UNIVERSITÀ DEGLI STUDI DI PARMA

DIPARTIMENTO DI MATEMATICA E INFORMATICA

Corso di Laurea in Informatica

Tesi di Laurea

Progettazione e Realizzazione di un'Applicazione di Gestione Eventi Integrata con il CRM Podio

Relatore:
Prof. Federico Bergenti

Candidato:
Daniel Lombardo

Anno Accademico 2014/2015

A Claudio, Maria
ed al Potente Thor
che hanno reso
tutto ció possibile.

Indice

Prefazione

Il lavoro della presente tesi nasce entro la collaborazione dell'Azienda Aicod (www.aicod.it) presso la quale ho svolto un tirocinio di 2 mesi e mezzo. Aicod è nata nel 2007 e prima di tutto studia i mercati per costruire strategie comunicative uniche che diano valore a prodotti e servizi, e che li distingua.
Prima di prender in carico un progetto Aicod analizza la situazione di partenza e definisce insieme al cliente che cosa può essere più efficace per loro.
L'azienda realizza applicativi per ottenere un gran quantità di risultati: dai moduli per inviare mail ai complessi datawarehouse ed in più offre un sistema di hosting per alcuni clienti.
Durante le fasi di Analisi, Progettazione e Realizzazione dell'applicazione si è utilizzata una metodologia di Project Management chiamata *agile*. Questa metodologia di sviluppo si contrappone al modello a cascata e altri processi software tradizionali, proponendo un approccio meno strutturato e focalizzato sull'obiettivo di consegnare al cliente, in tempo brevi e frequentemente software funzionante e di qualità.
Anche se il risultato di ogni singola iterazione non ha sufficienti funzionalità da essere considerato completo deve essere rilasciato e nel susseguirsi delle iterazioni, deve avvicinarsi sempre di più alle richieste del cliente. Alla fine di ogni iterazione è opportuno rivalutare le priorità di progetto. Tra le pratiche di sviluppo *agile* ci sono la formazione di team di sviluppo piccoli, cross-funzionali ed auto-organizzati, lo sviluppo iterativo e incrementale, la pianificazione adattiva ed il coinvolgimento diretto e continuo del cliente nel processo di sviluppo.

I principi fondamentali dello sviluppo *agile* sono:

- Le persone e le interazioni sono più importanti dei processi e degli strumenti ossia le relazioni e la comunicazione tra gli attori di un progetto software sono la miglior risorsa del progetto.
- É più importante avere software funzionante che documentazione bisogna infatti rilasciare nuove versioni del software ad intervalli frequenti e bisogna mantenere il codice semplice e avanzato tecnicamente riducendo la documentazione al minimo indispensabile.
- Bisogna collaborare con i clienti oltre che rispettare il contratto la collaborazione diretta offre risultati migliori dei rapporti contrattuali.
- Bisogna essere pronti a rispondere ai cambiamenti oltre che aderire alla pianificazione quindi il team di sviluppo dovrebbe essere pronto in ogni momento a modificare le priorità di lavoro nel rispetto dell'obiettivo finale.

Per far ciò è stato fondamentale l'utilizzo di strumenti di comunicazione e coordinamento in quanto non è stato sempre possibile avere dei meeting con il committente su base regolare.
Durante la progettazione e la realizzazione del progetto si sono rivelate fondamentali le nozioni apprese durante il mio percorso personale di studio in Informatica presso l'Università degli studi di Parma.

La tesi sara strutturata nel seguente modo:

- Il *Primo capitolo* parlerà degli obiettivi del progetto soffermandosi in particolare sull'Analisi dei Requisiti, il Metodo di Sviluppo e le entità da gestire. Inoltre ci sarà l'elenco degli attori ed Use-Case-Diagram e le relative entità in gioco. Verrà illustrata la situazione pre-progetto e mostrerà la soluzione proposta al cliente.

- Nel *Secondo capitolo* spiegherà le specifiche del software, il quale sono state decise in base alle esigenze del commitente ed al budget concordato, `MVC` model il workflow e verranno citate alcuni dei meccanismi di sicurezza utilizzati dall' applicativo. In più verrà spiegato come il progetto è stato strutturato, le interfacce utente e le numerose tecnologie utilizzate.

- Nel *Terzo* ed ultimo capitolo vi saranno le conclusioni.

Capitolo 1

Obiettivi del Progetto di Tesi ed Analisi

1.1 Analisi dei Requisiti

In Ingegneria del Software, l'analisi dei requisiti è un'attività preliminare allo sviluppo di un sistema software, il cui scopo è quello di definire le funzionalità che il nuovo prodotto deve offrire, ovvero i requisiti che devono essere soddisfatti dal software sviluppato per soddisfare le esigenze del commitente. L'analisi dei requisiti è una fase presente essenzialmente in tutti i modelli di ciclo di vita del software, pur con diverse enfasi e diverse connotazioni.

Nel tradizionale modello a cascata, l'analisi dei requisiti è la prima fase del processo di sviluppo, e deve concludersi con la stesura di una dettagliata specifica dei requisiti che descrive le funzionalità del nuovo software nella loro interezza ed tale specifica guida le fasi successive di sviluppo, che complessivamente sono volte a realizzare quanto previsto da tale specifica.
Data una specifica, esistono molti modi per realizzarla. Naturalmente la scelta fra le diverse possibiltà non è arbitraria, ma è guidata sia da vincoli di carattere economico, sia dalla necessità di conseguire un'adeguata qualità del prodotto o del progetto stesso.

Per qualità del progetto si intendono quelle caratteristiche che, pur non essendo visibili all'utente, determinano la qualità del prodotto e quelle caratteristiche che offrono un vantaggio economico al produttore del software, rendendo più efficace il processo di sviluppo.
Fra le caratteristiche che determinano la qualità del prodotto o del progetto, citiamo l'*affidabilità*, la *modificabilità*, la *comprensibilità* e la *riusabilità*.

L'esperienza accumulata finora dimostra che queste proprietà dipendono fortemente da un'altra, la *modularità*. Un sistema è definito modulare se è composto da un certo numero di sottosistemi, ciascuno dei quali svolge un compito ben definito e dipende dagli altri in modo semplice.

É chiaro che un sistema così strutturato è più comprensibile di uno la cui struttura venga oscurata dalla mancanza di una chiara suddivisione dei compiti fra i suoi componenti, e dalla complessità delle dipendenze reciproche.

La comprensibilità a sua volta, insieme alla semplicità delle interdipendenze, rende il sistema più facile da verificare, e quindi più affidabile.

Inoltre, il fatto che ciascun sottosistema sia il più possibile indipendente dagli altri ne rende più facili la modifica e il riuso. Prima di proseguire, è dunque doveroso fare un accenno a cosa sono queste caratteristiche sopracitate:

- *Correttezza*: Un programma o comunque un software si dice corretto se si comporta esattamente secondo quanto è previsto dalla sua specifica dei requisiti.

- *Affidabilità*: Un sistema si dice affidabile quando non esistono o comunque si presentano molto raramente dei malfunzionamenti.

- *Robustezza*: In termini molto generali un software si dice robusto quando si comporta in maniera ragionevole in situazioni impreviste che vanno dal inserimento di input scorretti a fallimenti sia del software che dell'hardware.

- *Efficienza*: un sistema si dice efficiente se è poco esoso sia in termini di memoria che tempo CPU.

Alla base della costruzione di un qualsiasi programma vi è un particolare problema elaborativo da risolvere.
Questo particolare problema però dovrà essere analizzato prima di partire a scrivere il Software atto a risolverlo.
Individuata cioè l'esigenza servirà esplicitarle e verificarne le condizioni di realizzabilità
Questa macrofase si può suddividere ulteriormente in due:

- Studio di fattibilità, nella quale si chiariscono le istanze dell'utente (ovvero della persona che intende risolvere un problema e intende farsi costruire un sistema di elaborazione).
- Definizione delle specifiche funzionali, nella quale si precisano le particolari funzioni che debbono essere soddisfatte e i limiti entro i quali si possono realizzare, (tutto ciò derivante dalla caratteristica del problema).

La prima, e quella che ci interessa maggiormente rivedere, delle due fasi di questa macrofase è lo studio di fattibilità, che serve per definire se il progetto è realizzabile o meno. Questa fase prende in esame:

- Le richieste dell'utente.
- Il contesto ambientale in cui verrà calato il prodotto finito (settore, ufficio,ecc).
- Le risorse fisiche disponibili ovvero personale dedicato al progetto, hardware esistente o acquistabile, gli uffici a cui fare riferimento per avviare una ricerca dettagliata delle richieste, le capacità professionali del personale, ecc.
- Le risorse economiche messe a disposizione dal committente che è il nostro l'cliente.
- Un'indagine sui progetti software esistenti che risolvono in parte o completamente il problema.

1.2 Entita da gestire

Le entità, rappresentano classi di oggetti fatti, cose, persone, che hanno proprietà comuni ed esistenza autonoma ai fini dell'applicazione di interesse. Un'occorrenza di un'entità è un oggetto o istanza della classe che l'entità rappresenta. Non si parla qui del valore che identifica l'oggetto ma dell'oggetto stesso. Un'interessante conseguenza di questo fatto è che un'occorrenza di entità ha un'esistenza indipendente dalle proprietà ad essa associate.

1.2.1 I Lead e le Aziende

I *Lead* non sono altro che i contatti delle persone che ha a disposizione Fiere di Parma. Tutti i dati anagrafici delle persone e delle numerose aziende al momento sono esclusivamente salvati dentro fogli di calcolo e dovranno essere importate dentro il Database di *Podio (www.podio.com)*.
Eseguire questa operazione manualmente sarebbe troppo oneroso in quanto nell'anagrafica tenuta da Fiere di Parma ci sono migliaia e migliaia di record, dunque proprio questo, si è utilizzato il modulo di import dei dati integrato in *Podio* che è totalmente compatibile con i fogli di calcolo.
Verrà dunque eseguita soltanto una mappatura dei nomi dei campi utilizzata grazie alla comoda interfaccia che *Podio* offre.

1.2.2 Gli Hotel e le Room

Alla maggioranza dei *Lead* invitati negli eventi di Fiere di Parma, viene offerto un pacchetto che comprende il pernottamento presso gli hotel convenzionati con Fiere di Parma. Dunque questi Hotel vanno mappati all'interno di Podio e proprio per questo abbiamo creato due app distinte chiamate *Hotel* e *Room* ma collegate di fatto tramite un vero e proprio modello entità-relazione. Per gli Hotel si tratta di ubicazione, date disponibilità ad accettare ospiti per gli eventi organizzati e numero di stanze disponibili e per le Room si tratta di tipologia di stanza Hotel di appartenenza e relativa disponibilità.

1.2.3 Campagna

Ad ogni inzio evento, Fiere di Parma dovrà creare nell'interfaccia di *Podio* un nuovo WorkSpace.
Questo Workspace è soprannominato Campagna e prenderà il nome dell'evento specifico organizzato (e.g Mercante in Fiera, Cibus, ecc) ed includerà tutte le app create al fine di importare i dati dei lead (che a questo punto diventano *Incoming*, in quanto sono i *Lead* invitati ad uno specifico evento) che saranno invitati.
Una Campagna, è caratterizzata da 3 fasi:

- *Inizio campagna* si procede al popolamento del DB della campagna prendendo i dati dal DB delle aziende e dal DB dei Lead
- *Gestione & Mantenimento* si organizzano e gestiscono gli incoming
- *Conclusione & Chiusura* a campagna terminata, viene impostato un flag che indica la chiusura.

1.3 Attori e Use Case Diagram

Gli attori secondo la terminologie di UML (Unified Modeling Language, www.uml.org), sono le categorie di entità esterne, tipicamente persone,che dovranno intereagire con il sistema.
Per il progetto attuale abbiamo due attori:

- L'utente che riceve il form con l'invito all'evento e che lo compilerà con i propri dati.
- L'utente che lavora su *Podio* e che interagisce con i dati Ricevuti e che utilizzerà il Rooming.

L'Use Case Diagram in UML sono i diagrammi dedicati alla descrizione delle funzioni o servizi offerti da un sistema, così come sono percepiti ed utilizzati dagli attori che interagiscono con il sistema stesso.
Ecco dunque un use-case-diagram per questo progetto:

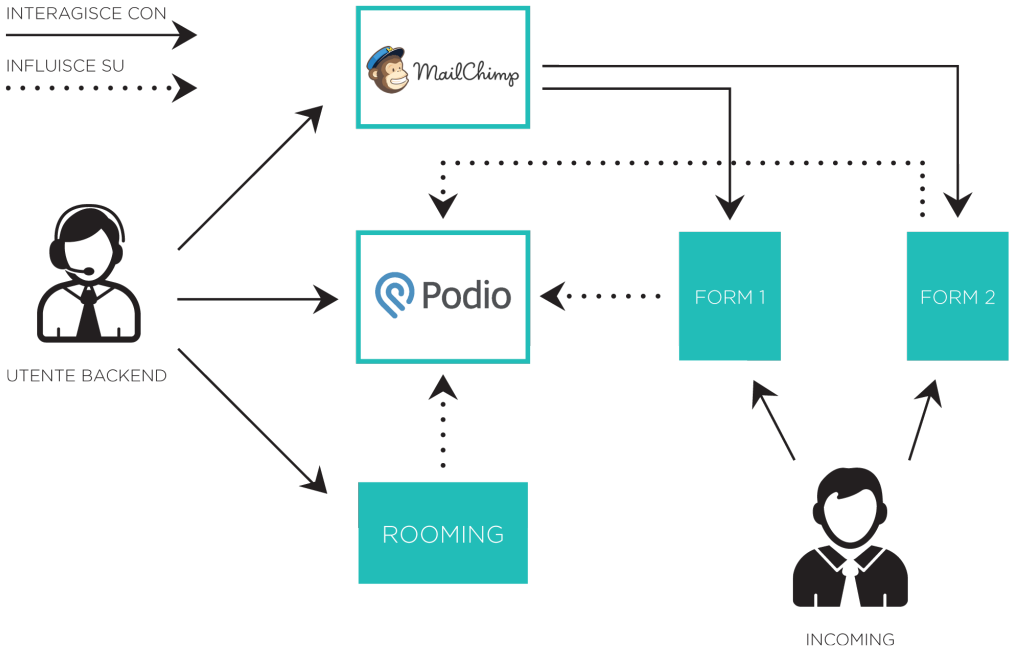

Figura 1.1: Use Case Diagram

Analizziamo ora le sue componenti:

- L'utente Back-End ha accesso a *Podio* e alla sua interfaccia, ha accesso al applicazione del rooming la quale andrà a scrivere i dati direttamente su Podio ed infine avrà accesso a *MailChimp* che utilizzerà per mandare le mail personalizzate con i link al *Form1* ed al *Form2* che il *Lead* andrà a compilare.
- L'utente Lead (ovvero in questo caso Incoming) che riceverà appunto via E-Mail i due form e procederà con la loro compilazione la quale se ha successo e non da errori va a scrivere i dati direttamente su *Podio*.

1.4 Situazione Pre-progetto

Prima della realizzazione del progetto i dati relativi ai contenuti del sito Web venivano inseriti o modificati attraerso l'utilizzo di enormi file excel editando direttamente a mano le celle.
Si ritiene che nel 2015 sia inaccettabile manipolare un enorme quantità di dati (si parla di migliaia di record) a mano o comunque utilizzando sistemi senza un minimo di automatizzazione.
Proprio per questo *Aicod* è stata contattata per migliorare i processi e renderli più dinamici ed automatizzati possibili.

1.5 Soluzione Proposta

1.5.1 I CRM e Podio

In economia aziendale il concetto di *Customer Relationship Management* o gestione delle relazioni con i clienti è legato al concetto di fidelizzazione dei clienti.
In un'impresa market-oriented il mercato non è più rappresentato solo dal cliente, ma dall'ambiente circostante, con il quale l'impresa deve stabilire relazioni durevoli di breve e lungo periodo, tenendo conto dei valori dell'individuo/cliente, della società e dell'ambiente dunque l'attenzione verso il cliente è cruciale e determinante. Per questo motivo il marketing management deve pianificare e implementare opportune strategie per gestire una risorsa così importante

Il `CRM` si spinge sostanzialmente secondo quattro direzioni differenti e separate:

- L'acquisizione di nuovi clienti (o clienti potenziali).
- L'aumento delle relazioni con i clienti più importanti (o clienti coltivabili).
- La fidelizzazione più longeva possibile dei clienti che hanno maggiori rapporti con l'impresa (definiti clienti primo piano).
- La trasformazione degli attuali clienti in procuratori, ossia consumatori che lodano l'azienda incoraggiando altre persone a rivolgersi alla stessa per i loro acquisti.

Esistono tre tipi di `CRM`:

- *CRM operativo*: soluzioni metodologiche e tecnologiche per automatizzare i processi di business che prevedono il contatto diretto con il cliente.
- *CRM analitico*: procedure e strumenti per migliorare la conoscenza del cliente attraverso l'estrazione di dati dal CRM operativo, la loro analisi e lo studio revisionale sui comportamenti dei clienti stessi.
- *CRM collaborativo*: metodologie e tecnologie integrate con gli strumenti di comunicazione (telefono, fax, e-mail, ecc.) per gestire il contatto con il cliente.

Riassumendo dunque in generale un `CRM` si occupa di:

- Analisi e gestione della relazione con i clienti: contatto con i clienti e analisi dei bisogni attraverso molteplicità di strumenti come mailing, lettere, telefonate, SMS, ecc.
  Il contatto è fondamentale se si vuole mappare ogni singolo cliente per

poi organizzare tutte le informazioni raccolte in un database strutturato. Queste informazioni sono preziose in quanto permettono di conoscere e se possibile, anticipare le esigenze del cliente.

- Lo sviluppo di contenuti e servizi personalizzati: i dati raccolti vengono gestiti per elaborazioni statistiche utili a segmentare i clienti in specifiche scale.
  Una volta organizzati, è possibile procedere con l'analisi dei dati per sviluppare una comunicazione e un'offerta commerciale e personalizzata.

- L'infrastruttura informatica: attivazione di strumenti informatici che aiutano in questo processo di gestione del cliente.

Un sistema fatto esclusivamente su misura sarebbe stato troppo oneroso sia dal punto di vista economico che come tempistiche, dunque proprio per questo essendo Aicod partner ufficiale di*Podio* abbiamo pensato di proporlo come `CRM` ed affiacarci un applicativo `PHP` il quale genera due form di compilazione uno generico ed uno scritto ad hoc per il tipo di persona invitata ed un sistema di gestione degli ospiti invitati i quali (in base al pacchetto scelto per il singolo ospite) verranno smistati nei vari hotel.
Il sistema d' invio dei Form è gestito tramite l'invio di una newsletter utilizzando `Mailchimp (www.mailchimp.com)`.
Podio e la piattaforma di lavoro collaborativo di proprietà della `Citrix (www.citrix.com)` che permette di organizzare flussi di lavoro, analisi clienti, anagrafiche e gestione dei prodotti con una flessibilità straordinaria.
Aicod stessa usa *Podio* per tener sotto controllo i propri flussi aziendali e lo utilizza per il proprio Project Management.
Ovviamente insieme all'applicativo ed agli account di podio verrà offerta alle fiere di parma un percorso di formazione.
Aicod infatti formerà il commitente.

1.5.2 MailChimp

MailChimp è uno degli strumenti più apprezzati nell'email marketing per la gestione e l'invio di newsletter.
Esso vanta fino a 12000 invii email al mese e un massimo di 2000 iscritti per il piano gratuito.
Ha un'interfaccia intuitiva e offre funzionalità professionali tra cui:

- Generazione di Sign-up form per l'iscrizione degli utenti con campi personalizzabili e facilmente esportabili.
- Grafica della newsletter facilmente editabile e compatibile anche con smartphone e tablet.
- Schedulazione degli invii, gestione delle liste e dei report.
- Applicazioni per iPhone/iPad e Android comodissime sia per l'editing della newsletter che per l'analisi dei report.

Sostanzialmente abbiamo creato un Template per l'invio del *Form1* e del *Form2* e tramite l'export da Podio dei dati e l'import dei dati in mailchimp saremo in grado di mandare migliaia di e-mail personalizzate contenenti i link per i due *Form* che il *Lead* dovrà compilare.

1.5.3 Form1 e From2

Il lead riceverà per e-mail un link personalizzato ed univoco nella quale verrà mostrato l'applicativo contenente i campi richiesti da Fiere di Parma per procedere con l'invito.
Ecco di seguito uno screenshot che illustra una parte dei campi presenti:

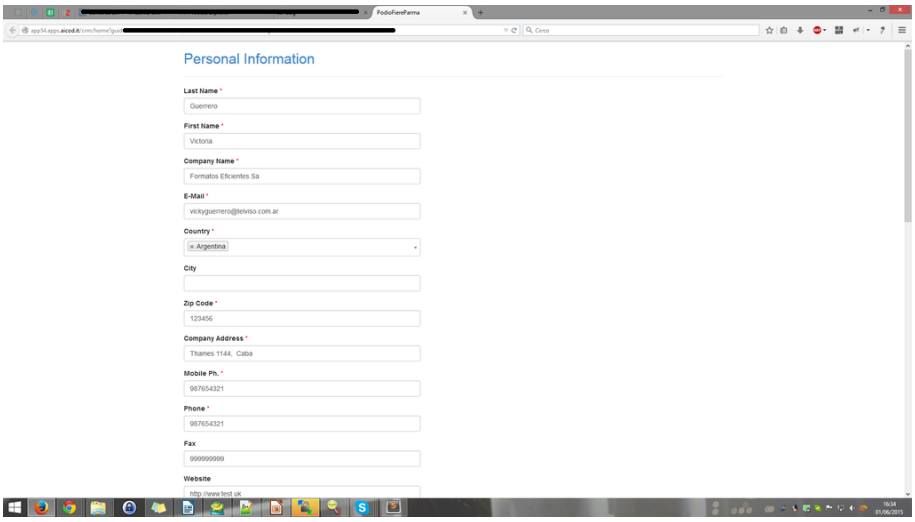

Figura 1.2: Esempio Form1

1.5.4 Rooming

L'applicativo del rooming, al contrario del *Form1* e del *Form2*, sarà utilizzato dall'utente back-end, dunque esclusivamente dagli impiegati di Fiere di Parma.
Entrando ora nel dettaglio, si può notare come in alto a sinistra ci sia un elenco di incoming i quali, avendo risposto con successo al *Form2*, sono pronti per essere gestiti e smistati nei vari hotel. Se sono contrassegnati con il colore *rosso* significa che hanno già una sistemazione, altrimenti devono ancora essere piazzati. Si può notare in oltre un box di *Ricerca* il quale è stato implementato per ovvi motivi, ovvero, la presenza di troppi record (si parla di migliaia). In alto abbiamo un select box dove si sceglie l'hotel della quale si vuole vedere le giornate e le relative camere disponibili. Una delle fea-

ture che abbiamo inserito per essere facile ed inuitivo è creare l'applicativo utilizzando il *drag&drop* ovvero gli incoming nella lista si posso trascinare all'interno della singola giornata nella singola camera. In alto sulla destra dell'interfaccia notiamo il cestino, infatti in caso di errore d'inserimento, è sempre possibile poter cancellare l'azione semplicemente trascinando l'incoming dalla prenotazione al cestino.
Una delle richieste del cliente, è che fosse facile ed intuitivo ecco uno screenshot:

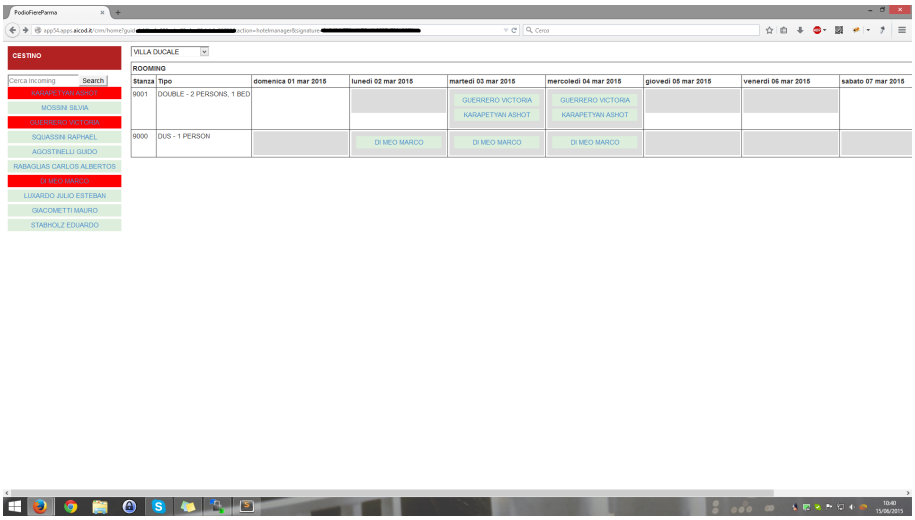

Figura 1.3: Applicativo Rooming

Capitolo 2

Progettazione ed Implementazione

La fase di progettazione si occupa di definire come il sistema dovrà rendere possibili le operazioni discusse nel processo di analisi.
Di fatto, può essere vista come la creazione di una soluzione ad un problema che sia ha ed è proprio per questo durante la progettazione bisogna tenere in considerazione i fattori, le potenzialità e i limiti realtivi alle tecnologie che verranno adoperate per la realizzazione.
L'obiettivo di questo processo è quindi ottenere una descrizione accurata di tutte le parti del sistema, dall'architettura software e del database di Podio fino all'interfacce grafiche con cui sia l'utente back-end che l'utente finale dovrà interagire. Cerchiamo ora di analizzare alcuni punti cruciali nonchè indespansibili in questa fase:

- Il processo di design non deve avere un approccio a tunnel. Un buon design deve tenere in considerazione diversi approcci, e giudicare in base ai requisiti al fine di risolvere il problema con le risorse disponibili sia in termini di budget che forza persone assegnate al progetto.

- Il design deve minimizzare quella che è la distanza tra il software e il problema esistente nel mondo reale. Proprio per questo la struttura del design Deve mimare la struttura del problema di partenza.

- Il design deve mostrare uniformatica ed integrazione. Un design ben fatto, da la parvenza che sia stata una sola persona l'abbia creato. Regole di stile e di formato dovrebbero essere definite da un team ancor prima che il lavoro di design parta.
- Un design ben fatto dovrebbe essere accomodante ai cambiamento, ovvero in caso di integrazione, non deve essere necessario fare tabula rasa.
- Design non è scrivere codice e scaricare codice non è design. Il livello di astrazione del modello è maggiore del codice sorgente. Le uniche scelte di design fatte a livello di codice devono essere implementazioni miniori che permettono al design di essere trasfomato in codice.

2.1 Specifiche del software

Data la natura del progetto, ossia orientata prettamente al Web, si è deciso di implementare il progetto come una applicazione web. La scelta del linguaggio è ricaduta sul `PHP` sia per la sua dinamicità che per la facile integrazione delle `API` di *Podio*, disponibili appunto nel linguaggio appena citato.
L'applicativo svilppato risiederà su un server acquistato da un hosting provider esterno all'azienda. Obiettivo della tesi è lo sviluppo di una applicazione web in Php per integrare il servizio Podio allo scopo di ottenere un `CRM` per la gestione degli incoming gestiti da fiere di Parma.
Dunque dovrà soddisfare i seguenti requisti:

- Implementazione di due DB con i dati forniti da Fiere di Parma
  - DB1: *Lead* ovvero l'elenco dei contatti.
  - DB2: *Aziende* ovvero l'elenco delle aziende.

  Questi due DB continuano ad essere popolati durante tutto l'anno dagli utenti back-end o da un operatore di data entry (agenzie esterne)
- Dovrà essere in grado di permettere una facile organizzazione degli incoming.

- Avere un'interfaccia grafica semplice in modo da renderlo utilizzabile anche da persone con una scarsa conoscienza delle tecnologie informatiche cercando di nascondere la logica, tavolta complessa, delle operazioni.

- I Form e il sistema del rooming saranno sviluppati in Php e dovranno essere in grado di interagire con la piattaforma Podio permettendo dunque tramite opportune chiamate alle `API` scritture sul sistema.

- É stato esplicitamente richiesto che la fase di invio delle e-mail contenenti i link al *Form1* ed al *Form2* sia rapida e poco onerosa e che sia possibile farlo facilmente anche tramite dispositivi mobile.

- Il cliente ha dichiarato di volere fare outsourcing e che quindi probabilmente utilizzerà un agenzia esterna per gestire i pernottamenti dei lead. Proprio per questo verrà forniti 2 account extra di *Podio* allo scopo di permettere questa richiesta.

2.2 Tecnologie utilizzate

2.2.1 HTML, CSS, JavaScript

Queste tre sono le teconlogie fondamentali che stanno alla base di tutte le web applicazion moderne. `HTML` *(Hyper Text Markup Language, http://www.w3.org/html/)* è un linguaggio di formattazione utilizzato per creare la struttura di una pagina web e inserirne i contenutio.La formattazione consiste nell'inserire nel tessto una serie di marcatori *(tag)* che definiscono come questo deve essere visualizzato e dispposto all'interno della pagina. Ogni documento di questo tipo inizi con un tag detto document typeche segnala al browser che versione di `HTML` è stata usato nella pagina. Dopo di questo, il documento `HTML` presenta una struttra ad albero annidato composta da sezioni delimitate da tag opportnu che al loro interno contengono a loro volta sottosezioni sempre delimitata a loro volta da tag. La struttura più esterna delimita l'intero documento ed è compresa tra i tag <html> e

</html>. All'interno dei tag <html> lo standard prevede la definizione di due sezioni ben distinte e disposte in sequenza ordinata:

- L'intestazione (Header), delimitata dai tag <head> e </html>, contiene informazioni di controllo come titolo, metatag e collegamenti a script e fogli di stile. Queste informazioni solitamente non appaiono nella pagina.

- Il corpo (body), compreso tra i tag <body> e </body>, contiene i contenuti veri e propri, ossia il testo, le immagini e i collegamenti che costituiscono la parte visualizzata dal browser.

L'ultima versione disponibile è `HTML5` che offre funzionalità decisamente più avanzate rispetto alle versione precedenti, ma essendo ancora in fase di standardizzazione e non ancora supportata da tutti i browser (soprattutto quelli Legacy) si è optato per l'utilizzo di `HTML4`.
`CSS` (Cascading style Sheet) è un linugaggio usato per definire la formattazione e lo stile di una pagina `HTML`.
Questa tecnologia è stata introdotta per separare i contenuti di una pagina `HTML` dalla formattazione; in questo modo strutturando correttamente un documento `HTML` è possibile cambiarne lo stile modificando semplicemnte alcune proprietà nel file `CSS` ad esso associato.
Infine JavaScript è un linguaggio di programmazione orientato agli oggetti. A differenza di altri linguaggi di programmazione, che permettono la scrittura di programmi stand-alone, `JavaScript` viene utilizzato soprattutto come linguaggio di scripting, ovvero viene integrato con altriu software per programmarne alcuni aspetti. Nel caso delle Web application un interprete `JavaScript` viene implementato come programma ospite dai browser che si occupano di riconoscere il codice contenuto in una pagina `HTML` e lo eseguono.
L'intergaccia che JavaScript utilizza per interagire con la struttura del documento visualizzato nel browser (lato client) è chiamato `DOM` (Document Object Model).
Questo permette di creare applicazioni web dinamiche, modificando la struttira del coumento senza richiede al server di riporocessare la pagina per in-

viarne al browser una versione modificata.

2.2.2 jQuery, jQueryUI

`jQuery`*(https://jquery.com/)* è una libreria JavaScript cross-browser, che fornisce metodi e funzioni per gestire gli aspetti grafici e strutturali di una pagina Web.
Il punto di forza di `jQuery` rispetto ad altre librerie concorrenti, è l'estrema compattezza dei comandi che consente, con poche righe di codice, di effettuare svariate operazioni su `DOM` e aggiungere alla pagina effetti grafici. Inoltre, essendo una libreria facilmente estendibile, si possono trovare online numerosi plugin che ne estendono le funzionalità. Una delle estenzioni più famose è appunto `jQueryUI` che comprende un insieme di comportamenti grafici personalizzabili con supporto di intereazioni, animazioni ed effetti avanzati.
La sintassi di `jQuery` è studiata per semplificare la navigazione dei documenti, la selezione degli elementi sul `DOM`, creare animazioni, gestire eventi e implementare funzionalità `AJAX`. Nel progetto sono stati utilizzati ampiamente come ad esempio l'effetto drag and drop utilizzato nell'applicazione del rooming.
`jQuery` offre dunque i seguenti vantaggi:

- *Incoraggia la separazione del `JavaScript` al `HTML`*: La libreria `jQuery` infatti facilita l'aggiunta di event handler al `DOM` usando `JavaScript` invece al posto dell'aggiunta di tag `HTML`. Questo di fatto, incoraggia gli sviluppatore a seprare completamente il codice JavaScript dal markup HTML;

- *Compatto e Chiaro*: `jQuery` vanta di essere chiaro e compratto grazie in quanto possiede feature come la concatenazione di funzioni e nomi di funzioni molto corti

- *Eliminazione di incompatibilità tra Browser*: L'engine `JavaScript` di browsers diversi, può avere piccole differenze quindi il codice che funziona correttamente su un dato browser potrebbe non lavorare cor-

rettamente su un altro. `jQuery` gestisce tutte queste inconsistenze cross-browser e fornisce una interfaccia che funziona su browser diversi.

- *Estendibile*: Estendere il framework `jQuery` è veramente molto semplice. Nuovi elementi, nuovi eventi e metodi sono si possono aggiungere con semplicità e possono essere riutilizzati come Plugin in seguito.

2.2.3 JSON

`JSON`, acronimo di *JavaScript Object Notation*, è una tecnologia, ma sarebbe meglio definirlo un formato, concepito per l'interscambio dei dati all'interno di applicazioni di tipo client-server, come per esempio quelle realizzate in `AJAX` e `PHP`, ma è largamente utilizzato anche in associazione a linguaggi di programmazione come `Java`, `C`, `C#` e `Perl`.
La diffusione di questo formato è dovuta, con tutta probabilità, alla sua estrema semplicità d'utilizzo, esso supporta inoltre una grande varietà di tipi di dato. Con un piccolo esempio mostro perchè è comodo utilizzare `PHP` in associazione al formato d'interscambio `JSON`:

```
<?php
// Array per la codifica
$vettore = array("albero", "ramo", "mela");
// codifica JSON
$stringa = json_encode($vettore);
// stampa del risultato
echo $stringa;
?>
```

Se si osserva il funzionamento dello script mostrato, la funzione *json_encode()* svolge il compito di restituire, partendo da una semplice array, una stringa contenente la rappresentazione `JSON` di un valore, in questo caso il Web server chiamato a gestire l'interscambio tra client e server che ospita l'applicazione, invierà una richiesta di stampa a video dell'operazione di codifica dei dati operata tramite *json_encode()* nel nostro caso [albero,-ramo,mela], a sua volta l'interprete del client effettuerà una decondifica per restituirli nuovamente sotto forma di vettore, per cui i valori rappresentati nella stringa saranno gli stessi che compongono l'array.
In questo altro caso invece, osserviamo la funzione *json_decode()* che svolge il compito di decodificare una stringa `JSON` passata sotto forma di oggetto:

```
...
<?php
// definizione dell'oggetto per il parsing
$j = '{"Anno": 2010}';
$oggetto = json_decode($j);
// lettura dello stream JSON
echo $oggetto->{'Anno'};
?>
```

In conclusione, La possibilità di effettuare codifiche e decodifiche praticamente al volo con `JSON`, ha messo in luce questo formato come alternativa al metalinguaggio `XML`, i vantaggi sono infatti evidenti: nel caso di `JSON` non dovrà essere generato alcun documento (cosa invece necessaria per quanto riguarda `XML`) e le prestazioni degli script ne risultano avvantaggiate non essendo più costrette ad effettuare il parsing di file che, con l'aumento del numero dei dati da gestire così come del traffico generato dai client, possono diventare anche estremamente pesanti.

2.2.4 API di Podio

Le API di Podio sono delle interafacce completamente programmabili ed hanno accesso a tutte le funzionalità offerte da Podio. Al momento sono disponibili PHP, .NET, Ruby, Java, Phyton, Android and Objective-C. Di fatto tutto il front-end di Podio è costruiito con le API, dunque è possibile costruire applicativi che possono registrare webhooks alle apps di Podio. Esiste un ratelimit a quante chiamate API si possono fare in un ora, il problema verrà discusso in seguito.

Per cominciare ad utilizzare le API di podio è necessario acere un account e acquisire una Chiave API per ogni App che si vuole costruire.
Le API sono scaricabili gratiuitamente ed utilizzano JSON come formato di interscambio. SSL è mandatorio ed è utilizzato per tutto. Per le Autorizzazioni e l'autenticazione viene utilizzato OAuth che non è altro che un protocollo che permette l'autorizzazione di API di sicurezza con un metodo standard e semplice.
Le API utilizzano i seguenti metodi HTTP per la manipolazione degli oggetti:

- GET: Utilizzato per il prelevamento degl oggetti.
- POST: Usato per la creazione degli oggetti e le azioni su di essi.
- PUT: Usato per azioni di update sugli oggetti.
- DELETE: Usato per la cancellazione degli oggetti.

Essendo che Podio utilizza *OAuth* come protocollo per l'autenticazione, è possibile da parte di applicazioni esterne accedere a Podio senza aver diretta conoscienza di username e password. Prima di cominciare ad utilizzare le API abbiamo dovuto registrare la nostra applicazione alla quale è stato assegnato un identificativo unico chiamato appunto *Client ID* ed un *Client Secret* e sono entrambe indispensabili per l'autenticazione.

Questo è il codice che abbiamo scritto per permettere ciò:

```
...
<?php
 function initPodio() {
  $opts = $this->conf['podio'];
  // Authenticate Podio API
  \Podio::set_debug($this['debug']?true:false);
  \Podio::setup(
    $opts['client_id'],
    $opts['client_secret'],
    array("session_manager" => "\util\PodioPhpSession")
  );
  if(!\Podio::is_authenticated()) {
    \Podio::authenticate_with_password(
     $opts['username'],
     $opts['password']
    );
  }
 }
?>
```

2.2.5 Silex

`Silex`*(http://silex.sensiolabs.org/)* è un microframework per `PHP` ed è basato su `Symfony2` e `Pimple` che fonisce le fondamenta per costruire applicazioni a file singolo. Analiziamo ora alcune peculiarità di `Silex`:

- *Conciso:* Offre delle `API` concise ed intuitive e molto facili da utilizzare.
- *Estendibile:* `Silex` gode di un sistema basato su Pimple che di fatto facilita l'utilizzo di librerie di terze parti.
- *Testabile:* `Silex` usa l'HttpKernel di `Symfony2` che astrae le richiezste e le rispooste. Questo meccanismo offre un ambiente ottimo ed adatto

a fare testing sulle applicazioni e sul framework stesso. Inoltre rispette le specifiche `HTTP` e ne incoraggia l'uso corretto.

2.3 MVC Model

Il Model-View-Controller o `MVC` , è un pattern architetturale molto diffuso nello sviluppo di sistemi software, in particolare nell'ambito della programmazione orientata agli oggetti, in grado di separare la logica di presentazione dei dati dalla logica di business.

Questo pattern si posiziona nel livello di presentazione in una Architettura multi-tier. L'MVC model si può riassumere con il seguente diagramma:

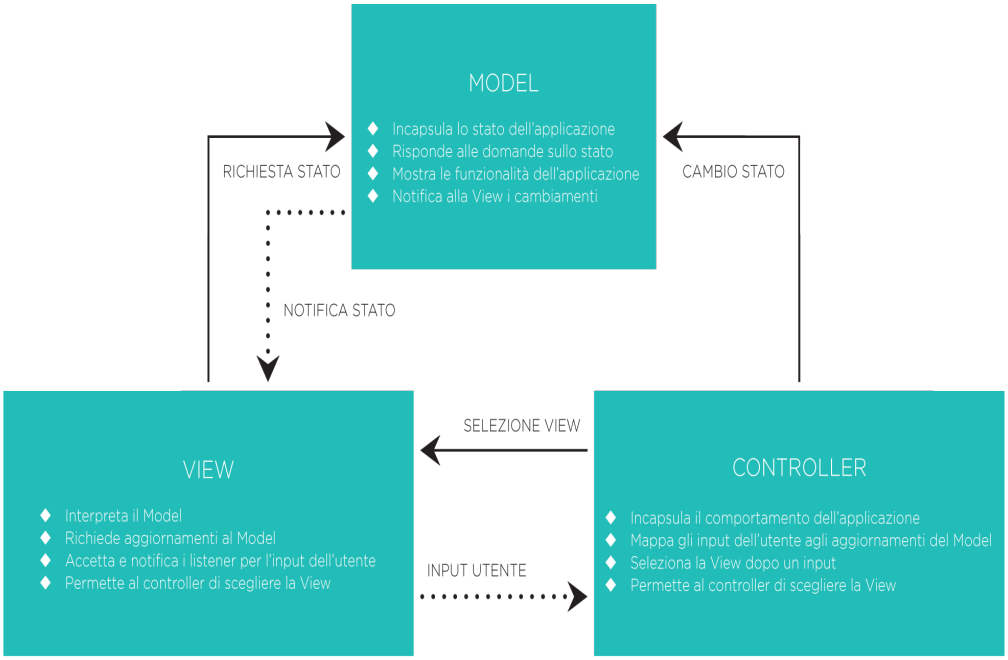

Figura 2.1: MVC Model

Il paradigma `MVC` consiste nel distinguere, all'interno di un sistema informatico, tre tipologie di componenti:

- quelle che rappresentano i dati in oggetto (*Model*);
- quelle che si occupano di presentare i dati all'utente (*View*);
- quelle che si occupano dell'interazione con l'utente (*Controller*).

Tale paradigma trova la sua applicazione principale nei programmi con interfaccia grafica (`GUI`) e nei sistemi con interfaccia web. Il paradigma detta la distinzione di responsabilità tra le tre categorie e il flusso di informazioni tra di esse. Nel nostro progetto il *Model* è rappresentato da *Podio* che gestisce i dati, la *View* sono i file *Twig* che si occupano di comporre ed assemblare la pagina `HTML` che verrà visualizzata all'utente e infine il *Controller* che saranno gli applicativi *Form1*, *Form2* ed infine il Rooming che fungono appunto da controller e gestiscono il flusso di dati in ingresso dal utente per andare a salvarli su Podio.

2.4 Workflow

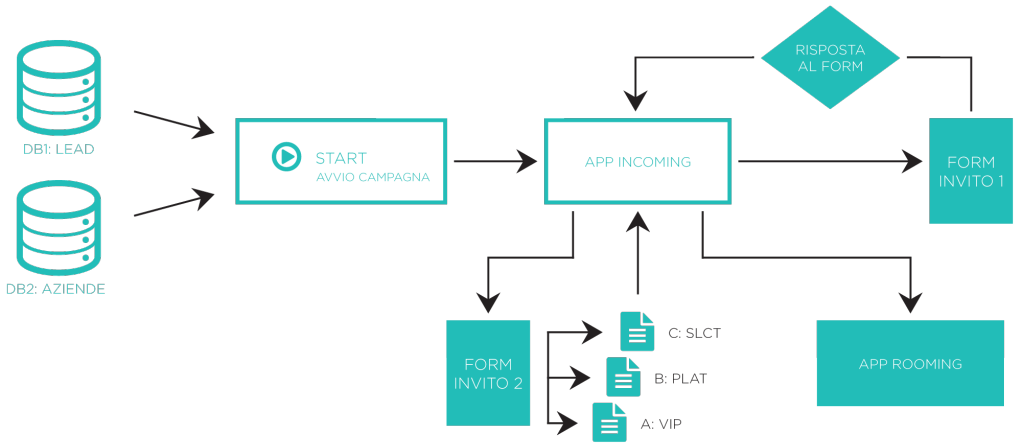

Figura 2.2: Diagramma del workflow

Analizziamo nel dettaglio quelli che possono essere definiti step di quello che è il Workflow:

- Il DataBase dei *Lead* ed il DataBase delle *Aziende* saranno popolati direttamente dal committente grazie agli strumenti di import offerti da *Podio*.

- Una volta popolati i DB, si potrà dunque creare una nuova campagna in cui verrano invitati i singoli lead che potenzialmente potrebbero essere interessanti alla partecipazione dell' evento ed allo stesso modo questi inviti saranno mandati alle aziende.

- Partita la campagna l'utente backend, grazie ai filtri di *Podio*, farà una selezione ed esporterà i dati delle persone a cui vuole mandare l'invito. Supponiamo ad esempio che si voglia invitare solo i lead di orgine Argentine tramite una opportuna selezione grazie ai filtri di Podio sarà possibile esportare la lista di tali *Lead* e successivamente procedere a caricare quest'ultima su *MailChimp* al fine di mandare il *Form1* a tutti

lead selezionati. A questo punto, in un automatico, all'interno di *Podio* ci sarà un Flag che verà impostato su Invito *Form1* Inviato.

- Il Lead riceverà dunque via mail l'applicativo *Form1* e provvederà a compilarlo con i propri dati anagrafici.
- Inviato il modulo, il valore del Flag precedentemente impostato su - Invito *Form1* Inviato verrà cambiato in Invito *Form1* Ricevuto.
- Stabilita una data per il termine ultimo in cui si accettano i *Form1*, si provvede a creare un ulteriore vista su Podio la quale selezionerà solo ed esclusivamente i lead che hanno il Flag valorizzato con Invito *Form1* Ricevuto. Si procederà dunque ad importare i dati in *MailChimp* per mandare il *Form2*, il quale contiene le personalizzazioni a secondo di che tipo di Lead invitato, ad esempio ad un VIP viene offerto il pacchetto che comprende il volo, ad un buyer-pass non viene offerto questo tipo di servizio. Sempre in automatico, il flag Invito form2 Inviato verrà valorizzato.
- Il Lead riceverà il *Form2* personalizzato in base alle richieste esplicitate nel primo e procederà dunque all invio.
- Il sistema, in automatico andrà a cambiare il valore del Flag Invito *Form2* Inviato in Invito *Form2* Ricevuto.
- Solamente a questo punto, sarà possibile visualizzare il Lead nell applicazione *Rooming* e quindi sarà possibile aggiungere una prenotazione tenendo conto delle richieste manifestate da esso, ad esempio in quale tipo di camera vuole alloggiare, se ha degli accompagnatori, date check-in check-out ecc.

2.5 Implementazione

La classe *home* fa da controller, in particolare *indexAction()* è un selettore il quale accetta 4 parametri tra cui: l'id, il numero dell app. Semplicemente controlla il parametro action ed in base a quello sceglie se fare la chiamata all'*hotelmanager* o al *form*

```
<?php
namespace fierecrm\crm;
class home {
  // Entering globale
  public function indexAction($view,$app,$req,$id) {
  // Verify args signature, can throw invalid args exception
    $args_verify_ok = $app->verifyArgsSignature($_GET);
    if($_REQUEST['action'] == 'hotelmanager') {
      return $this->hotelManager($view,$app,$req,$id);
    } else if(in_array($_REQUEST['action'],['form1','form2'])) {
      return $this->form($view,$app,$req,$id);
    }
  }
```

```
  public function form($view,$app,$req,$id) {
    $yaml_path = "~/.app/etc/{$_REQUEST['action']}.yaml";
    $fields = $this->app->yamlParseFile($yaml_path);
    $app->initPodio();
    $form = new \fierecrm\PodioForm();
    $form->setGuid($_REQUEST['guid']); //guid.
    $form->setMetadata($fields);
    $http_context = array(
      'POST' => $_POST,
      'REQUEST' => $_REQUEST,
      'SERVER'=> $_SERVER,
    );
```

```
    $form->handleForm($http_context);
    $model = array('form' => $form);
    $resp = $app->twig('.app/view/form.twig',$model,true);
    return $resp;
  }
```

Il metodo *form()*, crea il form in base sulla base dei parametri passati e crea i campi che saranno da compilare(o da leggere a seconda della richiesta) basandosi sul file di configurazione YAML seguente:

```
......
last_name:
  section: 'Personal Information'
  label: 'Last Name'
  required: true
  podio_obj: '#row.persona.nome'
first_name:
  label: 'First Name'
  required: true
  podio_obj: '#row.persona.nome-2'
company_name:
  label: 'Company Name'
  required: true
  podio_obj: '#row.persona.azienda.company-name'
......
```

Questo come possiamo vedere non è altro che appunto un file di configurazione, unico vincolo che abbiamo è che tutti i campi per essere mappati con il DB di Podio devono possedere l'attributo *podio_obj* per permettere la corretta scrittura e lettura del valore.

```
  public function hotelManager($view,$app,$req,$id) {
    $app->initPodio();
    $http_context = array(
      'POST' => $_POST,
      'REQUEST' => $_REQUEST,
      'SERVER'=> $_SERVER,
    );
    $rooming = new \fierecrm\PodioRooming($app,$http_context);
    $rooming->setGuid($_REQUEST['guid']); //guid
    if($_POST['ajax']) {
      $resp = $rooming->handleAjax();
    } else {
      $rooming->handleRooming();
      $model = array('rooming' => $rooming);
      $resp = $app->twig('.app/view/rooming.twig',$model,true);
    }
    return $resp;
  }
}
```

Infine questo metodo, gestisce la chiamata al applicativo rooming.

La funzione *podioInitPath()* ci semplifica la costruzione del form in quanto non è più necessario crearsi i singoli campi.
Infatti tramite il carimento di un file `YAML` opportunamente configurato è possibile costruire, modificare, aggiungere e togliere campi semplicemente editando il file di configurazione `YAML` senza andar a rimettere mano al codice.

```
<?
public function podioInitPath($path) {
  if(!$path) {
    return;
  }
  // Esempio stringa #row.persona #row.persona.azienda.relazione
      ecc ecc
  // Facciamo "l'explode" della stringa e come separatore usiamo il
      carattere "."
  $path_tokens = explode('.', $path);
  $current_node = &$this->root;
  foreach ($path_tokens as $token) {
    // Il current_node non sara altro che il valore
    $current_node = &$current_node['childs'][$token];
    if($current_node['item'] && !$current_node['app']) {
      $current_node['app'] =
          \PodioApp::get($current_node['item']->app->id);
    }
    if($last_node['item'] && !$current_node['field']) {
      $current_node['field_item'] = &$last_node['item'];
      $current_node['field'] =
          $current_node['field_item']->fields[$token];
      if(!$current_node['field'] && $last_node['app'] &&
          $last_node['app']->fields[$token] ) {
        $new_field_conf =
            $last_node['app']->fields[$token]->as_json(false);
        if($new_field_conf) {
          $class_name =
              "\Podio".ucwords($new_field_conf['type'])."ItemField";
          $new_item_field = new $class_name($new_field_conf);
          $current_node['field_item']->fields[$token] =
              $new_item_field;
          $current_node['field'] =
              &$current_node['field_item']->fields[$token];
```

```
      }
      $podio_field = $last_node['app']->fields[$token];
    }
  }
  if($current_node['field'] && $current_node['field']->type ==
      'app') {
    if(!$current_node['item'] &&
        $current_node['field']->values[0]) {
      $current_node['item'] = \PodioItem::get(
$current_node['field']->values[0]->item_id);
    }
    if(!$current_node['app']) {
      if($current_node['item']) {
        $current_node['app'] =
            \PodioApp::get($current_node['item']->app->id);
      } else if($current_node['field']->config->apps[0]) {
        $current_node['app'] = \PodioApp::get(
  $current_node['field']->config->apps[0]->id);
      }
    }
  }
    $last_node = $current_node;
}
```

Analizziamo ora il `JavaScript` che lavorerà in particolare sul `DOM` nell applicazione Rooming.
Questo file infatti tramite `jQuery` ed suo metodo *ajax* ed un opportuno utilizzo della libreria `Select2` renderà possibile l'effetto *Drag&Drop* ed inoltre permetterà di filtrare la ricerca degli incoming pronti ad essere smistati nei vari hotel disponibili.

```
// Inizializzazione jQuery su DOM
jQuery(function($){
  $('.select2').select2();
  var js_current_locale = 'it';
  var jq_locale_current =
      $.extend($.datepicker.regional[js_current_locale] ||
      $.datepicker.regional['en-GB'], { changeYear: true });
  $.datepicker.setDefaults(jq_locale_current);
  $('input.datepicker').datepicker();
  // Gestione selectbox dell hotel manager
  $(document).on('change','.curr_hotel',function(){
    var form = $(this).parents('form');
    if(form.length)
      form.submit();
  });
  var todo = [];
```

Stiamo definendo una funzione anonima e la passiamo a `jQuery` che la chiamerà a `DOM` ready. in particolare inizializza il `Select2`, il datepicker e gestirà il submit del form. array dichiarato nel contesto globale, che nel nostro caso, corrisponde alla pagina web.

```
function accept_drop(opts) {
  var drag_item = opts.drag_item;
  var drop_cnt = opts.drop_cnt;
  var event = opts.event;
  var drop_is_trash = drop_cnt.hasClass("trash");
  var drag_item_parent_is_clone =
      drag_item.parent().hasClass("clone-items");
  var drop_is_dropcontainer = drop_cnt.hasClass("drop-container");
  var ret = false;
  var currentRoom = drop_cnt.data('room')s
  var currentDay = drop_cnt.data('day')
  var already_present = $('.drag-item[data-incomingid='+drag_item
  .data('incomingid')+']',drop_cnt).length?true:false;
  var can_add = $('.drag-item',drop_cnt).length>=drop_cnt
  .data('roomtype')?false:true;
```

Definizione di tutte le variabili locali utilizzate dalla funzione. la funzione *accept_drop()* infatti accetta come parametro opts che è un oggetto JavaScript il quale contiene gli elementi su cui operare e l'azione da svolgere

```
  if(drop_is_trash) {
    if(event=='drop') {
      // Push su array todo l'azione Delete dell'id corrente
      todo.push({
          azione: "Delete",
          item: drag_item,
          to: drop_cnt,
          from: drag_item.parent(),
      });
      drag_item.remove();
    }

    ret = !drag_item_parent_is_clone;
```

```
  } else if(drop_is_dropcontainer) {

    if(event=='drop') {
      var elput = drag_item;
      if(drag_item_parent_is_clone) {
        elput = elput.clone();
      }
      elput.removeAttr('style');
      //drop_cnt.append(elput);

      if(!drag_item_parent_is_clone) {
        todo.push({
            azione: "Move",
            item: drag_item,
            to: drop_cnt,
            from: drag_item.parent(),
        });
      } else {
        // push su array todo l'azione ADD dell'id corrente
        todo.push({
          azione: "Add",
          item: elput,
          to: drop_cnt,
          from: elput.parent(),
        });
      }

      drop_cnt.append(elput);
    }
    ret = !already_present && can_add
  }
  return ret;
}
```

Ogni volta che l'utente effettua l'operazione di drop la funzione *accept_drop()* verifica dove si sta facendo l'azione. Nello specifico si verifica per primo se si sta droppando nel cestino oppure se si sta droppando su una delle celle della griglia stanza/giornata. In entrambi i casi viene accodato alla coda todo un oggetto literal che esprime l'azione da eseguire lato server.

```
// Gestione Drag&Drop
$(document).on('mouseover','.drag-item',function(){
  var opts = {
    revert: true,
    helper: function() {
      var bclone = $(this).parent().hasClass("clone-items");
      if(bclone)
        return $(this).clone();
      return this;
    },
  }
  $(this).draggable(opts);
})
```

Invece di inizilizzare staticamente gli eleenti su cui e possibile effettuare il drag, tramite l'apposita aggiunta di un evento live sul DOM, andiamo ad intercettare dinamicamente gli elementi draggable e su di essi, abilitiamo la corrispondente funzionalità di jQueryUI

```
$('.drop-container').each(function() {
  var opts = {
    accept: function(drag_item_arg) {
      var drag_item = $(drag_item_arg);
      var drop_cnt = $(this);
      return accept_drop({
  drag_item:drag_item,
  drop_cnt:drop_cnt,
  event:'accept
```

```
      '});
    },
    drop: function(e,ui) {
      var drag_item = $(ui.draggable);
      var drop_cnt = $(this);
      accept_drop({
  drag_item:drag_item,
  drop_cnt:drop_cnt,
  event:'drop'
      });
    },
  }
  $(this).droppable(opts);
});
```

Con questa chiamata inizializziamo le funzionalita di drag and drop per ogni elemento della classe `HTML` *drop-container* . è proprio in qeusto punto che associamo l'azione di drop alla funzione precendetemente definita *accept_drop()*

```
function consumeRolls() {
  if (todo.length) {
    // prelevo in testa all'array todo l'azione corrente da fare
    var obj = todo.shift();
    var payload = {
      ajax: true,
      incomingid: obj.item.data('incomingid'),
      reservationid: obj.item.data('reservationid'),
      from: {
        room: obj.from.data('room'),
        day: obj.from.data('day'),
```

```
        },
        to: {
          room: obj.to.data('room'),
          day: obj.to.data('day'),
        },
        hotelid: obj.item.data('hotel'),
      };
      // Ajax
      $.ajax({
        url: $('form').attr('action'),
        type: 'POST',
        cache: false,
        data: payload
      }).done(function(data) {
        obj.item.data('reservationid',data.reservationid);
      }).fail(function() {
        alert("Attenzione: Errore");
      }).always(function() {
        consumeRolls();
      });
    } else {
      setTimeout(consumeRolls, 500);
    }
  }
  consumeRolls();
});
```

La funzione controlla se esiste almeno un elemento che di fatto rappresenta un azione da eseguire sul server; Se esiste, prende il primo elemento dalla testa della pila ed utilizzando i dati in esso contenuito costruisce un oggetto payload da inviare al server.
Successivamente effettua un chiamata `HTTP` di tipo `POST` utilizzando il wrapper *ajax()* di `jQuery`.
L'`URL` della chiamata viene preso dall'attributo action presente nel form ed

i dati presenti nell oggetto payload verranno codificati in `JSON`. Le funzioni anonime passate ai metodi .done .fail .always verranno chiamate rispettivamente all'esecuzione del successo della chiamata, al faillmento e l'ultima in entrambi i casi.
Quindi in caso di successo,andiamo ad assolciare il risultato al corrispondente oggetto nel `DOM`, in caso di errore allertiamo l'utente ed a chiamata conclusa richiamamo di nuovo la funzione *consumeRolls()* in modo da processare l'eventuale elemento successivo.
Nel caso non esistessero elementi da gestire, accodiamo una nuova chiamata alla funzione *consumeRolls()* dopo un timeout di 500ms.

2.6 Sicurezza

Per rendere il nostro applicativo sicuro abbiamo usato tecniche di hashing. Quando un Record è salvato in Podio, gli viene assegnato un `ID`, ovvero un identificativo univoco che in questo caso specifico, non è altro che un semplice numero naturale.
Per ovvi motivi, questa cosa non è ottimale in quanto può portare a scenari indesiderati e potenzialmente dannosi.
Ad esempio, i link contenuti nelle e-mail che saranno mandate mostrano all'utente il proprio *Form1* e *Form2* indentificati univocamente per quel utente e non devono essere in nessun modo guessable da terzi.
Proprio per questo abbiamo creato un campo personalizzato su Podio, nella quale tramite una nostra chiave, abbiamo fatto hashing dei parametri utilizzando l'algoritmo `MD-5` al fine di poter produrre un UID più complesso e che non sia prevedibile da un utente.
Una funzione crittografica di hash trasforma dei dati di lunghezza arbitraria in una stringa di dimensione fissa chiamata valore di hash, *impronta del messaggio* o *somma di controllo*, in inglese nota come *message digest*.
La funzione crittografica di hash ideale deve avere tre proprietà fondamentali:

- Deve essere estremamente semplice calcolare un hash da qualunque tipo di dato;

- Deve essere estremamente difficile o quasi impossibile risalire al testo che ha portato ad un dato hash;
- Deve essere estremamente improbabile che due messaggi differenti, anche se simili, abbiano lo stesso hash;

Oltre a queste tre proprietà fondamentali c'è da aggiungere il cosiddetto *avalanche effect*, ossia la minima modifica del messaggio deve portare ad un'alterazione radicale dell'*Message Digest*.

Funzioni con queste proprietà sono utilizzate come funzioni di hash in diversi campi, anche al di fuori di quello prettamente crittografico.
Le applicazioni pratiche includono infatti i controlli sull'integrità dei dati mediante somme di controllo, le firme digitali semplici, l'autenticazione e varie applicazioni nella sicurezza informatica.
Un hash può anche agire come una rappresentazione concisa del messaggio o del documento da cui è stato calcolato, permettendo un facile indicizzamento di file dati unici o duplicati.
In diversi standard ed applicazioni le due funzioni crittografiche di hash più utilizzate sono `MD5` ed `SHA-1`.
Un hash è una funzione che prende in ingresso una stringa di lunghezza arbitraria producendo in uscita una nuova stringa di lunghezza predefinita che rappresenta una sorta di impronta digitale dei dati contenuti nella stringa di ingresso.
La funzione di Hash è a senso unico: conoscendo l'hash deve essere difficile trovare il messaggio originale mentre possedendo il messaggio originale è possibile stabilire il suo hash univoco.
Una funzione crittografica di hash deve approssimare una funzione random restando deterministica ed efficiente dal punto di vista computazionale.
In altre parole deve assicurare che ciascun output abbia sostanzialmente la stessa probabilità di essere associato ad un input e restituire sempre lo stesso output a fronte dello stesso input. Una funzione crittografica di hash è considerata insicura se una delle seguenti azioni è fattibile:

- Calcolare un messaggio che produca un hash dato.

- Trovare una collisione ovvero due messaggi diversi che producono lo stesso identico hash.

Un utente malintenzionato infatti che fosse in grado di eseguire una delle due operazioni citate, potrebbe ad esempio sostituire un messaggio autorizzato con un altro senza che si possa provare e verificare l'avvenuta contraffazione.
Una funzione crittografica di hash di qualità dovrebbe rendere difficile calcolare anche solo due messaggi con hash sostanzialmente simili oltre che garantire una ottima non-correlazione tra hash e messaggio: nessuna informazione del messaggio dovrebbe essere desunta dal suo hash.
Ad un eventuale attaccante resta solo l'informazione relativa all'associazione tra messaggio ed hash: esso può riconoscere che il messaggio è stato di nuovo usato vedendo una seconda volta lo stesso hash.

2.6.1 Utilizzo di MD5

L'`MD5` è un tipo di codifica che prende in input una stringa di lunghezza arbitraria e ne produce in output un'altra a 128 bit.
La codifica avviene molto velocemente e l'output noto anche come MD5 Hash restituito è tale per cui è altamente improbabile ottenere con due diverse stringhe in input uno stesso valore hash in output.
Ad oggi sono disponibili molte risorse online che hanno buone probabilità di riuscire a decriptare parole comuni codificate.
La crittografia tramite algoritmo `MD5` viene applicata in tutti i settori dell'informatica che lavorano con il supporto delle firme digitali o che comunque trattano dati sensibili. Ad esempio, viene utilizzata per controllare che uno scambio di dati sia avvenuto senza perdite, semplicemente attraverso il confronto della stringa prodotta dal file inviato con quella prodotta dal file ricevuto.
Con lo stesso metodo si può verificare se il contenuto di un file è cambiato, ad esempio questa funzionalità è largamente utilizzata dai motori di ricerca per capire se una pagina deve essere nuovamente indicizzata oppure no.

È diffuso anche come supporto per l'autenticazione degli utenti attraverso i linguaggi di scripting Web server-side (PHP in particolare): durante la registrazione di un utente su un portale internet, la password scelta durante il processo verrà codificata tramite `MD5` e la sua firma digitale verrà memorizzata nel database. Successivamente, durante il login la password immessa dall'utente subirà lo stesso trattamento e verrà confrontata con la copia in possesso del server, per avere la certezza dell'autenticità del login. Qui in seguito un breve esempio di come abbiamo integrato l'md5 per la generazione di quello che abbiamo deciso di chiamare *GUID*:

```
// START CUSTOM PARAMETERS
tab_guid = "56152259-b7b5-4542-8d42-f20c0ec79bd8"
created_at = moment(@Creato Il)
incremental_id = @ID Univoco
// END CUSTOM PARAMETERS
str = tab_guid + "-" + incremental_id + "-" + created_at
ret = md5(str)
```

Come si può osservare, il `GUID` è univoco praticamente al 100% in quanto è creato in base al risultato della concatenazione delle 3 seguenti stringhe:

- *tab_guid* che non è altro che una stringa nostra per renderlo ancora meno inindovinabile. Quando si usano queste stringhe in gergo si chiama Add salt.
- *created_at* è Timestamp dell'orario di creazione del record. (il Timestamp è numero intero di secondi dal 01/01/1970)
- *incremental_id* è l'ID generato automaticamente da Podio.

Allo stesso modo, abbiamo creato i parametri per gli `URL` completi che portano al *Form1* e al *Form2*:

```
// START CUSTOM PARAMETERS
row_guid = @guid
shared_key = "CRMINC93084239040234"
base_url = "http://app47.apps.aicod.it/crm/home"
custom_args = "action=form1"
// END CUSTOM PARAMETERS
url_args = "?guid="+row_guid
url_args += "&" + custom_args
url_args += "&signature="+md5(url_args+shared_key)
ret = base_url + url_args
```

Dove:

- row_guid è appunto il `GUID` creato nell'esempio precedente.
- shared_key è il nostro salt ovvero la stringa che aggiungiamo per complicare l'output.
- base_url è l'url base della nostra applicazione.
- custom_args è il parametro che ci permette di scegliere se si vuole compilare il *Form1* oppure il *Form2*.

Conclusioni

Il lavoro svolto è stato molto duro ma entusiasmante. L'applicativo e il sistema messo a punto sono stati un palese successo. Il codice sorgente ha una solida implementazione ed è aperto ad eventuali nuove estensioni senza dover andare a modificare il codice già scritto. Siamo riusciti a stare nei tempi e nel budget che il committente aveva messo a disposizione. Tutti i progetti Aicod nascono da esigenze dei clienti e vengono sviluppati sulla base degli obiettivi da raggiungere e proprio grazie ad una rapida ed ottima collaborazione con il committente siamo riusciti ad arrivare ad una soluzione adeguata e studiata appositamente per quest'ultimo. Abbiamo approfondito e visto come funzionano le `API` di *Podio* e siamo riusciti a creare degli applicativi (in particolare, in due form) che al 99% riusciremo a riutilizzare per altre ed eventuali futuri progetti. Avendo acquisito una certa familiartà con le `API` di *Podio* , Aicod sta seriamente valutando di creare altri tool e plugin, da offrire alla community di *Podio* .

Ringraziamenti

Passando ora ai ringraziamenti ci tengo a ringraziare tutti i colleghi, amici e familiari che mi hanno sostenuto sia nei momenti belli che difficoltosi di questo percorso. Un grazie di cuore ai miei genitori Claudio e Maria, a mio fratello Alessandro, alla mia morosa Katia Castagnetti ed ai colleghi universitari, Andrea Bolzoni, Martina Iasoni, Eleonora Turchi Sassi, Debora Ronzoni, Nazario D'Errico, Enrico Schiraldi, Simone Benassi, Bogdan Botezatu, Giuseppe Mollica, Gabriele Ravanetti, Francesco Rastelli, Lorenzo Furini, Mirko Aloi, Massimiliano Braglia e tanti tanti altri. Un grazie a Lanfranco Pellesi e Giovanni Lamberti, i due amici che più di tutti mi hanno sostenuto quando le cose non volevano andare proprio. Infine ringrazio Il professor Federico Bergenti per essere stato il correlatore di questa tesi e tutti i colleghi ed amici di Aicod srl.

Bibliografia

[1] JavaScript
http://www.javascript.com/

[2] jQuery
http://www.jquery.com/

[3] Podio
http://www.podio.com/

[4] MailChimp
https://www.mailchimp.com/

[5] UML
http://www.uml.org

[6] Citrix
http://www.citrix.com/

[7] W3C
http://www.w3.org/