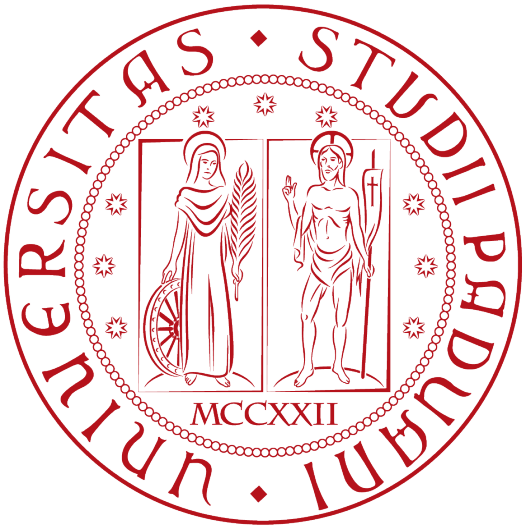
**Università degli studi di Padova**

DIPARTIMENTO DI MATEMATICA

CORSO DI LAUREA IN INFORMATICA




# Consolidamento di un sistema esperto per la gestione della sicurezza in azienda

*Tesi di laurea*

*Relatore*

Prof.ssa Ombretta Gaggi

*Laureando*

Fabio Ros
609724

Anno Accademico 2015-2016

# Indice

# Elenco delle figure

## Elenco delle tabelle

**Sommario**

Il presente documento descrive il lavoro svolto durante il periodo di stage, della durata di circa trecento ore, dal laureando Fabio Ros presso l'azienda Moku S.r.l. Il progetto di stage prevede l'analisi, la progettazione e lo sviluppo di alcune delle logiche del sistema esperto in coordinamento con il committente ed un consulente della sicurezza. Nello specifico è prevista l'implementazione delle componenti software sia lato backend, mediante l'utilizzo di Ruby on Rails, sia lato frontend simultaneamente alla definizione delle regole di gestione del sistema esperto utilizzando il framework Drools.

# 1 Introduzione

## 1.1 L'azienda

Figura 1: Logo dell'azienda ospitante - Moku S.r.l.

Moku S.r.l è una startup nata nel 2013 e situata in H-Farm. L'organizzazione del lavoro è supportata dalla suite Atlassian che offre un servizio cloud per la gestione dei compiti e dei progetti. L'azienda lavora a stretto contatto con il cliente definendo requisiti, user experience dei prodotti realizzati ed opportunità di business. Obiettivo primario dell'azienda è la qualità del software e l'utilizzo di tecnologie recenti poiché l'alto coefficiente innovativo dei prodotti realizzati rende necessaria una costante manutenzione che si vuole sia il più agevole possibile.

### 1.1.1 Origine

Il nome della startup, in hawaiiano, significa *isola*. Questo perché l'idea iniziale era una applicazione web per studenti dove un *"moku"* era visto come uno spazio personale (un'isola), ma allo stesso tempo uno spazio di collaborazione, rappresentando un *arcipelago* di isole collegate tra loro. In origine si basava sulla realizzazione di una applicazione il cui uso era riservato principalmente agli studenti. Ad ogni utente era data la possibilità di caricare i propri documenti in diverse tipologie di formato. Una volta aperti, era possibile prendere degli appunti su un layer posto sopra al documento caricato. Questo può essere condiviso con i propri collaboratori. Era anche possibile registrare la lezione e mettere delle note anche alla traccia audio, la quale si poteva sincronizzare con un particolare documento.

### 1.1.2 Evoluzione

Ora le attività dell'azienda si concentrano nella consulenza IT, in particolare nella realizzazione di software innovativo a supporto delle esigenze concrete dei clienti. Il progetto da me svolto è un chiaro esempio di questo mutamento, infatti il prodotto finito risulterà di completa proprietà di un'azienda esterna.
Ora l'azienda, quindi, sviluppa solo marginalmente software proprietario, ma sviluppa principalmente su commissione di aziende esterne.

## 1.2 Organizzazione del lavoro

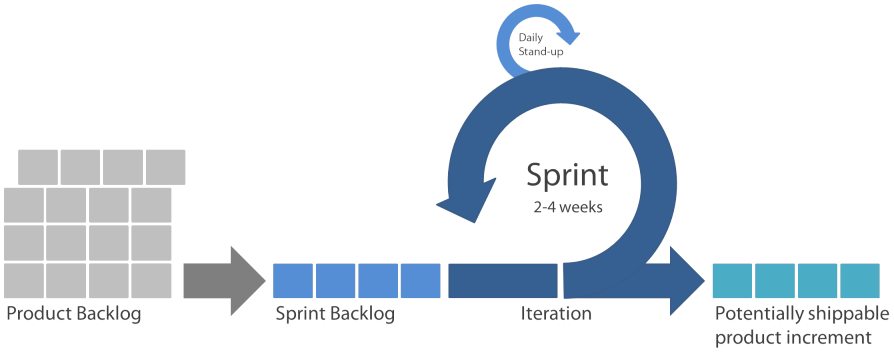



Figura 2: Metodo agile scrum.

All'interno dell'azienda il lavoro viene organizzato secondo il metodo agile scrum.
Il modello agile scrum prevede la suddivisione di un progetto in blocchi detti *Sprint* ognuno dei

quali deve produrre un avanzamento del software. Prevede, inoltre la suddivisione dei compiti in *task*, che possono essere visti come una attività abbastanza piccola da essere svolta da una sola persona nell'arco di 4/8 ore. È previsto un rapporto a stretto contatto con il cliente e la pianificazione di frequenti meeting con la funzione di punti di controllo, dove verificare lo stato di avanzamento del progetto rispetto a quanto atteso. È stato scelto questo modello poiché l'azienda sviluppa in maggioranza prodotti innovativi, quindi con tecnologie in evoluzione, e soggetti a continue variazioni dei requisiti rendendo necessaria un meccanismo di controllo flessibile. Nello specifico del progetto in analisi, la continua variazione delle norme di legge in vigore provoca continui mutamenti nell'implementazione del software ed un rapporto a stretto contatto con il cliente diventa indispensabile al fine di garantire la conformità del servizio.

## 1.3 Strumenti a supporto dell'organizzazione del lavoro

### 1.3.1 Atlassian Suite


Figura 3: Logo della suite Atlassian

Per l'organizzazione del lavoro, ci si è appoggiati alla suite offerta da Atlassian, che fornisce tutti gli strumenti necessari alla gestione di codice e compiti.
Fa seguito l'elenco degli strumenti utilizzati a supporto delle principali necessità.

### 1.3.2 Jira - Gestione dei Task e Sprint


Figura 4: Logo di Jira

Per la gestione dei *Task* e degli *Sprint* è stato utilizzato il software Jira della suite Atlassian. Esso permette di creare ed assegnare i task alle persone le quali possono annotare il numero di ore dedicate ad ogni specifico task. È possibile inoltre avere una visione d'insieme dello sprint corrente e dei precedenti, pianificando anche i successivi. È anche possibile collegare i task presenti su Jira con i *commit* presenti su Bitbucket facendo riferimento nel testo del *commit* al codice identificativo del *task* tenendo sotto controllo l'evoluzione del codice per ogni *task*.

### 1.3.3 Confluence - Gestione della documentazione


Figura 5: Logo di Confluence

Per la gestione della documentazione su norme aziendali e prodotti realizzati, viene utilizzato il software Confluence. Esso fornisce usa sorta di wiki integrato con tutte le applicazioni della suite collegata al progetto. Qui viene stesa la documentazione inerente a tutte le fasi del ciclo di vita del prodotto.

### 1.3.4 Bitbucket - Versionamento


Figura 6: Logo di Bitbucket

Per quanto riguarda il versionamento, viene utilizzato il servizio Bitbucket della suite. Esso utilizza `git` per la gestione del repository. Per agevolare la cooperazione è stato adottato l'approccio **git-flow**.

### 1.3.5 Slack - Comunicazione


Figura 7: Logo di Slack

Per quanto riguarda la comunicazione interna ai membri dell'azienda è stato scelto di utilizzare il software gratuito Slack. Si tratta di un sistema di messaggistica che distingue le comunicazioni dirette a persone da quelle dirette a canali inerenti ad un particolare argomento. Le norme aziendali prevedono la creazione di un canale per ogni progetto. Slack mette a disposizione anche un meccanismo di condivisione di file e, caratteristica essenziale, mantiene uno storico di tutti i messaggi di tutte le conversazioni al fine di mantenere un tracciamento delle comunicazioni interne. È fornita dal software anche una efficiente funzionalità di ricerca per recuperare le comunicazioni o i contenuti condivisi.

### 1.3.6 Errbit - Rilevazione degli errori

Figura 8: Logo di Errbit

Errbit è uno strumento per la rilevazione e gestione degli errori. Esso ha bisogno di un server dove girare e mantiene dei listener in ascolto dei messaggi di errore provenienti dall'applicazione alla quale è collegato. Per ogni errore rilevato, vengono resi disponibili su una pagina web le informazioni relative alla tipologia dell'errore, il backTrace, l'utente per il quale è avvenuto, ed i parametri sottomessi alla pagina in analisi. Sono inoltre esposte da questo framework delle API per AirBrake e Jira, in modo da favorirne l'integrazione. Nello stage è stato fatto uso di questo framework nel server in produzione per essere a conoscenza con esattezza delle criticità del prodotto realizzato in modo da essere reattivi nella risoluzione dei bug rilevati.

### 1.3.7 Airbrake - Gestione e notifica degli errori


Figura 9: Logo di Airbrake

Airbrake fornisce un sistema di notifiche push centralizzato. La sua caratteristica principale è quella di centralizzare la gestione delle notifiche permettendo un'accurata gestione dei flussi di spedizione dei messaggi e degli ambienti monitorati (sviluppo, staging, produzione).
È stato collegato Airbrake ad Errbit mediante le apposite API. Ad ogni eccezione catturata da ErrBit, AirBrake permette di visualizzare in una dashboard tutte le informazioni relative all'eccezione ed al contesto in cui essa è stata sollevata. È inoltre possibile collegare Airbrake ad altri strumenti ad esempio per la gestione di report periodici.
In questo progetto Airbrake è stato configurato in modo tale che le notifiche provenienti dal monitoraggio dell'ambiente di produzione siano spedite ad un canale di Slack appositamente creato.

# 2 Il progetto di Stage

Il progetto di stage consiste nell'estensione di un software web based, già sviluppato in azienda, a supporto delle aziende che intendono avvalersi dell'asseverazione in ambito della sicurezza sul lavoro, in particolare nel settore edilizio.
Con asseverazione si intende una scelta volontaria di una impresa edile al fine di dimostrare l'impegno per la prevenzione, salute e sicurezza nei luoghi di lavoro. Opera mediante la certificazione di conformità dei modelli di organizzazione e gestione aziendali e mediante visite a campione nei cantieri. L'asseverazione offre benefici economici alle aziende che la richiedono e garantisce efficacia esimente rispetto alla responsabilità amministrativa delle imprese.
Il software si pone come obiettivo la possibilità di fornire in ogni momento una fotografia dello stato della sicurezza in azienda, al fine di agevolarne il mantenimento nel tempo.
All'inserimento di nuove informazioni, verranno generate nuove domande, scadenze o vincoli. Se i risultati non sono conformi alle norme vigenti, verranno generati degli allarmi, che scompariranno solo nel momento in cui la violazione del vincolo che rappresentano non sia più verificata.
Dal punto di vista tecnico viene utilizzato un sistema esperto per mappare le norme in tema di sicurezza, individuare i punti deboli dell'azienda, ricordare al responsabile le scadenze, conservare ed indicizzare il patrimonio documentale in ambito sicurezza. Il sistema ha una funzione proattiva e dinamica, variando gli allarmi in base alla variazione delle norme e allo stato dell'azienda (es. al crescere o diminuire del numero dei dipendenti, al variare della superficie delle sedi aziendali, dei cantieri...). La webapp si comporta quindi come un consulente digitale per l'azienda, aiutandola a mantenere aggiornato il suo modello di sicurezza e a rispettarlo. Questo le permette all'azienda asseverata di accedere a significativi premi INAIL, di ottenere punti in graduatoria in bandi pubblici e di sollevare il datore di lavoro da responsabilità penali in caso di infortuni legati ad una mala gestione della sicurezza.
La filosofia generale del progetto prevede sia sempre possibile inserire i dati senza blocchi dettati dai vincoli di legge, in modo da evidenziare eventuali non conformità con allarmi al fine di risolverle.

## 2.1 Il prodotto esistente

Il progetto è in una fase di sviluppo avanzato (versione alpha). Alla data di inizio dello stage, il software supporta le aziende con i codici ATECO$_G$ in ambito edilizio, permette l'inserimento di informazioni su sedi, cantieri, dipendenti, organigramma aziendale e la gestione di abitabilità, certificato prevenzione incendi e della documentazione che emerge dal Documento di Valutazione dei Rischi (DVR)$_G$. Vengono poste all'utente più di 400 domande per individuare lo stato di sicurezza.
Il sistema esperto è funzionante, ma va irrobustito e vanno implementate le regole opportune per individuare le criticità nei dati inseriti dall'utente.

## 2.2 Obiettivi dello stage

Il team ha l'obiettivo di rilasciare una versione beta del software entro la fine del 2015 e quindi procedere al primo test con l'utente finale, una importante azienda del settore edilizio, già individuata. Durante lo stage sono stati posti i seguenti obiettivi:

- Analisi di alcune logiche del sistema esperto in coordinamento con il cliente ed il consulente della sicurezza;
- Progettazione delle funzionalità individuate nella fase di analisi;
- Implementazione del risultato del punto precedente;
- Stesura della documentazione relativa al codice prodotto;
- Rendere il sistema più efficace, semplice ed usabile da parte di un utente competente in materia di sicurezza.

## 2.3 Pianificazione del lavoro

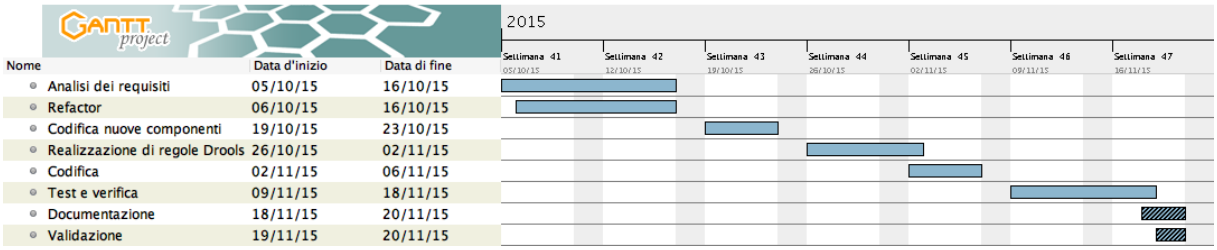



Figura 10: Diagramma di Gantt.

TODO Scrivo le cose relative al Gantt

Durante lo stage, le ore sono state mantenute tali alle previsioni, la loro distribuzione nel tempo però è cambiata. Nello specifico l'ordine di esecuzione con le relative ore impiegate sono state le seguenti:

1. 40 ore: studio di Ruby on Rails, Drools e del software esistente;
2. 20 ore: rilevamento delle criticità sulle componenti esistenti ;
3. 30 ore: analisi dei requisiti
4. 15 ore: riprogettazione delle componenti esistenti che presentano criticità
5. 20 ore: progettazione delle nuove entità richieste;
6. 80 ore: codifica di funzionalità e test;
7. 4 ore: colloquio con il committente per validare le nuove modifiche;
8. 4 ore: riprogettazione delle componenti da modificare;
9. 22 ore: codifica di funzionalità e test;
10. 40 ore: implementazione delle regole Drools;
11. 15 ore: stesura della documentazione;
12. 15 ore: verifica.

Il numero totale delle ore di stage è 305.

cose da dire: analisi e codifica spezzata in piu parti rispetto al previsto Come è possibile osservare dalla lista, l'attività di progettazione è stata divisa in due parti distinte: la prima relativa alle componenti già presenti al momento dell'inizio dello stage, la seconda relativa alle nuove componenti. aumento del tempo dedicato ad analisi e codifica dovuto a modifiche committente => sacrificati requisiti opzionali i test sono stati scritti durante la codifica

Alla fine della prima settimana è iniziata l'attività di analisi dei requisiti

# 3 Analisi del prodotto esistente

Al momento dell'arrivo in azienda era già presente una versione alpha del software.
Le aziende che il prodotto si prefigge di supportare in questa fase, sono quelle con i codici ATECO$_G$ in ambito edilizio. In particolare, al mio arrivo erano gestite le informazioni relative a:

- Sedi;
- Cantieri;
- Dipendenti;
- Organigramma aziendale;
- Abitabilità;
- Certificato di prevenzione degli incendi;
- DVR$_G$ e documentazione collaterale ad esso.

Il software esponeva una collezione di oltre 400 domande. Sulla base delle risposte a queste domande ed alle informazioni relative alle entità sopra indicate, venivano verificati alcuni vincoli mediante regole del sistema esperto.

## 3.1 Architettura del software

### 3.1.1 Architettura ad alto livello

Il software è gestito mediante tecnologia Software as a Service (SaaS)$_G$, un modello di distribuzione del software applicativo dove il fornitore del software si occupa della sua implementazione e manutenzione. Il servizio viene erogato al cliente mediante una applicazione web fruibile via internet.
L'applicazione web risiede fisicamente su una macchina virtuale della piattaforma cloud Amazon Web Services (AWS)$_G$, al fine di evitare oneri e spese di gestione di una infrastruttura informatica dedicata e garantire la scalabilità del servizio.

Come è possibile osservare nella Figura 11, ogni istanza di macchina virtuale contiene un clone dell'intera architettura. È stata scelta questa soluzione perché è prevista la possibilità di vendere il pacchetto a più utenti che possono fungere da distributori.
Ogni istanza è composta da quattro componenti principali:

- *Rule Engine;*
- *Database;*
- *Web Application;*
- *Storage.*

La componente *Storage*, in particolare, è stata dedicata al salvataggio di documenti in formato PDF esportabili in ogni momento da ogni azienda. Questa funzionalità non è ancora stata implementata.
Il routing delle richieste viene effettuato con un Reverse proxy$_G$, nello specifico *NGINX*.
In particolare Ruby on Rails permette nativamente di creare e versionare i database a seconda dell'ambiente nel quale si opera (*development, test, staging e production*). Questa caratteristica è tornata molto utile per lo sviluppo, dal momento che è stato possibile fare test accurati senza intaccare i dati in produzione. Inoltre ciò rende possibile evitare di utilizzare la console in sandbox di Rails, la quale presenta criticità nella consistenza dei dati visibili da shell ripetto a quelli visibili da browser$_G$.
La persistenza dei dati è stata gestita mediante il modulo `ActiveRecord` di Ruby on Rails (descritto nella sezione 5.3.1) il quale salva le informazioni su un database *PostgreSQL*.

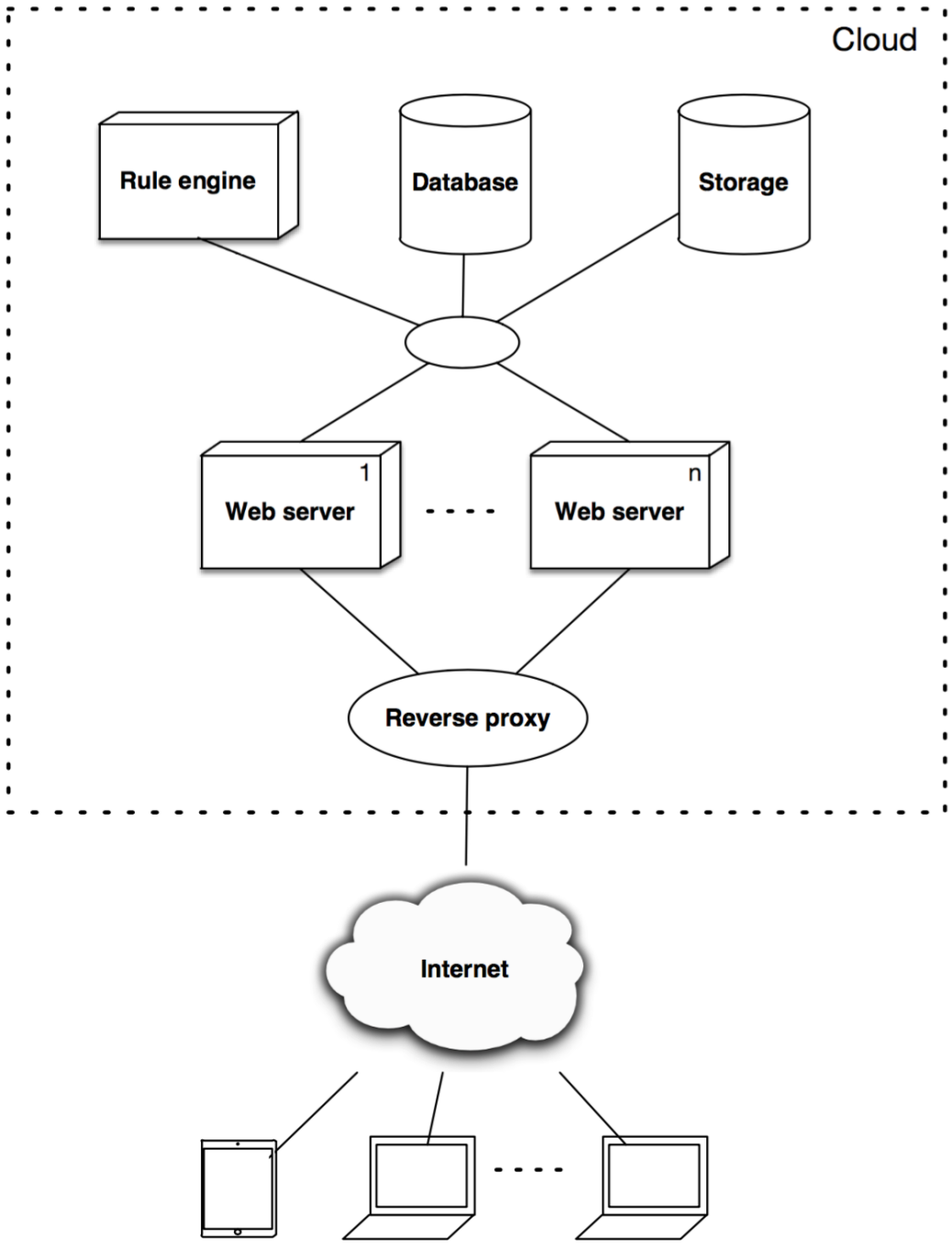



Figura 11: Architettura del sistema

### 3.1.2 Architettura a basso livello

L'applicazione web è organizzata secondo il pattern Model View Controller (MVC)$_G$. Per il raggiungimento delle viste e l'accesso alle informazioni necessarie al corretto funzionamento dell'applicazione è stata implementata una interfaccia REST$_G$.
Il sistema ha come entità principale il modello `Company` al quale sono riferite, direttamente o indirettamente, tutte le risorse.

Sono poi presenti numerosi modelli relativi alle entità che partecipano alla procedura di Asseverazione$_G$. I modelli più significativi sono:

- `Alert` per rappresentare gli allarmi;
- `Answer` per rappresentare le risposte alle domande;
- `Company` per rappresentare una azienda;
- `ConstructionSite` per rappresentare un cantiere;
- `Dpi` per rappresentare un dispositivo di protezione individuale;
- `Duty` per rappresentare una mansione;
- `FireExtinguisher` per rappresentare un estintore;
- `FirstAidBox` per rappresentare una cassetta di primo soccorso;
- `Individual` per rappresentare una persona;
- `Location` per rappresentare un edificio aziendale, ovvero una sede operativa, una sede amministrativa, una sede legale oppure un magazzino;
- `Machine, LiftingEquipment, ElectricTool` per rappresentare un mezzo oppure uno strumento presente nel parco macchine;
- `MedicalVisit` per rappresentare una visita medica;
- `Procedure` per rappresentare una procedura aziendale, sia essa di prassi o di sistema;
- `Question` per rappresentare una domanda;
- `Training` per rappresentare un corso.

Per ciascun modello, sono stati realizzati opportuni *controller* e *viste*.
Sono stati implementati, inoltre numerosi Concern$_G$ per modularizzare le funzionalità indipendenti dalla singola classe ed allo stesso tempo riutilizzabili da altre classi. Ad esempio, ogni modello che, se istanziato, provoca un aumento del numero delle domande in attesa di risposta, include un apposito Concern$_G$ per l'aggiornamento di un contatore dedicato a tale scopo. Per il calcolo del valore, il Concern$_G$ ricava il nome della classe dell'oggetto corrente mediante Reflection$_G$ ed incrementa automaticamente il valore del numero di domande correlate al tipo individuato.

### Relazioni tra domande, risposte ed allarmi

Per facilitare la comprensione di alcune delle soluzioni descritte di seguito è importante essere a conoscenza delle relazioni presenti tra domande, risposte ed allarmi.
Le risposte sono direttamente collegate alle domande ed ad una entità.
Quando un utente risponde ad una domanda, viene aggiornato il recod relativo a quella risposta.
A seguito di un inserimento o aggiornamento di una risposta, interviene Drools che valuta se le informazioni inserite rispettano tutti i vincoli previsti, altrimenti viene sollevato un allarme.
Per questioni di efficienza, gli allarmi relativi al vincolo di presenza di una risposta, vengono gestiti direttamente dalle funzionalità di validazione di Rails.
Come per le risposte, anche gli allarmi sono sempre collegati ad una risorsa, che di default è l'azienda corrente, ma può assumere come valore una qualsiasi istanza di un modello, purché non sia nulla. Particolarmente degna di nota è l'associazione di una risposta o di un *allarme* alla relativa risorsa. Per fare ciò si è utilizzato l'approccio standard di Rails per il supporto alle associazioni polimorfe.
Come si può vedere da Figura 12, indicando il tipo e l'id della risorsa interessata, l'accesso avviene tramite valutazione della classe e ricerca del relativo *id*.
Si può pensare, ad esempio, all'allarme scatenato alla scadenza di un estintore.
La Risposta (*Answer*) avrà i due campi `answerable_type` ed `answerable_id` impostati rispettivamente a *"FireExtinguisher"* ed al suo *id*. L'allarme avrà un riferimento alla domanda per la quale è stata data una risposta. In questo modo è possibile conoscere il punto esatto dove dirottare l'utente al click sull'allarme per raggiungere il punto di non conformità. Questo aspetto è stato considerato di fondamentale importanza poiché la mole delle informazioni nel software è molto grande, è quindi necessario fornire il maggior numero di strumenti possibili all'utente per facilitarne la navigazione e l'orientamento nel sistema.

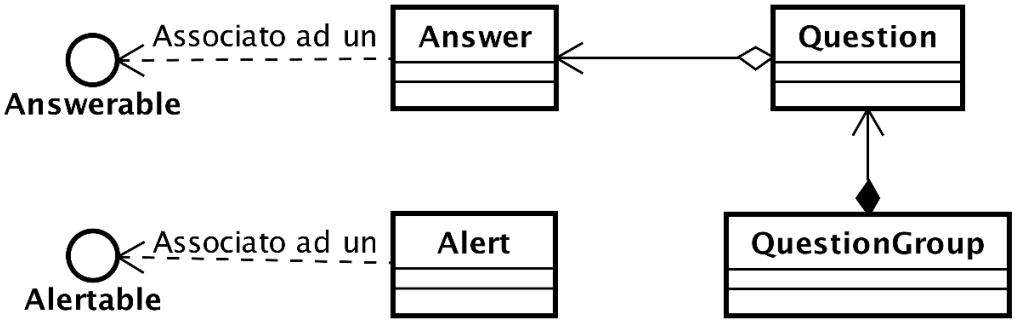



Figura 12: Diagramma delle classi delle associazioni polimorfe di Risposte ed Allarmi.

### 3.1.3 Integrazione tra Ruby on Rails e Drools

Il codice è stato scritto per la maggior parte utilizzando Ruby on Rails ma il rule engine (Drools) è un framework$_G$ scritto in Java. I due linguaggi non sono nativamente compatibili. Per ovviare a questo problema, è stato utilizzato JRuby, un interprete del linguaggio Ruby scritto in Java, quindi in esecuzione su una Java Virtual Machine (JVM)$_G$.
Per il corretto funzionamento di Drools, è necessario inserire le informazioni nella *Working Memory* come *"fatti"* che sono richiesti come oggetti di tipo JavaBean$_G$ o POJO$_G$.
Per far cooperare i due ambienti, è stato implementato un apposito Concern$_G$ chiamato `"act_as_fact"`. Questo modulo viene incluso nelle classi delle quali è necessario tenere traccia nella *Working Memory* e vengono generati, mediante Reflection utilizzando i template di Rails (ERB$_G$), le classi Java corrispondenti.

### 3.1.4 Flusso dei dati ed interazione

Un aspetto che merita di essere esaminato è il flusso con il quale vengono generate e valutate domande e risposte .

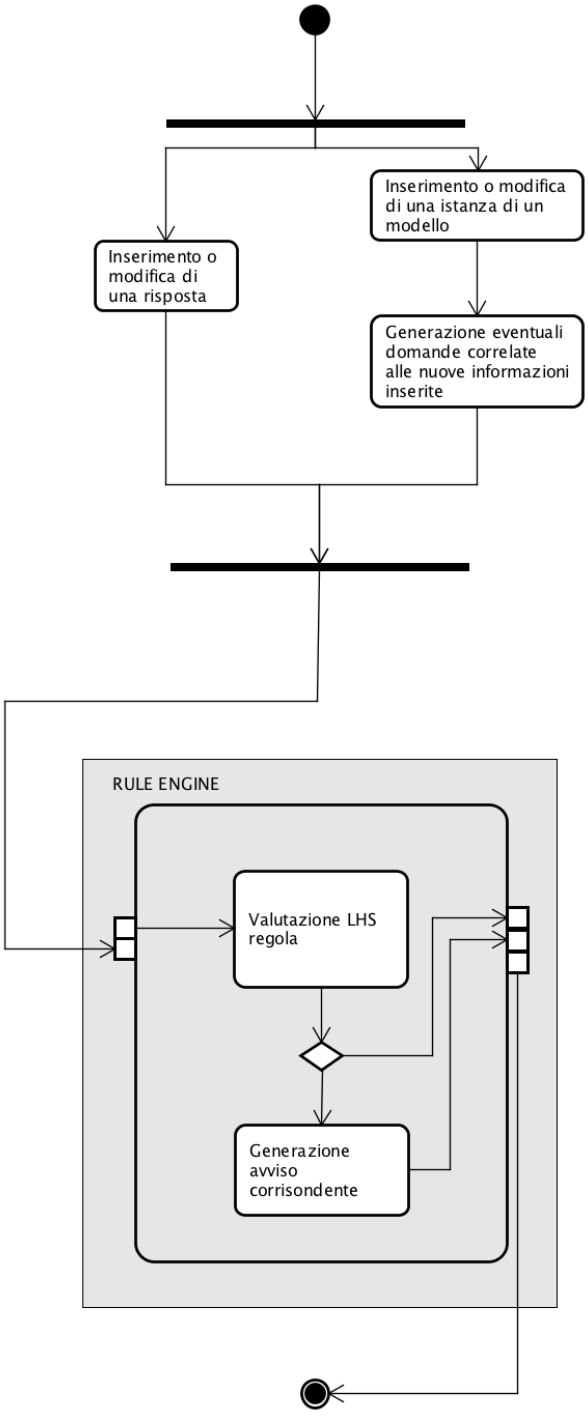



Figura 13: Diagramma di attività del flusso di una risposta o l'inserimento di un oggetto a modello

Come è possibile osservare dalla Figura 13, nel momento in cui un utente inserisce o modifica un'istanza di un modello oppure risponde ad una domanda, viene aggiornata la *Working Memory*.
Nel caso un cui venga generata o aggiornata un'istanza di modello, può essere necessario l'in-

serimento di alcune domande ad essa direttamente correlate.
Un esempio è rappresentato dall'aggiunta di un estintore ad un cantiere. Ad ogni estintore in ogni cantiere deve essere associata una posizione specifica che deve essere riportata nel layout di cantiere per permetterne il facile reperimento in caso di incendio. La posizione nel layout di un estintore è rappresentata dal software come una risposta ad una domanda in un apposito questionario generato ad ogni associazione di un estintore ad un cantiere. Se tale informazione non è specificata viene sollevato un allarme.
Sia per l'inserimento o aggiornamento di una risposta, sia per la generazione o modifica di una istanza di modello, il passo successivo è rappresentato dalla valutazione delle informazioni inserite sulla base della *Knowledge Base*.
È in questo momento che agisce il rule engine Drools che per ogni regola presente nella *Knowledge Base* valuta la $\text{LHS}_G$ sulle nuove informazioni inserite e, se soddisfatta, solleva gli allarmi corrispondenti.
Per maggiori dettagli sulle regole, si rimanda alla sezione 6.3.

# 4 Analisi dei requisiti

## 4.1 Definizione dei casi d'uso

Al momento dell'arrivo in azienda, molte componenti e funzionalità erano già parzialmente sviluppate. Alcune di esse necessitavano di adattamenti strutturali per rendere il software aderente alle norme vigenti, altre invece necessitavano di una ristrutturazione dell'interfaccia utente.
Per la maggior parte del tempo si è sviluppato lato back-end ma, dal momento che l'obiettivo dello stage stato consolidare il sistema per permettere il rilascio di una versione alpha, sono stati trattati anche aspetti sia lato front-end. Sono stati dunque riportati nei casi d'uso tutti gli aspetti toccati durante lo stage, specificando nella descrizione le modifiche che sono state apportate alla componente interessata.
I casi d'uso per i quali è stata eseguita un riprogettazione sono stati differenziati da quelli per i quali era necessaria una soluzione da zero mediante il colore di sfondo.

### 4.1.1 Legenda

Sono stati rappresentati con sfondo azzurro i casi d'uso per i quali è stato eseguita una riprogettazione; quelli su sfondo bianco invece hanno richiesto progettazione e sviluppo di una soluzione da zero.

- *Notazione di caso d'uso relativo a componenti progettate da zero:*


Figura 14: Notazione di caso d'uso relativo ad una componenti progettate da zero;

- *Caso d'uso relatico a vomponenti riprogettate dal punto di vista strutturale o dell'interfaccia utente:*


Figura 15: Notazione di caso d'uso relativo a componenti riprogettate.

### 4.1.2 Attori

nuova sezione

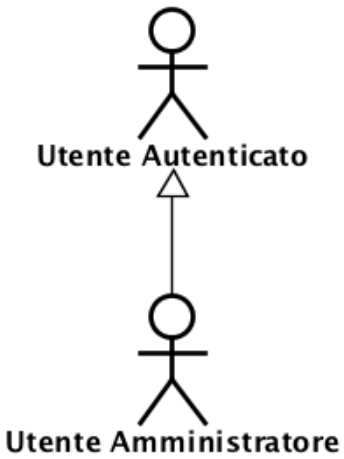



Figura 16: Attori coinvolti.

Sono state individuate due categorie di attori:

- `Utente amministratore:` Utente che ha effettuato con successo il login nel sistema con privilegi da amministratore;
- `Utente autenticato:` Utente che ha effettuato con successo il login nel sistema.

L'`utente amministratore` specializza l'`utente autenticato`.

### 4.1.3 Caso d'uso UCP - Scenario Principale

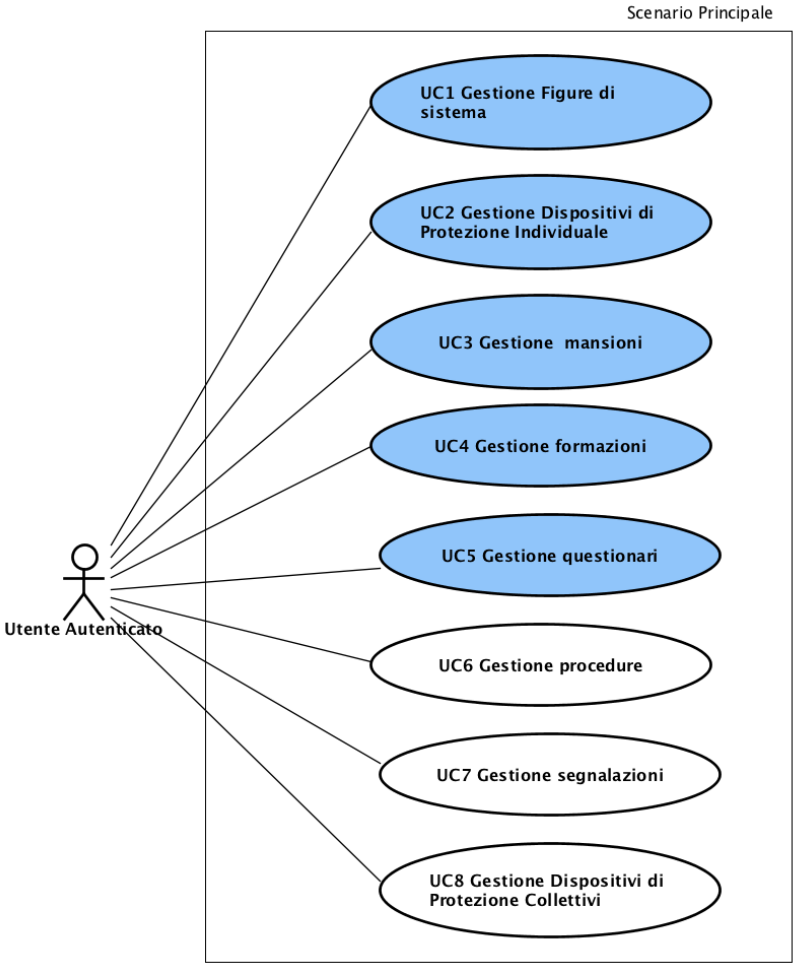



Figura 17: Diagramma dei casi d'uso generale.

- **Scopo:** Il diagramma generale dei casi d'uso UCP (Figura 17), ha lo scopo di rappresentare ad alto livello i casi d'uso necessari al soddisfacimento di tutti i requisiti.
  In particolare un utente che abbia effettuato con successo il login nel sistema, deve poter gestire: figure di sistema; dispositivi di protezione individuali, mansioni, formazioni, questionari, procedure; segnalazioni e dispositivi di protezione collettivi. A seguire la spiegazione e l'esplosione in sotto casi d'uso per ognuno dei casi d'uso sopra elencati.
- **Attori Coinvolti:** Utente Autenticato;
- **Flusso principale degli eventi:**
  - *L'utente autenticato gestisce le Figure di sistema (UC1);*
  - *L'utente autenticato gestisce i Dispositivi di Protezione Individuale (UC2);*
  - *L'utente autenticato gestisce le mansioni (UC3);*
  - *L'utente autenticato gestisce le formazioni (UC4);*
  - *L'utente autenticato gestisce i questionari (UC5);*
  - *L'utente autenticato gestisce le procedure (UC6);*
  - *L'utente autenticato gestisce le segnalazioni (UC7)*
  - *L'utente autenticato gestisce i Dispositivi di Protezione Collettivi (UC8).*

### 4.1.4 UC1 Gestione delle Figure di sistema

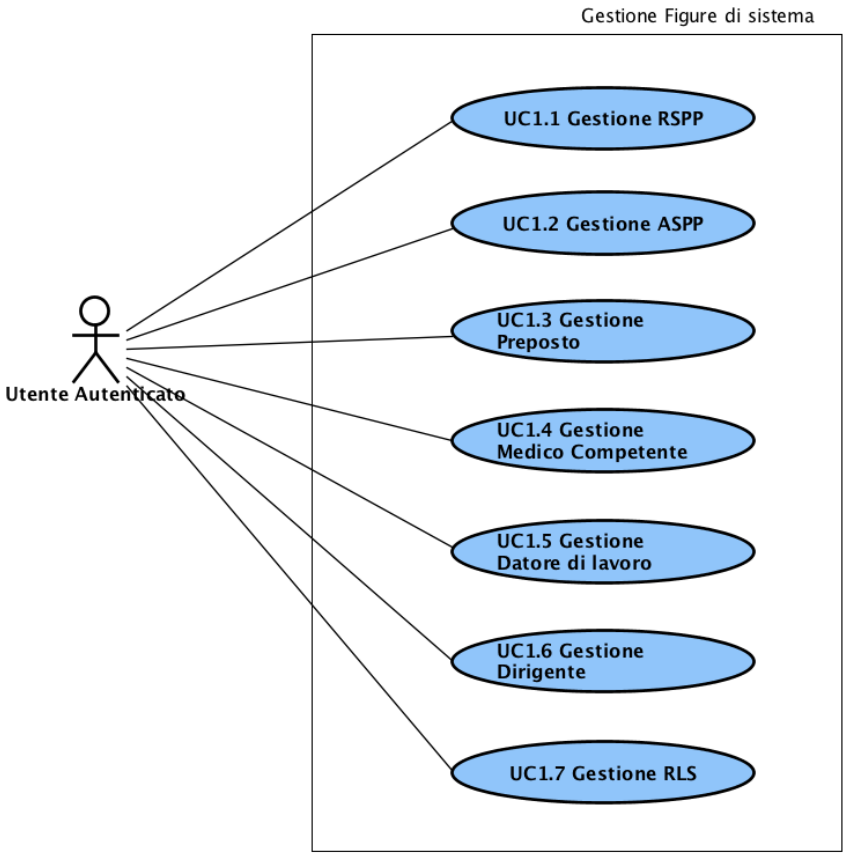



Figura 18: Diagramma dei casi d'uso UC1 - Figure di sistema.

- **Scopo:** Il diagramma generale dei casi d'uso UC1 (Figura 18), ha lo scopo di rappresentare ad alto livello i casi d'uso necessari al soddisfacimento di tutti i requisiti rigruardanti le figure di sistema;
- **Attori Coinvolti:** Utente Autenticato;
- **Flusso principale degli eventi:**
  - *L'utente autenticato gestisce gli Responsabile del Servizio di Prevenzione e Protezione (RSPP)$_G$ (UC1.1);*
  - *L'utente autenticato gestisce gli Addetto al Servizio di Prevenzione e Protezione (ASPP)$_G$ (UC1.2);*
  - *L'utente autenticato gestisce i preposti (UC1.3);*
  - *L'utente autenticato gestisce il medico competente (UC1.4);*
  - *L'utente autenticato gestisce il datore di lavoro (UC1.5);*
  - *L'utente autenticato gestisce i dirigenti (UC1.6);*
  - *L'utente autenticato gestisce gli Rappresentante dei Lavoratori per la sicurezza (RLS)$_G$ (UC1.7).*

### 4.1.5 UC1.1 Gestione degli RSPP

Tutte le figure di sistema presentano requisiti analoghi. Per evitare di affaticare la lettura con sezioni ridondanti è stato scelto di riportare soltanto il diagramma dei casi d'uso riguardante gli RSPP$_G$. Si intende che per tutte le altre figure di sistema il diagramma sia analogo.

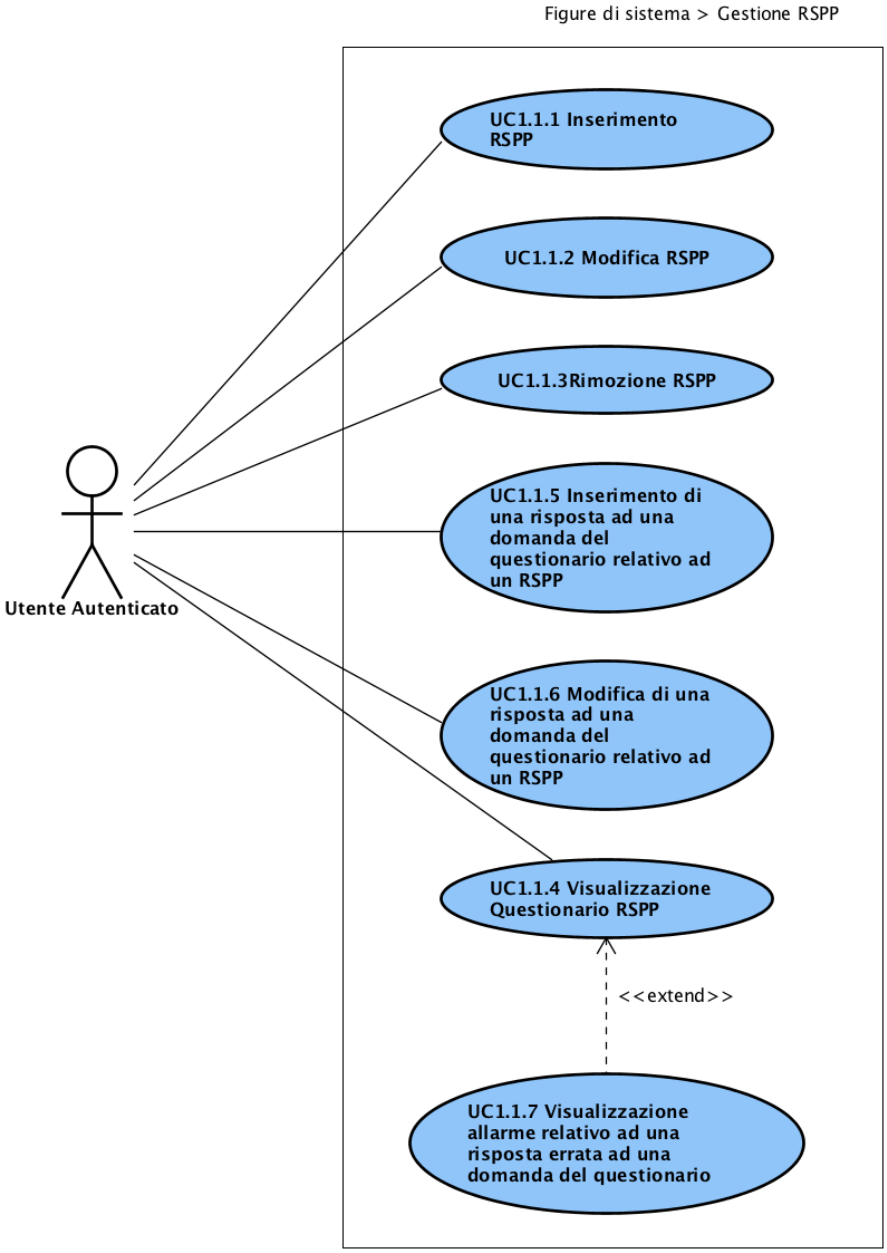



Figura 19: Diagramma dei casi d'uso relativo all'inserimento degli RSPP

- **Scopo:** Il diagramma relativo alla gestione degli RSPP (Figura 17), ha lo scopo di rappresentare i casi d'uso necessari al soddisfacimento di tutti i requisiti riguardanti la gestione degli RSPP.
  In particolare un utente che abbia effettuato con successo il login nel sistema, deve poter inserire, rimuovere, modificare RSPP$_G$. Deve essere messo in condizione inoltre di Visualizzare un questionario relativo ad uno specifico RSPP$_G$, rispondendo alle domande riportate su di esso. In caso di errore, deve essere sollevato un allarme che comunica la domanda esatta ed un messaggio esplicativo.

- **Attori Coinvolti:** Utente Autenticato;
- **Flusso principale degli eventi:**
  - *L'utente autenticato inserisce un $RSPP_G$ (UC1.1.1);*
  - *L'utente autenticato modifica un $RSPP_G$ (UC1.1.2);*
  - *L'utente autenticato rimuove un $RSPP_G$ (UC1.1.3);*
  - *L'utente autenticato visualizza il questionario associato ad un $RSPP_G$ (UC1.1.4);*
  - *L'utente autenticato inserisce una risposta ad una domanda del questionario (UC1.1.5);*
  - *L'utente autenticato modifica di una risposta ad una domanda del questionario relativo ad un $RSPP_G$ (UC1.1.6);*
  - *L'utente autenticato visualizza un allarme relativo ad una risposta errata ad una domanda del questionario (UC1.1.7).*

### 4.1.6 UC2 Gestione dei DPI

La gestione dei Dispositivi di Protezione Individuale $(DPI)_G$ va suddivisa in due scenari distinti. Il primo scenario è relativo alla gestione delle tipologie di $DPI_G$ disponibili da parte degli utenti amministratori. Il secondo scenario riguarda la gestione dei $DPI_G$ che compongono la dotazione personale di un dipendente.
Seguono i diagrammi dei casi d'uso dei $DPI_G$ relativi agli scenari sopra indicati.

#### UC2.1 Gestione delle tipologie di DPI dal pannello di amministrazione

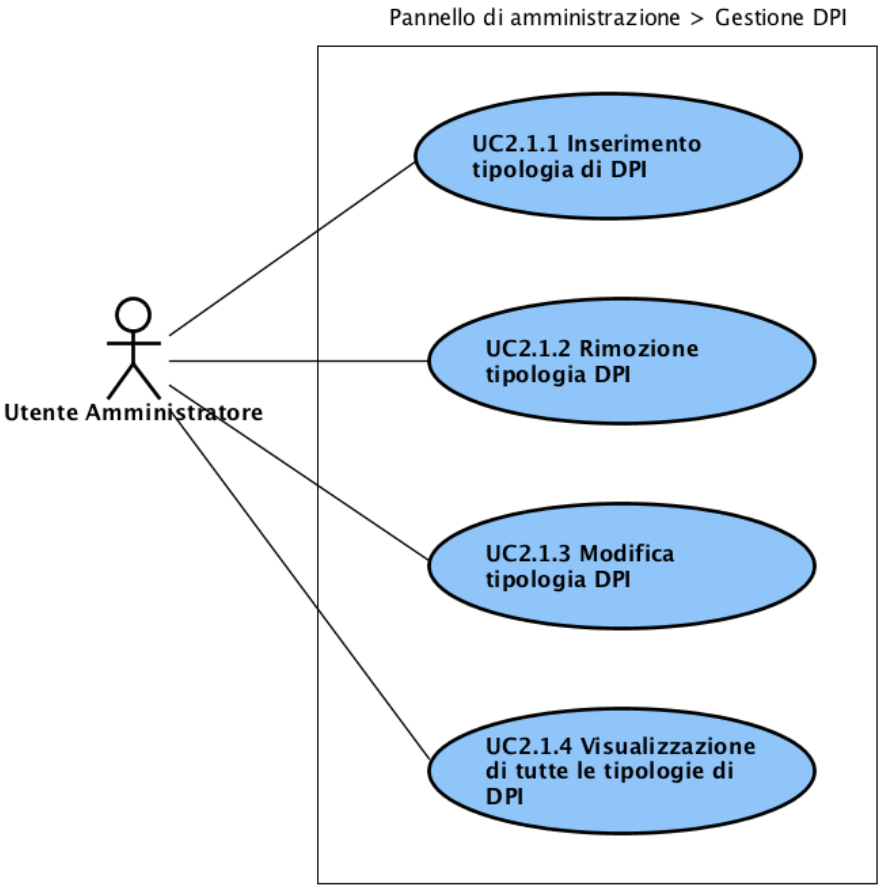



Figura 20: Diagramma dei casi d'uso UC2.1 - Gestione tipologie di DPI dal pannello di amministrazione.

- **Scopo:** Il diagramma presentato nella Figura 20, ha lo scopo di rappresentare i casi d'uso necessari al soddisfacimento di tutti i requisiti riguardanti la gestione delle tipologie dei $DPI_G$.
  Un utente amministratore deve poter gestire in modo autonomo le tipologie di $DPI_G$ che ritiene più opportune e metterle a disposizione degli utilizzatori del sistema.
- **Attori Coinvolti:** Utente Amministratore;
- **Flusso principale degli eventi:**
  - *L'utente amministratore inserisce una tipologia di $DPI_G$ (UC2.1.1);*
  - *L'utente amministratore rimuove una tipologia di $DPI_G$ (UC2.1.2);*
  - *L'utente amministratore modifica una tipologia di $DPI_G$ (UC2.1.3);*
  - *L'utente amministratore visualizza tutte le tipologie di $DPI_G$ (UC2.1.3).*

### UC2.2 Gestione dei DPI assegnati ai dipendenti

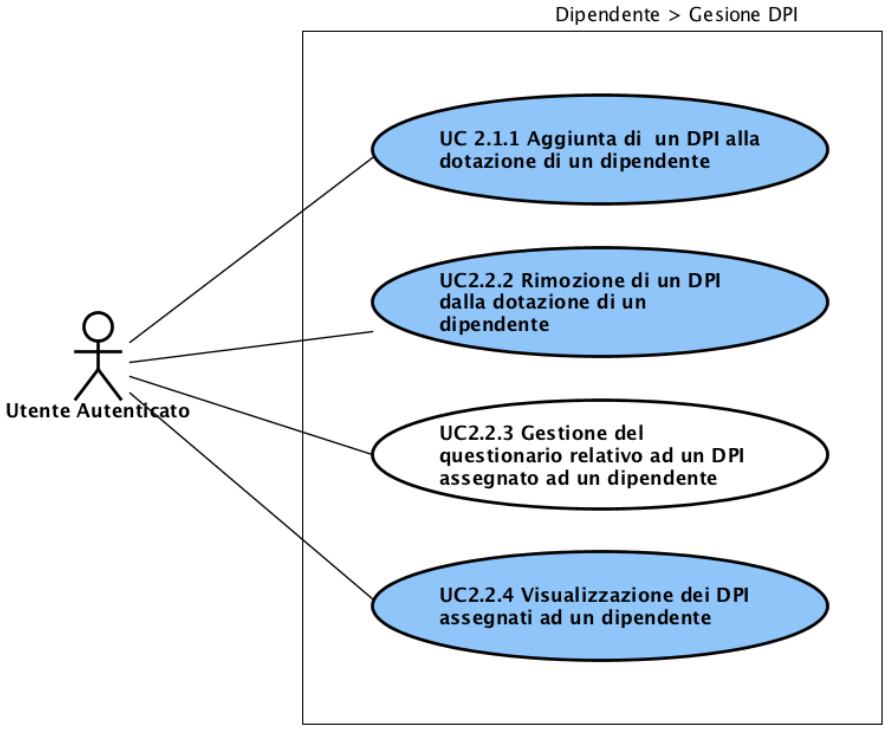



Figura 21: Diagramma dei casi d'uso UC2.2 - Gestione dei DPI relativi ad un dipendente.

- **Scopo:** Il diagramma presentato nella Figura 21, ha lo scopo di rappresentare i casi d'uso necessari al soddisfacimento di tutti i requisiti riguardanti la gestione dei $\text{DPI}_G$ dati in dotazione ai dipendenti.
  Un utente autenticato deve poter assegnare o rimuovere $\text{DPI}_G$ ad ogni dipendente. Deve essere possibile assegnare $\text{DPI}_G$ con molteplicità variabile, scegliendo la tipologia da una lista predefinita (Sezione: 4.1.6).
  Per ogni $\text{DPI}_G$ deve essere possibile rispondere ad un questionario;
- **Attori Coinvolti:** Utente Autenticato;
- **Flusso principale degli eventi:**
  - *L'utente autenticato aggiunge un $\text{DPI}_G$ alla dotazione di un dipendente (UC2.2.1);*
  - *L'utente autenticato rimuove un $\text{DPI}_G$ dalla dotazione di un dipendente (UC2.2.2);*
  - *L'utente autenticato gestisce il questionario relativo ad un $\text{DPI}_G$ assegnato ad un dipendente (UC2.2.3);*
  - *L'utente autenticato visualizza tutti i $\text{DPI}_G$ assegnati ad un dipendente (UC2.2.4).*

### 4.1.7 UC3 Gestione delle mansioni

Le mansioni rappresentano le attività che un individuo interno all'azienda è abilitato a svolgere.
L'associazione agli individui e la gestione delle mansioni disponibili è stato eseguito allo stesso modo dei $DPI_G$.
Per evitare di affaticare la lettura con sezioni ridondanti è stato scelto di riportare soltanto il diagramma dei casi d'uso riguardante i $DPI_G$. Si intende che per le mansioni il diagramma e la spiegazione del caso d'uso sia analogo alla sezione 4.1.6.

### 4.1.8 UC4 Gestione delle formazioni

Con il termine formazione si intende una certificazione relativa ad un corso abilitante ad una o più mansioni.
La gestione delle formazioni è del tutto simile a quella dei $DPI_G$, fatta eccezione per la gestione delle scadenze. È richiesto infatti che un utente amministratore possa assegnare un periodo di validità della formazione in mesi dal pannello di controllo. Un utente autenticato deve visualizzare un allarme nel momento in cui una formazione sia scaduta.

#### UC4.1 Gestione delle formazioni dal pannello di amministrazione

Tutte le considerazioni della sezione 4.1.6 valgono anche per le formazioni.
A differenza dei $DPI_G$ va inserita una informazione aggiuntiva obbligatoria: il periodo di validità della formazione.
Il diagramma dei casi d'uso è del tutto analogo a quello della Figura 20.

#### UC4.2 Gestione delle formazioni dei dipendenti e del datore di lavoro

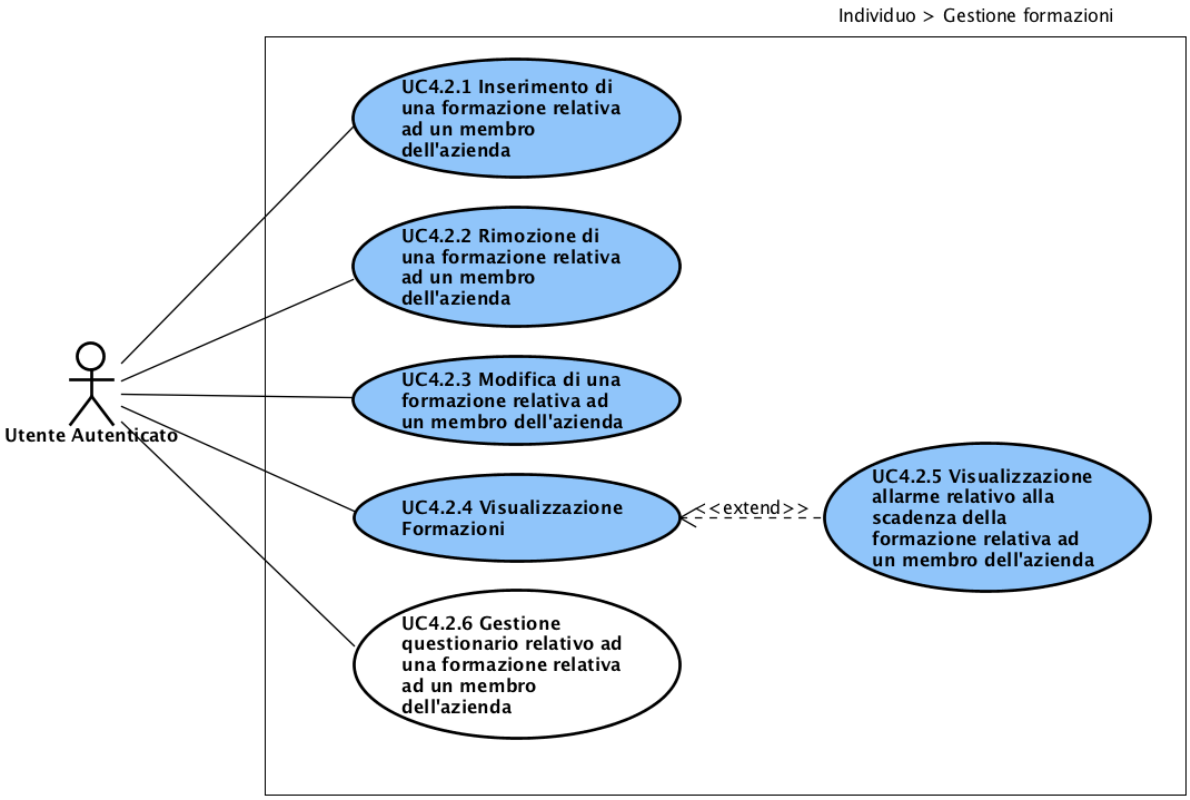



Figura 22: Diagramma dei casi d'uso relativo alla gestione delle formazioni di un dipendente.

- **Scopo:** Il diagramma presentato nella Figura 22, ha lo scopo di rappresentare i casi d'uso necessari al soddisfacimento di tutti i requisiti riguardanti la gestione delle formazioni relative ad un qualunque membro dell'azienda.
  Un utente autenticato deve poter assegnare o rimuovere formazioni ad ogni dipendente.
  Ogni formazione relativa ad un dipendente deve essere dotata di un questionario.
  La scadenza della formazione deve essere segnalata trenta giorni prima della data ultima mediante un allarme.
- **Attori Coinvolti:** Utente Autenticato;
- **Flusso principale degli eventi:**
  - *L'utente autenticato aggiunge una formazione ad un membro dell'azienda indicando descrizione e data (UC4.2.1);*

- *L'utente autenticato rimuove una formazione ad un membro dell'azienda (UC4.2.2);*
- *L'utente autenticato modifica una formazione ad un membro dell'azienda (UC4.2.3);*
- *L'utente autenticato visualizza le formazioni relative ad un membro dell'azienda (UC4.2.4);*
- *L'utente autenticato visualizza l'allarme relativo alla scadenza di una formazione in possesso di un membro dell'azienda (UC4.2.5);*
- *L'utente autenticato gestisce il questionario relativo ad una formazione in possesso di un membro dell'azienda (UC4.2.6).*

### 4.1.9 UC5 Gestione dei questionari

TODO Rimuovere e sistemare tutta la numerazione degli uc successivi + riferimenti nella tabella dei requisiti

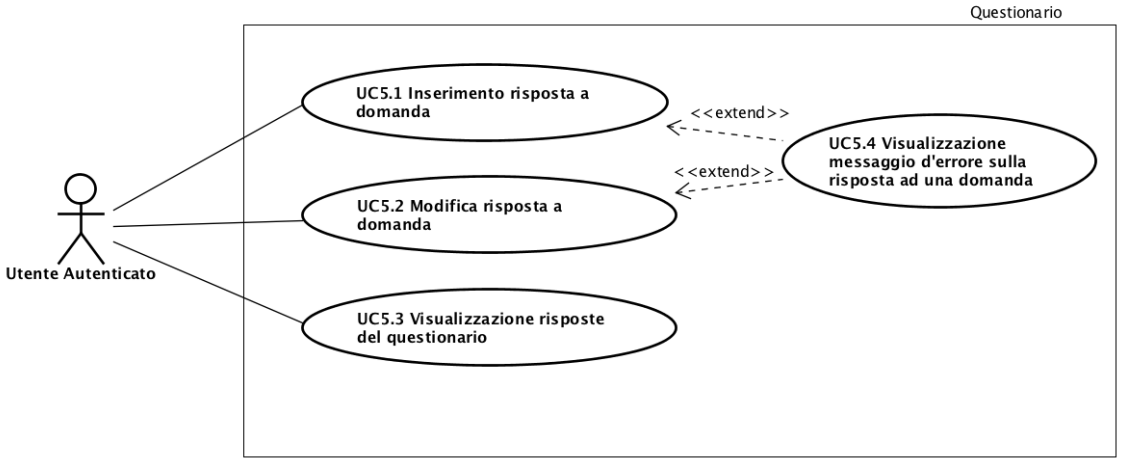



Figura 23: Diagramma dei casi d'uso relativo alla gestione di un questionario.

- **Scopo:** Il diagramma presentato nella Figura 23, ha lo scopo di rappresentare i casi d'uso necessari al soddisfacimento dei requisiti riguardanti la gestione di un questionario di pertinenza di un utente autenticato.
  Gli utenti amministratori devono poter gestire le domande poste nei questionari da un pannello d'amministrazione. Questo aspetto non è stato trattato in dettaglio poiché analogo a quanto descritto nella sezione 4.1.6.
- **Attori Coinvolti:** Utente Autenticato;
- **Flusso principale degli eventi:**
  - *L'utente autenticato risponde per la prima volta ad una domanda (UC5.1);*
    - *Viene mostrato un messaggio d'errore se la risposta fornita non è corretta (UC5.4);*
  - *L'utente autenticato modifica una risposta ad una domanda (UC5.2);*
    - *Viene mostrato un messaggio d'errore se la risposta fornita non è corretta (UC5.4);*
  - *L'utente autenticato visualizza tutte le risposte al questionario (UC5.3). Se è la prima volta che lo apre, le risposte saranno tutte vuote.*

### 4.1.10 UC6 Gestione delle procedure

Le procedure rappresentano un aspetto di primaria importanza e si suddividono in due categorie: di prassi e di sistema.
Le procedure di prassi derivano direttamente da un documento fornito dall'Istituto Nazionale per l'Assicurazione contro gli Infortuni sul Lavoro (INAIL)$_G$. Queste procedure sono state individuate dall'analisi delle cause di infortunio maggiormente frequenti rilevate dall'INAIL$_G$ nel tempo allo scopo di migliorare la sicurezza dei lavoratori. Le procedure di sistema, invece, sono stese dall'azienda e fanno parte del DVR$_G$.

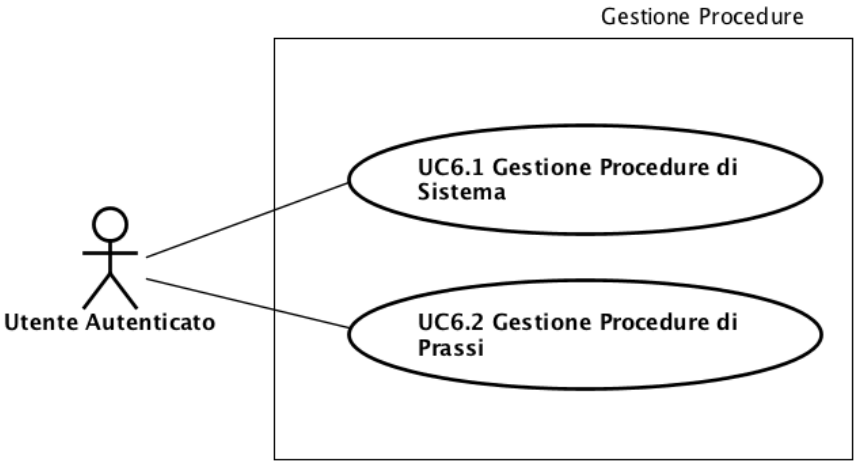



Figura 24: Diagramma dei casi d'uso relativo alla gestione delle procedure.

- **Scopo:** Il diagramma presentato nella Figura 24, ha lo scopo di rappresentare i casi d'uso necessari alla gestione delle procedure all'interno dell'azienda.

  Un utente autenticato deve poter gestire le procedure di ogni dipendente dell'azienda. Ogni procedura, di sistema, deve essere dotata di descrizione, codifica ed informazioni relative alle revisioni delle quali è stata oggetto nel tempo.
  Le procedure di prassi sono state gestite con un questionario (vedi sezione: 4.1.9) in quanto derivanti direttamente da una lista di domande proveniente dall'INAIL$_G$;
- **Attori Coinvolti:** Utente Autenticato;
- **Flusso principale degli eventi:**
  - *L'utente autenticato gestisce le procedure di sistema(UC6.1);*
    - *L'utente autenticato inserisce una nuova procedura di sistema(UC6.1.1);*
    - *L'utente autenticato modifica una procedura di sistema(UC6.1.2);*
    - *L'utente autenticato rimuove una procedura di sistema(UC6.1.3);*
    - *L'utente autenticato visualizza tutte le procedure di sistema(UC6.1.4).*
  - *L'utente autenticato gestisce le procedure di prassi (UC6.2);*
    - *L'utente autenticato inserisce una nuova procedura di prassi (UC6.2.1);*
    - *L'utente autenticato modifica una procedura di prassi (UC6.2.2);*
    - *L'utente autenticato rimuove una procedura di prassi (UC6.2.3);*
    - *L'utente autenticato visualizza tutte le procedure di prassi (UC6.2.4).*

### 4.1.11 UC7 Gestione delle segnalazioni

Le segnalazioni sono delle comunicazioni ufficiali indirizzate all'organo di vigilanza. Esse sono dotate di: descrizione, segnalante, data di comunicazione del segnalante all'alta direzione, data di comunicazione dall'alta direzione all'organo di vigilanza, data di risposta dell'organo di vigilanza al soggetto interessato.

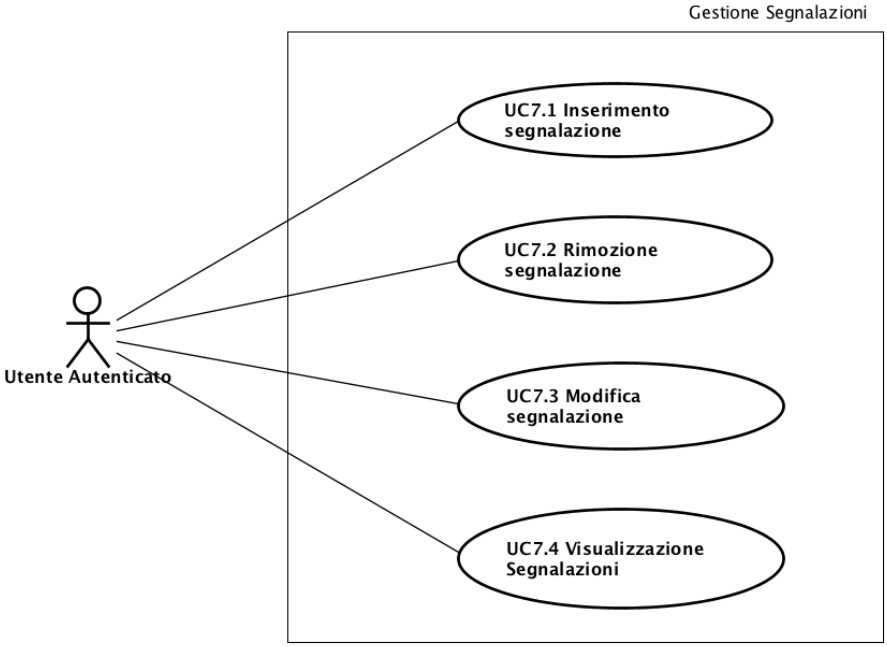



Figura 25: Diagramma dei casi d'uso relativo alla gestione delle segnalazioni.

- **Scopo:** Il diagramma presentato nella Figura 25, ha lo scopo di rappresentare i casi d'uso necessari al soddisfacimento di tutti i requisiti riguardanti la gestione delle segnalazioni all'interno dell'azienda.

- **Attori Coinvolti:** Utente Autenticato;
- **Flusso principale degli eventi:**
  - *L'utente autenticato inserisce una segnalazione (UC7.1);*
  - *L'utente autenticato rimuove una segnalazione (UC7.2);*
  - *L'utente autenticato modifica una segnalazione (UC7.3);*
  - *L'utente autenticato visualizza tutte le segnalazioni (UC7.4);*

### 4.1.12 UC8 Gestione dei Dispositivi di Protezione Collettivi

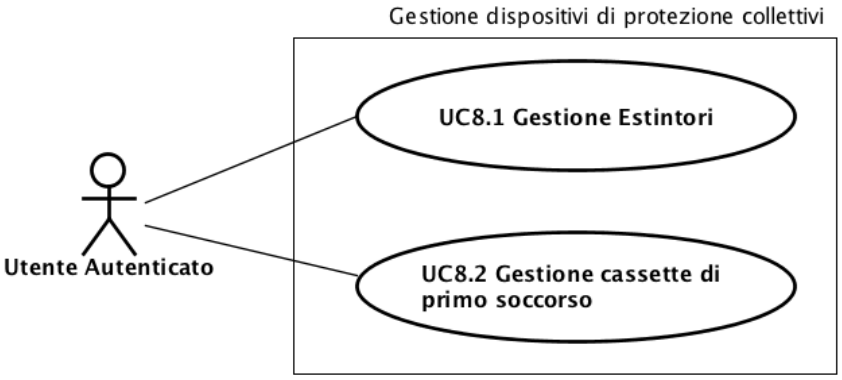



Figura 26: Diagramma dei casi d'uso relativo alla gestione dei Dispositivi di Protezione Collettiva (DPC).

- **Scopo:** Il diagramma presentato nella Figura 26, ha lo scopo di rappresentare i casi d'uso necessari al soddisfacimento di tutti i requisiti riguardanti la gestione dei $DPC_G$ all'interno dell'azienda.

- **Attori Coinvolti:** Utente Autenticato;
- **Flusso principale degli eventi:**
  - *L'utente autenticato gestisce gli estintori(UC8.1);*
    - *L'utente autenticato inserisce un nuovo estintore (UC8.1.1);*
    - *L'utente autenticato modifica un estintore(UC8.1.2);*
    - *L'utente autenticato rimuove un estintore (UC8.1.3);*
    - *L'utente autenticato visualizza tutti gli estintori (UC8.1.4).*
  - *L'utente autenticato gestisce le cassette di primo soccorso (UC8.2);*
    - *L'utente autenticato inserisce una nuova cassetta di primo soccorso (UC8.2.1);*
    - *L'utente autenticato modifica una cassetta di primo soccorso (UC8.2.2);*
    - *L'utente autenticato rimuove una cassetta di primo soccorso (UC8.2.3);*
    - *L'utente autenticato visualizza tutte le cassette di primo soccorso (UC8.2.4).*

## 4.2 Requisiti

nuova sezione Al fine di chiarire ciò che si è svolto durante lo stage, è riportata la seguente tabella relativa ai requisiti che devono essere soddisfatti.
I requisiti sono stati stesi sulla base di diversi colloqui con il committente e da vincoli interni all'azienda. Dopo ogni colloquio, è stato steso il relativo verbale.
Tutti i requisiti provenienti dai verbali che fanno seguito ai colloqui sono stati catalogati come fonte interna.
La codifica di ogni requisito è così composta:

- Una *R* iniziale per sottolineare il fatto che si tratta di un requisito;
- Un qualificatore del livello di importanza:
  - *obb*: obbligatorio;
  - *des*: desiderabile;
  - *opz*: opzionale.
- Un qualificatore relativo alla tipologia:
  - *F*: funzionale;
  - *Q*: qualitativo;
  - *V*: vincolo.
- Un numero progressivo.

| **Codice** | **Descrizione** | **Fonti** |
|---|---|---|
| *RobbV0* | Il codice deve essere scritto utilizzando Ruby on Rails. | Azienda |
| *RobbF0* | Un dipendente deve poter ricoprire più cariche contemporaneamente. | fonte interna, UC1 |
| *RobbF1* | Un dipendente deve poter ricoprire una carica in un dato luogo. | fonte interna, UC1 |
| *RobbF1.1* | Per ogni carica di ogni dipendente devono essere gestite le informazioni relative alla nomina. | fonte interna, UC1, |
| *RobbF1.2* | Per ogni carica di ogni dipendente devono essere segnalate eventuali formazioni mancanti per poterla ricoprire. | fonte interna, UC1 |
| *RobbF1.3* | Un $\text{ASPP}_G$ oppure un $\text{RSPP}_G$ deve poter essere sia un membro interno sia esterno all'azienda. | fonte interna, UC1, UC1.1 |
| *RobbF1.4* | Tutte le informazioni relative alle *figure di sitema* devono essere gestibili in un unica sezione. | fonte interna |
| *RobbF2* | Un utente amministratore deve poter gestire le tipologie di $\text{DPI}_G$ messe a disposizione dal sistema. | fonte interna, UC2.1 |
| *RobbF2.1* | Un utente autenticato deve poter aggiungere un $\text{DPI}_G$ alla dotazione di un dipendente selezionandolo esclusivamente tra quelli messi a disposizione da RobbF2. | UC2.2.1 |
| *RobbF2.2* | Un utente autenticato deve poter inserire le informazioni aggiuntive relative ad ogni $\text{DPI}_G$ di ogni dipendente. | fonte interna, UC2.2.3 |
| *RdesF2.3* | La scadenza di un $\text{DPI}_G$ deve essere segnalata con trenta giorni di preavviso. | fonte interna |
| *RobbF3* | Un utente amministratore deve poter gestire le denominazioni delle mansioni messe a disposizione dal sistema. | UC3.1 |
| *RobbF3.1* | Un utente autenticato deve poter assegnare una mansione ad un dipendente selezionandola esclusivamente tra le denominazioni messe a disposizione da RobbF3. | UC3.2.1 |
| *RobbF3.2* | Un utente autenticato deve poter inserire le informazioni aggiuntive relative ad ogni mansione assegnata ad ogni dipendente. | UC3.2.3 |

| Codice | Descrizione | Fonti |
|---|---|---|
| *RobbF4* | Un utente amministratore deve poter gestire le denominazioni delle formazioni messe a disposizione dal sistema. | fonte interna, UC4.1 |
| *RobbF4.1* | Un utente autenticato deve poter inserire una formazione di un dipendente selezionandola esclusivamente tra le denominazioni messe a disposizione da RobbF4. | fonte interna, UC4.2.1 |
| *RobbF4.2* | Un utente autenticato deve poter inserire le informazioni aggiuntive relative ad ogni formazione conseguita da ogni dipendente. | fonte interna, UC3.2.3 |
| *RdesF4.3* | Un utente amministratore deve poter inserire le formazioni necessarie allo svolgimento di una data mansione. | fonte interna |
| *RdesF4.4* | La mancanza delle formazioni necessarie per poter svolgere una mansione deve essere segnalata. | fonte interna |
| *RdesQ5* | Deve essere previsto un aiuto all'utente nella compilazione di un questionario con una quantità rilevante di domande. | Azienda |
| *RobbF6* | Devono essere gestite in una apposita sezione le procedure aziendali che compongono il $DVR_G$. | fonte interna, UC6 |
| *RobbF6.1* | Devono essere poste tutte le domande che compongono le procedure di prassi fornite dall'INAIL agli asseveratori. | fonte interna, UC6 |
| *RobbF7* | Un utente autenticato deve poter gestire le segnalazioni all'organo di vigilanza. | fonte interna, UC7 |
| *RobbF8* | Deve essere possibile gestire i DPC aziendali indicati in estintori e cassette di primo soccorso. | fonte interna, UC8 |
| *RobbF8.1* | Deve essere possibile associare un DPC ad una sede o ad un cantiere. | fonte interna |
| *RobbF9* | Deve essere possibile impostare regole di validazione delle informazioni inserite condizionata dal valore o la presenza di altre . | fonte interna |
| *RopzQ9.1* | Devono essere disponibili degli script che caricano le informazioni minime funzionali all'installazione del sistema. | fonte interna |
| *RopzF9.2* | Un utente amministratore deve poter impostare regole di validazione da un editor What You See Is What You Get (WYSIWYG) . | Azienda |
| *RdesQ10* | Stesura della documentazione relativa al codice prodotto. | Azienda |

# 5 Tecnologie e strumenti

## 5.1 Git

Git è un sistema di controllo di versione sviluppato per facilitare la cooperazione nello sviluppo di software. Questo software di versionamento è gratuito ed open-source; è stato sritto da *Linus Torvalds* per lo sviluppo del kernel linux nel 2005 ed attualmente mantenuto da *Junio Hamano*.
Questo strumento permette di sottomettere al server le modifiche e lasciare ad esso l'onere di verificare se la versione appena inoltrata va in conflitto con quella esistente indicando esattamente in quale punto si è presentato il problema.
Viene data la possibilità di generare diversi rami (*branch*) con l'intento di differenziare una nuova versione del codice (*fork*) o per lo sviluppo di una nuova funzionalità, mentre nel primo caso il ramo diverge dal ramo da quale ha origine, nel secondo caso il *branch* ha come fine il ricongiungimento con la sorgente da cui proviene una volta soddisfatto il suo contratto. È proprio da questa idea che nasce il concetto di `git-flow`.

### 5.1.1 `git-flow`

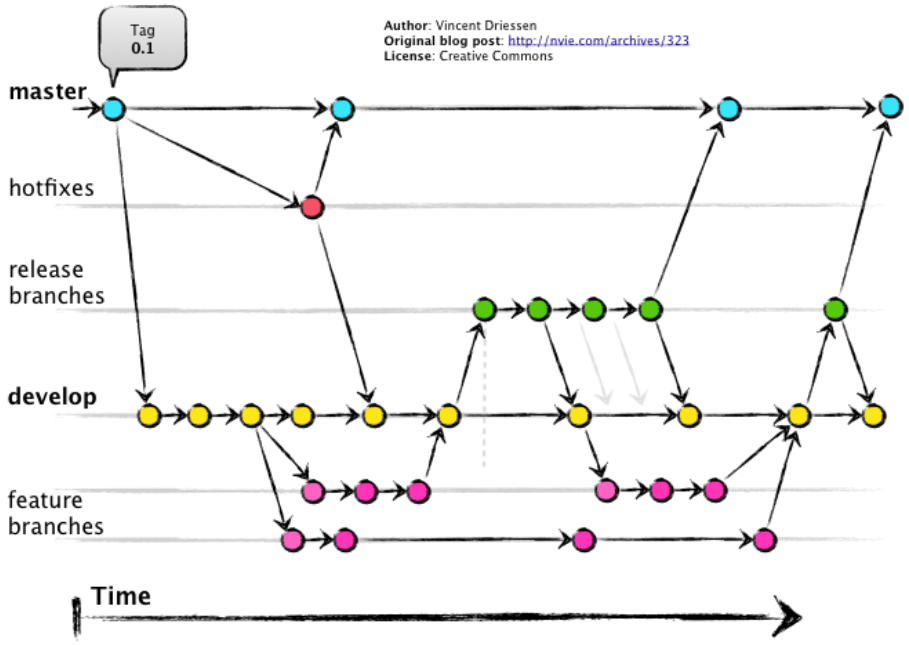



Figura 27: Grafico dei branch di `git-flow`

Git-flow è un set di estensioni di git che offre dei comandi di alto livello sul repository per utilizzare il modello di branching (vedi Figura 27) di *Vincent Driessen*, fornendo il supporto a branching e release management. Esso prevede due branch principali:

- `"master"`: ramo contenente il codice pronto per essere inoltrato nell'ambiente di produzione;
- `"develop"`: ramo di riferimento per l'attività di sviluppo.

Sono previste inoltre altre tre tipologie di branch:

- `feature`: questo tipo di branch viene creato a partire dal ramo principale `develop`; dopo aver portato a compimento una particolare funzionalità, il branch `feature` viene fuso con lo stesso `develop`;
- `release`: questo branch è dedicato alla preparazione di un prodotto alla release; qui è possibile effettuare solo piccoli interventi di rimozione di bug (minor bugfix) in vista di un imminente rilascio nel branch master simultaneamente alla pubblicazione della nuova versione con il comando `git tag`;

- `hotfix`: questo ramo ha lo scopo di risolvere velocemente eventuali bug riscontrati; ha sempre origine dal branch master e, andando a modificare del software già rilasciato, provoca un avanzamento di sul numero di versione.

### Vantaggi

- Permette di mantenere una storia chiara e concisa del progetto poiché tutti i commit parziali avvengono nei branch di tipo feature. Nel ramo develop, saranno presenti solo commit derivanti dalla fusione di rami di tipo feature che soddisfano pienamente il contratto per i quali sono stati creati;
- Evita di creare confusione nel repository con commit parziali potenzialmente dannosi e, dualmente, evitare le situazioni in cui lo sviluppo avviene localmente con un solo commit di grosse dimensioni al compimento del contratto del requisito perdendo i vantaggi dello storico dei commit offerti da git.

### Svantaggi

- L'utilizzo di git-flow richiede agli sviluppatori di dedicare del tempo in più rispetto al metodo classico. Questo però è vero solo per progetti di piccole dimensioni poiché, al crescere dei progetti, il tempo impiegato a risolvere i conflitti derivanti dall'operare in modo classico risulta superiore a quello dedicato ad un corretto utilizzo di git-flow.
- Un ulteriore svantaggio è dato dal fatto che gli strumenti offerti dagli IDE per la gestione di git-flow non sono sempre all'altezza del loro compito.

## 5.2 Rubymine


Figura 28: Logo di RubyMine

Rubymine è un IDE prodotto da jetBrains, disegnato appositamente per lo sviluppo con Ruby on Rails.

### Vantaggi

- Vengono messi a disposizione molti plugin gratuiti che offrono funzionalità specifiche, non incluse nel pacchetto base;
- Il salvataggio di un file avviene automaticamente ogni volta che esso perde il focus, dando la garanzia di lavorare sempre con file aggiornati;
- Il software è strutturato in modo tale da fornire aiuto attivo nel momento della stesura del codice, come autompletamento ed analisi statica, rendendo più veloce ed efficace il lavoro.

### Svantaggi

- Il plugin responsabile del versionamento non funziona sempre alla perfezione, in particolare quando si utilizza git-flow.

## 5.3 Ruby on Rails

Ruby on Rails, è un framework open source per lo sviluppo di applicazioni web. La sua architettura si basa sul pattern Model-View-Controller (MVC), realizzando a tutti gli effetti un framework full-stack; è dunque possibile realizzare sia la parte di backend che quella di front-end$_G$.

### Vantaggi

- Risorse e componenti sono integrati in modo tale che i collegamenti non debbano mai essere impostati manualmente, ma in modo automatico;
- È possibile lavorare utilizzando ActiveRecord, descritto nella sezione 5.3.1. Questo è un grosso vantaggio perché velocizza la stesura del codice e risponde bene ai cambiamenti. Nel progetto svolto, l'utilizzo di ActiveRecord è un fattore determinante poiché è necessario che il software sia sempre conforme alle normative vigenti che cambiano nel tempo;
- È possibile disaccoppiare il backend ed il front-end$_G$ perché in Rails il routing delle risorse è gestito con una interfaccia REST$_G$;
- Sono inoltre messi a disposizione i seguenti ambienti per uno stesso software, al fine di creare una separazione nel progetto:
  - development
  - test
  - production

  Ambienti distinti solitamente risiedono anche su server distinti.
- Sono presenti infine numerose librerie dette *Gemme*, che offrono molteplici funzionalità velocizzando il lavoro.

### Svantaggi

- Ruby on Rails è meno performante rispetto ad altri linguaggi, ad esempio Java o Scala, ma di un ordine di grandezza non influente su progetti di medie dimensioni.

### 5.3.1 ActiveRecord

Active Record è il modulo di Ruby on Rails che gestisce la persistenza dei dati seguendo il pattern *Active Record* di Martin Fowler[1].
Il modulo ActiveRecord di Rails si serve di un database e prevede:

- Ogni tabella del database relazionale è gestita attraverso una classe;
- Una singola istanza della classe corrisponde ad una riga (record) nella tabella associata;
- Alla creazione di una nuova istanza viene creata una nuova riga all'interno della tabella che viene aggiornata ad ogni modifica dell'istanza associata.

Le colonne della tabella rappresentano gli attributi della classe.

### Vantaggi

- È possibile associare un diverso database per ogni ambiente predisposto da Rails separando i record memorizzati per lo sviluppo, per i test, o per l'ambiente di produzione;
- È disponibile un meccanismo di gestione delle relazioni molto efficiente che facilita dichiarazione ed utilizzo delle diverse tipologie di relazione tra le tabelle del database;
- Viene messo a disposizione un meccanismo specifico per la gestione dell'evoluzione dello schema del database mediante le `migrazioni` (vedi Appendice A).

### Svantaggi

- Richiede una progettazione spesso diversa da quella classica, poiché l'ereditarietà risulta molto più difficile da gestire quando una classe corrisponde ad una tabella;
- Ad ogni migrazione, è necessario pensare bene a tutti gli effetti collaterali che potrebbero avvenire sugli statement che utilizzano la tabella modificata.

[1] La spiegazione dettagliata del pattern Active Record di Martin Fowler è descritta all'indirizzo `http://www.martinfowler.com/eaaCatalog/activeRecord.html`.

### 5.3.2 ActiveAdmin


Figura 29: Logo di ActiveAdmin

ActiveAdmin è una libreria (Gemma) di Ruby on Rails che permette di creare un sistema di amministrazione (backoffice). Mediante un pannello di gestione, permette di inserire, eliminare ed aggiornare istanze di modelli e relazioni direttamente da browser.

#### Vantaggi

- Questa libreria permette ad un utente del sito di essere autonomo nell'aggiornamento dei contenuti senza dover attendere i tempi tecnici dell'azienda fornitrice;
- È fortemente personalizzabile, per questo è spesso utilizzato per la creazione di Dashboard o altre funzionalità dedicate ad utenti ai quali sono assegnati particolari privilegi d'accesso.

#### Svantaggi

- La grafica di ActiveAdmin risulta datata, è quindi spesso necessario riprogettare la veste grafica del pannello di amministrazione fornito di default.

### 5.3.3 I18n


Figura 30: Logo di I18n

I18n è l'abbreviazione della parola "internationalization", deriva dal suo spelling che interpone 18 lettere tra la "i" iniziale e la "n" finale.
Lo scopo è gestire la distribuzione del software in diverse lingue. Il meccanismo prevede un codice identificativo per ogni stringa da visualizzare, che assumerà un diverso valore per ogni traduzione.
In Ruby, esiste una apposita gemma, denominata appunto *I18N*, che fornisce le direttive per adottare correttamente questo approccio.
Spesso il concetto di internazionalizzazione, viene associato a quello di localizzazione che è l'insieme dei processi di adattamento di un software , pensato e progettato per un mercato e contesto predefinito, in modo specifico ad altre nazione e culture.
Tutte le informazioni di pertinenza di una lingua sono raccolti in un gruppo di parametri chiamato *locale*.

#### Vantaggi

- Grazie alla gemma *i18n* , è possibile riutilizzare il codice responsabile della logica di business dell'applicazione e localizzare le stringhe per ogni lingua;

- L'aggiunta di nuove lingue richiede solo lo sforzo riguardante la traduzione ed eventuali conversioni nelle unità di misura appropriate;
- È possibile riutilizzare in contesti diversi le stringhe già definite, evitando quindi di introdurre errori di battitura;
- L'aggiunta di nuove lingue non provoca alcuna modifica nell'applicazione esistente, ma solo l'aggiunta del file in formato `yml` con le traduzioni.

#### Svantaggi

- La scrittura di codice con questo approccio richiede generalmente più tempo se il software esiste in una sola lingua.

## 5.4 JRuby

Figura 31: Logo di JRuby

JRuby è un'implementazione in Java del linguaggio di programmazione Ruby.
Sebbene non siano coperte tutte le implementazioni delle librerie standard offerte da Ruby, è comunque possibile utilizzare tutti la maggior parte delle funzionalità del linguaggio.

#### Vantaggi

- JRuby è un software completamente gratuito;
- Gira su una JVM, quindi è possibile integrare l'interprete Ruby in una applicazione Java ed, allo stesso tempo, scrivere direttamente codice Java.

#### Svantaggi

- È possibile incontrare situazioni in cui si necessita di una libreria che non è stata implementata;
- L'utilizzo di JRuby provoca un significativo appesantimento del software ed un conseguente peggioramento delle performance.

## 5.5 Drools


Figura 32: Logo di Drools

Drools è un Business Rules Management System (BRMS) basato su forward and backward chaining inference (vedi Appendice B).
Questo strumento permette di definire delle regole che, al soddisfacimento delle condizioni

iniziali, permettono a Drools di prendere delle decisioni e di conseguenza determinare in che modo il software deve agire.

### 5.5.1 Architettura

Ad alto livello, Drools si può vedere come tre componenti che cooperano (vedi Figura 33): *Production Memory*, *Working Memory* ed *Inference Engine*. Quest'ultimo si compone di due sotto componenti: *Pattern matcher* ed *Agenda*.

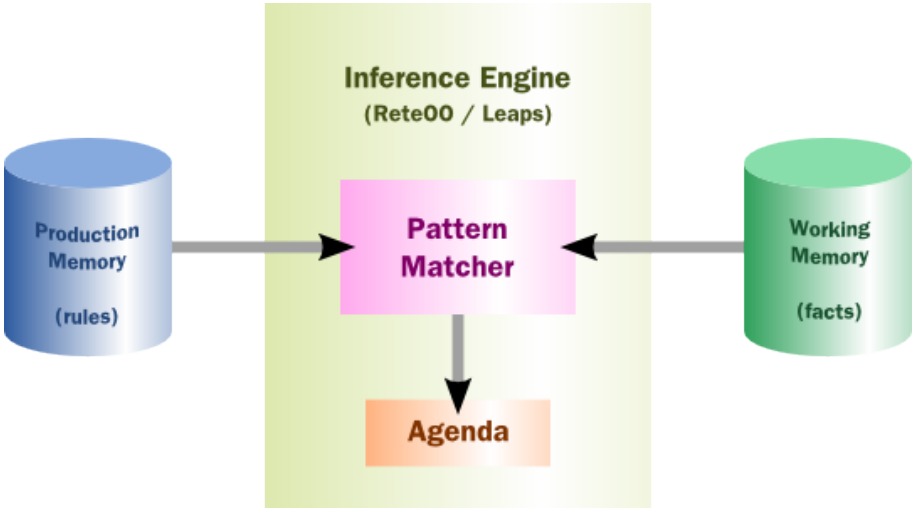



Figura 33: Architettura di Drools

- **Production Memory:**
  Componente che contiene tutte le regole che compongono la Knowledge Base (KB);
- **Working Memory:**
  Componente che contiene tutti i fatti. In Drools, i fatti sono degli oggetti Java.
  Si chiama *Working Memory*, poiché è possibile effettuare operazioni di modifica, inserimento e rimozione dei fatti che contiene;
- **Inference engine:**
  Macrocomponente che ha la responsabilità di valutare i fatti della *Working Memory* sulla base della Knowledge Base presente in *Production Memory*. Per fare ciò si avvale di due sottocomponenti:
  - **Pattern matcher:**
    Componente che ha lo scopo di verificare, quando richiamata dall'inference engine, quali regole soddisfano la condizione presente nella $\text{LHS}_G$ sulla base dei fatti presenti nella *Working Memory* al momento dell'invocazione. Verranno poi eseguite le azioni presenti nella $\text{RHS}_G$ di tali regole. Per fare ciò in modo efficiente con molti dati, viene utilizzato l'algoritmo Rete, brevemente descritto nell'Appendice D;
  - **Agenda:**
    Ha lo scopo di risolvere i conflitti tra le regole. Essi avvengono nel momento in cui più regole soddisfano la condizione nella loro $\text{LHS}_G$ con gli stessi fatti.
    In questi casi, interviene l'Agenda che decide un ordine di esecuzione tra le regole che vanno in conflitto.

#### Vantaggi

- In sistemi con grandi quantità di dati e vincoli, un approccio di questo tipo facilita notevolmente il mantenimento della verità ed il controllo sui dati inseriti. La definizione dei vincoli e l'aggiornamento degli stessi, risulta molto meno oneroso e di più facile verifica;
- L'elaborazione, risulta essere più veloce e performante rispetto ad un approccio tradizionale.

### Svantaggi

- Per poter integrare questo sistema con Ruby on Rails é necessario utilizzare jRuby ed implementare in Java, le classi corrispondenti ai modelli sui quali si vogliono definire delle regole. Tecnicamente, qusto si fa con dei file di template java.erb di Rails che ottengono le informazioni riguardo classi e membri mediante Reflection$_G$;
- Per lavorare in modo efficace con questa tecnologia, è necessario investire molto tempo per conoscerne a fondo tutte le sfaccettature.

## 5.6 Foundation


Figura 34: Logo di Foundation

Foundation è una collezione di framework$_G$ per lo sviluppo della componente front-end$_G$ di applicazioni web.
In particolare è composto da tre pacchetti:

- **Foundation for Sites:** per la realizzazione di siti web;
- **Foundation for Emails:** per la realizzazione di email formattate in html;
- **Foundation for Apps:** per la realizzazione di applicazioni web.

La struttura di ogni framework$_G$ è modulare e consiste essenzialmente di fogli di stile in formato SASS$_G$ - SCSS$_G$.
Per limitare il peso dei fogli di stile, è possibile selezionare le sole componenti di cui si necessita al momento del download: saranno le sole presenti nel pacchetto da includere nel sito o da scaricare con un CDN$_G$.
Come la maggioranza dei framework$_G$ moderni per il front-end$_G$ Foundation fornisce un sistema di posizionamento a griglia dei componenti. Sono previste dodici colonne, visibili una accanto all'altra quando la larghezza dello schermo è maggiore di 940 pixel. Il numero di colonne viene adattato automaticamente a seconda dello spazio a disposizione, rendendo il sito, l'applicazione o l'email responsive.

### Vantaggi

- Oltre ai comuni elementi HTML, sono messi a disposizione componenti aggiuntivi per la realizzazione delle sezioni di interfaccia maggiormente usate, ad esempio:
  - Gruppi di bottoni;
  - Menu a tendina;
  - Breadcrumb$_G$;
  - Messaggi formattati per avvisi, errori, aiuto.
- Foundation permette di realizzare interfacce grafiche accattivanti senza dedicare eccessivo tempo alla loro implementazione;
- La collezione di framework$_G$ che compone Foundation è stata testata sui principali browser$_G$ ottenendo ottimi risultati (vedi Figura 35 per il dettaglio sulla versione utilizzata, la 5.5).

| Browser/OS | The Grid | Layout/UI | JS |
|---|---|---|---|
| Chrome | ✓ | ✓ | ✓ |
| Firefox | ✓ | ✓ | ✓ |
| Safari | ✓ | ✓ | ✓ |
| IE10 | ✓ | ✓ | ✓ |
| IE11 | ✓ | ✓ | ✓ |
| IE9 | ✓ | ✓ | ✓ |
| IE8 | ✗ | ✗ | ✗ |
| IE7 | ✗ | ✗ | ✗ |
| iOS (iPhone) | ✓ | ✓ | ✓ |
| iOS (iPad) | ✓ | ✓ | ✓ |
| Android 2, 4 (Phone) | ✓ | ✓ | ✓ |
| Android 2, 4 (Tablet) | ✓ | ✓ | ✓ |
| Windows Phone 7+ | ✓ | ✓ | ✓ |
| Surface | ✓ | ✓ | ✓ |

Figura 35: Compatibilità di Foundation 5.5

### Svantaggi

- Per utilizzare questo framework$_G$ è necessario investire del tempo per apprenderne appieno le caratteristiche.

## 5.7 jQuery


Figura 36: Logo di jQuery

jQuery è una libreria Javascript$_G$ per applicazioni web. Nasce con l'obiettivo di velocizzare la gestione del Document Object Model (DOM)$_G$ e semplificare l'utilizzo delle chiamate Ajax$_G$ e degli eventi.

### Vantaggi

- Permette di eseguire chiamate Ajax$_G$ in modo veloce, rendendo così l'applicazione web più dinamica;
- Velocizza l'accesso e la gestione degli elementi del DOM$_G$ delle pagine HTML$_G$;
- È stata testata sui principali browser$_G$;
- Permette di semplificare e ridurre il codice, velocizzando il raggiungimento degli obiettivi.

### Svantaggi

- È sempre necessario importare tutto il pacchetto. Tuttavia, la versione compressa, occupa poco meno di 100 KiloBite.

# 6 Progettazione e codifica

A partire dalle specifiche fornite dal committente, è stata svolta una accurata analisi per scomporre il questionario di oltre mille domande in questionari di dimensione minore e direttamente correlati alla risorsa di pertinenza.
In particolare sono state oggetto di analisi le seguenti componenti:

- *Figure di sistema;*
- *Dispositivi di protezione individuale;*
- *Mansioni;*
- *Formazioni;*
- *Questionari;*
- *Segnalazioni;*
- *Procedure;*
- *Dispositivi di protezione collettivi.*

Per ogni componente, è stato realizzato o restaurato il modello corrispondente, aggiornate le viste che ne facevano uso. Sono state implementate inoltre le regole Drools per verificare la conformità delle informazioni inserite alle norme vigenti in ambito di sicurezza.

## 6.1 Riprogettazione delle componenti esistenti

### 6.1.1 Riprogettazione della componente: *Figure di sistema*

Con *figure di sistema* si intendono tutte le cariche che hanno la responsabilità di garantire la sicurezza in azienda.
Come specificato dal testo unico della salute e sicurezza sul lavoro (D.lgs. 81/2008), esse sono:

- Datore di lavoro;
- Dirigente;
- Preposto;
- Medico Competente;
- Responsabile del servizio di prevenzione e Protezione (RSPP);
- Addetto al servizio di prevenzione e Protezione (ASPP);
- Responsabile dei Lavoratori per la Sicurezza (RLS).

L'operato delle figure sopra elencate deve essere supervisionato da un *Organo di vigilanza.*

#### Situazione precedente alla riprogettazione

La situazione iniziale prevedeva che la classe *Individual* avesse un attributo booleano per ogni tipologia di figura di sistema.
Si è resa necessaria una riprogettazione dal momento che alcune figure dovevano ricoprire lo stesso ruolo contemporaneamente in più contesti e questa situazione non poteva essere gestita con l'architettura esistente.
Un esempio di tale problema è rappresentato da un ASPP assegnato a due cantieri contemporaneamente. Tale situazione si può verificare se i due cantieri sono allocati nello stesso periodo e nel primo si lavora il mattino, mentre nel secondo di pomeriggio oppure a giorni alterni.
I questionari relativi a a tutte le figure di sistema erano presentati in una pagina contenente tutte le domande. I problemi che tale soluzione procurava erano sostanzialmente due: era possibile gestire al più un individuo per ogni ruolo e le domande a cui rispondere in una sola sezione erano più di centocinquanta.

#### Modifiche apportate

L'architettura è stata riprogettata centralizzando tutti i ruoli in una apposita classe `Role` e Mantenendo la relazione molti a molti mediante una tabella intermedia `IndividualRoles`.
Questo approccio si è rivelato molto vantaggioso perché ha permesso di associare ogni classe rappresentante un ruolo, alle formazioni minime che un individuo deve possedere per poter ricoprire quella carica.

Questa soluzione (Figura 37) soddisfa appieno tutti ri requisiti.

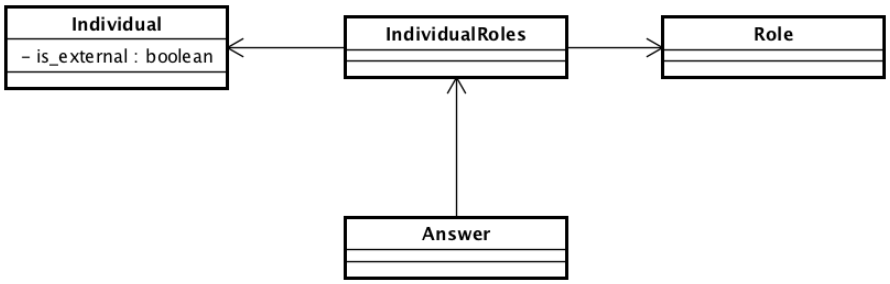



Figura 37: Diagramma delle classi per la gestione delle figure di sistema ed i relativi questionari.

Obiettivo primario della riprogettazione è stato centralizzare tutte le informazioni relative alle figure di sistema ed i loro questionari in un unica sezione. La schermata presente nella Figura 38 vuole rappresentare il risultato del lavoro svolto per le figure riguardanti l'azienda.

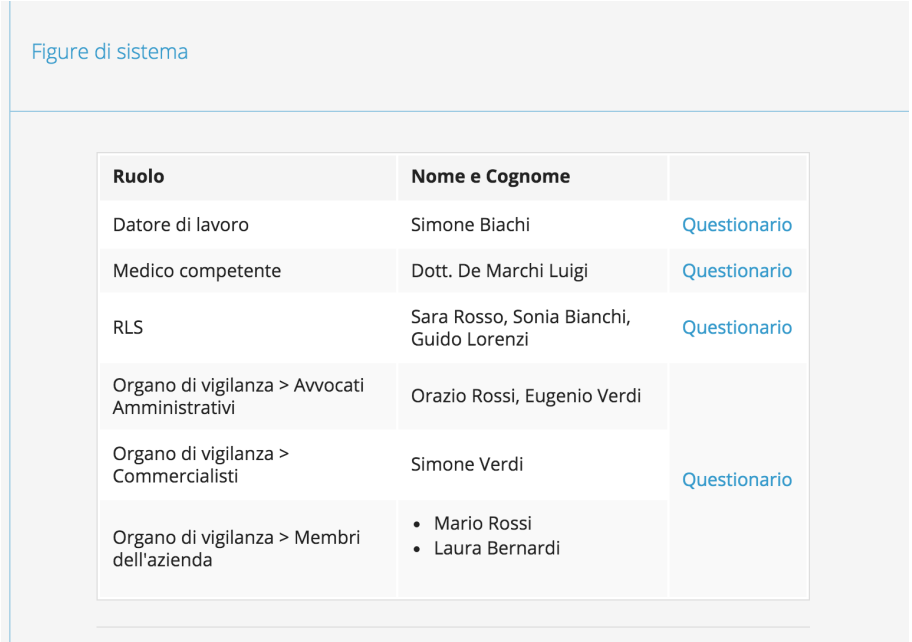



Figura 38: Schermata iniziale della sezione *Figure di sistema*.

È stato implementato inoltre un pannello dedicato alla gestione delle figure impiegate in specifiche sedi o cantieri. In questo modo è sempre possibile allocare gli individui nei luoghi d'interesse senza vincoli di molteplicità.


Figura 39: Schermata iniziale della sezione *Pannnello di gestione delle figure impegnate in sedi e cantieri.*.

Per facilitare l'inserimento delle informazioni relative alla selezione dei dipendenti, è stata utilizzata la libreria *Selectize.js* per rendere i form più gradevoli dal punto di vista grafico e per permettere l'autocompletamento (vedi Figura 40 ).
Una criticità si è presentata nell'inserimento dinamico delle figure che possono essere presenti con molteplicità maggiore di uno. In particolare gli RSPP e gli ASPP presentano l'ulteriore complicazione rappresentata dal fatto che possono essere sia membri interni all'azienda, sia esterni.
In entrambi i casi, la situazione è stata gestita mediante chiamate Ajax, evitando quindi di dover ricaricare la pagina ad ogni inserimento e generando un record della classe *Individual* identificato da un particolare flag (`is_external`) nel caso si tratti di una figura esterna all'azienda.

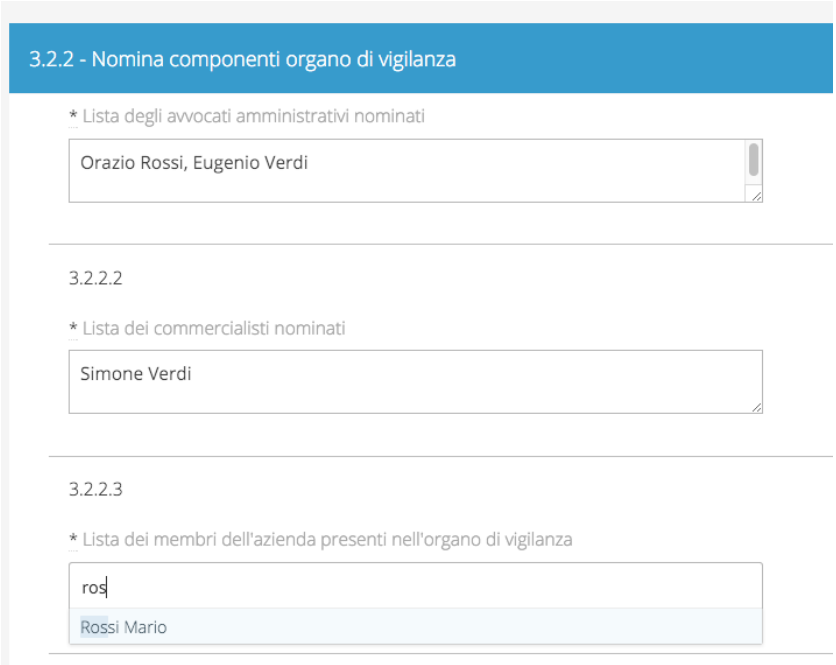



Figura 40: Schermata relativa all'inserimento di alcune informazioni in merito all'organo di vigilanza

Ad ogni figura di sistema è stato associato un questionario nel quale sono specificate le informazioni in merito alla carica, tra tutte la data di accettazione e scadenza dell"incarico.

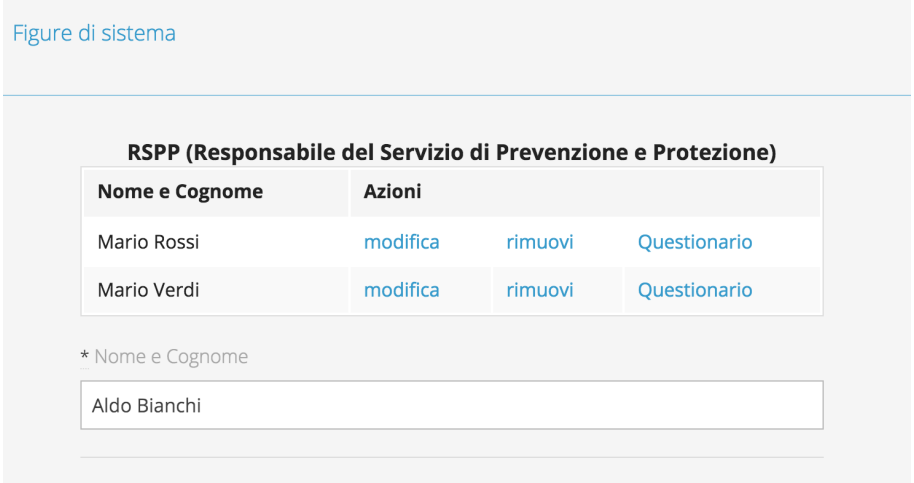



Figura 41: Schermata relativa all'inserimento di un RSPP.

L'associazione ad un questionario avviene generando tutte le risposte ad esso relative. Per ogni risposta saranno preimpostati i seguenti valori:

- `response: nil`;
- `answerable_type: "IndividualRoles"`;
- `answerable_id:` codice identificativo dell'istanza specifica di `"IndividualRoles"`.

Facendo Riferimento alla Figura 41 e supponendo che *Aldo Bianchi* non sia attualmente presente nel database, alla pressione del tasto invio verrà generato un nuovo *Individual* relativo ad *Aldo Bianchi*. Successivamente verrà generato un questionario con le domande relative agli RSPP in relazione ad *Aldo Bianchi*.

### 6.1.2 Riprogettazione della componente: *DPI*

nuova sezione Il termine *DPI* è un acronimo di *Dispositivi di Protezione Individuale*.
Appartengono a questa categoria tutti i dispositivi utilizzati per la prevenzione degli infortuni dei lavoratori ad esempio guanti, occhiali e caschetti.

#### Situazione precedente alla riprogettazione

Al momento dell'arrivo in azienda, la situazione presente era la seguente:

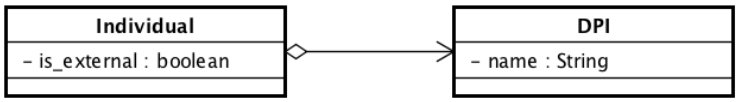



Figura 42: Diagramma delle classi relativo ai DPI all'inizio dello stage .

Come si può osservare dalla Figura 42, questa soluzione non impone vincoli sul nome del $DPI_G$. È necessario però segnalare se un utente non è in possesso tutti i $DPI_G$ necessari allo svolgimento di una data mansione.
Questa soluzione necessitava di essere restaurata poiché, non imponendo vincoli sul nome, rende impossibile gestire il caso sopracitato poiché lo stesso $DPI_G$ può essere descritto utilizzando stringhe diverse.
Il questionario relativo ai $DPI_G$ era stato implementato riportando tutte le domande in un unico blocco, senza differenziare le domande relative ad uno specifico $DPI_G$ da quelle generiche sui $DPI_G$.

#### Modifiche apportate

Le modifiche hanno coinvolto sia il back-end$_G$ sia il front-end$_G$ dell'applicazione.
In particolare, nel back-end$_G$ sono stati centralizzati i nomi dei DPI in un unica classe.
Questa soluzione risolve il problema dell'impossibilità di riconoscimento del $DPI_G$ perché il nome è rappresentato da un riferimento ad una lista di possibili valori definiti nel pannello di amministrazione.

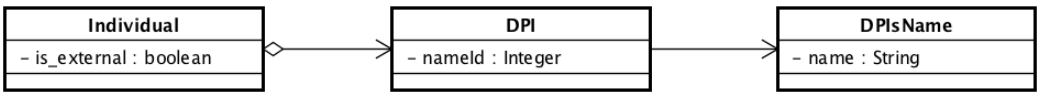



Figura 43: Diagramma delle classi relativo ai DPI dopo la riprogettazione.

Dopo un'attenta analisi, è stata rivista tutta l'organizzazione delle domande relative ai $DPI_G$.
In particolare sono stati differenziate le domande relative ad uno specifico $DPI_G$ da quelle generali in merito alla gestione dei $DPI_G$ in azienda.
Lato front-end$_G$, è stato reso possibile, per un utente amministratore, la gestione delle tipologie dei $DPI_G$ da pannello di controllo. Questo passaggio è stato reso estremamente semplice e veloce da implementare dall'utilizzo della gemma ActiveAdmin di Ruby on Rails.

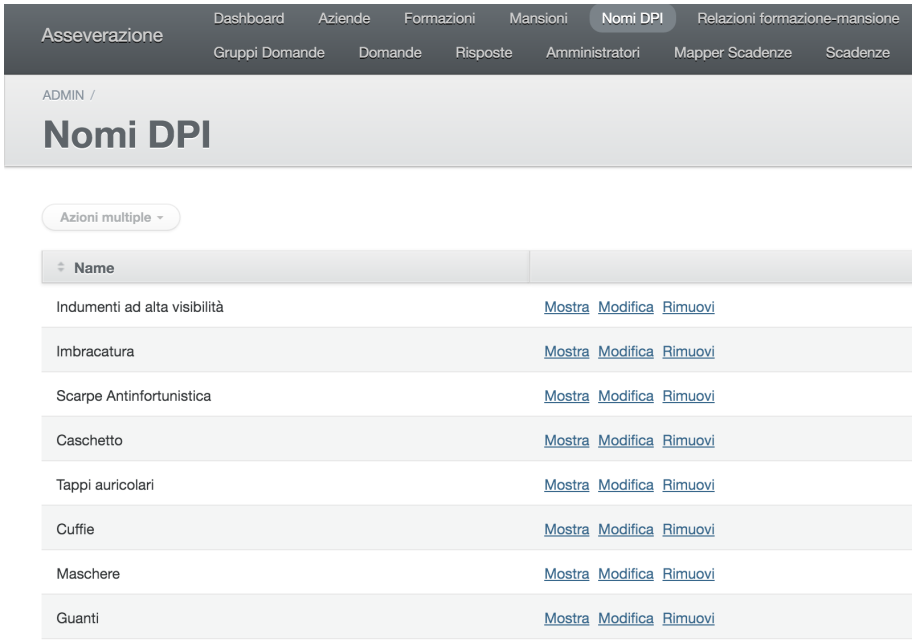



Figura 44: Schermata relativa alla digestione dei DPI nel pannello di amministrazione.

E' stata completamente riprogettata l'interfaccia relativa alla gestione dei $\text{DPI}_G$ assegnabili ad un dipendente.
In particolare è stato inserito un questionario a scomparsa contenente le domande relative alla formazione obbligatoria del datore di lavoro.
L'input dei $\text{DPI}_G$ è stato facilitato mediante autocompletamento basato sui record della tabella `DpisName` .

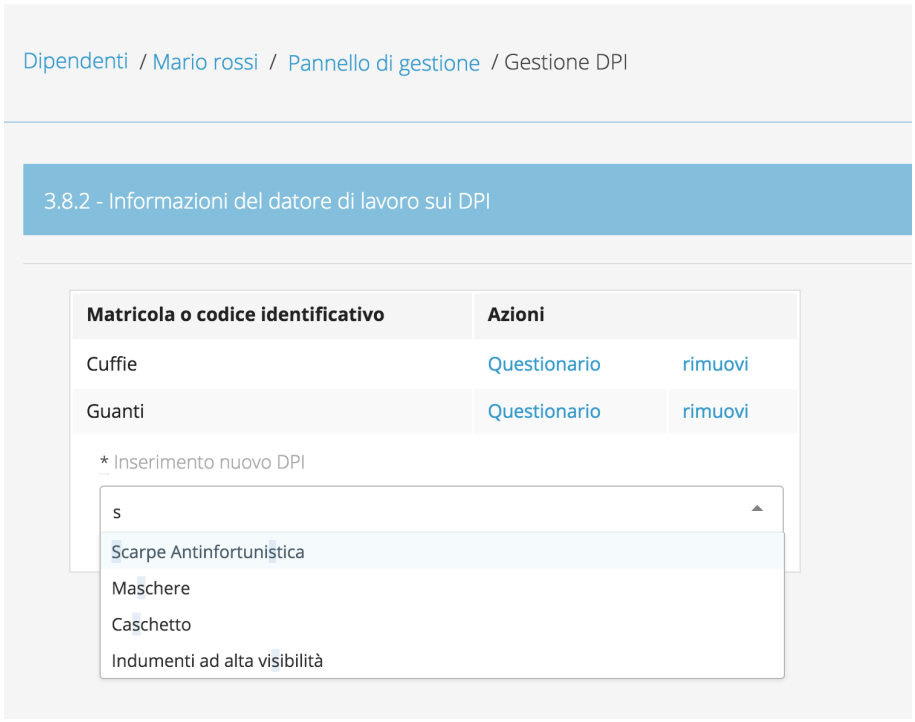



Figura 45: Schermata relativa alla digestione dei DPI nell'interfaccia utente.

### 6.1.3 Riprogettazione di mansioni e formazioni

nuova sezione
Uno degli aspetti cardine del progetto è stato la gestione della relazione che intercorre tra le mansioni che svolge un dipendente e le formazioni di cui è in possesso.
Questa funzionalità ha richiesto una riprogettazione delle classi relative a formazioni e mansioni oltre alla definizione delle regole relative.

#### Situazione precedente alla riprogettazione

Al momento dell'inizio dello stage, le mansioni e le formazioni erano delle classi con una chiave esterna verso la classe `Employee` che rappresenta il dipendente in possesso di quella formazione oppure a cui è stata affidata quella mansione.
Anche in questo caso, la descrizione era gestita con un campo testuale provocando ambiguità nel riconoscimento di una specifica mansione o formazione.
La gestione lato front-end$_G$ era poco intuitiva poiché solo abbozzata.

#### Modifiche apportate

scrivere meglio e completare

Le mansioni e le formazioni sono state riprogettate allo stesso modo dei DPI$_G$, ovvero centralizzando la definizione del nome in una tabella definita allo scopo.
Per evitare di affaticare la lettura non verranno trattati i dettagli implementativi relativi alle singole componenti.
Degna di nota è invece l'integrazione tra le due componenti finalizzata alla verifica della presenza delle formazioni necessarie allo svolgimento di una mansione.

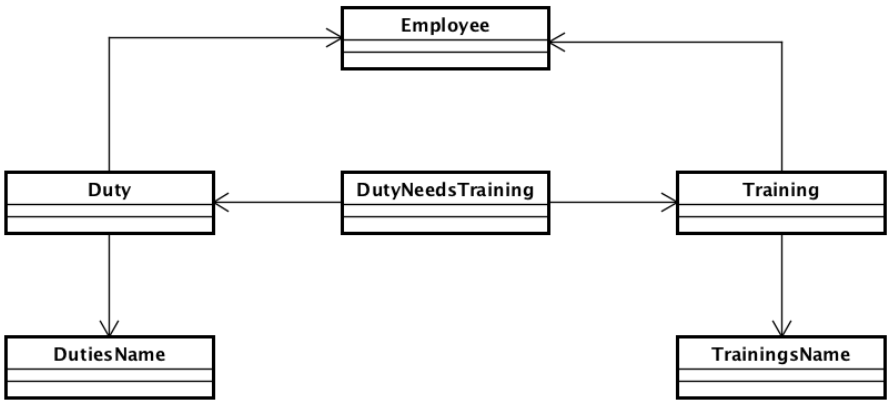



Figura 46: Diagramma delle classi di formazioni, mansioni e della relazione tra esse.

Come è possibile osservare dalla Figura 46, ogni record della tabella `DutyNeedsTraining` contiene un riferimento alla classe `Training` ed uno alla classe `Duty`.
Ogni coppia identifica una relazione che, se non soddisfatta per qualche dipendente, solleverà un allarme ad esso associato.
Un aspetto critico è stato dare la possibilità ad un utente amministratore di definire le regole di dipendenza tra mansioni e formazioni in modo autonomo.
La tabella `DutyNeedsTraining`, introdotta per mappare le dipendenze, è stata utilizzata per generare una regola relativa la verifica della conformità tra relazioni e formazioni. La regola appena descritta è riportata nella sottosottosezione 6.3.1. La gestione dal pannello di controllo stata realizzata utilizzando la gemma ActiveAdmin, rendendo autonomo l'utilizzatore finale nella gestione delle relazioni necessità tra mansioni e formazioni.


Figura 47: Schermata relativa alla digestione delle formazioni necessarie allo svolgimento di determinate mansioni.

Per un utente autenticato, l'immissione delle mansioni e delle formazioni dei dipendenti avviene allo stesso modo dei $\text{DPI}_G$. Un esempio dell'assegnazione delle mansioni è visibile nella figura Figura 48.

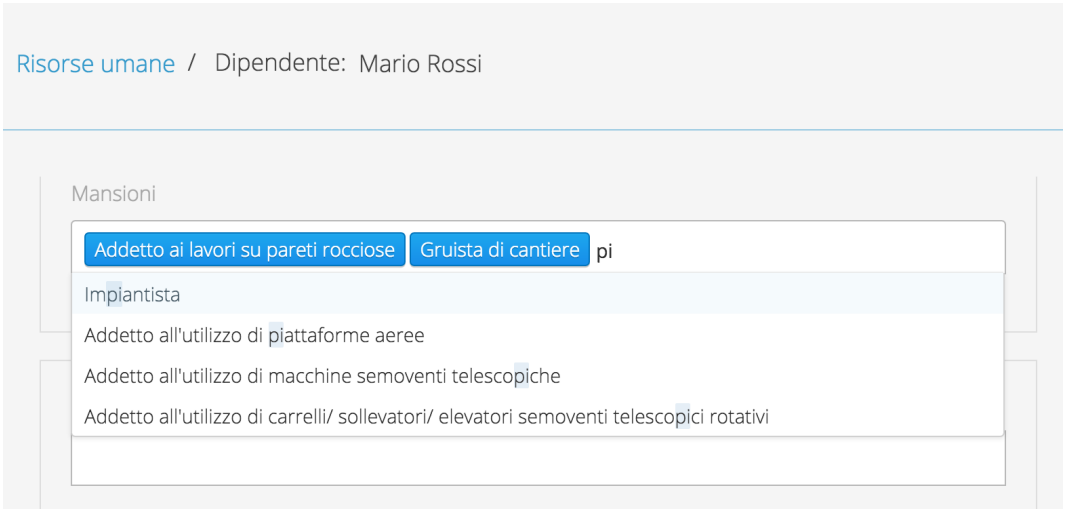



Figura 48: Schermata relativa all'assegnazione delle mansioni da parte di un utente autenticato.

Eventuali non conformità sulle formazioni in possesso di un dipendente in relazione alla mansione che svolge saranno segnalate con un allarme.
Questo approccio sposa la filosofia generale del progetto la quale prevede che deve essere sempre possibile inserire i dati senza blocchi dettati dai vincoli di legge, in modo da evidenziare eventuali non conformità con allarmi al fine di risolverle.

### 6.1.4 Restyling della componente: *Questionari*

I questionari sono stati riprogettati solo lato front-end$_G$. Al momento dell'arrivo in azienda, le domande venivano poste in sequenza organizzate in capitoli. Questo approccio rischia di disorientare l'utente poiché al crescere del numero delle domande può trovare difficoltà nel ricordare quale sia il capitolo in questione.
Per ovviare a questo problema, tutte le domande sono state inserite in un accordion e bloccando il titolo in alto durante lo scroll. In questo modo, all'apertura della pagina sono facilmente visibili i capitoli e, durante la compilazione del questionario, il titolo del capitolo è sempre visibile.

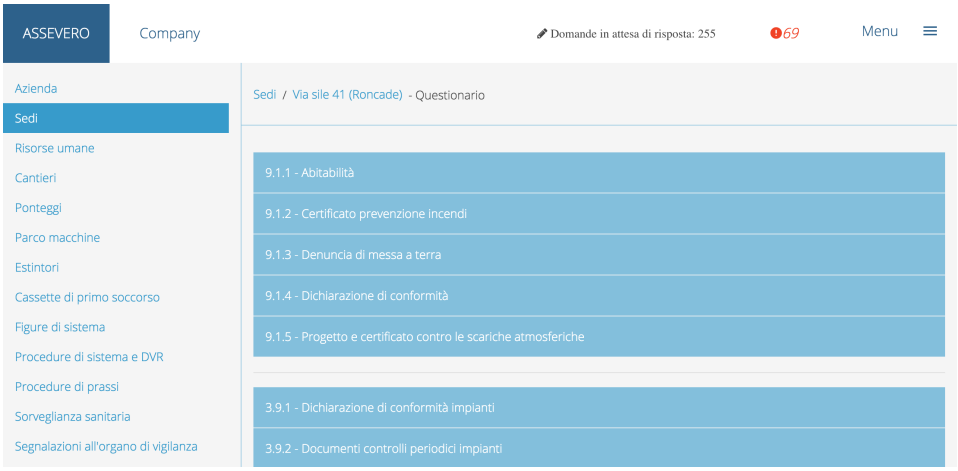



Figura 49: Schermata relativa al questionario relativo alla sicurezza nelle sedi al momento dell'apertura.

Nella figura seguente si può notare il fatto che il titolo del capitolo rimane sempre visibile quando l'accordion è aperto.

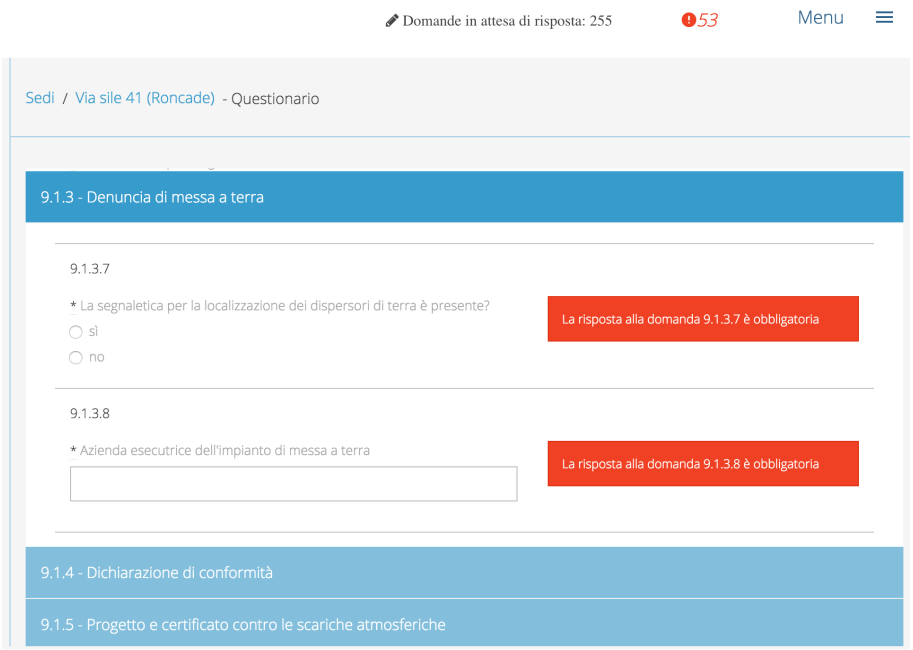



Figura 50: Schermata relativa al questionario relativo alla sicurezza nelle sedi al momento della risposta ad una domanda.

## 6.2 Nuove componenti

Le componenti descritte di seguito, sono state progettate e implementate nella loro interezza durante lo stage.
TODO Scrivere di più nell'introduzione

### 6.2.1 *Segnalazioni*

Le segnalazioni sono delle comunicazioni inviate da un qualunque membro interno all'azienda ed indirizzate verso l'organo di vigilanza.
Le segnalazioni hanno lo scopo di sollevare il responsabile dalle responsabilità derivanti da violazioni avvenute sotto la sua supervisione.
Le comunicazioni all'organo di vigilanza infatti, non possono essere anonime e demandano ufficialmente la responsabilità delle eventuali violazioni a tale organo il quale dovrà decidere per eventuali azioni correttive.

#### Progettazione

Come è possibile osservare dalla Figura 51, le segnalazioni sono state implementate come una classe (`Report`) con una chiave esterna verso la classe `Individual`.
La presenza del riferimento verso un'istanza della classe `Individual` è obbligatoria, soddisfacendo quindi il vincolo di non anonimato delle segnalazioni.

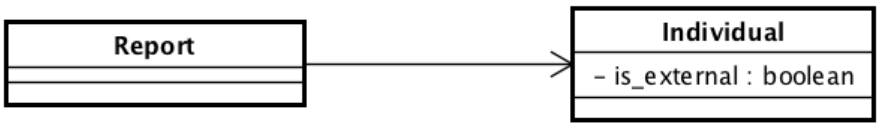



Figura 51: Diagramma delle classi relativo alle segnalazioni.

Specifiche direttive del committente hanno richiesto l'inserimento di tre date rilevanti al fine della gestione delle segnalazioni:

- *Data di invio della segnalazione all'alta direzione;*
- *Data di invio della segnalazione dall'alta direzione all'organo di vigilanza;*
- *Data di invio della risposta alla segnalazione da parte dell'organo di vigilanza al soggetto interessato.*

In accordo con la filosofia del progetto, sono state gestite la presenza e sequenzialità delle date con degli allarmi in caso di errato inserimento.
Per facilitare la gestione delle segnalazioni, nell'interfaccia utente relativa ad esse sono riportate tutte le date. Questo approccio permette di tenere sempre traccia dello stato delle segnalazioni ed agevola la verifica da parte delle autorità competenti.
Uno sviluppo futuro, prevede l'archiviazione delle segnalazioni le quali hanno data di invio della risposta alla segnalazione da parte dell'organo di vigilanza al soggetto interessato più vecchia di 30 giorni. Questa soluzione è stata progettata ma non implementata nel periodo di stage a causa dell'esaurimento del monte ore disponibile.


Figura 52: Schermata relativa alla gestione delle segnalazioni.

### 6.2.2 *Procedure*

Nuova sezione
Da rivedere
Le procedure si distinguono in due caegorie distinte:

- *Procedure di prassi:*
  Procedure derivanti un analisi dell'INAIL$_G$ sulla base dell'analisi degli infortuni sul lavoro avvenuti nel tempo.
- *Procedure di sistema:*
  Procedure proprie dell'azienda. La loro composizione compone il DVR (Documento di Valutazione dei Rischi).

#### Progettazione

##### Procedure di prassi

Queste procedure sono state fornite come collezione di domande. È sembrato quindi gestirle sotto forma di questionari.
Dopo un'attenta analisi delle domande, sono stati individuati sette filoni principali che sono stati usati per spezzare un questionario composto di 189 domande.
Ad ognuno di questi è associato un questionario come è possibile osservare dalla Figura 53.

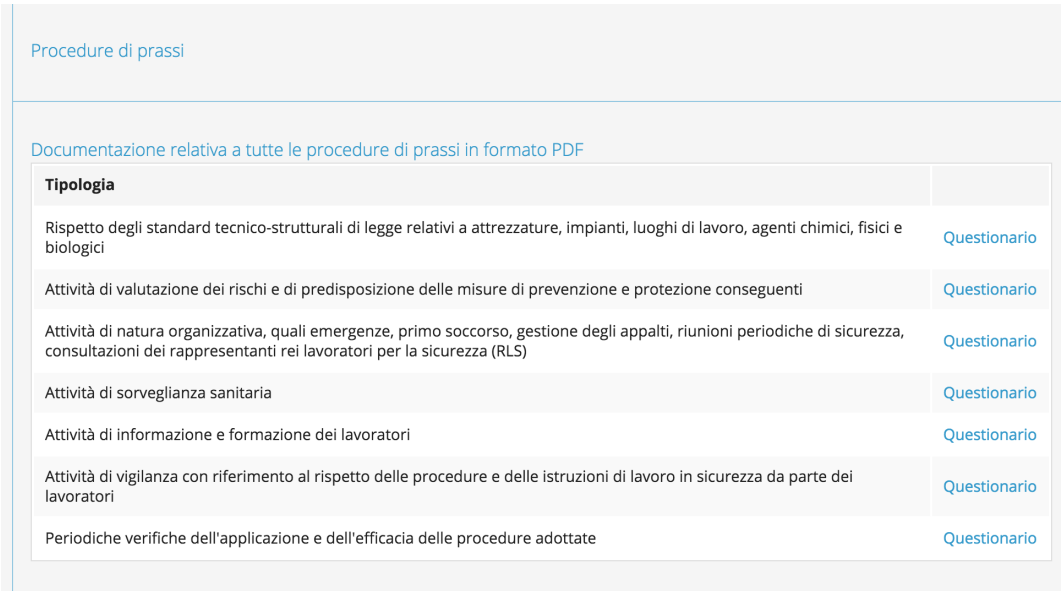



Figura 53: Schermata relativa alla gestione delle procedure di prassi.

Per avere un quadro generale della conformità dell'azienda rispetto alle procedure di prassi è stata implementata l'esportazione di tutte le domande in un unico documento in formato PDF.

##### Procedure di sistema

Le procedure di sistema sono state implementate mediante un apposito modello `Procedure` direttamente collegata a `Company`. Sono state inserite le informazioni relative alle revisioni di ogni procedure.
Come è possibile osservare dalla Figura 54, sono state inoltre inserite le domande relative al DVR$_G$ nell'intefaccia utente con un questionario a scomparsa.


Figura 54: Schermata relativa alla gestione delle procedure di sistema.

Per mettere a disposizione un quadro generale delle procedure di sistema è stata implementata l'esportazione di tutte le domande in un unico documento in formato PDF.

**Criticità incontrate** Per mantenere una coerenza logica, tutti gli aspetti inerenti le procedure sono state gestite con un singolo controller.
Questo ha subito evidenziato numerose criticità in quanto è stato necessario realizzare due index, ma l'url
`procedures` può corrispondere ad una sola azione (`index`).
È stato necessario creare delle apposite rotte per permettere di raggiungere entrambe le risorse.

### 6.2.3 *Dispositivi di protezione collettivi*

nuova sezione
rileggere
La gestione dei $DPC_G$ richiede di tenere traccia delle informazioni relative al loro posizionamento.
In particolare sono stati gestiti Estintori e cassette di primo soccorso.
Le immagini Figura 56 e Figura 57, arrecano esempi relativi agli estintori, ma la trattazione dei due modelli è analoga. La differenza è individuabile solo nella gestione della scadenza che per gli estintori è unica, mentre per le cassette di primo soccorso chiede di memorizzare quella dei due componenti più vicini alla data di scadenza.

#### Progettazione

La soluzione individuata prevede la creazione di una associazione con l'interfaccia *Locationable* come è possibile osservare dalla Figura 55.
Questo approccio è proprio di Ruby on Rails e permette di riferire un oggetto specificandone il nome della classe e l'id a patto che la classe interessata dichiari di essere essere un *Locationable*.
Nel caso di specie, le classi che dichiarano di essere raggiungibili mediante l'interfaccia *Locationable* sono`Location` e `ConstructionSite`, permettendo quindi di assegnare $DPC_G$ a sedi e cantieri.


Figura 55: Diagramma delle classi relativo ai $DPC_G$.

Al momento dell'assegnazione dei $DPC_G$ ad una sede oppure ad un cantiere, sono stati resi selezionabili solamente quelli che non risultano essere impegnati in altri luoghi.
È stata implementata una pagina nella quale sono visibili tutti gli estintori (e rispettivamente per le cassette di primo soccorso) nella quale è indicata la loro allocazione al fine di facilitare la localizzazione agevolando le operazioni di manutenzione.

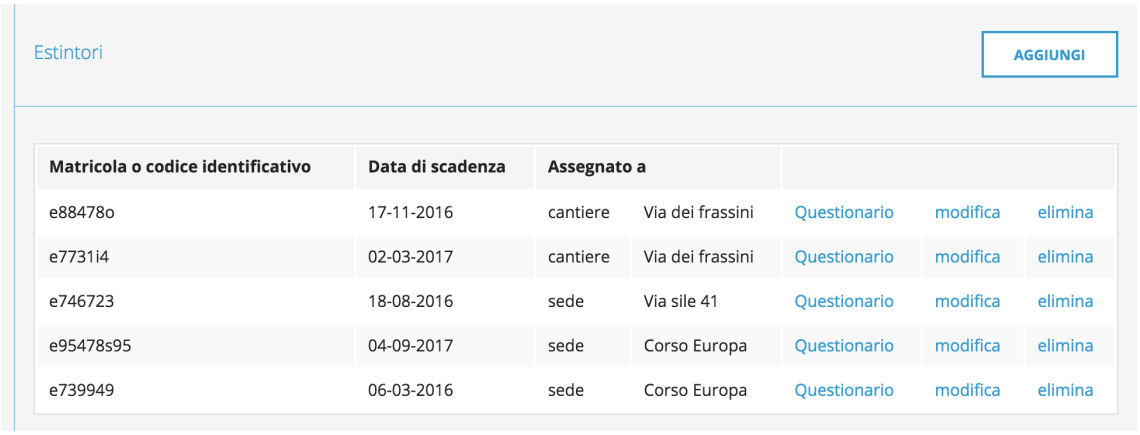



Figura 56: Schermata relativa agli Estintori in azienda.

#### Criticità incontrate

Scrivere meglio
Al fine di rendere agevole ed intuitiva la gestione dei $DPC_G$, è stato creato un meccanismo di gestione dinamica mediante chiamate $Ajax_G$.
Fare ciò è stato necessario introdurre un nuovo tipo di file di template di Rails(**.js.erb*) e strutturare i template **.html.erb* in modo tale da essere compatibili con il nuovo pattern

mediante invocazioni remote.
Questo tipo di attività, essendo molto verbosa e delicata nella gestione, ha richiesto più tempo rispetto a quanto pianificato.


Figura 57: Schermata relativa alla dotazione di estintori in un luogo .

## 6.3 Regole Drools

In questa sezione vengono riportate alcune tra le regole Drools più significative.

### 6.3.1 Riconoscimento di un indoviduo che ricopre una mansione per la quale non è formato

```
rule "Individuo ricopre una mansione per la quale non è formato"
when
$t: Training()
$d: Duty(trainings contains $t)
$i: Individual(duties contains $d, trainings not contains $t)
then
System.out.println($i.getFirstName() + ' ' + $i.getLastName() + "ha la mansione" +
$d.getName() + " ma non la formazione " + $t.getName() );
end
```

- $t filtro sulle formazioni;
- $d filtro sulle mansioni che necessitano della formazione $t;
- $i filtro su tutti gli individui che svolgono la mansione $d ma non sono in possesso della formazione $t.

Le corrispondenze di questa regola sono tutti gli individui individui che svolgono una qualunque mansione per la quale non sono correttamente formati.
Al verificarsi di queste condizioni, viene generato un allarme il cui contenuto è specificato dalla stampa disposta dalla $\text{RHS}_G$.

### 6.3.2 Mancanza del CPI per sedi con superficie maggiore di trecento metri quadri

```
rule "CPI assente e superficie e la superficie della sede è  maggiore di 300 metri quadri"
when
$s:    Location(surfaceArea > 300)
$a:    Answer(answerableType == "Location", answerableId == $s.getId(),
questionCode == "location_needs_CPI", response == "no")
then
System.out.println("La sede all\'indirizzo"+ $s.getAddress() +
 ", ha una superficie maggiore di 300 metri quadri, richiede quindi il CPI. ");
end
```

- $s filtro sulle sedi;
- $a filtro sulle risposte riferite alla sede $s relative alla presenza del Certificato Prevenzione incendi $(\text{CPI})_G$;

Le corrispondenze di questa regola sono tutte le risposte alle domande con risposta negativa relative alla presenza del CPI (Certificato Prevenzione Incendi) nelle sedi con più di trecento metri quadri di superficie.

# 7 Verifica e validazione

TODO Il codice prodotto è stato parzialmente sottoposto a test utizzando la gemma *Minitest* di Ruby on Rails.

# 8 Considerazioni finali

TODO Rivedere e scrivere qualcosa di decente Al termine dello stage sono stati analizzati i risultati ottenuti e sul valore aggiunto che esso ha portato.
Personalmente mi ritengo soddisfatto di come si è svolto lo stage soprattutto poiché mi ha permesso di lavorare in un progetto di dimensioni molto maggiori di quelle a cui sono stato abituato finora. Ho avuto l'opportunità di dare il mio contributo ad un software il cui obiettivo è aiutare le aziende a prevenire gli infortuni e mi piace credere che qualcuno di essi verrà evitato anche grazie al codice che ho scritto.
Mi sono trovato ad affrontare un'esperienza da full-stack developer, dovendomi assumere la responsabilità di codice sia lato back-end sia lato front-end. Ho potuto lavorare e studiare codice scritto da sviluppatori senior dal quale ho imparato molto. In particolare l'integrazione tra Drools e Ruby on Rails mi ha affascinato poiché mi ha fatto capire che anche linguaggi molto diversi e non compatibili nativamente, possono essere manipolati in modo da diventarlo.

Questo stage mi ha fatto crescere moltissimo in ambito lavorativo che personale. TODO

# A Migrazioni

Una migrazione è un file nel quale viene indicato cosa cambia nella struttura del database rispetto allo stato precedente ad essa e le variazioni ai dati ad essa conseguente. Questo permette a tutti gli sviluppatori di avere lo stato della struttura del database sempre aggiornato. Qui di seguito un esempio di migrazione che crea una tabella Bookmark (nel metodo statico up), e specifica cosa fare in caso di annullamento della migrazione (nel metodo statico down).

```
class CreateBookmarks < ActiveRecord::Migration
  def self.up
    create_table :bookmarks do |t|
      t.string :url
      t.string :title
      t.text :description

      t.timestamps
    end
  end

  def self.down
    drop_table :bookmarks
  end
end
```

Una volta eseguita la migrazione verrà creata una tabella Bookmarks con i tre campi *url*, *title* e *description*.

# B Inferenza e Motori di inferenza

Con il termine inferenza si intende il processo con il quale si passa da una proposizione vera ad una seconda proposizione la cui verità è derivata dal contenuto della prima.
Questo processo avviene mediante l'applicazione di regole, dette regole di inferenza. Al verificarsi di una o più condizioni (*antecedent*), queste regole traggono delle conclusioni ed agiscono di conseguenza sul sistema, sulla base di ciò che è definito nella seconda parte della regola (*consequent*).
Prendiamo ad esempio la regola `"Se piove apro l'ombrello"`.
L'*antecedent* è rappresentato dalla condizione: `"Oggi piove?"`. Se la condizione dell'*antecedent* è soddisfatta, verrà tratta la conclusione descritta nel *consequent*, in questo caso `"apro l'ombrello"`.
Questo genere di approccio assume maggior significato se applicato contemporaneamente ad un insieme di fatti o circostanze.
Un motore inferenziale (*inference engine*), è un software il cui algoritmo simula le modalità con cui la mente umana trae delle conclusioni logiche attraverso il ragionamento utilizzando le regole di inferenza. Le due modalità con cui si opera in questi contesti sono principalmente le seguenti:

- **Backward chaining**
  Rappresenta il ragionamento di tipo induttivo, il quale dall'analisi di informazioni di carattere particolare permette di ricavare informazioni di carattere generale.
  Nella pratica, si considerano una serie di obiettivi e si cerca tra i dati disponibili se ce ne sono di disponibili tali da supportare tutti gli obiettivi fissati.
  Un motore inferenziale utilizza questo approccio cercando tra le regole di inferenza finché non ne individua una che abbia il *consequent* che soddisfa l'obiettivo. Se non è provata la verità dell'*antecedent* della regola individuata, allora l'*antecedent* stesso diventa un nuovo obiettivo poiché, una volta soddisfatto, sarà soddisfatto anche l'obiettivo precedentemente individuato.
- **Forward chaining**
  Rappresenta il ragionamento di tipo deduttivo, ovvero permette di partire da principi di carattere generale per estrarne uno o più di carattere particolare.
  L'approccio utilizzato è quello di iniziare con i dati già disponibili ed usare le regole di inferenza per estrarne di nuovi fino a quando non si è raggiunto l'obiettivo fissato.
  Un inference engine che usa il forward chaining, itera sulle regole di inferenza eseguendo quelle che hanno disposizione tutte le informazioni per poter calcolare i nuovi dati e fermandosi quando è stato raggiunto l'obiettivo fissato.
  A differenza del backward chaining, questo è un approccio orientato ai dati ed effettua i *"ragionamenti"* al fine di ottenere delle risposte.
  È consigliabile rispetto al backward chaining nelle situazioni in cui i dati cambiano frequentemente, perché in seguito alla modifica, alla cancellazione ed all'immissione di nuovi dati, viene innescato il processo di inferenza dilazionando il costo computazionale nel tempo.

# C Sistemi Esperti

Un sistema esperto è un programma che cerca di riprodurre le capacità di decisione di una o più persone esperte in un determinato campo di attività.
Per il corretto funzionamento, è necessario che siano fornite procedure di inferenza sufficienti alla risoluzione dei problemi alla quale si vuole fornire risposta.
Un sistema esperto è sempre in grado di esibire i passaggi logici che hanno portato ad una particolare decisione.
Le principali componenti sono:

- **Base di conoscenza (Knowledge Base)**
  Componente che contiene tutte le regole necessarie al sistema per prendere le decisioni; non contiene i dati;
- **Motore inferenziale**
  (vedi Appendice B);
- **Interfaccia Utente**
  Componente che permette l'interazione fra il soggetto umano ed il software che deve dare risposta ad un suo particolare problema.

I sistemi esperti si dividono in due categorie principali: quelli basati su regole e quelli basati su alberi. Per la realizzazione del progetto di stage è stato utilizzato *Drools*, un framework per la realizzazione di sistemi esperti basati su regole. Non verranno quindi trattati su questa tesi i sistemi basati su alberi.
I sistemi esperti basati su regole sono dei programmi composti da regole della forma `IF condizioni THEN azione`.
La parte condizionale viene detta Left Hand Side ($\text{LHS}_G$), mentre quella relativa all'azione viene detta Right Hand Side ($\text{RHS}_G$).
Le regole vengono valutate ad ogni variazione sui dati che avviene sempre mediante l'introduzione di nuovi fatti.
Esempio:
Fatti :

- Mal di Testa
- Raffreddore
- Temperatura >=38

```
IF( (Mal di testa) AND (Raffreddore) AND (Temperatura >=38))
THEN ( Diagnosi: Influenza)
```

# D Algoritmo RETE

L'algoritmo RETE è stato inventato da Charles Forgy nel 1978 e successivamente perfezionato nel 1979.
Può essere separato in due parti:

- Compilazione delle regole;
- Esecuzione runtime.

Questo algoritmo prevede l'utilizzo di una sorta di rete che filtra i dati mano a mano che essi la attraversano. I nodi all'inizio della rete avranno, con probabilità molto alta, un numero elevato di match. Mano a mano che si scende in profondità, le corrispondenze tenderanno sempre più a diminuire di numero. In fondo alla rete, troveremo i nodi terminali.
Il funzionamento prevede la creazione di un nodo radice, attraverso il quale tutti gli oggetti entrano nella rete. Ogni oggetto, dopo essere passato per il nodo radice, viene subito indirizzato verso un altro nodo chiamato *ObjectTypeNode* che ha lo scopo di assicurare che ogni nodo venga inoltrato verso nodi che sanno gestire oggetti con un tipo compatibile a quello dell'oggetto in analisi[2].
Gli *ObjectTypeNode*, propagano gli oggetti verso dei nodi detti *AlphaNode*, ognuno dei quali verifica una condizione.
Tramite altre due tipologie di nodi (*JoinNode* e *NotNode*), si verificano condizioni che coinvolgono oggetti di diverso tipo, per poi giungere infine ai nodi terminali.
Una volta raggiunto un nodo terminale, si ha la garanzia che la regola è stata soddisfatta e viene eseguita l'azione descritta nella $\text{RHS}_G$ corrispondente.

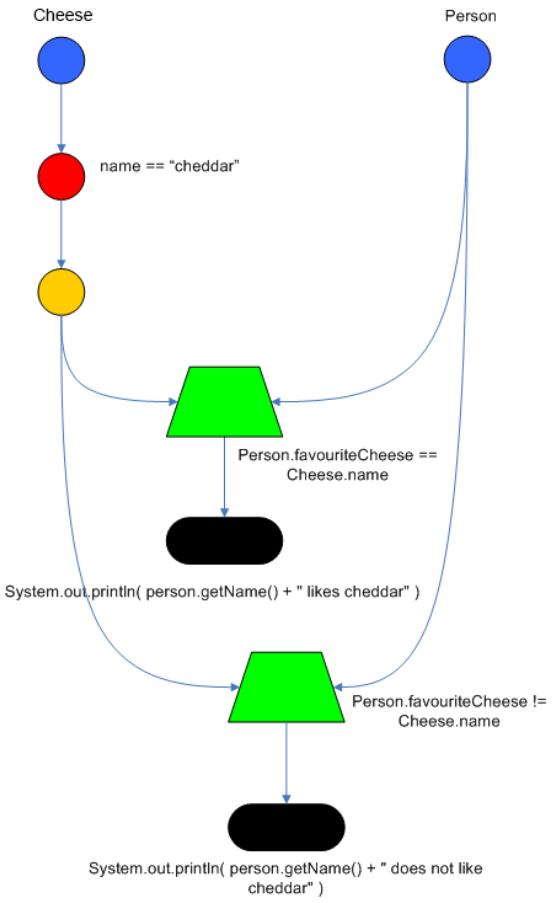



Figura 58: Esempio di flusso di valutazione di una regola Drools.

[2]Tecnicamente questo passaggio avviene mediante l'invocazione dell'operatore `instanceof` di Java.

Un esempio di flusso di funzionamento è raffigurato nella Figura 58.
In particolare i nodi azzurri sono degli *ObjectTypeNode* e rappresentano nell'ordine formaggi e persone.
Il nodo rosso, invece, è di tipo *AlphaNode* e verifica la condizione che il nome del formaggio sia "cheddar".
Il nodo giallo è di tipo *LeftInputAdapterNode* e serve per scegliere dove dirottare il flusso a seguito del soddisfacimento o meno di una condizione prevista da un *AlphaNode*.
Se la condizione prevista dall'*AlphaNode* è verificata, il flusso viene inoltrato al primo punto di join (identificato da un trapezio verde) che verifica se il formaggio preferito dalla persona è il *cheddar*. si dice punto di join perché per operare ha bisogno delle informazioni provenienti da due diverse entità gestite da degli *ObjectTypeNode* distinti.
se la condizione prevista dall'*AlphaNode* non è verificata, si passa al secondo punto di join.
In caso di verifica delle condizioni dei *JoinNode*, si giunge ai nodi terminali ovvero al soddisfacimento delle condizioni che permette di prendere una decisione. In questo caso viene solo stampata una stringa, ma in generale è possibile scegliere l'azione più adatta al contesto di utilizzo.

# Glossario

**Ajax** Tecnica di sviluppo software per la realizzazione di applicazioni web interattive. Lo sviluppo di applicazioni HTML con AJAX si basa su uno scambio di dati in background fra web browser e server, che consente l'aggiornamento dinamico di una pagina web senza esplicito ricaricamento da parte dell'utente.

**ASPP** Addetto al Servizio di Prevenzione e Protezione

**Asseverazione** Con asseverazione si intende una scelta volontaria di una impresa edile al fine di dimostrare l'impegno per la prevenzione, salute e sicurezza nei luoghi di lavoro. Opera mediante la certificazione di conformità dei modelli di organizzazione e gestione aziendali e mediante visite a campione nei cantieri. L'asseverazione offre benefici economici alle aziende che la richiedono e garantisce efficacia esimente rispetto alla responsabilità amministrativa delle imprese.

**ATECO** Sigla della classificazione ISTAT(Istituto Nazionale di Statistica) per determinare la tipologia delle attività economiche (**AT**tività **ECO**nomiche).

**AWS** Amazon Web Services

**back-end** É la componente software responsa di gestire le interazioni provenienti dall'esterno del sistema e restituire le risposte a seguito delle opportune elaborazioni. Si occupa inoltre di gestire la persistenza dei dati interagendo con eventuali basi di dati. Il back-end è responsabile dell'utilizzo dei dati forniti dal front-end.

**Breadcrumb** Letteralmente si traduce in *briciole di pane*. Rappresenta il percorso dalla home page, o comunque da una radice, al punto corrente della navigazione.
Il nome deriva dalla fiaba di Hansel e Gretel, in cui Hansel sparge lungo il suo cammino delle briciole di pane per poter riconoscere la strada precedentemente percorsa e poter tornare sui suoi passi.

**browser** Un browser è un programma che consente di visualizzare i contenuti delle pagine web e di interagire con esse.

**CDN** Sta per *Content Delivery Network*, ovvero rete di consegna di contenuti. Questo meccanismo viene utilizzato per reperire il codice relativo agli strumenti di terze parti direttamente dalla rete, senza tenere una versione del codice stesso nel proprio server web.

**Concern** Un concern rappresenta un modulo di Rails utilizzato per facilitare la gestione di funzionalità assegnabili a più classi ma allo stesso tempo gestibili in modo atomico.

**CPI** Certificato Prevenzione incendi

**DOM** Document Object Model

**DPC** Dispositivi di Protezione Collettivi

**DPI** Dispositivi di Protezione Individuale

**DVR** Documento di Valutazione dei Rischi

**ERB** Sta per Embedded RuBy.
È il sistema per la gestione dei template in Rails. ERB permette di generare file in moltissimi formati generando a tutti gli effetti dei template. In particolare, le informazioni vengono reperite mediante Ruby On Rails, ma la struttura del file nel formato specificato rimane invariata, adattando a tutti gli effetti le informazioni presenti al mommento della invocazione ad una struttura definita in precedenza.

**framework** Architettura software sulla quale un software può essere progettato e realizzato.

**front-end** Parte di un sistema software che gestisce l'interazione con l'utente o con sistemi esterni che producono dati di ingresso.

**HTML** HyperText Markup Language. Linguaggio di markup per la strutturazione delle pagine web.

**INAIL** Istituto Nazionale per l'Assicurazione contro gli Infortuni sul Lavoro

**JavaBean** Letteralmente si traduce in "Chicchi di Java". Le JavaBean sono classi scritte in Java seguendo le seguenti convenzioni:

- Un costruttore senza parametri;
- Devono essere definiti tutti i metodi accessori per le variabili manipolabili (in particolare getter e setter);
- La classe dovrebbe essere serializzabile;
- La classe non dovrebbe contenere metodi per la gestione degli eventi.

Le classi JavaBean, sono spesso utilizzate per incapsulare più oggetti in un singolo oggetto in modo da passare un solo oggetto ai metodi che necessitano di tutti qusti oggetti come parametri.

**Javascript** TODO

**JVM** Java Virtual Machine

**LHS** Sta per Left Hand Side. Rappresenta l'insieme delle condizioni che verificano una regola del rule engine Drools.

**MVC** Model View Controller

**POJO** Sta per `Plain Old Java Object`. Rappresenta un oggetto Java non legato ad alcuna restrizione diversa da quelle costrette dalla specifica del linguaggio Java.

**Reflection** Capacità di un programma di eseguire elaborazioni che hanno per oggetto il programma stesso, ed in particolare la struttura del suo codice sorgente. Un utilizzo comune, è l'utilizzo di questo approccio per individuare il nome della classe oppure degli attributi a partire da una istanza di un oggetto.

**REST** Tipo di architettura software per i sistemi di ipertesto distribuiti come il World Wide Web, si riferisce ad un insieme di principi di architetture di rete, i quali delineano come le risorse sono definite e indirizzate.

**Reverse proxy** Si tratta di un tipo particolare di proxy che recupera le informazioni per conto di un client da uno o più server. Le risorse ottenute vengono restituite al client come se provenissero direttamente dal Proxy server, rendendo trasparente il recupero distribuito delle informazioni all'utente. Questo tipo di approccio risulta molto utile nei contesti in cui il recupero delle informazioni risulta oneroso perché, permettendo la distribuzione su più macchine, si ottiene un bilanciamento del carico.

**RHS** Sta per Right Hand Side. Rappresenta l'insieme delle azioni da intraprendere se tutte le condizioni presenti nella LHS del rule engine Drools sono soddisfatte.

**RLS** Rappresentante dei Lavoratori per la sicurezza

**RSPP** Responsabile del Servizio di Prevenzione e Protezione

**SaaS** Software as a Service

**SASS** È un preprocessore CSS che analizza il codice scritto in SCSS e ne elabora il risultato. Si tratta a tutti gli effetti di una estensione di CSS3 che aggiunge:

- Regole annidate;
- Variabili;
- Mixin;
- Ereditarietà tra i selettori.

Home page del progetto: `http://sass-lang.com/`.

**SCSS** Sta per Sassy CSS ed è la sintassi utilizzata da SASS.

**WYSIWYG** What You See Is What You Get