte del tutor e del team di sviluppo nel caso ci siano chiarimenti da fornire	Alto
Dovendo interagire con un software già funzionante e molto grande, le parti da me sviluppate potrebbero creare conflitti	Evitare di modificare metodi che possano essere utilizzati da alre componenti	Basso

Capitolo 4

Progettazione

Questo capitolo illustra le caratteristiche dell'applicazione già esistente e lo studio eseguito per comprenderle e motivare le progettuali adottate nello sviluppo del prototipo da me realizzato durante l'esperienza di stage.

4.1 Architettura preesistente

Trattandosi di effettuare la reimplementazione di una funzionalità preesistente, il primo ostacolo è stato quello di capire come il mio prodotto dovesse integrarsi con il resto dell'architettura cui avrebbe fatto parte, per questo motivo la prima settimana di stage è stata dedicata quasi esclusivamente alla conoscenza dell'ambiente e delle componenti.

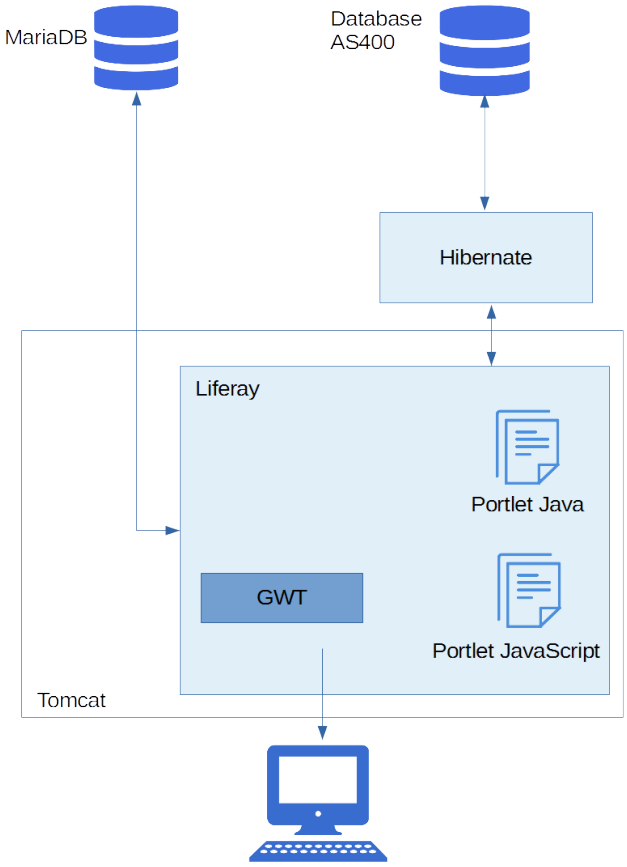


Figura 4.1: Schema generale di interconnessione tra le componenti

4.1.1 Visione generale

la figura 4.1 illustra, ad alto livello, le relazioni che intercorrono tra le varie tecnologie utilizzate dal software JGalileo CRM.
Partendo dalle basi di dati fino ad arrivare all'interfaccia utente, l'architettura del sistema si snoda nel seguente modo:
Entrambe le basi di dati, sia quella contenuta nel database del sistema operativo AS400 che quella contenuta sul server, si interfacciano con il Framework Hibernate per creare delle classi, equivalenti alle tabelle del database, con cui permettere l'interazione.
La sezione dell'appendice A.1 riporta più nel dettaglio come vengono create le entità Hibernate.
L'interazione con le entità di Hibernate avviene per mezzo di Liferay, che si occupa anche della gestione degli accessi degli utenti alla pagina, introducendo il concetto di Single Sing On, e della corretta visualizzazione delle portlet all'interno delle pagine del portale. Queste sono caricate di volta in volta, tramite dei servizi REST, dal server Tomcat e possono essere scritte in AngularJS, come ad esempio la portlet per la geolocalizzazione o quella che gestisce la timeline, oppure in Java per essere successivamente tradotte in JavaScript tramite i servizi offerti da GWT.

Le azioni dell'utente vengono gestite poi da Liferay, che si occupa anche del routing tra le varie pagine del portale. Nel caso ci fossero dei dati da salvare, come nel caso dell'inserimento di un nuovo lead, vengono lanciati dei servizi REST che andranno ad interfacciarsi con le entità gestite da Hibernate, il quale gestisce la persistenza dei dati.

4.1.2 Portlet life cycle

Al fine di capire al meglio come progettare la nuova form, mi sono concentrato sullo studio del ciclo di vita di una portlet:
L'ambiente in cui una essa viene creata ed utilizzata è denominato *portlet container* che, come suggerisce la parola, è un contenitore che gestisce l'aggregazione tra le portlet, mantiene memoria delle preferenze dettate dall'utente su di esse ed inoltre funge da *Controller* tra l'interfaccia utente(*View*) ed i dati (*Model*) nel paradigma di programmazione MVC (Model View Controller), illustrato nella sezione A.2.

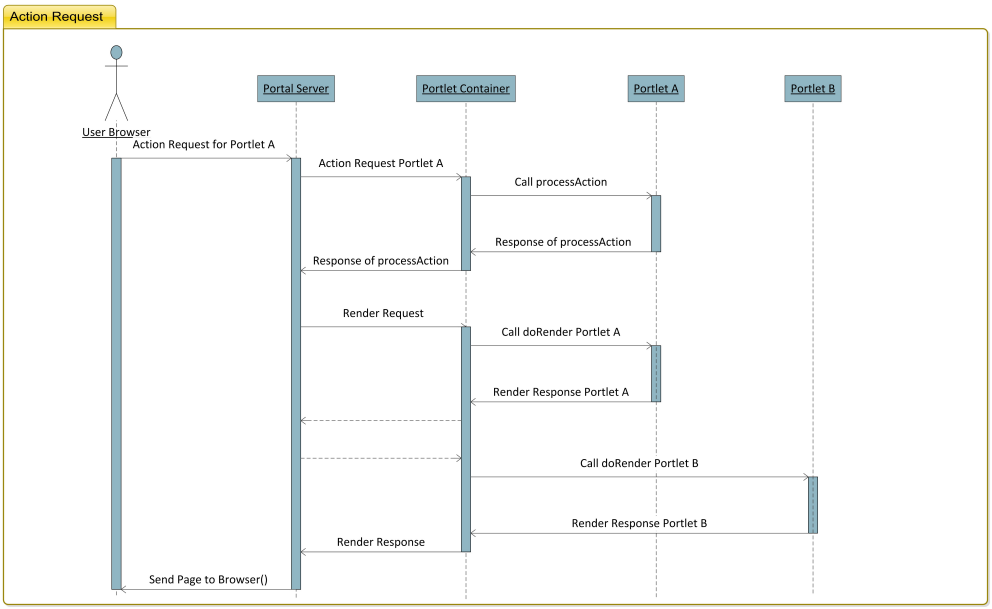


Figura 4.2: Diagramma di sequenza di ProcessAction

Il ciclo di vita di una portlet è caratterizzato da 6 metodi principali che lo regolano:

* **init()**: chiamato dal portlet container, questo metodo crea la portlet desiderata. Durante il ciclo di vita della stassa viene invocato una sola volta;

* **render()**: questo metodo è responsabile della generazione di codice HTML che sarà quindi visualizzato a schermo. Ci sono alcune restrizioni di sicurezza, viene impedita ad esempio la generazione di tag quali <html>, <head>, <body>, che andrebbero a rompere l'intera pagina.

* **processAction()**: processAction viene invocato quando viene eseguita un'azione, ad esempio l'azione di salvataggio di una form. dopo processAction viene

automaticamente invocato il metodo render() della portlet e delle altre portlet, qualora ve ne fossero.

* **processEvent()**: utilizzato per gestire gli eventi, viene poi chiamato il metodo render() della sola portlet interessata;
* **serveRescource()**: metodo utilizzato per reperire risorse utilizzando un URL;
* **destroy()**: distrugge la portlet;

4.2 Form portlet

La form la cui reimplementazione costituisce il mio progetto di stage consiste in un unica portlet,scritta in Java.
La portlet è costituita da una grande classe, di oltre 6.000 linee di codice, che si appoggia poi ad altre classi per la gestione di aspetti più particolari: ad esempio i campi di tipo picker, cioè dei campi di ricerca che recuperano le voci su cui effettuarla dal database, è separata e lunga circa 4.000 linee di codice.
Data l'estrema lunghezza, in rapporto alla mia esperienza di studente, ed alla mancanza di documentazione dovuta all'adozione della metodologia agile, lo studio del funzionamento della portlet che gestisce la form ha richiesto più tempo del previsto.
L'analisi ha portato allo sviluppo del diagramma di flusso riguardandte la creazione della portlet, illustrato nelle figure 4.3 e 4.4. Il diagramma è molto generale a causa della grande complessità di metodi e alternative dettate dal fatto che la form viene creata dinamicamente e deve quindi coprire tutte le possibili casistiche in cui è possibile incorrere.

4.2.1 Descrizione del flusso

Il flusso di esecuzione per la creazione di una form può essere essenzialmente diviso in X parti:

* **settaggio dei parametri e costruzione dei bottoni**;
* **recupero del datasource**;
* **creazione delle azioni**;
* **creazione dei fields descritti dal datasource**;

4.2.1.1 Settaggio dei parametri e costruzione dei bottoni

In questa prima parte del flusso, vengono settati alcuni parametri utili alla creazione della portlet stessa.
Al costruttore della classe vengono infatti passati alcuni parametri, contenuti in oggetti di tipo String od in file JSON, contenenti informazioni quali ad esempio:

* l'utente connesso con i relativi permessi;
* l'id della portlet;
* la configurazione della portlet, contenente parametri utili al recupero del datasource;

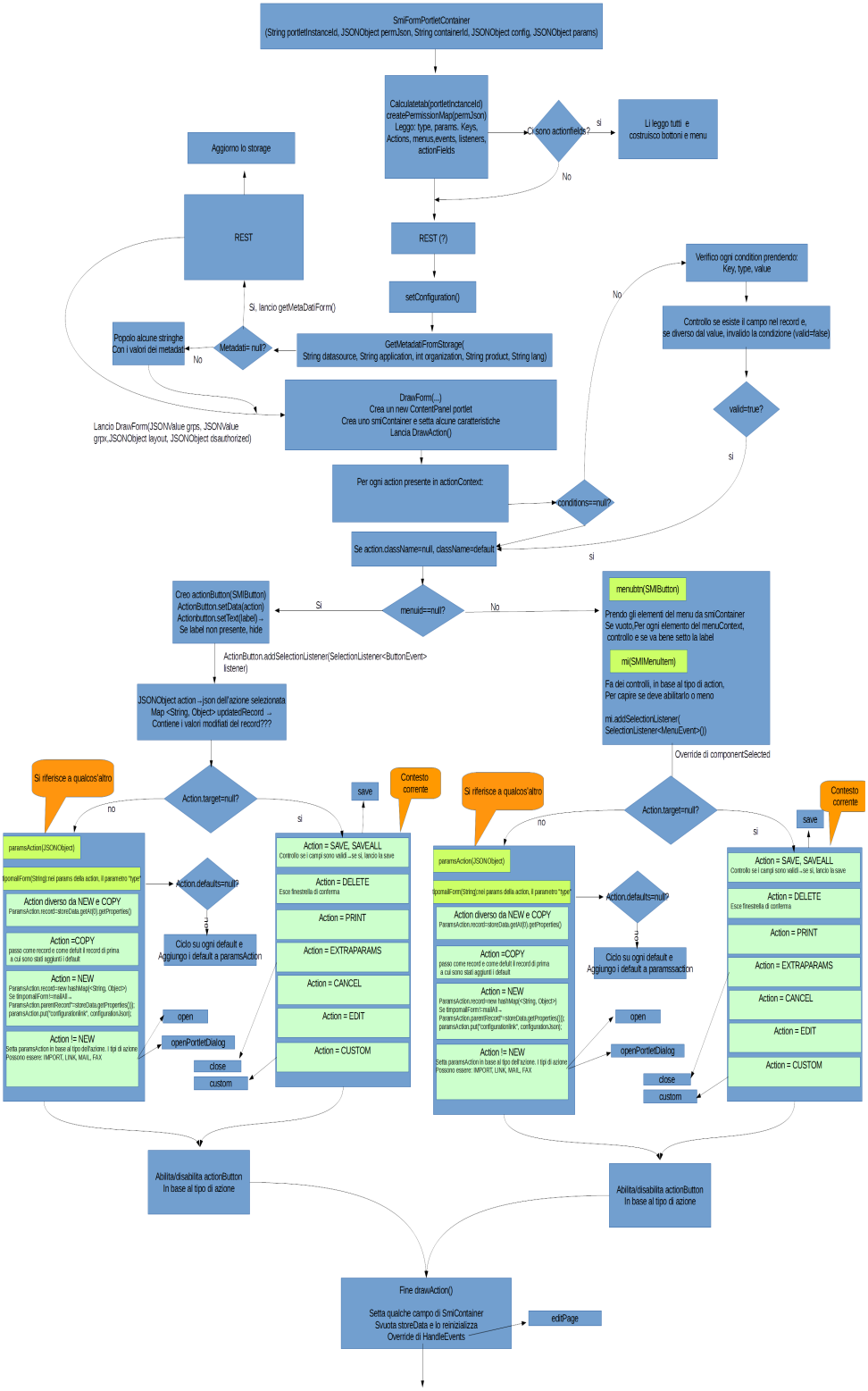


Figura 4.3: Diagramma di flusso della form preesistente- prima parte

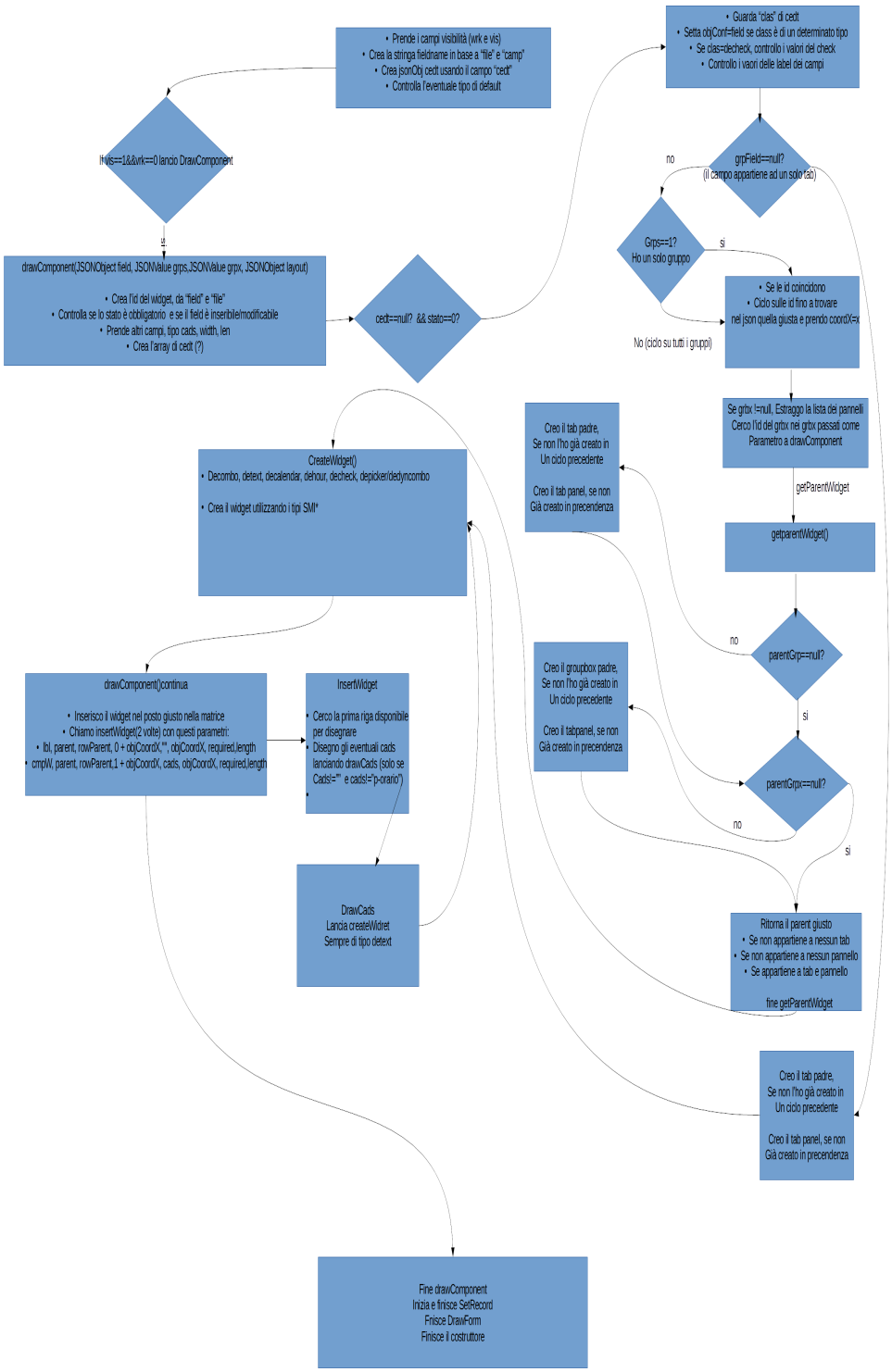


Figura 4.4: Diagramma di flusso della form preesistente - seconda parte

* il record dei dati (vuoto nel caso di una form per l'inserimento di dati).

Dalla configurazione si prelevano poi parametri sulla presenza o meno di chiavi per il recupero di un record in particolare, delle portlet eventualmente in ascolto di cambiamenti e parametri che indicano la presenza di bottoni, che vengono eventualmente costruiti immediatamente.

4.2.1.2 Recupero del datasource

Successivamente è necessario recuperare dal server il file JSON contenente l'insieme dei campi dati da visualizzare: viene dapprima controllato se esso è già disponibile nel localStorage e, nel caso esso non fosse già stato caricato in precedenza, viene richiesto al server Tomcat. La richiesa avviene attraverso delle chiamate REST:

4.2.1.3 Creazione delle azioni

Una volta ottenuto il datasource, è possibile passare alla costruzione delle parti che andranno ad interagire con l'utente:
Dapprima vengono lette dal datasource le azioni che la form può eseguire. Il flusso di esecuzione qui si dirama molto a seconda che l'azione abbia o meno una seconda portlet target, che sia contenta o meno in un menù e comunque in base al tipo di azione che si dovrà creare.
I tipi di azione sono i seguenti:

* **new**;
* **copy**;
* **import**;
* **link**;
* **mail**;
* **save**;
* **print**;
* **delete**;
* **custom**;

I metodi invocati sono diversi per ogni azione, e si occupano della gestione della funzionalità allorché l'utente prema il relativo pulsante.

4.2.1.4 Creazione dei fields

Infine viene eseguito un ciclo su tutti i campi dati rappresentati nel datasource. Questa è la parte più complessa del flusso in quanto ogni campo contiene avariate caratteristiche che è necessario andare a verificare per la corretta costruzione dello stesso. Esso è descritto nel datasource, con alcune variazioni (a volte piuttosto importanti) a seconda della tipolgia di campo, come riportato nella figura 4.5, che rappresenta i parametri relativi al campo "mail".
Vengono letti i campi relativi all'effettiva rappresentazione sullo schermo del campo, i

```
{
  "flle":0,
  "flds":"",
  "grbx":null,
  "fldr":"?",
  "stato":1,
  "soab":0,
  "flgt":0,
  "flne":0,
  "cadr":"",
  "cads":"",
  "flbt":0,
  "repl":0,
  "id":426311,
  "grp":9070,
  "camp":"MAIL",
  "tot":"00",
  "clbl":0,
  "dec":0,
  "flge":0,
  "file":"TLEADM",
  "flin":0,
  "rnd":0,
  "dft":"",
  "upab":1,
  "cls":0,
  "cedt":{
    "clas":"detext",
    "mask":"#email",
    "frmt":"",
    "up":0
  },
  "fleq":0,
  "tsep":0,
  "width":19.23,
  "align":0,
  "len":255,
  "inab":1,
  "vis":1,
  "sign":0,
  "t":"S",
  "klb":"email",
  "lbl":{
    "lung":"Email",
    "col1":"",
    "brev":"",
    "col2":""
  },
  "fllt":0,
  "flab":0,
  "seq":16,
  "wrk":0,
  "req":0,
  "fllk":0,
  "loadv":0,
  "pk":0
},
```


Figura 4.5: Esempio di un campo dati come rappresentato nel datasource

premessi di scrittura, l'eventuale valore di default, che viene inserito in un particolare tipo di record e vengono letti i valori, rappresentati in un punto differente del file, relativi al posizionamento del campo all'interno della form.
Vengono quindi create le tab ed i groupbox per contenere i campi.
Viene poi letta la tipologia di campo, che può essere

- **text**;
- **combobox**;
- **date**;
- **picker**;

Per ognuna di queste tipologie vengono chiamati dei metodi appropriati che costruiscono il campo: il più complesso è il campo picker, che tra i suoi parametri contiene le indicazioni per il recupero dei dati che saranno poi oggetto della ricerca da parte dell'utente.
Infine viene effettuata un'ulteriore chiamata REST per recuperare il record completo, tenendo anche conto dei valori di default recuperati dal ciclo appena effettuato.

Capitolo 5

Sviluppo del prototipo

Questo capitolo illustra il funzionamento delle tecnologie che ho utilizzato nello sviluppo del prototipo, la logica di realizzazione ed il risultato ottenuto

5.1 AngularJS

La libreria che ho scelto di utilizzare, AngularJS, nasce nel 2012 nei laboratori di Google.
Esso potenzia l'aspetto dichiarativo del linguaggio HypetText Markup Language (HTML), offrendo al contempo tutti gli strumenti necessari alla realizzazione di SPA.

5.1.1 Direttive e controller

Una delle peculiarità di AngularJS è che consente di aggiungere al codice HTML nuovi attributi, denominati **direttive**.
Tali direttive possono essere definite dall'utente, che comunque ha a disposizione anche un buon numero di direttive standard fornite dal *framework*. Queste strutture consentono al compilatore HTML di AngularJS (attivate dal comando `$compile()`, di creare un legame tra l'attributo in cui sono inserite ed il codice JavaScript che va ad eseguire il corpo della direttiva, tipicamente una funzione.
La possibilità di definire direttive customizzate estende di molto le funzionalità di base, lasciando alla fantasia del programmatore la creazione di strutture pensate appositamente per l'applicazione in sviluppo, ed consente di utilizzare quasi esclusivamente l'HTML per la creazione delle interfacce grafiche.
L'invocazione delle direttive può avvenire utilizzando il nome della direttiva come tag(`<direttiva></direttiva>`), oppure come attributo ad un tag regolare HTML(`<div direttiva></div>`).
Nel mio progetto ho utilizzato esclusivamente direttive standard di AngularJS, che ne ha create di specifiche per la creazione di form.

L'altro importante componente di AngularJS è il controller. Esso viene invocato tramite la direttiva`ng-controller` ed il suo scope concerne tutto ciò che si trova all'interno dei tag nei quali è definito. Un controller, grazie al concetto di dependency injection, può utilizzare le funzionalità definite in moduli esterni che vengono inseriti come dipendenza all'interno del controller.

É possibile importare, oltre ad interi metodi, anche singoli oggetti: il più utilizzato è sicuramente l'oggetto di sistema `$scope`. Quest'ultimo è un semplice oggetto JavaScript, per cui è possibile aggiungervi nuovi attributi e metodi semplicemente digitando `$scope .nuovoattributo = valore;`. Lo scope, assieme alle funzionalità definite nei moduli dichiarati come dipendenza, funge da *model* nel patter MVC descritto in A.2.
Gli attributi dello `$scope` definiti in un controller sono condivisi all'interno della view (quindi nel codice HTML) unicamente nell'area in cui è definito il controller. AngularJS implementa infatti il concetto di **two-ways data binding**: il data binding è il meccanismo di sincronizzazione automatica dei dati tra il modello e la view. La maggior parte dei sistemi di template supporta il data binding in una sola direzione, tipicamente dal modello dei dati verso la view. Questo vuol dire che i dati del modello vengono combinati con il template HTML per generare la view visibile all'utente. Se però il modello viene modificato, le modifiche non si riflettono automaticamente sulla view. Analogamente, se l'utente modifica la view, queste modifiche non vengono automaticamente riportate sul modello dei dati. Per sincronizzare view e modello occorre in genere scrivere del codice che lo faccia.
Il data binding di AngularJS invece è bidirezionale, cioè ogni modifica al modello dei dati si riflette automaticamente sulla view e ogni modifica alla view viene riportata sul modello dei dati.
Per poter utilizzare un attributo dello `$scope` nella view è sufficiente indicarlo, all'interno di un tag di testo, con la seguente sintassi: `ng-model = nuovoattributo`. Questa sintassi attiva il cosiddetto *ciclo di digest*.

5.1.2 Ciclo di digest

Il ciclo di digest è ciò che permette di avere il two ways data binding. Ogni volta che viene usata la sintassi `ng-model = nuovoattributo`, viene automaticamente creato un **watch** che ascolta i cambiamenti del valore della variabile a cui è associato, nel nostro caso la variabile "nuovoattributo".
Quando viene cambiato il valore di una variabile, il *ciclo di digest* controlla tutti i watch presenti nell'applicazione e, nel caso verifichi il cambiamento di alcune variabili rispetto al ciclo precedente, esegue delle operazioni al fine di aggiornare il DOM nei punti in cui essa è utilizzata.
Nel caso di variabili che vengono modificate all'interno di un watch, ad esempio perchè il valore di una variabile dipende dal valore di un'altra, il ciclo viene ripetuto finchè non si presentano più modifiche.
Questo avviene in maniera del tutto automatica e trasparente sia all'utente che allo sviluppatore, che può però creare dei watch attraverso l'attributo di sistema `$watch( attribute, function([data]){})`.
Questo nel caso ci siano delle dipendenze da rispettare, ma non ci sia alcun binding con la view. Nel mio progetto, ad esempio, utilizzo questa funzionalità per poter gestire le chiamate asincrone ai servizi REST, sfruttando il cambio di valore della variabile da recuperare.

5.1.3 Eventi

Gli attributi assegnati allo `$scope` non sono comunque strettamente ristretti al controller in cui sono definiti: è infatti possibile il passaggio degli attributi tra controller annidati come avviene in un comune contesto di ereditarietà: un controller figlio eredita tutti gli attributi definiti nel controller padre.

Nel contesto di una gerarchia di controller è anche possibile sollevare degli eventi sia un controller figlio a quello padre, utilizzando `$emit('event name', [data])`, che dal padre verso i figli, con `$broadcast('event name', [data])`.
Gli eventi vengono poi raccolti da `$on('event name', function([data]){...})`, che definisce il comportamento da adottare una volta raccolto l'evento.

5.2 Codifica

La creazione della componente form utilizzando angularJS è iniziata cercando di aderire ai principî dell'incapsulamento e di *information hiding* tipici della programmazione ad oggetti.
Questi principî, che sanciscono la necessità di nascondere le proprietà intrinseche di un oggetto rispetto alla sua rappresentazione, sono qui utilizzati per separare i compiti tra i diversi elementi della form tramite l'utilizzo di vari controller, ognuno destinato alla gestione di una porzione specifica della form stessa.
Vengono quindi utilizzati dei controller generici, che vengono instanziati una sola volta:

* **FormController**: un controller generale, che recupera le informazioni utili alla form nel suo complesso, come record e datasource;
* **ActionsController**: un controller che gestisce la creazione delle azioni e il loro comportamento; indipendente dall'altro;
* **FieldsController**: un controller che si occupa della creazione dei campi dati che costituiscono il corpo della form.

e specifici in cui, per ogni button del tipo interessato, viene instanziato un nuovo controller:

* controller figli di ActionsController:
  - **ActionButtonController**: un controller per gestire il singolo button d'azione;
* controller figli di FieldsController:
  - **PickersController**: un controller per il controllo dei campi di tipo picker;
  - **DetextController**: un controller per il controllo dei campi di tipo text;
  - **GeneralFieldController**: un controller per il controllo dei campi diversi dal tipo picker e text.
    Quest'ultimo, in particolare, è stato pensato in ottica manutentiva: se è necessario aggiungere delle proprietà ad un campo generico, è possibile farlo andando a modificare questo controller.

5.2.1 Ciclo di creazione

5.2.1.1 Recupero dei dati

La creazione di una form avviene seguendo parzialmente il flusso di creazione utilizzato dalla precedente implementazione e visto nel precedente capitolo, ma ho comunque

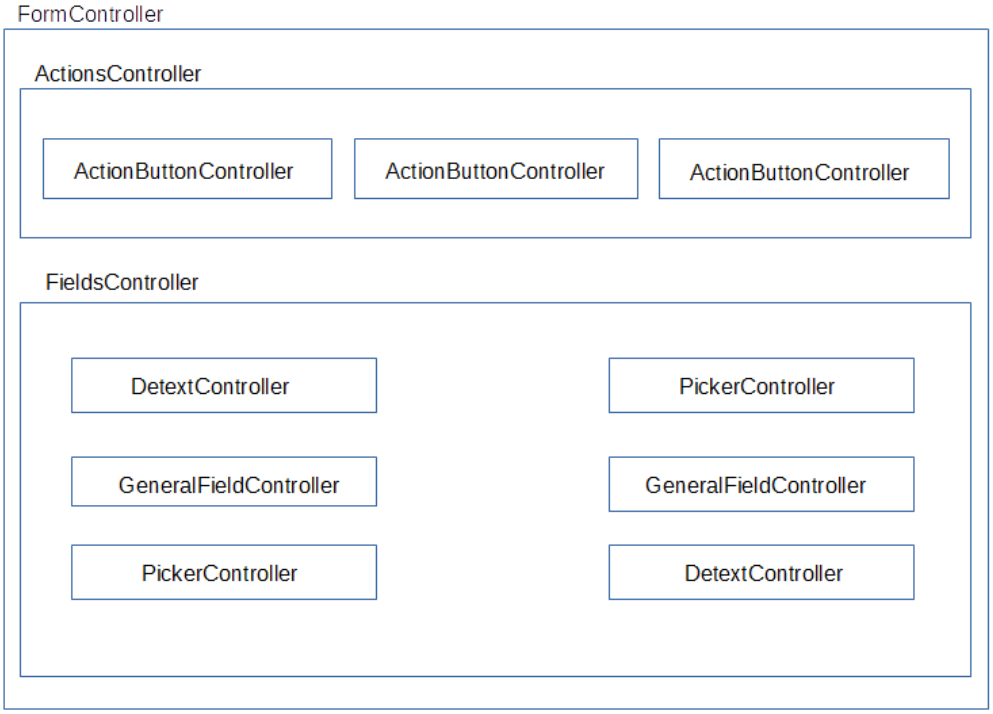


Figura 5.1: gerarchia di controller in una form

cercato di utilizzare meno risorse possibili per aumentare la velocità di creazione e migliorare l'esperienza utente. La velocità di creazione è inoltre aumentata dalla scelta di caricare dinamicamente alcune componenti della form, come i tab, solamente se l'utente vi interagisce realmente.
La direttiva `ng-app`, che inizializza l'applicazione AngularJS, può essere utilizzato solamente una volta all'interno di un documento HTML: in JGalileo CRM era già utilizzata per inizializzare un modulo comune, chiamato SMICommonModule, che conteneva anche alcuni servizi REST per il reperimento di risorse quali datasource e record. Tali servizi sono stati successivemente integrati con nuove funzionalità. É stato quindi sufficiente importare il modulo come dipendenza del controller principale della form per poter correttamente inizializzare il tutto.
Al controller principale vengono passati alcuni parametri di configurazione, con cui viene recuperato il datasource corretto tramite un servizio REST.
Tramite il meccanismo di `$watch` descritto in precedenza, al ritorno del datasource viene chiamato il metodo `setValues()` contenuto in FieldsController, che si occupa di creare degli array contenti le azioni, i tab della form e, per ogni tab, i groupbox che lo conpongono. Successivamente vengono letti tutti i campi dati descritti nel datasource e posti negli array, per facilitarne la rappresentazione.
AngularJS mette infatti a disposizione la direttiva `ng-repeat`, che consente di iterare lungo una lista od un array di elementi.
Al termine di queste operazioni, vengono inizializzati nel record i campi contententi i valori di default. Questo record viene poi utilizzato per recuperare dal server il record completo, composto dai campi di default precedentemente settati e da alcuni altri

```
setFields: function(grpsId){ //imposta l'array di fields da visualizzare per il tab selezionato
    var self = this;
    var validField = false; //serve per capire se effettivamente ci sia almeno un campo valido
    var tab = Object.assign({}, $filter('filter')(this.tabs.grp, grpsId));
    if(tab.hasOwnProperty("0"))//se c'erano più tab, resta l'indice da togliere
        tab = tab[0];
    this.tabFields = [];
    this.tabGrbxFields = [];
    if(tab.hasOwnProperty("fld")){//solo se la tab ha qualche campo dati da mostrare
        for(var i = 0; i<this.fields.length; i++){ // ciclo sui campi dati propri della tab selezionata
                if(this.fields[i].grp == tab.id){//se il field appartiene al gruppo
                    var element = Object.assign({}, this.fields[i]);
                    if(element.grbx == null){  //controllo che il campo non appartenga a nessun grupBox
                        this.tabFields.push(element);
                        if(element.vis == "1" && element.wrk == "0")
                            validField = true;
                    }
                    else{//grbx != null
                        for(var k = 0; k < this.grx.length; k++){//ciclo sui vari groupbox
                            if(element.grbx == this.grx[k].id){//cerco il groupbox di appartenenza
                                var found = false;
                                for(var z = 0; z<this.tabGrbxFields.length && !found; z++){//cerco se il groupbox è già presente
                                    if(this.tabGrbxFields[z].id == element.grbx){ //c'è e quindi è anche già presente l'array fld)
                                        this.tabGrbxFields[z].fld.push(element);
                                        if(element.vis == "1" && element.wrk == "0")
                                            validField = true;
                                        found = true;
                                    }
                                }
                                if(!found){ // non c'è, lo creo, creo il campo fld e inserisco il campo
                                    var grbx = Object.assign({}, this.grx[k]);//copio l'elemento per non fare side effect su grx
                                    grbx["fld"] = [];
                                    grbx.fld.push(element);
                                    this.tabGrbxFields.push(grbx);
                                }
                            }
                        }
                    }
                }
    }
```

Figura 5.2: setFields(), che inserisce i valori negli array

campi che vengono inseriti dal server.
Una volta ottenuto il record completo, i valori che sono stati recuperati vengono visualizzati direttamente nella view grazie al *two ways data binding*.

5.2.1.2 Creazione della view

Il file che costituisce la view è unico e rispecchia, utilizzando tag innestati, la gerarchia dei controller vista nell'immagine 5.1

Vengono creati i buttons per le azioni e dei buttons che poi andranno a costituire i tab di cui è composta la form.
In seguito il file è essenzialmente diviso in due parti: la prima, che tramite l'esecuzione di un ciclo sull'array corretto si occupa della creazione dei campi dati che non appartengono ad alcun groupbox, e la seconda, che utilizzando due cicli gestisce la creazione delle groupbox e dei campi contenuti in ciascuna di esse. Nella view sono poi inserite, come attributo di tag, svariate direttive, come il condizionale `ng-if`, `ng-readonly`, `ng-show` che servono alla corretta visualizzazione di un determinato campo dati sulla base delle proprietà descritte per quel campo nel datasource recuperato.
Ogni campo della form utilizza la direttiva `ng-model`, in maniera che venga attivato il ciclo di digest per ogni campo, e che il valore descritto da esso venga automaticamente salvato nel record.
La sintassi `{{variable}}` viene invece utilizzata per rappresentare a schermo il valore della variabile `variable`. Nel mio progetto queta direttiva è utilizzata per visualizzare il nome del campo.

```
<div data-ng-controller= "ActionsController" data-block-ui="blockUI-{{form.portletInstanceId}}" data-ng-cloak data-ng-init="actions.checkValidity()">
    <div>
        <button data-ng-controller= "ActionButtonController" ng-repeat="rows in actions.actions" ng-click="actions.clicked(rows, CRMform.$valid)" ng-hide="rows.hideActio
            ng-disabled={{actionButton.disabled}}>{{rows.label}}</button>
    </div>
<div data-ng-controller= "FieldsController" data-block-ui="blockUI-{{form.portletInstanceId}}" data-ng-cloak>
    <div>
        <div>
            <button ng-if="rows.fld" ng-repeat="rows in fields.tabs.grp" ng-click="fields.setFields(rows.id)" >{{rows.lbl.lung}}</button>
        </div>
    </div>
            <div ng-repeat="item in fields.tabFields">
            <!--  <div ng-switch-when="0">-->
                    <div ng-controller ="DetextController" ng-if="item.cedt.clas == 'detext'" ng-show="item.wrk =='0' && item.vis=='1'" ng-init="detext.setField(item);">
                        <input name={{item.lbl.lung}} type={{detext.type}} step={{detext.step}} ng-model="form.record[item.camp.toLowerCase()]" ng-required="item.req" ng
                    </div>
                    <div data-ng-controller="PickersController" ng-if=" (item.cedt.clas == 'depicker' || item.cedt.clas == 'dedyncombo')"  ng-show="item.wrk =='0' && ite
                    <a ng-if="pickers.hasUrl == true" herf="#" ng-click="pickers.openTab(item)">{{item.lbl.lung}}</a>
                    <span ng-if="pickers.hasUrl == false">{{item.lbl.lung}}</span>
                        <input type="text" ng-model="pickers.fld.inputQuery" ng-readonly="!item.upab">
                        <select  name={{item.lbl.lung}}  ng-model="form.record[item.camp.toLowerCase()]" ng-required="item.req" ng-readonly="!item.upab"
                            ng-options="field[pickers.fld.mapValues[0].da.toLowerCase()] as field[pickers.fld.searchField] for field in pickers.fld.dati.records" ng-read
                                <!--  <option ng-repeat="keys in pickers.fld.dati.records" value="{{keys[pickers.fld.searchField]}}"  ng-selected="form.record[item.camp.
                        </select>
                    </div>
                    <div data-ng-controller="GeneralFieldController" ng-if="item.cedt.clas == 'decheck'"  ng-show="item.wrk =='0' && item.vis=='1'">{{item.lbl.lung}}
                        <input name={{item.lbl.lung}}  type="checkbox" ng-model="form.record[item.camp.toLowerCase()]" ng-required="item.req" ng-readonly="!item.upab" ng
                    </div>
                    <div data-ng-controller="GeneralFieldController" ng-if="item.cedt.clas == 'decombo'"  ng-show="item.wrk =='0' && item.vis=='1'">{{item.lbl.lung}}
                        <select name={{item.lbl.lung}}  ng-model="form.record[item.camp.toLowerCase()]" ng-required="item.req" ng-readonly="!item.upab">
                            <option ng-repeat="keys in item.cedt.kvs.kv" value="{{keys.val}}" ng-selected="form.record[item.camp.toLowerCase()]== keys.val || $index=='0'
```

Figura 5.3: parte della view

5.2.2 Validazione dei campi inseriti

AngularJS mette a disposizione molte funzionalità dedicate alla realizzazione di form. Una di queste è la validazione della form a livello client. AngularJS, in maniera del tutto autmatica, valuta se i dati inseriti siano del tipo dichiarato per ogni campo e se vengono rispettate le obbligatorietà. Un campo viene dichiarato obbligatorio tramite la direttiva `ng-required`
La validazione dell'intera form è possibile tramite l'oggetto di sistema `$valid`, una variabile booleana che assume il valore `true` solamente nel caso tutti i campi dati rispettino i vincoli descritti sopra.

Capitolo 6

Conclusioni

6.1 Obiettivi personali

Nell'intraprendere il mio percorso di stage, oltre agli obiettivi stabiliti nel Piano di Lavoro, avevo anche degli obiettivi personali da raggiungere, di seguito elencati:

* Conoscere uno dei rami lavorativi in cui è possibile inserirsi al termine del percorso di studi;
* Conoscere tecnologie nuove e non affrontate durante il percorso di studi;
* Confrontarmi con software di larga scala;
* Confrontarmi con altre persone già inserite nel mondo del lavoro;
* Migliorarmi grazie all'aiuto del tem di lavoro;
* Migliorarmi nella gestione di tempi/risorse nel contesto di un lavoro assegnatomi.

Appendice A

Appendice A

A.1 Funzionamento di Hibernate

Come avviene la conversione delle tabelle in classi Java?
Hibernate deve conoscere la configurazione del database e per farlo necessita di un file, estesi generalmente in **.cfg.xml**. Al framework è inoltre indispensabile fornire dei files

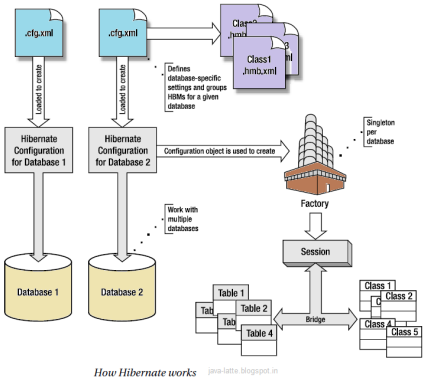


Figura A.1: Schema del funzionamento di Hibernate

di mapping, uno per ogni tabella e generalmente estesi con i suffissi **.hmb.xml**, che contengono le informazioni riguardanti le colonne della singola tabella da trasformare in un oggetto Java.

Figura A.2: XML di un'entità Hibernate di JGalileo CRM

Questi files vengono poi utilizzati per creare una *SessionFactory* globale e thread-safe che funge da *gateway* per l'interrogazione del database.
I vantaggi dell'utilizzo della tecnica di programmazione ORM (Object-Relational Mapping) sono molteplici ed in particolare consentono:

* **Disaccoppiamento dal DBMS**;
* **Elevata portabilità**;
* **Drastica riduzione del codice sorgente**, a causa dei semplici comandi che mascherano complesse istruzioni;
* **Elevata modularità**.

A.2 Model View Controller

Il pattern *MVC, Model-View-Controller*, è un design pattern, cioè uno schema di progettazione che costituisce una soluzione progettuale ad un problema ricorrente.
É uno dei pattern più famosi ed utilizzati per la separazione tra la logica di presentazione e la logica di business nei sistemi software, ed in particolare nelle applicazioni web.
Come evidenziato dalla figura A.3, vi sono 3 attori principali:

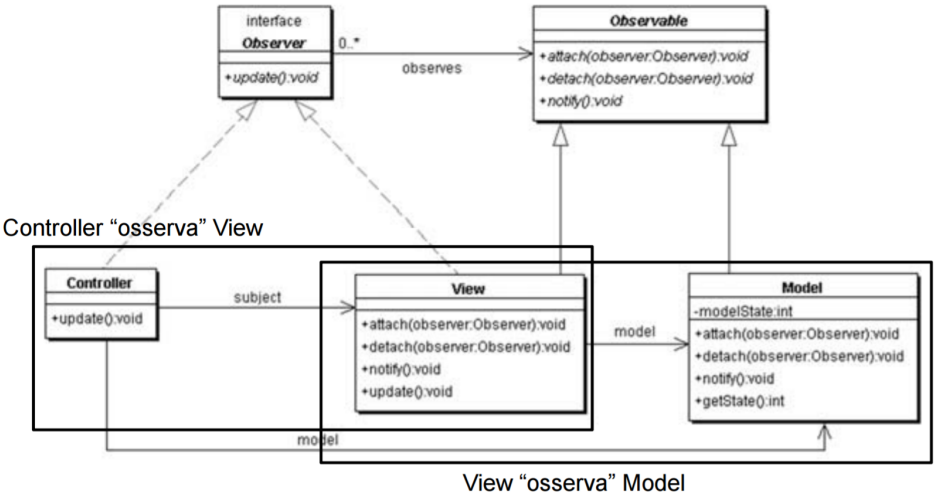


Figura A.3: Design pattern MVC

- **Model**: rappresenta la cosiddetta *logica di business*, cioè l'insieme di dati e metodi per eseguire operazioni;
- **View**rappresenta come i dati vengono visualizzati nell'interfaccia utente, cioè la *logica di presentazione*;
- **Controller**:si pone come intermediario tra view e model;

Il controller, che osserva la view, riceve da quest'ultima le richieste di elaborazione e, una volta che il model restituisce i dati, si occupa di inviarli alla view.
Questo design pattern ha molti lati positivi, tra i quali la divisione dei compiti ed il riuso di codice.

Glossario

DOM l'acronimo *DOM, Document Object Model* (ing. modello ad oggetti del documento) è una forma di rappresentazione di documenti, rappresentato come modello orientato agli oggetti.
Tipicamente, nella rappresentazione di un documento HTML, questo modello corrisponde ad un albero dove la radice è l'elemento più generale del documento (il tag <html>), e le foglie i tag più annidati.. 10

MIDDLEWARE con il termine *middleware* si intende un insieme di programmi informatici che fungono da intermediari tra diverse apllicazioni e componenti software.. 9

PHP l'acronimo ricorsivo *PHP, PHP: Hypertext Preprocessor* (ing. PHP: preprocessore di ipertesti) è un linguaggio open-source di scripting concepito per la programmazione di pagine web dinamiche, anche se attualmente il suo uso più comune risiede nelle applciazioni web lato server.. 8

PORTLET una portlet è un modulo web riutilizzabile all'interno di portale web. Solitamente, infatti, una pagina di un portale è costituito da finestre il cui contenuto è diverso a seconda della portlet che andiamo ad inserire. Ad esempio possono esserci portlet per le previsioni meteo, per la geolocalizzazione, per l'inserimento di dati,..
Queste portlet, in quanto finestre, sono adattabili alle esigenze del singolo utente e possono quindi esse chiuse, allargate/ridotte, spostate.. 7, 11–13

SCRIPT uno *script*, nel linguaggio informatico, è un particolare tipo di programma. Si differenzia infatti dai normali programmi da questi fattori:

* Bassa complessità;
* Uso di un linguaggio interpretato;
* Mancanza di un'interfaccia grafica.

Solitamente uno script è quindi una piccola funzionalità che risolve un prolblema specifico, inserita all'interno di un contesto più grande.. 9

SPA in informatica, per *SPA, Single Page Application* (ing. applicazioni a pagina singola) s'intende un'applicazione web che sia, a livello di esperienza utente, più simili alle applicazioni desktop. Infatti in una *SPA* il codice necessario viene caricato dinamicamente all'occorrenza, in questo modo la pagina non si ricaricherà in nessun punto del del processo. Spesso si rende necessaria una comunicazione dinamica con il web server.. 10, 11