

ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

TESI DI LAUREA

in

Fondamenti di Informatica T-1

**PROGETTAZIONE E SVILUPPO DI UN'APPLICAZIONE WEB A
SUPPORTO DI UN SISTEMA ESPERTO IN AMBITO LEGALE**

CANDIDATO
Davide Laffi

RELATORE:
Prof. Federico Chesani

Anno Accademico 2018/19

Sessione I

INDICE

INTRODUZIONE	4
1. ANALISI DEI REQUISITI	6
1.1 REQUISITI UTENTE	6
1.2 CASI D'USO	7
1.3 SCENARI CASI D'USO	8
1.4 DIAGRAMMI DI SEQUENZA	12
1.5 REQUISITI DI SISTEMA	15
2. BACKGROUND TECNOLOGIE SPRING E PROLOG	20
2.1 SPRING MVC	20
2.2 SPRING WEBFLOW	23
2.3 SPRING DEPENDENCY INJECTION	24
2.4 PROLOG E JPL	26
3 PROGETTO	28
3.1 PROGETTAZIONE ARCHITETTURALE	28
3.2 MODELLO DEL DOMINIO	29
3.3 PROGETTO DI DETTAGLIO	30
3.4 ARCHITETTURA LOGICA: INTERAZIONE	33

4. IMPLEMENTAZIONE	35
4.1 LIMITI E SVILUPPI FUTURI.....	35
4.2 INTERFACCE GRAFICHE	36
4.3 CONFIGURAZIONE DELL'APPLICAZIONE	42
 CONCLUSIONI.....	46
 BIBLIOGRAFIA	47

Introduzione

Un problema sorto in ambito giuridico è quello di determinare la giurisdizione competente a risolvere una certa azione legale. In modo molto semplificato, l'obiettivo è quello di riuscire a capire quale stato e, più precisamente, quale corte ha il compito di risolvere un contenzioso tra convenuti di nazionalità diverse. Ovviamente, non ci sono dubbi che, nel caso in cui i convenuti siano residenti o domiciliati entrambi nello Stato italiano, sia applicabile la giurisdizione italiana.

Un discorso diverso può essere fatto, però, quando l'azione legale viene intrapresa tra convenuti di nazionalità diversa, per esempio tra una persona italiana e una francese. Sorge spontaneo chiedersi se a giudicare la controversia sarà competente l'autorità italiana o quella francese.

Ancora più articolato il caso in cui l'azione legale riguardi due persone di nazionalità diverse ed il fatto oggetto del contenzioso accada in un terzo stato differente.

Un problema apparentemente risolvibile senza troppe considerazioni, richieda in realtà un vero e proprio studio di tutte le variabili in gioco.

Oltre alla residenza delle persone, devono essere considerati altri parametri per trovare una soluzione. Infatti, vi è un grande cambiamento se la persona di una certa nazionalità è colei che ha iniziato l'azione legale oppure si sta difendendo dall'attacco di un'altra.

L'idea è quella di creare un'applicazione che permetta di acquisire in ingresso i parametri di una azione legale e di riuscire in qualche modo a fornire come risultato lo stato e la corte competente per risolvere la causa.

Il CIRSFID (centro interdipartimentale di ricerca in storia del diritto, filosofia e sociologia del diritto e informatica giuridica) ha richiesto di realizzare questa applicazione esponendo il problema.

Dopo una riunione con i committenti dell'applicazione, è stato effettuato uno studio delle tecnologie che sono state utilizzate ed è stata prima progettata e poi sviluppata l'intera applicazione Web. Dai ricercatori del CIRSFID è stato fornito un componente esterno che effettua il vero e proprio calcolo della giurisdizione competente e che è stato perfettamente integrato all'interno dell'applicazione.

Nel primo capitolo viene affrontata l'analisi dei requisiti che è la prima fase nella creazione di un progetto. Durante questa fase è avvenuto l'incontro con i committenti che hanno spiegato il ruolo dell'applicazione e le funzionalità che dovevano essere implementate. In seguito, sono stati redatti casi d'uso e diagrammi di sequenza, in modo da spiegare in modo chiaro come l'applicazione si sarebbe comportata.

Nel secondo capitolo sono spiegate le tecnologie principali che sono state utilizzate all'interno del progetto. In particolare, il framework Spring, il linguaggio Prolog e la libreria JPL per interfacciamento tra ambiente Java e Prolog.

All'interno del terzo capitolo, si entra nel cuore della progettazione dell'applicazione. Sono presenti diagrammi del dominio, diagrammi dei package e delle classi. Inoltre, viene effettuata la scelta del pattern architetturale.

Nell'ultimo capitolo, infine, si affronta la parte di implementazione del sistema. Vengono mostrate le interfacce grafiche dell'applicazione e alcune parti di codice che sono ritenute significative, in particolare per la configurazione dell'applicazione.

1. Analisi dei Requisiti

1.1 Requisiti Utente

È stata effettuata una riunione con i committenti, durante la quale sono state delineate le caratteristiche che deve avere l'applicazione web che verrà sviluppata. Il compito principale dell'applicazione è, in generale, quello di riuscire a calcolare stato e corte competente rispetto ad una certa causa inserita nel sistema. Più nel dettaglio, l'applicazione si serve di un programma fornito esternamente, scritto in linguaggio Prolog, che effettuerà il vero e proprio calcolo, mentre un'altra parte dell'applicazione raccoglierà i dati dall'utente. Verrà quindi progettato un ottimo collegamento tra le parti del sistema in modo che i dati inseriti in linguaggio naturale dall'utente siano convertiti automaticamente in input del programma Prolog e, allo stesso modo, il risultato del programma Prolog sia visualizzabile all'utente.

All'interno dell'applicazione, l'utente può scegliere se autenticarsi o rimanere utente provvisorio. Nel caso di utente provvisorio, egli potrà soltanto inserire una nuova causa e ricevere il risultato che gli interessa. L'utente registrato, invece, avrà in aggiunta la possibilità di visualizzare l'elenco delle cause precedentemente inserite e di ottenere un report PDF delle cause salvate. Infine, esiste una sezione dedicata all'amministratore del sistema, che può decidere di caricare un file Excel o un file Prolog. Questa sezione serve per aggiornare in modo semplice le domande e le possibili risposte, nel caso del foglio Excel, il programma che esegue il calcolo, nel caso del file Prolog.

Nell'aggiunta di una nuova causa, l'applicazione deve permettere di visualizzare un form in cui sono presenti domande in linguaggio naturale e a cui l'utente è tenuto a rispondere. Le domande e le risposte verranno acquisite dinamicamente dal file Excel caricato all'interno dell'applicazione. Una volta risposto alle domande necessarie, l'utente cliccherà in un'apposita area e gli input inseriti nell'applicazione verranno passati al programma Prolog che calcolerà il risultato. Infine, in un'ultima ulteriore schermata, all'utente saranno mostrati lo stato e la corte competente per risolvere il caso inserito. Il form mostrerà o meno alcune domande, sulla base delle risposte precedentemente fornite.

1.2 Casi D'uso

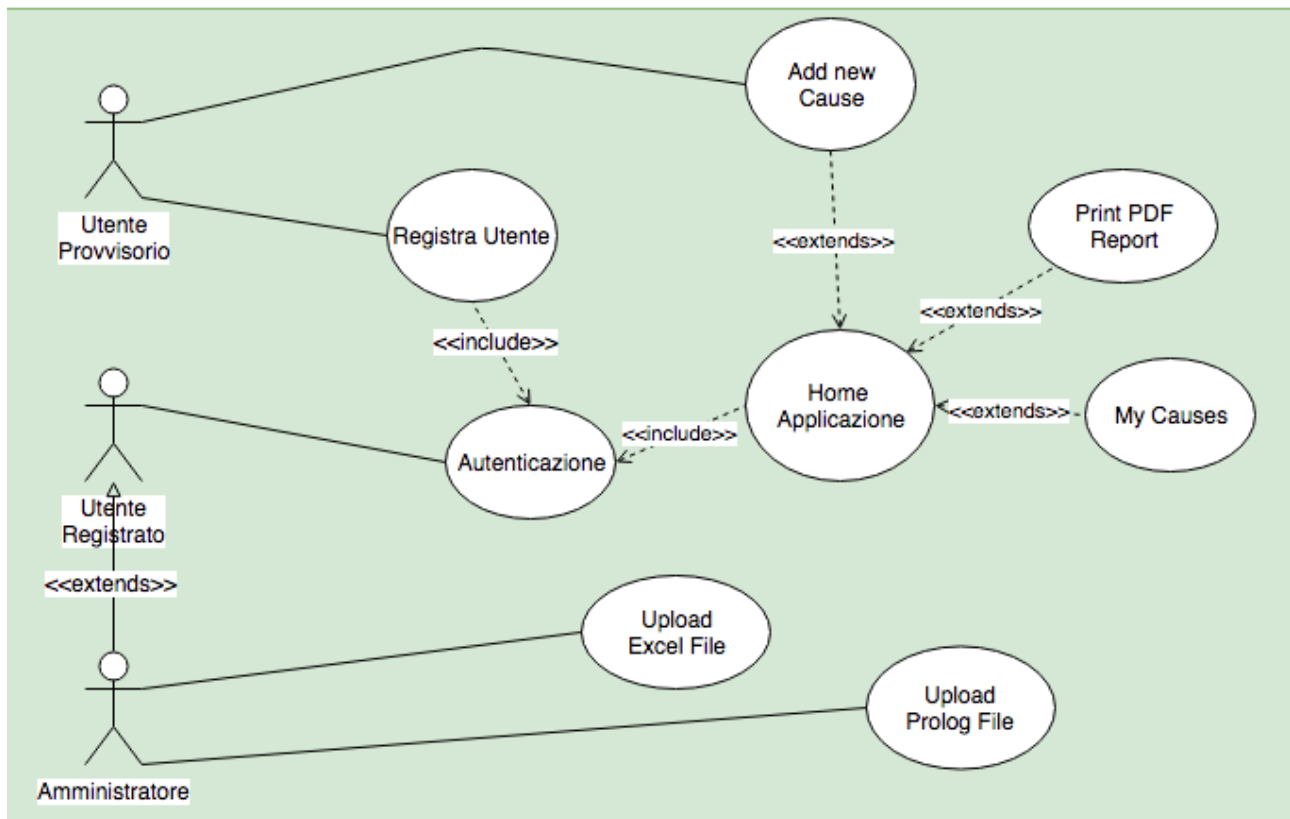


Figura 1.1 – Diagramma dei Casi d'uso

L'utente provvisorio ha la possibilità di registrarsi all'interno dell'applicazione ottenendo gli stessi privilegi di un utente registrato, altrimenti potrà semplicemente aggiungere una nuova causa e ottenere il risultato dal programma.

L'utente registrato, invece, può, dopo aver effettuato l'autenticazione con le proprie credenziali, accedere alle altre sezioni dell'applicazione:

- **My Causes:** visualizzazione delle cause precedentemente aggiunte
- **Print PDF report:** permette di stampare un report della causa appena inserita con relativo risultato

Infine, l'amministratore ha i privilegi di poter caricare un nuovo file Prolog per aggiornare e modificare il programma Prolog sottostante e un nuovo file formato Excel per aggiornare e modificare gli input dell'applicazione e le possibili risposte.

1.3 Scenari Casi d'uso

Titolo	Autenticazione
Descrizione	Modalità di accesso al sistema da parte dell'Utente
Attori	Utente Registrato
Relazioni	Registra Utente, Home Applicazione
Precondizioni	
Postcondizioni	L'Utente è autenticato presso il sistema
Scenario principale	<ol style="list-style-type: none"> 1. Il sistema presenta la schermata di accesso all'Utente 2. L'Utente immette le proprie credenziali 3. L'Utente preme il bottone di accesso 4. Il sistema verifica la correttezza delle credenziali 5. Viene mostrata la pagina dedicata all'Utente Registrato
Scenari alternativi	<p>Scenario: Le credenziali non sono corrette</p> <ol style="list-style-type: none"> 1. Viene mostrato un avviso all'Utente 2. Il sistema mostra nuovamente la schermata di accesso al sistema
Requisiti non funzionali	La password inserita non deve essere visibile in maniera esplicita sulla schermata: sicurezza delle informazioni

Titolo	Registra Utente
Descrizione	L'utente non registrato decide di registrarsi all'applicazione
Attori	Utente Provvisorio
Relazioni	Autenticazione
Precondizioni	Possedere un indirizzo mail non ancora registrato
Postcondizioni	L'utente sarà registrato al sistema e avrà delle credenziali per poter accedere ad esso
Scenario principale	<ol style="list-style-type: none"> 1. L'utente avrà la possibilità di inserire username e password 2. L'utente inserisce username e password 3. Verrà visualizzato un messaggio di conferma, l'utente sarà ora registrato all'applicazione
Scenari alternativi	<ol style="list-style-type: none"> 1. L'utente inserisce una mail già presente nel sistema

	2. Verrà visualizzato un messaggio di errore e si darà all'utente la possibilità di aggiungere una mail non registrata
Requisiti non funzionali	1. Facilità di navigazione delle schermate 2. Velocità di esecuzione delle operazioni richieste

Titolo	Home Applicazione
Descrizione	Schermata di home dell'utente
Attori	Utente Registrato
Relazioni	Autenticazione, Add new Cause, Print PDF Report, My Causes
Precondizioni	Aver effettuato l'autenticazione al sistema
Postcondizioni	
Scenario principale	1. L'utente visualizza le opzioni disponibili da poter scegliere 2. L'utente sceglie una delle opzioni
Scenari alternativi	
Requisiti non funzionali	1. Facilità e chiarezza di visualizzazione della schermata

Titolo	Add new Cause
Descrizione	Inserire una nuova Causa all'interno del sistema
Attori	Utente Registrato, Utente Provvisorio
Relazioni	Home Applicazione
Precondizioni	
Postcondizioni	1. Se l'utente è registrato, la causa verrà salvata all'interno di "My Causes"

Scenario principale	<ol style="list-style-type: none"> 1. All'utente viene visualizzato un form con domande e possibili risposte a proposito della nuova causa 2. L'utente compila il form e conferma il risultato 3. All'utente viene visualizzato il risultato della giurisdizione competente
Scenari alternativi	
Requisiti non funzionali	<ol style="list-style-type: none"> 1. Chiarezza e coerenza delle domande all'interno del form

Titolo	Print PDF Report
Descrizione	Permette di scaricare un file PDF con i risultati dell'applicazione
Attori	Utente Registrato
Relazioni	Home Applicazione
Precondizioni	<ol style="list-style-type: none"> 1. L'utente deve aver inserito almeno una causa
Postcondizioni	<ol style="list-style-type: none"> 1. L'utente avrà un file PDF da poter scaricare e stampare
Scenario principale	<ol style="list-style-type: none"> 1. L'utente sceglie una causa tra quelle precedentemente salvate 2. L'utente clicca su "ottiene file PDF" 3. L'utente visualizza un report con i dati della causa inseriti e il corrispondente risultato calcolato dall'applicazione
Scenari alternativi	
Requisiti non funzionali	<ol style="list-style-type: none"> 1. Completezza del report PDF

Titolo	My Causes
Descrizione	Raccoglie le cause precedentemente inserite dall'utente
Attori	Utente Registrato
Relazioni	Home Applicazione
Precondizioni	
Postcondizioni	
Scenario principale	<ol style="list-style-type: none"> 1. Vengono visualizzate all'utente tutte le cause precedentemente aggiunte 2. L'utente ha la possibilità di stampare il report di una delle cause o di eliminarle
Scenari alternativi	
Requisiti non funzionali	

Titolo	Upload Excel File
Descrizione	Permette di caricare un file Excel
Attori	Amministratore
Relazioni	
Precondizioni	Il file Excel deve essere ben formato
Postcondizioni	Gli utenti vedranno le domande e le possibili risposte per aggiungere una nuova causa aggiornate
Scenario principale	<ol style="list-style-type: none"> 1. L'amministratore effettua l'upload di un nuovo file Excel contenente domande e possibili risposte
Scenari alternativi	
Requisiti non funzionali	<ul style="list-style-type: none"> • Velocità di upload e sostituzione del file Excel

<i>Titolo</i>	Upload Prolog File
<i>Descrizione</i>	Permette di caricare un file Prolog
<i>Attori</i>	Amministratore
<i>Relazioni</i>	
<i>Precondizioni</i>	Il programma Prolog deve essere funzionante e ben scritto
<i>Postcondizioni</i>	L'algoritmo di calcolo della giurisdizione competente sarà aggiornato
<i>Scenario principale</i>	1. L'amministratore effettua l'upload di un nuovo file Prolog
<i>Scenari alternativi</i>	
<i>Requisiti non funzionali</i>	<ul style="list-style-type: none"> • Velocità di upload e sostituzione del file Prolog

1.4 Diagrammi di Sequenza

Vengono riportati in modo molto schematico i diagrammi di sequenza di ogni caso d'uso. In generale in blu è indicato l'attore protagonista, in rosso le view, che saranno le schermate che vengono visualizzate all'utente e, in verde, i controller che implementano la logica di business dell'applicazione (metodi e algoritmi dell'applicazione)

Add new Cause

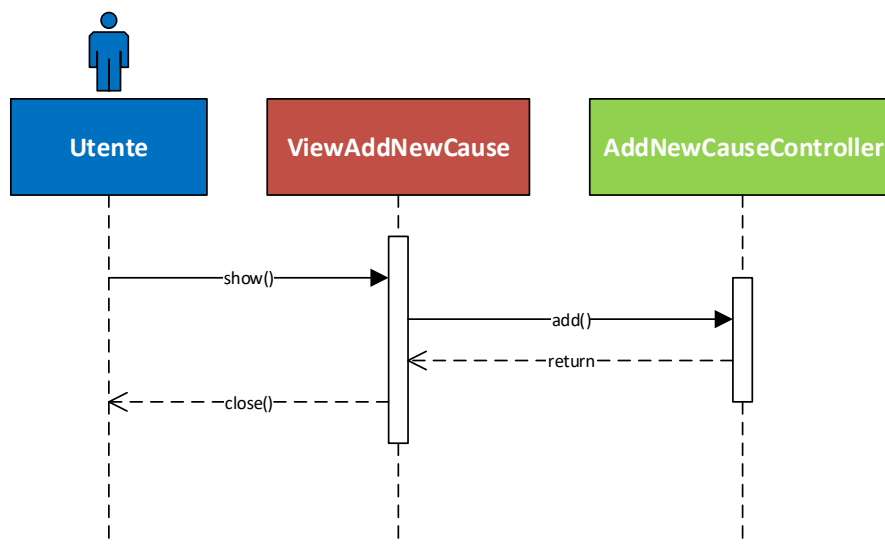


Figura 1.2 – Diagramma di sequenza aggiunta nuova Causa

Il diagramma mostra, in modo molto schematico, come protagonista l'utente a cui viene visualizzata la "ViewAddNewCause" in cui saranno presenti tutti i form in cui inserire i dati per aggiungere una nuova causa. Una volta terminato, verrà richiesto al controller, che implementa la logica di business dell'applicazione di aggiungere la causa.

My Causes

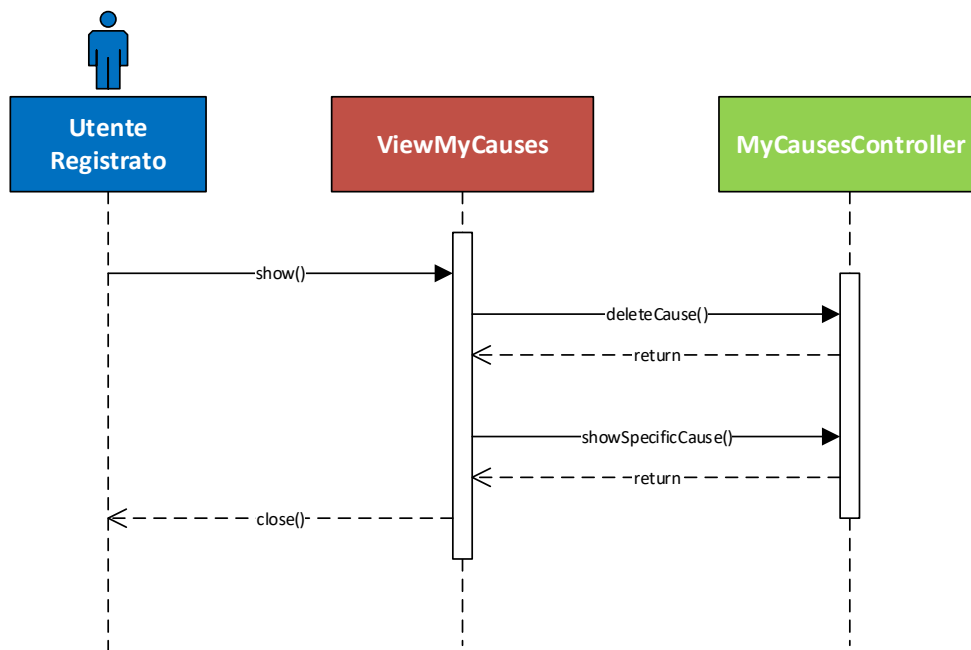


Figura 1.3 – Diagramma di sequenza my Causes

All'utente registrato che sceglierà di visualizzare la sezione "My Causes" verrà visualizzato l'elenco delle cause precedentemente inserite e potrà scegliere di visualizzarne una in modo dettagliato, ovvero con tutti i parametri inseriti e il risultato ottenuto, oppure di cancellarne una.

Print PDF Report

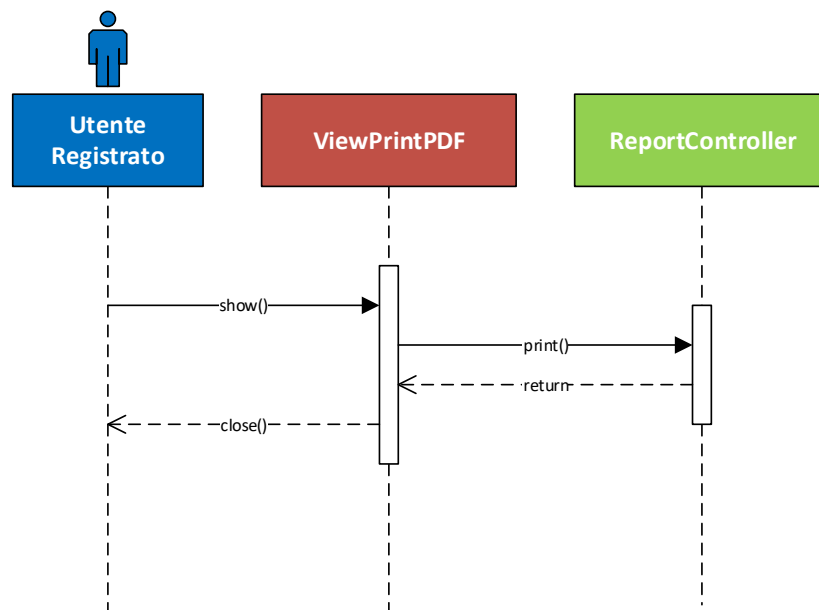


Figura 1.4 – Diagramma di sequenza print PDF Report

All'utente registrato verrà visualizzato un elenco schematico delle cause precedentemente aggiunte. Scegliendo una delle cause, potrà ottenere il report della causa in formato PDF tramite l'opportuno controller.

Upload Prolog File

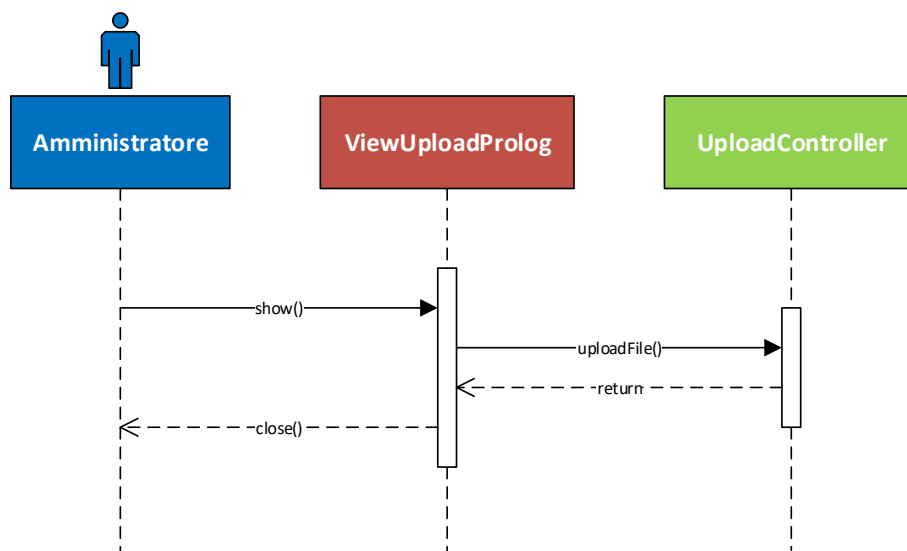


Figura 1.5 – Upload Prolog File

L'amministratore, una volta autenticato correttamente, potrà caricare un file Prolog. All'amministratore verrà visualizzata una semplice schermata in cui vi sarà l'opzione di caricare un nuovo file Prolog che con l'aiuto del controller sostituirà quello precedente.

Upload Excel File

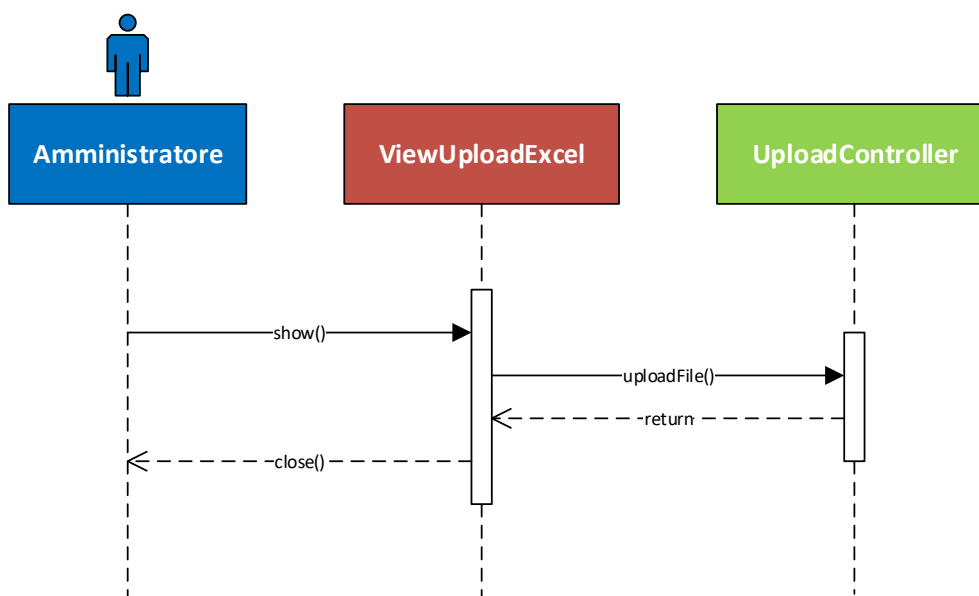


Figura 1.6 – Upload Excel File

La sequenza di operazione è analoga al caricamento di un file Prolog. L'amministratore visualizzerà una schermata dove gli sarà permesso di effettuare l'upload di un file Excel. Mediante un controller, il file caricato sostituirà quello precedente.

1.5 Requisiti di Sistema

Requisiti Funzionali

L'applicazione permette di inserire i dati relativi ad una causa nelle diverse categorie: Claim, Person, Contract. In cui Claim è l'oggetto della causa per cui si è creato un conflitto, Person sono le persone coinvolte all'interno della causa e Contract, se presente, è il contratto oggetto della discussione.

1. Sezione parametri di Input:

- Il sistema permette all'utente di inserire una nuova causa

- **Per ogni nuova causa**, l'applicazione prende in ingresso diversi parametri che possono essere divisi in tre macro-categorie:
 - Claim (Affermazione)
 - Person (Persona)
 - Contract (Contratto)
- **Per ogni nuova causa**, deve esserci un solo **Claim** ed ha come parametri di ingresso:
 - Matter: Civil/Commercial
 - Grade: First/Appeal
 - Type: Original/Counter/Incidental
 - Object: Contract/Tort/Ownership/Rights in Rem/Liability/Trust/Criminal Proceeding/Damages Restitution/No more than 6 months
 - Seised Country
 - Seised City
 - State which has Jurisdiction for civil proceedings
 - City which has Jurisdiction for civil proceedings
- I parametri "Object" e "Jurisdiction for civil proceedings" non sono singoli, ma possono essere presenti più volte (da 0 a N volte) per ogni Claim
- Ogni Claim ha un campo "id" che è un numero casuale associato ad essa ed è univoco all'interno del sistema, il campo può essere definito dall'utente oppure viene associato direttamente dal sistema in modo automatico (default)
- **Per ogni nuova causa**, possono esserci un numero indefinito di **Person** (da 1 a N persone) ed i parametri di ingresso sono:
 - Nature: Legal/Natural
 - Role: Plaintiff/Defendant/Third Party
 - Type: Consumer/Business/Employer/Employee/Insurer/Trust
 - Country Work
 - City Work
 - Country Activity
 - City Activity
 - Country Domicile
 - City Domicile
 - Country Establishment
 - City Establishment

- Ogni Person ha un campo “id” che è un numero casuale associato ad essa ed è univoco all’interno del sistema, il campo può essere definito dall’utente oppure viene associato direttamente dal sistema in modo automatico (default)
- **Per ogni nuova causa, può esserci al massimo un Contract (0 o 1):**
 - Obligation: Sale/Loan/Transport/Accommodation
 - Obligation Target Use: Commercial/Private/Sale
 - Consideration: Sale/Loan/Transport/Accommodation/Insurance/Employment/Consumer/Tenancy
 - Obligation Target: Goods/Service/Ship/Credit/Baggage/Immovable Property
 - Immovable Property Country
 - Immovable Property City
 - Place of Provision Country
 - Place of Provision City
 - Type: Sale of Goods/Provision of Service
- Uno stesso contratto può avere Consideration (da 0 a N)
- Uno stesso contratto può avere più Type (da 0 a N)
- Uno stesso contratto può avere più Place of Provision (da 0 a N)
- Ogni Contract ha un campo “id” che è un numero casuale associato ad essa ed è univoco all’interno del sistema, il campo può essere definito dall’utente oppure viene associato direttamente dal sistema in modo automatico (default)

2. Sezione Output:

- L’applicazione, una volta completato l’inserimento di tutti i parametri, mediante un apposito pulsante, procederà al calcolo del risultato
- In risposta all’utente vengono forniti: “Stato”, “Giudice Competente”, “Legislazione Competente”

3. Sezione relativa al programma Prolog

- All’interno dell’applicazione è presente un programma Prolog che effettua il calcolo del risultato dati certi parametri di ingresso
- Il sistema permette di caricare un file Prolog

- Una volta caricato il file, esso sostituirà automaticamente il vecchio programma Prolog con la nuova versione in modo da facilitare l'aggiornamento dell'applicazione

4. Sezione Report PDF

- Il sistema permette di generare un file PDF con indicati i parametri di ingresso inseriti dall'utente e il risultato calcolato dall'applicazione

5. Sezione salvataggio Query e Autenticazione

- Il sistema permette di autenticarsi come utenti in modo da poter salvare i propri casi inseriti e usarli in futuro
- L'applicazione permette di salvare momentaneamente i parametri di input inseriti per una particolare causa senza perderli in caso di chiusura

6. Sezione relativa al caricamento automatico dei parametri da un foglio Excel

- L'applicazione permette di caricare dinamicamente le domande da porre all'utente e le possibili risposte disponibili per ognuna da un foglio Excel
- Vi è la possibilità di caricare nuove versioni del foglio Excel in modo da apportare modifiche

È stata indicata come **priorità** da parte dei committenti, quella di creare l'architettura web dell'applicazione e implementare la funzionalità di trattazione di una nuova causa e calcolo del risultato del programma Prolog. Gli altri requisiti verranno implementati in futuro

Requisiti non Funzionali

- Facilità d'uso dell'applicazione, in particolare nell'aggiunta di una nuova causa
- Velocità del calcolo del risultato dopo aver inserito i parametri all'interno del form, quindi comunicazione rapida tra applicazione web e programma Prolog
- Rappresentazione chiara e semplice delle interfacce e in particolare di quelle in cui si necessita di inserimento dei dati. In particolare, semplicità e chiarezza delle domande in linguaggio naturale all'interno dei form
- Protezione dell'area dedicata all'amministratore, non deve succedere che un utente normale o registrato ottenga i privilegi dell'amministratore

- Non deve succedere che una volta inserito un nuovo file (Prolog o Excel), l'applicazione non funzioni più

2. Background tecnologie Spring e Prolog

Viene fornita una trattazione delle tecnologie principali utilizzate per la progettazione dell'applicazione.

2.1 Spring MVC

Per la struttura e l'implementazione dell'applicazione, è stato scelto di utilizzare Spring. Questa soluzione si sposa perfettamente con il pattern architetturale MVC, infatti Spring implementa pienamente l'approccio del pattern, mantenendo invariati sia i concetti che la nomenclatura [1]. Anche all'interno di un'applicazione Spring MVC troveremo quindi:

1. **Model**, caratterizzati dalle classi che rappresentano gli oggetti del sistema, generalmente ogni classe rappresenta un oggetto definito come Bean (oggetti POJO, ovvero con un costruttore vuoto e soltanto metodi get e set per accedere alle proprietà dell'oggetto, serializzabili)
2. **View**, in un'applicazione Web come questa, sono generalmente caratterizzate da pagine JSP o HTML che realizzano la grafica dell'applicazione
3. **Controller**, sono classi che restano in ascolto di una richiesta e, all'atto di segnalazione di una classe del model o delle view, gestiscono la richiesta dell'utente

Spring MVC framework fornisce elementi precostruiti che possono essere utilizzati per sviluppare un'applicazione basata su pattern MVC. L'intero modello è costruito intorno al “**DispatcherServlet**”, che è una servlet (infatti eredita dalla classe *HttpServlet*) che ha il compito di gestire tutte le richieste e risposte HTTP. Lo schema successivo rappresenta come avviene la gestione di una richiesta e la creazione della relativa risposta HTTP.

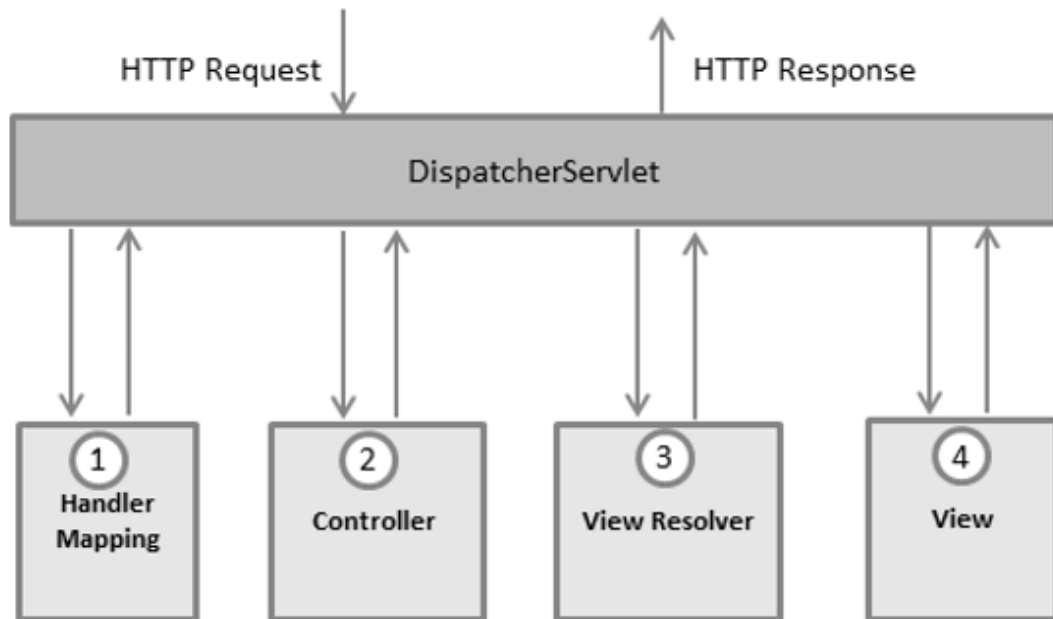


Figura 2.1 – Gestione HTTP request e response dal DispatcherServlet

Quando al sistema arriva una qualunque HTTP request, essa viene gestita dalla servlet principale del sistema. L'obiettivo del DispatcherServlet è quello di trovare l'oggetto (il controller) in grado di gestire in modo efficace ed esaustivo la richiesta e, per farlo, si affida al "Handler Mapping". Quest'ultimo, che in *Figura 2.1* vediamo marcato con (1), è un'interfaccia che può essere implementata dal programmatore e il cui scopo è di passare la richiesta HTTP al Controller più adatto. In seguito, il controller, in *Figura 2.1* marcato con (2), elabora la richiesta, modifica i dati del model se necessario e restituisce al DispatcherServlet il nome della view che permetterà la visualizzazione dei dati richiesti. A questo punto, la servlet richiede l'utilizzo dell'oggetto "View Resolver", in *Figura 2.1* marcato con (3), che ha il compito di mappare i nomi delle View con le rispettive View vere e proprie. Infine, il DispatcherServlet passa il model trattato dal Controller alla view che verrà visualizzata all'utente e invia la HTTP response al browser che si occuperà del render della view. [2]

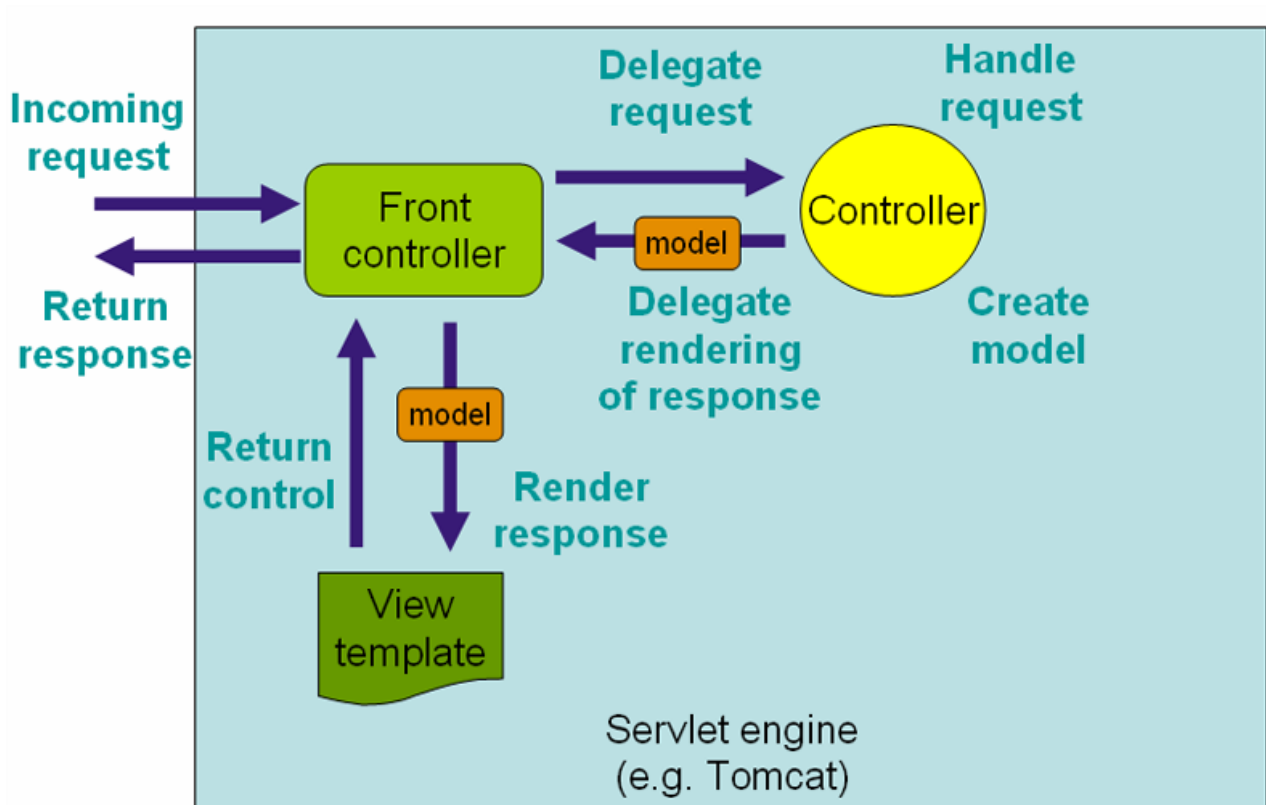


Figura 2.2 – Gestione HTTP request e response dal DispatcherServlet

Un altro schema che rappresenta la successione di azioni svolte all'arrivo di una HTTP request è quello di *Figura 2.2*. Quello che in figura è chiamato "Front controller" corrisponde al DispatcherServlet, mentre sono impliciti nello schema gli oggetti "Handler Mapping" e "View Resolver" che agiscono, però, rispettivamente tra il Front Controller e il Controller, e tra il Front Controller e il View Template.

2.2 Spring WebFlow

Un'altra caratteristica fondamentale, e che è stata ampiamente utilizzata per progettare e implementare l'applicazione web, è il “web flow”. Utilizzare un web flow all'interno di un'applicazione significa, in generale, creare una sequenza di passi che guidano l'utente all'interno di un processo [3]. L'utente si troverà di fronte ad una serie di interfacce in cui potrà effettuare delle scelte (scegliere dei prodotti, o, come in questo caso, compilare form) che influenzeranno la visualizzazione delle interfacce successive, mostrandone una piuttosto che un'altra.

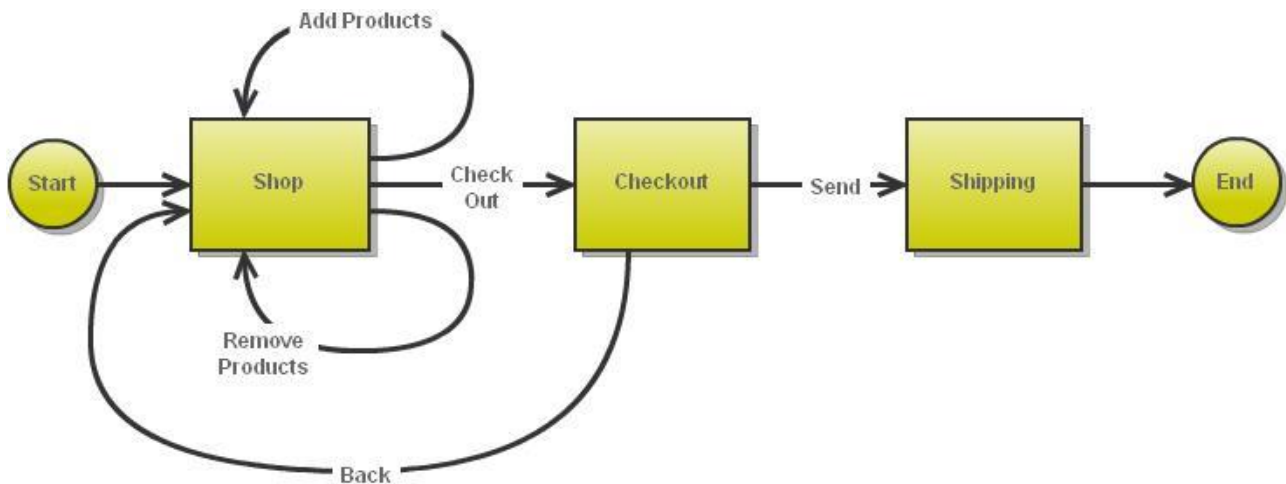


Figura 2.3 – Esempio Web Flow

In *Figura 2.3* è schematizzato un semplice e comune esempio di web flow di uno shop online. L'utente nello shop può aggiungere o rimuovere un certo numero di prodotti dalla stessa schermata. Quando l'utente decide di effettuare il “check out”, si passerà all'interfaccia successiva.

Un web flow ha la caratteristica di avere un determinato punto di inizio e un determinato punto di fine. Tra questi due punti, l'utente percorre una serie di interfacce in un ordine specifico.

Spring propone un metodo di implementazione del web flow tramite “Spring Web Flow” che estende Spring MVC. La configurazione del flow viene effettuata tramite un file XML in cui devono essere definiti tutti gli step, che prendono il nome di “state”, attraverso cui l'utente deve passare per concludere il flow.

Gli elementi fondamentali all'interno di Spring Web Flow sono:

- **View-state:** rappresenta uno step all'interno del flow che visualizza all'utente una certa interfaccia (esempio pagina JSP). Dopo la dichiarazione del view-state deve essere definito un campo “id” che indichi la view che si vuole visualizzare. In aggiunta, si può utilizzare la parola chiave “model” che permette di trattare oggetti del modello all'interno delle view.

- Transition: elemento usato per gestire eventi che accadono all'interno di un particolare "state". Transition definisce il comportamento futuro del flow: a seconda di quello che accade all'interno dello "state" decide quale sarà il prossimo "state".
- Action-state: rappresenta uno step all'interno del flow che, però, a differenza del view-state, non visualizza un'interfaccia all'utente. Permette di scegliere il prossimo "state" del flow invocando un'azione il cui risultato determinerà la transizione. [4]

2.3 Spring Dependency Injection

Come detto in precedenza, Spring è un framework a container leggero. Il ruolo principale del container è quello di applicare la dependency injection, chiamata anche principio di "Inversion of Control", che ha rivoluzionato il metodo di gestione delle dipendenze. In generale, la dependency injection è una tecnica che permette di risolvere le dipendenze automaticamente senza che sia il cliente o il programmatore a indicarle. In particolare, il programmatore non ha più l'obbligo di prendersi carico della risoluzione delle dipendenze perché un componente esterno se ne occupa al posto suo [5]. Quando per esempio si crea un oggetto che possiede istanze di altri oggetti, normalmente è il programmatore a istanziare questi oggetti all'interno del primo.

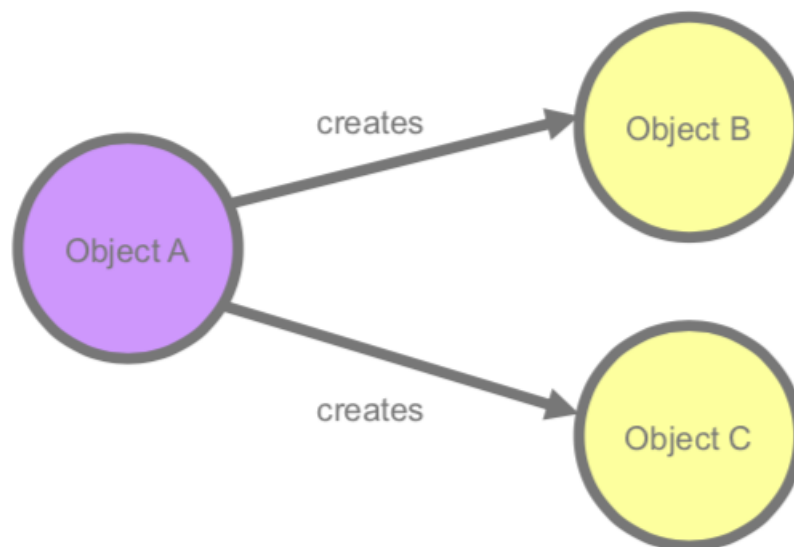


Figura 2.4 – Creazione di un oggetto con istanze di altri oggetti senza Dependency Injection [6]

Un semplice esempio è quello di *Figura 2.4*, in cui un oggetto A possiede un'istanza dell'oggetto B e una dell'oggetto C. Senza Dependency Injection, il programmatore deve creare, all'interno dell'oggetto A, gli oggetti B e C come istanze del primo.

Applicando, invece, il principio di Inversion of Control, la responsabilità di creare gli oggetti non è più del programmatore, bensì di un componente esterno che “inietta” le dipendenze all’interno dell’oggetto. [6]

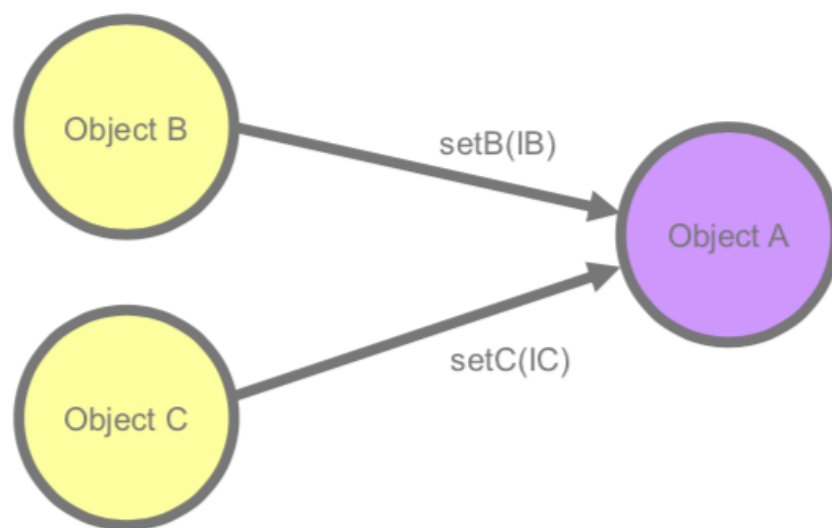


Figura 2.5 – Creazione di un oggetto con istanze di altri oggetti con Dependency Injection [6]

Lo stesso esempio di *Figura 2.4* viene rappresentato in *Figura 2.5* con l’utilizzo della Dependency Injection. Il programmatore si limita a settare i due oggetti all’interno dell’oggetto A tramite metodi `set()` (possibilità anche attraverso costruttore) e un componente esterno si occuperà di iniettare effettivamente gli oggetti quando necessario.

I vantaggi dell’utilizzo di questa tecnica sono diversi. In primo luogo, la possibilità, per il programmatore, di scrivere meno codice, che implica direttamente la riduzione della probabilità di errore. Inoltre, l’aumento del disaccoppiamento delle classi, seguendo il principio del “Single Responsibility” delle classi per cui ogni classe dovrebbe avere un solo compito. Applicando il principio di Inversion of Control, infatti, si riduce l’accoppiamento tra le classi poiché esse non dipendono più strettamente tra di loro. Un ulteriore vantaggio è la testabilità del codice, poiché non vi è più la necessità di dipendere da risorse esterne. Le classi diventano indipendenti e testabili singolarmente portandoci verso un altro beneficio, ovvero la manutenibilità. Non deve più essere modificato direttamente il codice, ma direttamente i file di configurazione che definiscono la Dependency Injection. [7]

Entrando nel dettaglio di Spring, il “componente esterno” di cui sopra è stato trattato e che si occupa della risoluzione delle dipendenze dei componenti è il container leggero, attraverso l’opportuna configurazione dell’implementazione dell’oggetto.

In Spring vi sono tre diverse varianti della Dependency Injection:

1. **Constructor-Based** dependency injection: il container chiama un costruttore con gli argomenti che si vogliono settare. Spring risolve la dipendenza di ogni argomento del costruttore prima per tipo, poi per nome e infine per id.
2. **Setter-Based** dependency injection: il container invoca i metodi setter della classe di cui si vogliono risolvere le dipendenze dopo aver istanziato l'oggetto (bean) tramite un costruttore vuoto.
3. **Field-Based** dependency injection: possono essere iniettate le dipendenze marcando con l'annotazione “@Autowired” gli oggetti o i metodi. [7]

2.4 Prolog e JPL

Come è già stato ampiamente trattato, il vero e proprio calcolo della corte competente viene effettuato da un componente esterno scritto in linguaggio Prolog. Prolog è un linguaggio di programmazione che adotta il paradigma di programmazione logica. Semplificando molto la trattazione, un programma Prolog è costituito da un certo numero di fatti, che dichiarano un certo stato di cose, da un insieme di regole, che, invece, definiscono le relazioni tra gli stati di cose, e infine da obiettivi (o domande) a cui il programma deve rispondere [8]. Il problema che si è creato è stato quello di interfacciare e far comunicare la parte di programma Prolog con l'applicazione che è stata progettata. Per farlo si è fatto uso di una libreria chiamata “JPL”, il cui scopo è proprio quello di permettere la comunicazione tra ambiente Java e programma scritto in Prolog. La libreria consente, quindi, alle applicazioni Prolog di sfruttare le classi, gli oggetti e i metodi creati in linguaggio Java, alle applicazioni Java di utilizzare i predicati e i componenti Prolog e, infine, di creare applicazioni ibride (che abbiano una componente Java e una Prolog) e di sfruttare al meglio i vantaggi di entrambe le parti [9]. In generale, un programma Prolog è sempre formato da un Database che contiene la lista di fatti e di regole. Per interrogare un Database è necessario prima caricarlo tramite il comando “consult” e poi utilizzare delle Query per effettuare le domande effettive.

I componenti principali che vengono utilizzati in ambiente Java della libreria JPL sono:

- Query: permette di interrogare il Database Prolog. È formata, in generale, da una stringa che dà il nome all'interrogazione e da un array di Term. Per ottenere la risposta dal programma si utilizza il comando “hasNext()” che, invocato sulla Query, itera su tutte le soluzioni restituendo ciascuna all'utente.
- Term: sono gli elementi che costruiscono l'interrogazione al Database Prolog.
- Atom: viene creato passando una stringa come parametro. Sono le parti che compongono un Term insieme a Variable.

- Variable: sono le parti che compongono un Term. A differenza degli Atom non hanno un valore preciso ma possono assumerne qualunque.
- Compound: è un particolare Term che contiene un nome e un array di Term. [10]

3 Progetto

In questo capitolo, si entrerà maggiormente nel dettaglio, evidenziando come nella pratica sono state progettate le soluzioni. Inoltre, verrà spiegata la scelta del pattern architetturale per progettare l'applicazione, così come la tecnologia di cui ci si servirà.

3.1 Progettazione Architetturale

Dall'analisi dei requisiti, è emerso che il cuore dell'applicazione è la trattazione di una nuova causa e il calcolo del corrispondente risultato. È stata quindi progettata l'intera architettura web che supporta l'applicazione e, in seguito, la funzionalità che permette di aggiungere e trattare una nuova causa

Pattern MVC

c

Figura 3.1 – Schema Pattern MVC

Per la progettazione e l'implementazione dell'applicazione è stato adottato il pattern architetturale MVC. La scelta è ricaduta su di esso perché permette un'ottima separazione tra la parte di presentazione dei dati all'utente e la logica di business che racchiude la logica applicativa che rende operativa l'applicazione. Il pattern architetturale divide il progetto in tre macro-aree:

1. **Model:** definisce metodi e operazioni per accedere e modificare i dati dell'applicazione
 2. **View:** ha il compito di visualizzare i dati del model e si occupa dell'interazione tra l'utente e il controller
 3. **Controller:** riceve comandi dall'utente attraverso la view e modifica i dati contenuti nel model o visualizzati all'interno delle view. Implementa la logica di business dell'applicazione
- [11]

3.2 Modello del Dominio

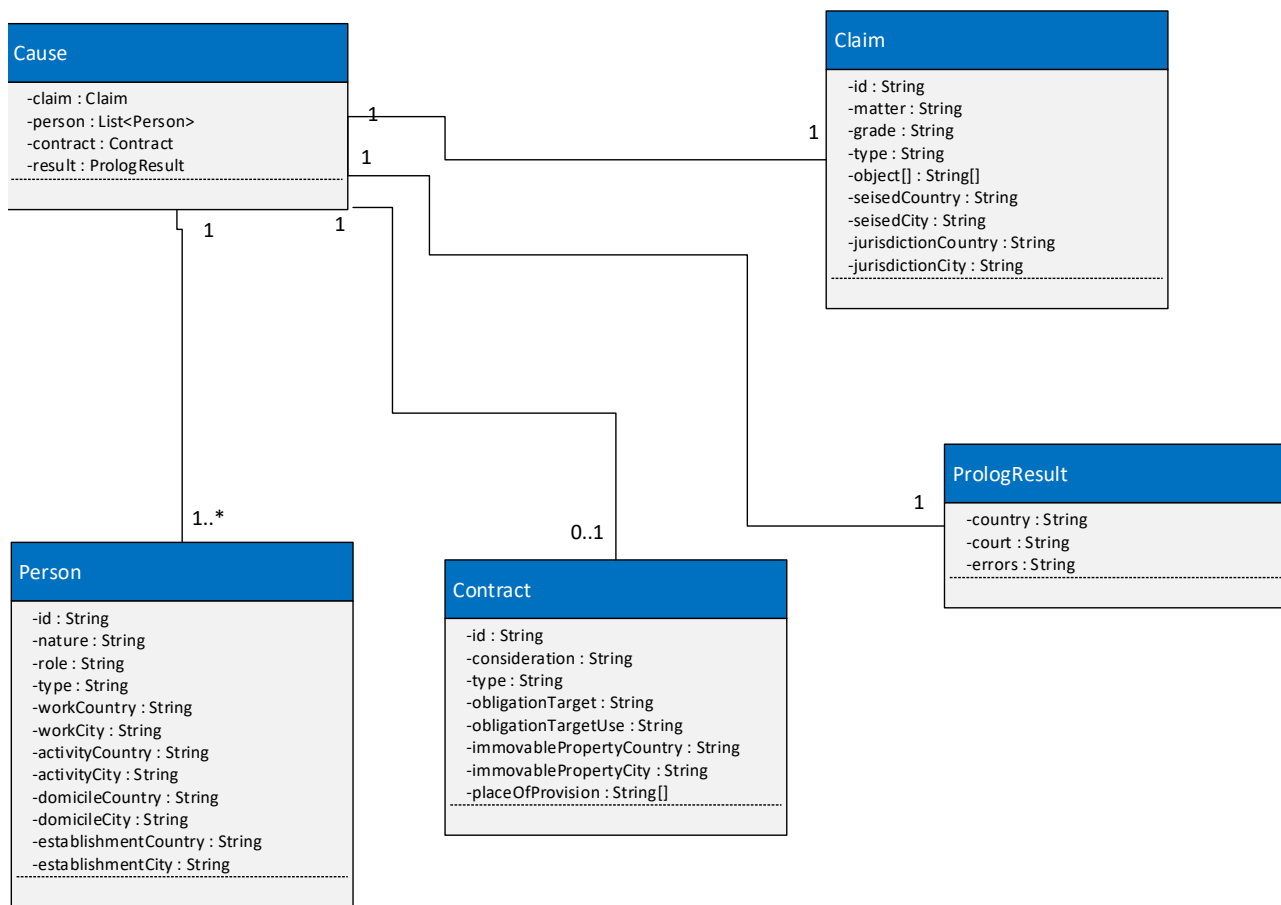


Figura 3.2 – Modello del Dominio

L'entità principale del modello è **Cause** che al suo interno avrà tre campi privati:

1. Claim
2. Person
3. Contract

Come ben definito durante l'analisi dei requisiti, ogni causa avrà uno e un solo claim e, allo stesso modo, un claim può essere associato ad una sola causa. Un discorso diverso, invece, deve essere fatto per l'attributo **Person** che ha una relazione 1 a N con Cause. Infatti, una singola causa può essere associata ad una o più persone, per questo al suo interno troviamo come campo un array di Person. Infine, la classe **Contract** sarà associata ad una sola causa, mentre una causa può avere un contract o non averne proprio, come indicato dalla cardinalità dell'associazione. Inoltre, ogni classe (Claim, Contract, Person) ha come campi privati i parametri che sono stati concordati in sede di riunione durante l'analisi dei requisiti.

È stato scelto di definire il campo "id" come stringa in modo da permettere all'utente di associare un nome facile da usare e da ricordare all'entità che sta creando. Questa scelta risulta efficace soprattutto per la creazione di una nuova persona all'interno della causa per ovvi motivi.

All'interno di Claim, "object" è stato definito come array per mantenere la coerenza con l'analisi dei requisiti, che richiedevano che ognuna di queste entità potesse avere un numero indefinito di object. Un discorso analogo può essere effettuato per il campo "place of provision" della classe Contract, anch'esso può essere di numero indefinito.

Inoltre, è stata inserita una classe ausiliaria, "PrologResult" che viene utilizzata per il salvataggio del risultato del programma Prolog. Una volta effettuato il calcolo, infatti, i campi "Country" e "Court" verranno aggiornati con i valori che sono stati calcolati secondo i parametri inseriti. Se l'operazione è andata a buon fine e non si sono incontrati errori durante il calcolo, all'interno dei primi due campi compariranno i risultati con lo stato in cui deve essere risolta la Causa e la corte di competenza. Nel caso in cui, invece, si siano incontrati errori, verrà aggiornato soltanto il campo "errors" con un messaggio di errore che verrà visualizzato all'utente. Gli errori possono essere di vario tipo, il più comune può essere una combinazione di input che non è ancora stata inserita all'interno dell'applicazione, e che quindi il sistema stesso non supporta. Questo tipo di errore genera un'eccezione che deve essere opportunamente gestita in fase di implementazione.

3.3 Progetto di Dettaglio

Diagramma dei package

Di seguito viene proposto lo schema del diagramma dei package aggiornato dopo la scelta del pattern architetturale e della tecnica di Spring WebFlow

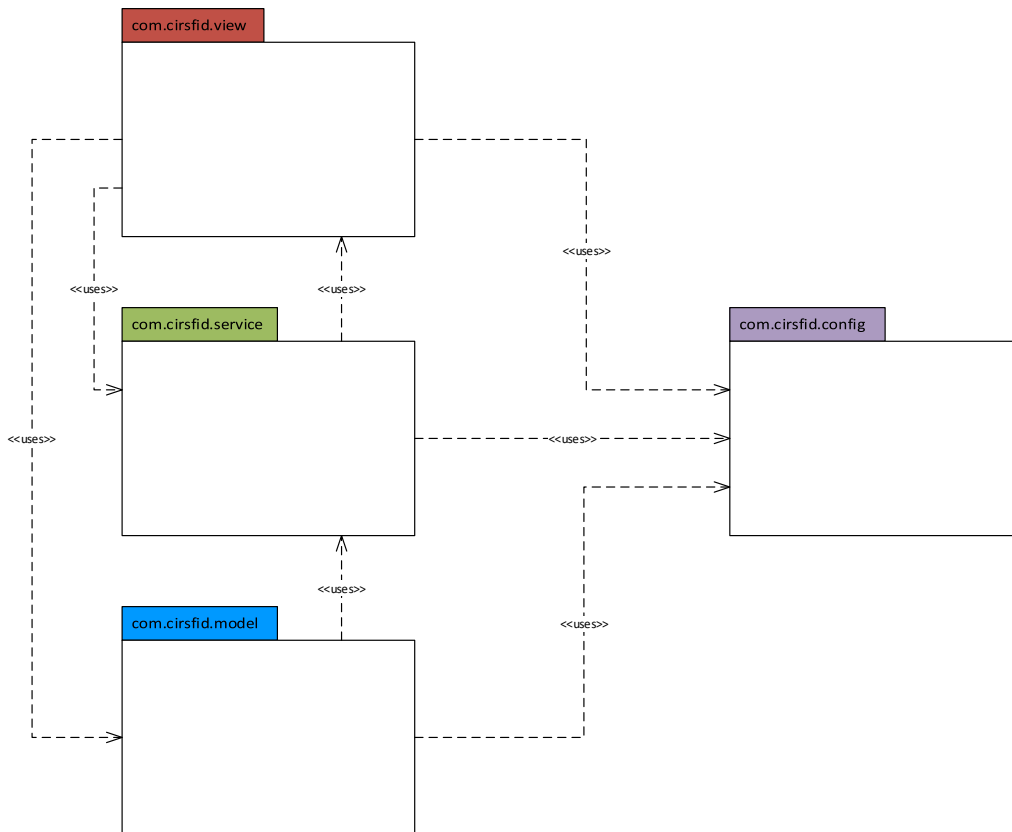


Figura 3.3 – Diagramma dei package

La struttura dei package è basata sul pattern architetturale che è stato scelto di utilizzare, ovvero il pattern MVC. Più precisamente, viene introdotto un package in cui sono raccolte le classi di configurazione del sistema “com.cirsfid.config”, in modo da utilizzare Spring MVC. Nel package, è infatti presente una classe “WebMvcConfig” che contiene metodi per la configurazione dell’applicazione. Sono presenti per esempio “ViewResolver” e “Handler Mapping” che permettono rispettivamente di ricavare le view e di mappare le richieste.

Un’altra classe rilevante nel package di configurazione è “WebInitializer” che estende la classe astratta “AbstractAnnotationConfigDispatcherServletInitializer” e permette di configurare il DispatcherServlet di cui sopra è stato parlato.

Il package riservato alle view contiene tutte le pagine che vengono visualizzate all’utente. Dipende dal package di configurazione per motivi già trattati.

“com.cirsfid.service” è il package che racchiude le classi che implementano la parte di business logic dell’applicazione. All’interno si trovano i due Controller principali del sistema:

- ParameterController: che permette di acquisire e registrare tutti gli input inseriti dall’utente all’interno degli oggetti
- PrologController: che interfaccia l’applicazione con il programma Prolog passando i parametri e ottenendo il risultato

Infine, il package “com.cirsfid.model” unisce le classi del modello e che sono state rappresentate in *Figura 3.2* e ampiamente trattate.

Diagramma delle Classi

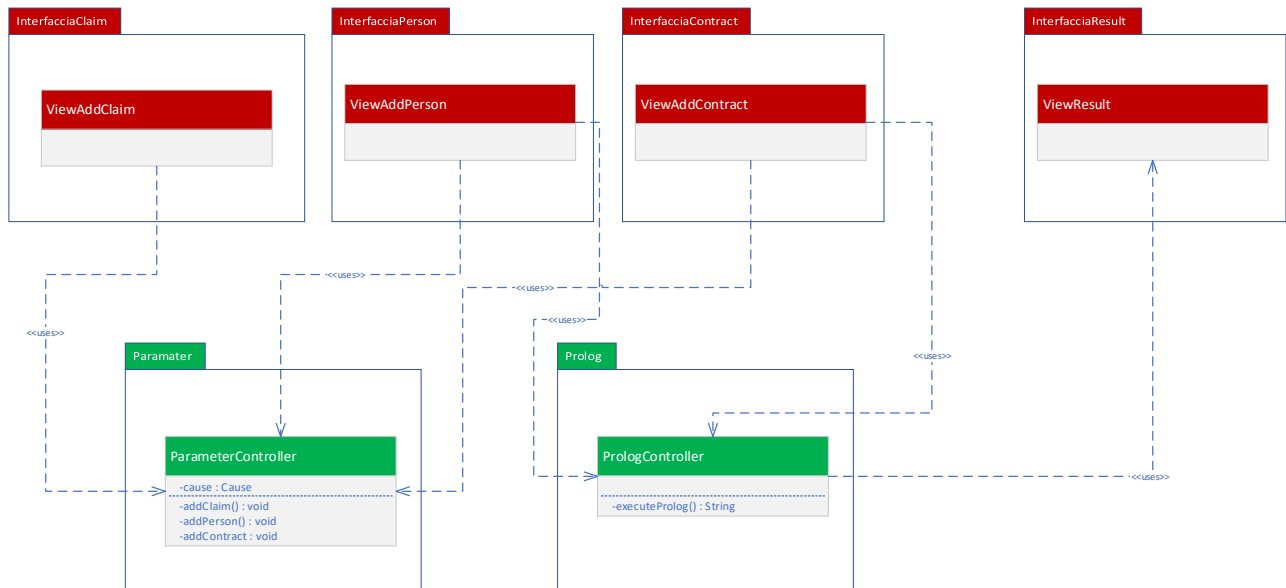


Figura 3.3 – Diagramma delle Classi

Entrando nel dettaglio, **ParameterController** conterrà tre metodi corrispondenti alle tre interfacce. Ogni interfaccia visualizzerà all’utente il form da completare e il controller permetterà di raccogliere i dati inseriti e associarli all’oggetto contenuto come campo.

È importante notare come avviene il passaggio all’utilizzo del programma Prolog. Infatti, come definito nell’analisi dei requisiti e modellato nel dominio, una causa può non avere contratti. Perciò, potrà essere passato il comando al PrologController dalla “ViewAddPerson”, nel caso non ci sia nessun contratto, oppure dalla “ViewAddContract”, nel caso in cui, dopo aver aggiunto una persona, l’utente desideri aggiungere anche un contratto.

Infine, è molto importante il verso della freccia tra PrologController e ViewResult. Sarà infatti il controller che invocherà la view dopo aver eseguito l’algoritmo e, con essa, visualizzerà il risultato. La classe **PrologController** si limiterà ad effettuare il calcolo vero e proprio cuore dell’applicazione, invocando tramite il metodo “executeProlog” il programma Prolog sottostante e visualizzando tramite la view il risultato finale della giurisdizione competente.

3.4 Architettura Logica: Interazione

Nel seguito vengono riportati i due esempi di aggiunta di una nuova causa e esecuzione dell'algoritmo:

1. Aggiunta causa senza Contratto
2. Aggiunta causa con Contratto

Aggiunta Causa senza Contratto

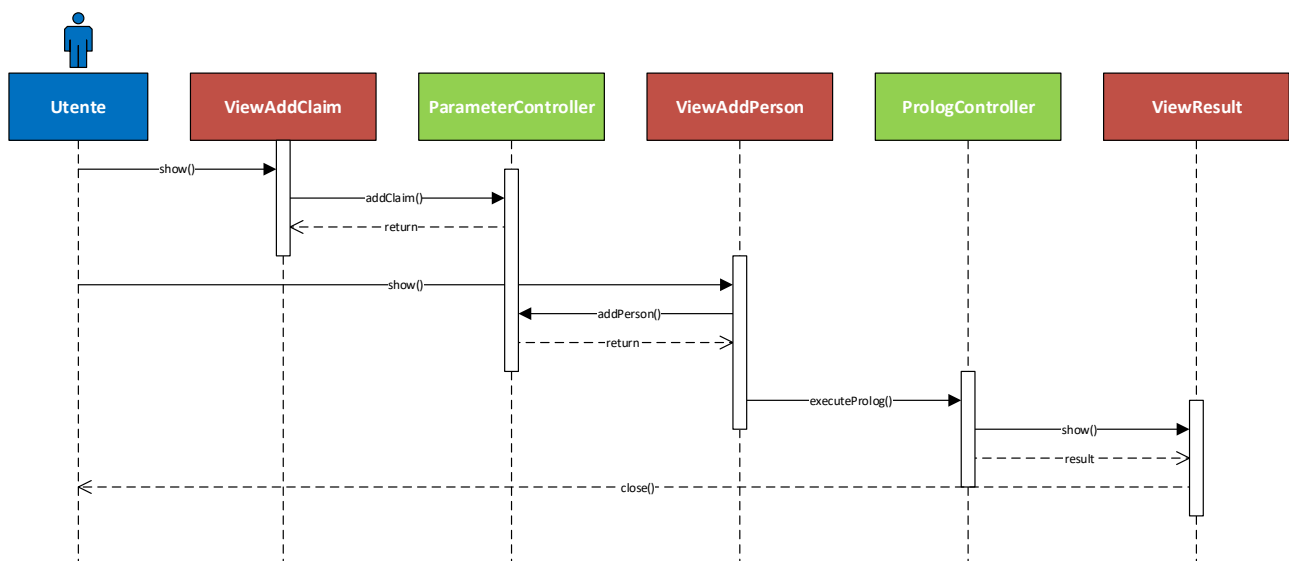


Figura 3.4 – Diagramma delle Sequenze senza Contratto

Aggiunta Causa con Contratto

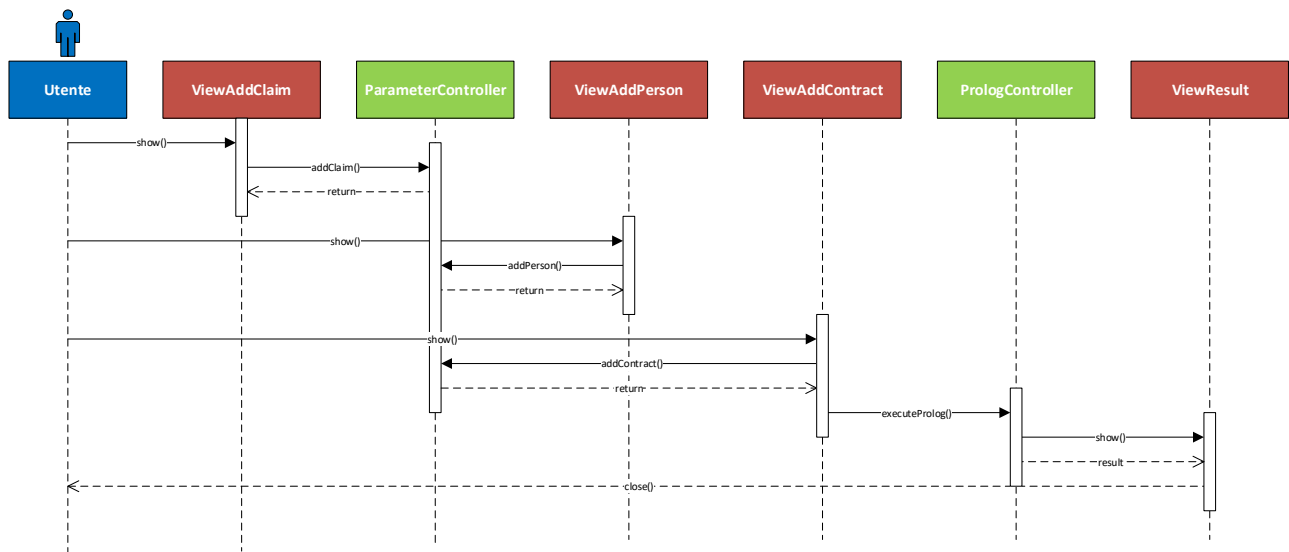


Figura 3.5 – Diagramma delle Sequenze completa con Contratto

com.cirsfid.service



Figura 3.6 – package service

In *Figura 3.6* sono rappresentate le due classi contenute nel package `com.cirsfid.service`. `ParameterController` possiede tre metodi per l'aggiunta di parametri all'istanza di `Cause`, e un quarto metodo per controllare se all'interno della causa si sta inserendo un contratto o no.

L'altro Controller invece ha un unico metodo che calcola il risultato finale passando i parametri al programma `Prolog` e ricevendo il risultato.

4. Implementazione

In questo capitolo verrà trattata l'applicazione vera e propria, non più a livello di progettazione, ma di implementazione.

4.1 Limiti e sviluppi futuri

Per questioni di tempo non è stato possibile implementare l'intera applicazione web come designata nell'analisi dei requisiti. La caratteristica principale dell'applicazione resta l'aggiunta di una nuova causa e il calcolo dello stato e della corte competente a giudicarla, che è stata quasi completamente implementata. Le differenze, rispetto all'accordo con i committenti, riguardano alcuni campi che non sono stati realizzati come richiesto. Infatti, i campi "ClaimObject" di un Claim e "PlaceOfProvision" di un Contract erano stati progettati in modo che potessero essere presenti in numero indefinito, ovvero l'utente potesse inserire da 0 a N ClaimObject e da 0 a N PlaceOfProvision. Nell'implementazione, però, si è preferito, per facilità e mancanza di tempo, permettere all'utente di inserire al massimo due ClaimObject e due PlaceOfProvision.

Per quanto riguarda, invece, il campo "Jurisdiction for civil proceedings" di un Claim e i campi "Consideration" e "Type" di un Contract, si è preferito dare la possibilità all'utente di inserire un solo campo per ognuno, anche se erano stati, anch'essi, richiesti in numero indefinito.

Inoltre, verrà sviluppata in futuro la possibilità di aggiungere un foglio Excel per ottenere le domande e le possibili risposte dell'applicazione. Infatti, questo richiederebbe un cambiamento significativo nella progettazione dell'applicazione che non avrebbe più campi definiti per ogni oggetto.

Per alcuni campi, per esempio per il domicilio di una persona, è stato richiesto dai committenti di dare la possibilità agli utenti di inserire lo stato e, in seguito, la città collegata. Ovviamente, il metodo migliore è quello di aggiornare la scelta delle possibili città in relazione allo stato scelto dall'utente. Non avrebbe infatti senso che all'utente, dopo aver scelto un determinato stato, per esempio Italia, venissero visualizzate tutte le possibili città, tra cui, rimanendo nell'esempio, quelle francesi, ma, piuttosto, soltanto le città italiane. Per implementarlo, l'idea migliore è quella di effettuare una chiamata Ajax asincrona ad una servlet che, comunicando con un Database, restituisce l'elenco corretto delle città. Questo metodo richiede, però, l'utilizzo di un Database che, in questa prima versione dell'applicazione, non è ancora stato costruito. La soluzione che è stata utilizzata è quella di aggiornare la scelta delle città tramite metodo Javascript. Javascript permette di aggiornare in modo dinamico una pagina Web, quindi, quello che è stato fatto è di collegare ogni campo che richiedesse l'aggiunta, da parte di un utente, di uno stato, ad un metodo "onChange" Javascript che aggiornasse

il campo relativo alle possibili città. In questo modo, alla scelta dell'utente di uno stato di domicilio come l'Italia, il metodo Javascript si preoccupa di aggiornare le possibili città di domicilio, con solo quelle italiane.

Come già detto in precedenza, non è ancora stato creato un Database relativo all'applicazione Web. Questo rende impossibile, almeno per il momento, l'aggiunta della sezione "My Causes" che prevede il salvataggio delle cause precedentemente inserite e allo stesso tempo anche l'autenticazione, poiché le credenziali devono essere salvate su un relativo Database.

4.2 Interfacce Grafiche

La parte di interfaccia grafica dell'applicazione è implementata attraverso pagine JSP all'interno del package "com.cirsfid.view". Le pagine sono state scritte per la maggior parte in HTML, linguaggio a marcatori, con l'aggiunta di codice Javascript, come sopra trattato. Inoltre, sono presenti alcune parti di codice Java, per eseguire semplici operazioni e di Expression Language per accedere ai bean istanziati da Spring. Infatti, per utilizzare gli elementi del model in un WebFlow, essi devono essere definiti come Spring Bean. Per rendere questi elementi disponibili nelle pagine JSP, si ricorre all'uso di EL, expression language, che è un linguaggio molto conciso utile per eseguire operazioni semplici e di visualizzazione di dati.

Un ultimo accenno è sulla grafica vera e propria dell'applicazione web. È stato utilizzato largamente il codice CSS per modificare posizione, colori e grandezza di tutti gli elementi della pagina. Inoltre, sono state realizzate alcune parti della grafica grazie all'aiuto di Bootstrap. Bootstrap è una raccolta di stili, colori, ma soprattutto elementi grafici da poter utilizzare, costruiti proprio per le applicazioni Web [12]. Essendo compatibili con quasi tutti i browser, esso risulta molto utile perché propone temi e componenti già creati e che devono solo essere aggiunti alla propria pagina Web.

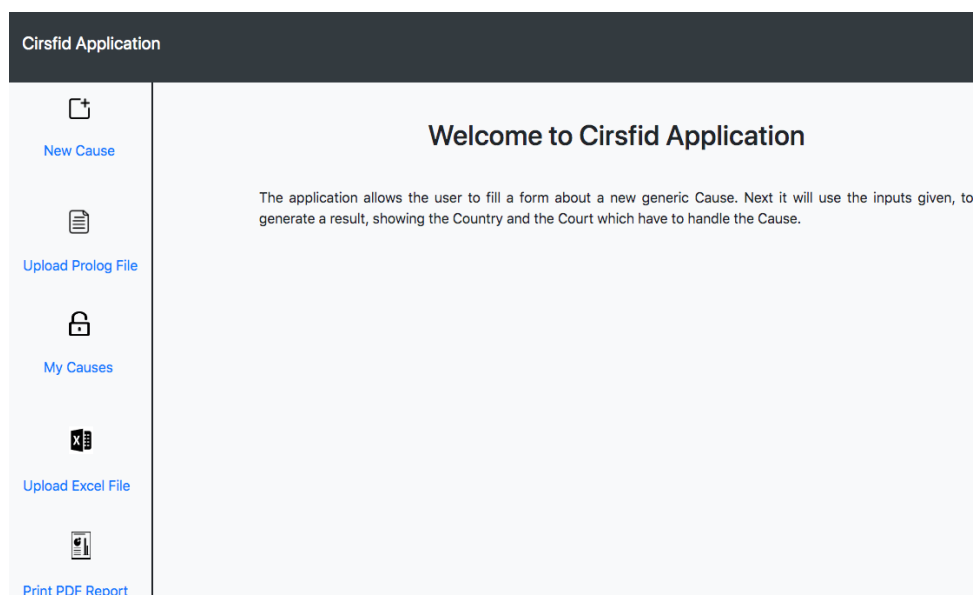


Figura 4.1 – Welcome Page applicazione

In *Figura 4.1* è rappresentata la Home Page dell'applicazione. Al centro vi è una semplice descrizione dello scopo dell'applicazione, mentre sulla sinistra vi è una barra di navigazione contenente i link a tutte le funzionalità. Ovviamente, in questa prima versione, solo il primo link è funzionante, mentre gli altri, in caso di click, mostrano un messaggio che spiega all'utente che la pagina è ancora in costruzione.

Figura 4.2 – Schermata di aggiunta di un Claim

In *Figura 4.2* è mostrata la pagina di aggiunta di una nuova Claim. Per ogni domanda in linguaggio naturale, vi è una lista di possibili risposte implementata tramite il tag `<select>` di HTML.

Cirsfid Application

*Choose Claim Object:

In which state has the proceeding been seised?

In which court of the state has the proceeding been seised?

Which state has Jurisdiction for civil proceedings?

Which court of the previously selected state has Jurisdiction for civil proceedings?

Confirm


Add a second Claim Object

-- select an option --

-- select an option --
Contract
Tort
Ownership
Rights in Rem
Liability
Trust
Criminal Proceeding
Damages Restitution
No more than 6 months

-- select an option --

Sempre nella schermata di aggiunta di un Claim. In *Figura 4.3*, è presente il pulsante di conferma “confirm” e un pulsante per permettere all’utente di aggiungere un secondo campo “ClaimObject”.



Home Page Person


List of people already added to the Cause

#	ID	Domicile Country	Domicile City	Nature	Role	Type	Work Country	Work City	Activity Country	Activity City	Establishment Country	Establishment City
<div style="display: flex; justify-content: space-around; margin-top: 10px;"> Add new Person Finish Go Back </div>												

Figura 4.4 – Schermata HomePage Person

Una volta confermata l'aggiunta di un Claim, l'utente passa alla schermata di riepilogo delle persone aggiunte fino a quel momento. Come sappiamo, per una sola causa, possono essere presenti un numero indefinito di persone. Per questo l'applicazione è stata progettata per avere una home page in cui sono presenti tutte le persone aggiunte e i dati inseriti per ognuna di esse. Quando l'utente desidera aggiungere una nuova persona, premerà sul pulsante apposito e verrà spostato alla schermata di aggiunta di una nuova persona.

Cirsfid Application



Add New Person

Please complete the form about the new Person

Insert person ID (blank for default):

Silvia

Choose the nature of the Person:

-- select an option --

*What is the Role of the Person in the Cause?

Defendant

What is the type of the Person?

-- select an option --

Which state for the Work?

-- select an option --

Figura 4.5 – Aggiunta di una nuova persona

L'utente in questa schermata ha la possibilità di aggiungere i dati che preferisci di una persona, partendo dall'ID.

Cirsfid Application

Which state for the Work?

-- select an option --

Select the city work:

-- select an option --

Which state for the Activity?

-- select an option --

*Select the domicile of the person:

Italy

*Domicile city:

-- select an option --

✓ Bologna

Rome

Milan

Florence

Where is the Establishment?

-- select an option --

Establishment city:

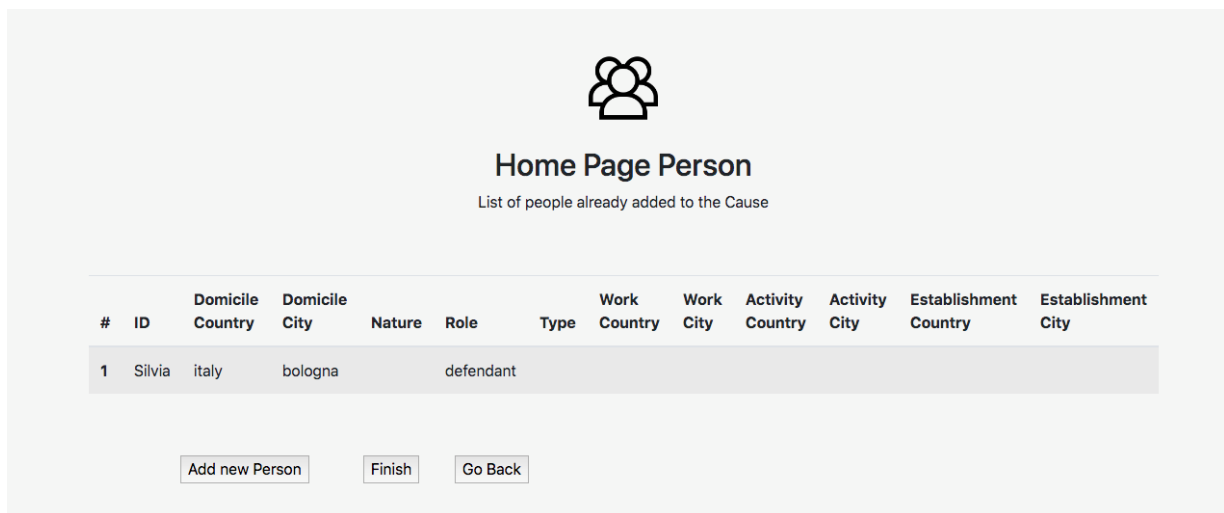
-- select an option --

Confirm

Figura 4.6 – Aggiunta di una nuova persona

In *Figura 4.6* si nota che le città, come già detto in precedenza, cambiano a seconda dello stato relativo che è stato inserito.

Una volta completato il form di aggiunta di una persona, l'utente finalizzerà la sua scelta attraverso il pulsante "Confirm" alla fine delle domande e verrà riportato alla schermata di riepilogo delle persone.



The screenshot shows a web interface titled "Home Page Person" with the subtitle "List of people already added to the Cause". At the top, there is an icon of three people. Below the title, there is a table with 14 columns: #, ID, Domicile Country, Domicile City, Nature, Role, Type, Work Country, Work City, Activity Country, Activity City, Establishment Country, and Establishment City. The table contains one row with the following data: # 1, ID Silvia, Domicile Country italy, Domicile City bologna, Nature, Role defendant, Type, Work Country, Work City, Activity Country, Activity City, Establishment Country, and Establishment City. At the bottom of the interface, there are three buttons: "Add new Person", "Finish", and "Go Back".


#	ID	Domicile Country	Domicile City	Nature	Role	Type	Work Country	Work City	Activity Country	Activity City	Establishment Country	Establishment City
1	Silvia	italy	bologna		defendant							

Buttons: Add new Person, Finish, Go Back

Figura 4.7 – Aggiunta persona con domicilio e ruolo

A questo punto, vi possono essere due strade diverse. Se l'utente, durante l'aggiunta di un Claim, aveva selezionato come "ClaimObject" un "Contract", una volta premuto il pulsante "Finish" di questa schermata, verrà visualizzato il form di aggiunta di un nuovo contratto. Nel caso invece il campo "ClaimObject" non avesse come valore "Contract", alla conclusione di aggiunta di persone, verrà subito calcolato e visualizzato il risultato, passando all'ultima schermata dell'applicazione.

Cirsfid Application



Add New Contract

Please complete the form about the new Contract

Insert Contract ID (blank for default):

What is the consideration of the contract? -- select an option --

What is the Obligation Target? -- select an option --

What is the type of contract? -- select an option --

Figura 4.8 – Aggiunta di un contratto

Nel caso in cui si debba inserire un contratto all'interno della causa, in *Figura 4.8* è visualizzato il form da completare. In fondo è presente un pulsante che permette di mostrare un campo per aggiungere un secondo “PlaceOfProvision” e un altro pulsante per procedere alla prossima schermata.

Cirsfid Application

Result

That is the result of your question:

Country: italy

Court: milan

[Torna alla home page](#)

Figura 4.9 – Risultato Finale

Per concludere, è stato inserito un esempio di una causa all'interno del sistema, che ha portato come risultato quello mostrato in *Figura 4.9*. La schermata, molto semplice, mostra lo stato e la corte competente a prendersi carico della causa. Inoltre, è presente un link per riportare l'utente alla HomePage e permettergli di inserire una nuova causa.

4.3 Configurazione dell'applicazione

L'intera struttura dell'applicazione è racchiusa all'interno del package “com.cirsfid.config”. Le due tecnologie principali che sono state utilizzate sono Spring WebFlow e Spring MVC, entrambe realizzate all'interno di questo package. Con la nascita del framework Spring, ogni configurazione doveva essere realizzata tramite file XML. Con gli ultimi aggiornamenti, invece, si è andati verso una “Java Based Configuration” che sfrutta le annotazioni per indicare al framework Spring quali sono gli elementi di configurazione. All'interno dell'applicazione sono state create quattro classi per definire la struttura.

La prima classe realizzata è stata il “WebInitializer”.

```
public class WebInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {  
    public WebInitializer() {  
        super();  
    }  
    // API  
    protected Class<?>[] getRootConfigClasses() {  
        return new Class<?>[] { WebMvcConfig.class, WebFlowConfig.class };  
    }  
    @Override  
    protected Class<?>[] getServletConfigClasses() {  
        return null;  
    }  
    @Override  
    protected String[] getServletMappings() {  
        return new String[] { "/" };  
    }  
    @Override  
    protected void customizeRegistration(final Dynamic registration) {  
        super.customizeRegistration(registration);  
    }  
}
```

Figura 4.9 – WebInitializer

Come si vede dalla *Figura 4.9*, la classe è in realtà molto semplice. Estende una classe astratta già creata all'interno del framework Spring e esegue l'override di alcuni metodi. Con il metodo “getRootConfigClasses()” viene indicato all'applicazione quali sono le classi dedicate alla configurazione. Con il metodo “getServletMappings()”, invece, vengono, in generale, indicate le servlet presenti all'interno del sistema. In questo caso, il mapping avviene all'URL “/”, questo vuol dire che ogni richiesta che viene effettuata all'applicazione, deve essere risolta dalla stessa servlet, che nell'applicazione è il “DispatcherServlet” di cui è stato parlato nel capitolo 2 [14].

```

@EnableWebMvc
@Configuration
@ComponentScan({"com.cirsfid.config", "com.cirsfid.model"})
public class WebMvcConfig implements WebMvcConfigurer {

    @Autowired
    private WebFlowConfig webFlowConfig;

    public void addResourceHandlers(ResourceHandlerRegistry registry) {
        registry.addResourceHandler("/resources/**").addResourceLocations("/resources/");
    }

    @Bean
    public ViewResolver viewResolver() {
        InternalResourceViewResolver viewResolver = new InternalResourceViewResolver();
        viewResolver.setPrefix("/WEB-INF/view/");
        viewResolver.setSuffix(".jsp");
        return viewResolver;
    }

    @Bean
    public FlowHandlerMapping flowHandlerMapping() {
        FlowHandlerMapping handlerMapping = new FlowHandlerMapping();
        handlerMapping.setOrder(-1);
        handlerMapping.setFlowRegistry(this.webFlowConfig.flowRegistry());
        return handlerMapping;
    }

    @Bean
    public FlowHandlerAdapter flowHandlerAdapter() {
        FlowHandlerAdapter handlerAdapter = new FlowHandlerAdapter();
        handlerAdapter.setFlowExecutor(this.webFlowConfig.flowExecutor());
        handlerAdapter.setSaveOutputToFlashScopeOnRedirect(true);
        return handlerAdapter;
    }
}

```

Figura 4.10 – WebMvcConfig

Un'altra classe molto importante è “WebMvcConfig”, di cui in *Figura 4.10* si può osservare l'implementazione. All'interno di questa classe vengono realizzati gli oggetti che regolano la risoluzione di una HTTP request, come il “ViewResolver” e “HandlerMappings”. Il primo serve a definire come ottenere le view. In questo modo, all'interno dell'applicazione, è sufficiente, per invocare la view, inserire il nome di essa senza preoccuparsi di prefisso e suffisso che vengono invece aggiunti dal “ViewResolver”. Il secondo oggetto viene qui implementato come un “FlowHandlerMapping”, poiché il sistema è organizzato come un WebFlow.

È importante anche notare le tre annotazioni che sono state aggiunte prima della classe. Esse definiscono il ruolo del codice che è stato scritto. La prima annotazione, “@EnableWebMvc”, è usata semplicemente per attivare la configurazione di Spring MVC. La seconda, “@Configuration”, serve ad indicare al container di Spring che la classe che è stata creata definisce dei Bean. Questo permette di collegarsi all'annotazione presente prima dell'implementazione di ogni metodo, ovvero “@Bean”, che indica che ogni operazione avrà come risultato un oggetto che deve essere registrato come Bean all'interno del contesto dell'applicazione Spring.

Infine, l'annotazione “@ComponentScan”, definisce quali sono i package in cui si troveranno altri componenti [15].

```

@Configuration
public class BeanDefinition {

    @Bean
    public Claim claim() {
        return new Claim();
    }

    @Bean
    public Contract contract() {
        return new Contract();
    }

    @Bean
    public Person person() {
        return new Person();
    }

    @Bean
    public ArrayList<Person> listPerson(){
        return new ArrayList<Person>();
    }

    @Bean
    public Cause cause(Claim c, ArrayList<Person> list, Contract cont, PrologResult result) {

        Cause cause = new Cause();
        cause.setClaim(c);
        cause.setPersons(list);
        cause.setContract(cont);
        cause.setPrologResult(result);
        return cause;
    }

    @Bean
    public PrologResult result() {
        return new PrologResult();
    }
}

```

Figura 4.11 – BeanDefinition

All'interno della classe “BeanDefinition”, sono stati costruiti come Bean i vari oggetti del dominio che saranno utilizzati dall'applicazione. Anche questa classe è marcata come “Configuration” poiché ogni metodo ha come risultato un Bean.

```

@Configuration
public class WebFlowConfig extends AbstractFlowConfiguration {

    @Autowired
    private WebMvcConfig webMvcConfig;

    @Bean
    public FlowDefinitionRegistry flowRegistry() {
        return getFlowDefinitionRegistryBuilder(flowBuilderServices()).addFlowLocation("/WEB-INF/flows/activation-flow.xml", "activationFlow").build();
    }

    @Bean
    public FlowExecutor flowExecutor() {
        return getFlowExecutorBuilder(flowRegistry()).build();
    }

    @Bean
    public FlowBuilderServices flowBuilderServices() {
        return getFlowBuilderServicesBuilder().setViewFactoryCreator(mvcViewFactoryCreator()).setDevelopmentNode(true).build();
    }

    @Bean
    public MvcViewFactoryCreator mvcViewFactoryCreator() {
        MvcViewFactoryCreator factoryCreator = new MvcViewFactoryCreator();
        factoryCreator.setViewResolvers(Collections.singletonList(this.webMvcConfig.viewResolver()));
        factoryCreator.setUseSpringBeanBinding(true);
        return factoryCreator;
    }
}

```

Figura 4.12 – WebFlowConfig

L'ultima classe di configurazione è “WebFlowConfig”. Il suo ruolo è di particolare importanza, poiché indica a Spring dove si trova il foglio XML in cui è descritto il percorso del flow. Oltre a

indicare il percorso per raggiungere il file, il metodo “FlowDefinitionRegistry” definisce un nome con cui il flow può essere “chiamato” all’interno dell’applicazione.



```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <flow xmlns="http://www.springframework.org/schema/webflow"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xsi:schemaLocation="http://www.springframework.org/schema/webflow
5                           http://www.springframework.org/schema/webflow/spring-webflow.xsd">
6
7     <var name="claim" class="com.cirsfid.model.Claim" />
8     <var name="cause" class="com.cirsfid.model.Cause" />
9     <var name="person" class="com.cirsfid.model.Person" />
10    <var name="contract" class="com.cirsfid.model.Contract" />
11    <var name="result" class="com.cirsfid.model.PrologResult" />
12
13
14
15    <view-state id="addClaim" model="cause">
16      <transition on="confirm" to="addClaimParameter"/>
17    </view-state>
18
19    <action-state id="addClaimParameter">
20      <evaluate expression="sampleAction"/>
21      <transition on="success" to="homePagePerson"/>
22    </action-state>
23
24
25    <view-state id="homePagePerson" model="cause">
26      <transition on="add" to="addPerson"/>
27      <transition on="finish" to="checkContract"/>
28    </view-state>
29
30    <view-state id="addPerson" model="cause">
31      <transition on="confirm" to="addPersonToIst"/>
32    </view-state>
33
34    <action-state id="addPersonToIst">
35      <evaluate expression="sampleAction"/>
36      <transition on="success" to="homePagePerson"/>
37    </action-state>
38
39    <action-state id="checkContract">
40      <evaluate expression="sampleAction"/>
41      <transition on="yes" to="addContract"/>
42      <transition on="no" to="evaluateResult"/>
43    </action-state>
44
45    <view-state id="addContract" model="cause">
46      <transition on="confirm" to="addContractParameter"/>
47      <transition on="back" to="homePagePerson"/>
48    </view-state>
49
50    <action-state id="addContractParameter">
51      <evaluate expression="sampleAction"/>
52      <transition on="success" to="evaluateResult"/>
53    </action-state>
54
55
56    <action-state id="evaluateResult">
57      <evaluate expression="prologAction"/>
58      <transition on="success" to="resultProlog"/>
59    </action-state>
60
61    <view-state id="resultProlog" model="cause"/>
62
63  </flow>
```

Figura 4.13 – activation-flow.xml

Per concludere questa trattazione, in Figura 4.13, è mostrato il flow vero e proprio che l’utente seguirà all’interno dell’applicazione. Tramite il tag “var” vengono creati gli elementi del dominio che saranno passati alle view attraverso la parola chiave “model”. In seguito, sono presenti dei “view-state” che definiscono le view che l’utente visualizzerà e su cui potrà prendere delle decisioni e inserire dei parametri. A seconda dei parametri inseriti o dei tasti premuti, l’utente attraverserà i vari step del flusso fino ad arrivare all’ultimo stato in cui visualizzerà il risultato.

Gli “action-state” sono stati collegati ai controller che eseguiranno operazioni per determinare il prossimo step del flow [16].

Conclusioni

La trattazione che è stata effettuata mostra la progettazione e lo sviluppo completo di un'applicazione web. Partendo da un problema proposto da alcuni committenti, è stato svolto un intero percorso che ha portato alla realizzazione della completa applicazione. Ad una riunione con i committenti, che hanno esposto il problema in modo chiaro, è seguita un'attenta analisi dei requisiti che ha portato a stilare un documento di analisi che sposasse perfettamente i loro obiettivi. Una volta concordato che l'analisi dei requisiti rispecchiasse correttamente le volontà dei committenti, è stato effettuato un attento e completo studio delle tecnologie che sarebbero state utilizzate all'interno dell'applicazione. In primo luogo, è avvenuta un'analisi del linguaggio Prolog, in modo da aver ben chiaro il programma esterno fornito dai committenti e di potersi interfacciare con esso senza problemi. In seguito, è stato capito a fondo il framework Spring che realizza l'architettura dell'applicazione. Lo studio di Spring è stato diviso in una prima parte in cui sono stati affrontati i concetti generali di Spring MVC e di Dependency Injection, e in una seconda parte di Spring WebFlow. Infine, è stato affrontato lo studio della libreria JPL che permette la comunicazione tra ambiente Java e linguaggio Prolog.

Alla fase di studio, ne è seguita una di progettazione del sistema. È stato definito il pattern architetturale dell'applicazione web e il dominio dell'applicazione con cui sono state create le classi base. In seguito, sono stati progettati i package che compongono il sistema seguendo il modello di Spring MVC e, più nel dettaglio, le classi che ne faranno parte con la dichiarazione dei metodi principali.

Infine, si è passati alla parte implementativa in cui dal progetto si effettua lo sviluppo vero e proprio dell'applicazione. Essa è stata implementata e configurata su una macchina virtuale che funziona come server. Sono state mostrate alcune delle principali interfacce grafiche all'interno dell'ultimo capitolo e sono stati descritti i limiti che l'applicazione possiede.

Attualmente, l'applicazione web è funzionante e può essere raggiunta via web mediante un indirizzo IP associato. Il collegamento è stato fornito ai committenti che sono in una fase di "testing" dell'applicazione, in modo da individuare possibili bug, malfunzionamenti o requisiti che non sono stati rispettati e che necessitano di modifiche all'applicazione.

Bibliografia

- [1] Guida Spring MVC <https://www.html.it/guide/guida-al-framework-spring-mvc/>
- [2] Spring MVC Tutorials 02 - An Introduction to Spring MVC framework (Spring MVC Architecture) <https://www.youtube.com/watch?v=qHllF5pl1PA>
- [3] Spring Web Flow <https://projects.spring.io/spring-webflow/>
- [4] Spring Web Flow Reference Guide <https://docs.spring.io/spring-webflow/docs/current/reference/html/>
- [5] What is Dependency Injection? | Why | Spring <https://www.youtube.com/watch?v=Eqi-hYX50MI>
- [6] 5.02 Spring <http://www-db.deis.unibo.it/courses/TW/PDF/5.02.Spring.pdf>
- [7] Intro to Inversion of Control and Dependency Injection with Spring <https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring>
- [8] Un linguaggio di programmazione logica: il Prolog <http://www.ce.unipr.it/research/HYPERPROLOG/prolog.html>
- [9] JPL Introduction <https://jpl7.org/>
- [10] JPL Java API – Overview <https://jpl7.org/JavaApiOverview>
- [11] Il Pattern MVC <https://www.html.it/pag/18299/il-pattern-mvc/>
- [12] Bootstrap <https://getbootstrap.com/>
- [13] Spring - Java Based Configuration https://www.tutorialspoint.com/spring/spring_java_based_configuration
- [14] web.xml vs_INITIALIZER with Spring <https://www.baeldung.com/spring-xml-vs-java-config>
- [15] Spring MVC Tutorial <https://www.baeldung.com/spring-mvc-tutorial>
- [16] Guide to Spring Web Flow <https://www.baeldung.com/spring-web-flow>