UNIVERSITA' DEGLI STUDI DI
NAPOLI FEDERICO II

Facoltà di Ingegneria
Corso di Studi in Ingegneria Informatica

tesi di laurea

Realizzazione di un software di gestione allocazione risorse e analisi dei movimenti finanziari correlati

Anno Accademico 2012/2013

relatore
Ch.mo Prof. Porfirio Tramontana

correlatore
Dott. Antonio Agliata

candidato
Giuseppe Petrone
matr. 534/1413

...like a convolution
before you flip and
then delay

Indice

Introduzione

La complessità di gestione delle attività di un'organizzazione, l'allocazione del personale e degli strumenti utilizzati, l'analisi dettagliata di tutti i movimenti finanziari che nascono all'interno di queste realtà, sono tra le attività più delicate ed allo stesso tempo fondamentali per la nascita ed il corretto funzionamento di un'attività commerciale e/o organizzazione.

Si prenda ad esempio una qualunque azienda e si provi a mandarla avanti senza un'adeguata gestione del personale e delle attrezzature. Sarebbe impossibile gestire tale situazione e soprattutto non si saprebbe quanto quest'attività stia producendo o contro producendo.

Oggi il mercato propone alcune soluzioni software in grado di agevolare la gestione aziendale o l'analisi finanziaria, alcune molto complesse, altre inadeguate per tutte le realtà.

Nell'elaborato di tesi che presentiamo, vi proporremo una soluzione che unisca la gestione e l'analisi, fornendo strumenti semplici e riutilizzabili per qualunque realtà oggi esistente.

Tale elaborato è frutto di un'attività di tirocinio svolta presso la T.&C. Systems Group, società di formazione e consulenza convenzionata con l'università Federico II, operante su tutto il territorio nazionale ed europeo, con una durata complessiva di tre mesi.

Il progetto proposto dalla società è quello di analizzare, progettare e sviluppare un software, una web application, in grado di fornire strumenti di gestione risorse, allocazione di queste ed analisi finanziaria, il tutto basato su Java Enterprise Edition.

Prima di iniziare con la prima fase, ci sono state fornite le conoscenze adeguate per poter sviluppare in modo corretto il software, quindi con una serie di corsi di formazione siamo riusciti in breve tempo ad acquisire le conoscenze necessarie per poter sviluppare il progetto.

Questo elaborato sarà suddiviso in cinque capitoli, nei quali andremo ad affrontare:

- **Capitolo 1**

  Descrizione del progetto proposto, delle problematiche ad esso legate e breve panoramica sulle soluzioni software presenti sul mercato.

- **Capitolo 2**

  Analisi delle richieste del committente, spiegheremo nel dettaglio cosa vogliano dire i termini Risorsa, Allocazione e Movimento.

  Dopo un'adeguata analisi daremo un esempio di possibili usi di tale software e presenteremo quella che è la nostra soluzione alle richieste giunte e come intendiamo strutturare questa web application.

- **Capitolo 3**

  Sviluppo del Data Access Layer. In questo capitolo mostreremo la struttura della base di dati e le operazioni che andremo ad effettuare su di essa.

  Illustreremo i passi necessari per configurare Hibernate, in modo da poter comunicare con il database Oracle, e la sua integrazione con Spring.

  Analizzeremo Entity, DAO, Service e Controller di Progetto A.R.E.M.

  Analizzeremo le criticità incontrate durante lo sviluppo del layer DAL e le soluzioni adottate.

- **Capitolo 4**

  Sviluppo del web layer. In questo capitolo viene mostrata la struttura del livello web e tutti gli strumenti utilizzati per lo sviluppo di questo layer.

  Spiegheremo nel dettaglio il funzionamento di Spring MVC, cosa voglia dire usare il pattern Model-View-Controller.

Mostreremo le View più importanti della web application e i Controller necessari al loro funzionamento ed analizzeremo quali sono stati i punti critici di tale livello e quali le soluzioni adottate.

- **Capitolo 5**

  Capitolo conclusivo della tesi. Faremo delle osservazioni e considerazioni sugli obiettivi preposti.

  Daremo una descrizione di quali funzionalità verranno implementate nell'immediato futuro, ovvero la trasformazione di questa web application in una portlet per un portale basato su Liferay e la possibilità di predizione dei profitti futuri.

Capitolo 1

Aspetti generali

In questo capitolo parleremo delle richieste fatte dal committente cercando di capire a grandi linee il prodotto che desidera. Per fare ciò, ci ricondurremo ad un esempio pratico che poi sarà generalizzato per poter sfruttare il software in diversi scenari per più ampie e complesse realtà.

Studieremo anche la presenza o meno sul mercato di software capaci di fare ciò che si vuole sviluppare, in modo da poter capire cosa sfruttare di tali software, se necessario, e cosa dover migliorare o integrare per avere un prodotto finito che risponda a pieno alle esigenze del committente.

Il capitolo sarà dunque sviluppato nei seguenti paragrafi:

- 1.1 Descrizione del problema;
- 1.2 Panoramica sulle applicazioni esistenti.

1.1 Descrizione del problema

Il committente richiede che venga sviluppato un software in grado di gestire le Allocazioni tra le Risorse presenti nella sua azienda e di poter effettuare analisi sui Movimenti che tali Allocazioni generano.

La struttura delle risorse può essere gerarchica, nel senso che una risorsa è collegata ad una risorsa padre che a sua volta può essere collegata ad una o più risorse figlie.

Prendiamo come esempio un Poliambulatorio, la cui struttura può avere al suo interno più piani, su ogni piano possono trovarsi vari studi, ed in ogni studio lavorano uno o più dipendenti che utilizzano determinati materiali.

La nostra problematica è quindi gestire la struttura gerarchica dell'azienda e le transazioni economiche legate ad essa mediante un'analisi dei movimenti relativi ad una o più allocazioni, in base a diversi parametri:

- rispetto alla singola risorsa;
- al tempo;
- al tipo di movimento.

Inoltre si richiede che tale software sia utilizzabile in qualunque momento e luogo da determinati soggetti che ne hanno l'autorizzazione e che sia in grado di fornire uno storico delle Allocazioni e dei Movimenti creati durante l'arco di utilizzo dell'applicazione.

1.2 Panoramica sulle applicazioni esistenti

La maggior parte dei prodotti esistenti sul mercato sono degli Human Resources Management System (HRMS), ovvero dei software multi-funzione per la gestione del personale (comprese timbrature e mansioni), delle commissioni e della gestione delle retribuzioni.

Altre applicazioni offrono servizi simili in base alla tipologia dell'azienda interessata. La domanda sorge spontanea: perché non utilizzare questi software?

Gli HRMS si limitano alla gestione delle risorse umane, oltre ad avere funzionalità inutili al nostro scopo, mentre le altre applicazioni non permettono di svincolarsi dal settore di competenza dell'azienda committente.

Inoltre, non è disponibile nessun software in commercio che analizzi ciò su cui abbiamo deciso di concentrare l'attenzione, ossia l'aspetto economico.

Quindi per lo sviluppo del nostro software non ci siamo basati su nessuno di questi già presenti nel mercato. Abbiamo scelto di modellare l’applicazione in base alle richieste del committente, ma soprattutto in base a possibili usi futuri che un’applicazione del genere potrebbe riscontrare.

Nel prosieguo della tesi si farà riferimento sempre ad un’azienda concreta, un Poliambulatorio, ma tale software nasce con l’obiettivo di poter essere applicato in qualunque realtà necessiti una gestione ed un’analisi intrapresa in questo progetto.

Capitolo 2

Progetto A.R.E.M.

Il **Progetto A.R.E.M**, dove A.R.E.M sta per **Analisi Risorse E Movimenti**, è un software per la gestione e l'analisi delle risorse, delle allocazioni tra queste ultime e dei movimenti generati.

Tale progetto si propone di dare degli strumenti in grado di poter gestire l'allocazione delle risorse di un'azienda e/o di una qualunque realtà che ne abbia bisogno, gestire i movimenti finanziari derivanti dalle allocazioni e farne un'analisi su diverse dimensioni.

In questo capitolo studieremo i requisiti necessari al funzionamento del software e come abbiamo, in modo generale, strutturato il progetto.

I paragrafi presenti nel capitolo sono:

- 2.1 Analisi del problema;
- 2.2 Raccolta e specifica dei requisiti;
- 2.3 Analisi dei requisiti;
- 2.3.1 Use Case Diagram;
- 2.4 Dominio del problema;
- 2.4.1 La struttura del software.

2.1 Analisi del problema

Durante una serie di incontri con il cliente siamo riusciti a comprendere meglio le richieste formulate in prima battuta, si veda il paragrafo 1.1, e ad individuare quello che sarebbe stato il punto critico del software, ovvero la creazione di una struttura che potesse variare nell'arco di poche ore ma che mantenesse dei vincoli gerarchici fissi.

La Figura 2.1 è la descrizione grafica del problema, per far comprendere meglio il disegno al cliente abbiamo sfruttato l'esempio da lui proposto ovvero quello del Poliambulatorio:

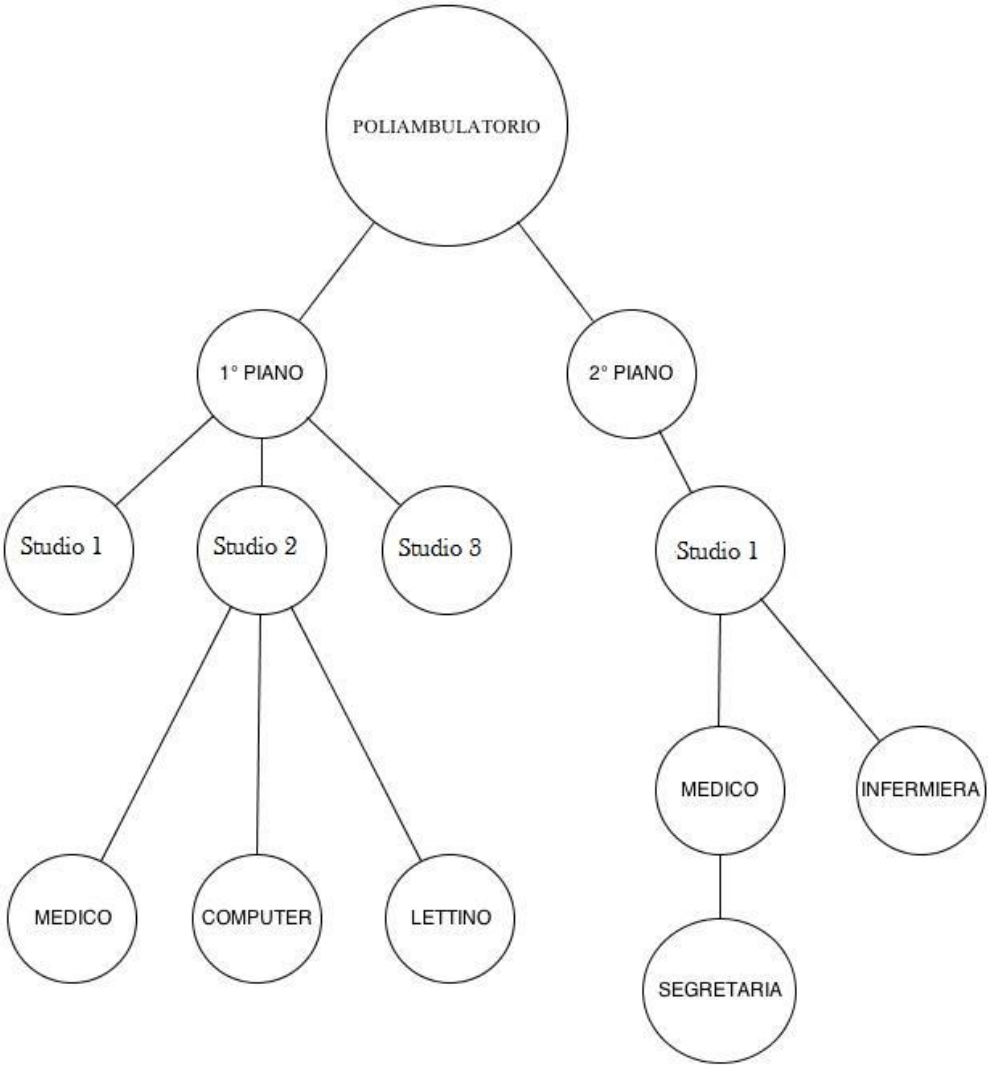

Figura 2.1 Bozza Struttura ad Albero

La problematica nella realizzazione di tale struttura consta nella gestione delle variazioni temporali, le quali comportano una modifica strutturale dell'albero. Tale modifica deve essere automatizzata in funzione del tempo di allocazione tra due risorse verificando l'impossibilità di allocare una risorsa figlia a due padri nello stesso tempo.

La struttura in figura è servita non solo per chiarire su quale problema dovremo prestare maggiore attenzione, ma ha reso chiaro fin da subito come dovrà esser presentato il software al cliente.

Durante i colloqui avvenuti anche in fase di progettazione e sviluppo egli ha manifestato forte interesse affinché fosse visibile questo tipo di struttura ad albero per le operazioni di gestione ed analisi del software.

2.2 Raccolta e specifica dei requisiti

Per prima cosa, dobbiamo specificare il significato dei termini citati ad inizio capitolo.

Col termine "Risorsa" si intende qualunque entità di un solo "Tipo", allocabile, che genera un costo o un ricavo.

Per "Allocazione" si intende l'assegnazione di una Risorsa ad un'altra per un periodo specifico, la quale determina la creazione di una struttura gerarchica.

Un "Movimento" può essere un costo o un ricavo, in generale un movimento finanziario, legato ad un'allocazione, e non ad una singola risorsa, rispetto alla sua durata, all'importo e alla causale.

Per chiarire questi concetti riprendiamo l'esempio usato in precedenza, quello del Poliambulatorio, in cui tutti gli studi, i medici, le segretarie ed i macchinari sono Risorse, rispettivamente di tipo *studio*, *medico*, *segretaria* e *macchinario.*

Tuttavia anche il Poliambulatorio stesso è una Risorsa che avrà delle allocazioni e dunque genera movimenti finanziari.

Un medico è assegnato ad uno studio tramite un'Allocazione che lega le due Risorse per un dato periodo.

Il punto di vista da analizzare è quello del proprietario dell'azienda, secondo il quale tutto il denaro in uscita rappresenta un costo, quello in entrata un ricavo.

Nel Poliambulatorio, l'affitto e gli stipendi sono costi, mentre le prestazioni pagate dai clienti sono ricavi.

2.3 Analisi dei requisiti

Nella fase di analisi andiamo a definire quali sono le funzionalità che il software deve offrire e che tipo di utenti accedono ai suoi servizi.

L'applicazione offre i seguenti servizi:

- *Gestione delle Allocazioni*: l'utente interessato ha la possibilità di accedere alla sezione per creare le allocazioni tra risorse e modificare i relativi periodi. Questa funzionalità include la gestione delle risorse.
- *Gestione delle Risorse*: al momento della creazione di un'allocazione può essere creata una nuova risorsa, se non è presente, o modificarne una esistente.
- *Gestione dei Movimenti*: l'utente può accedere alla sezione per la creazione di movimenti legati alle allocazioni di uno specifico periodo.
- *Analisi delle Allocazioni*: l'utente ha la possibilità di visualizzare le allocazioni di un periodo selezionato.
- *Analisi dei Movimenti*: l'utente può analizzare i movimenti aggregandoli rispetto ad una singola risorsa, al tempo e al tipo di movimento.
- *Analisi del Profitto*: l'utente ha la possibilità di analizzare il profitto dell'azienda.

Le persone che utilizzeranno il software sono :

- *Amministratore*: si occupa della parte amministrativa, ossia la gestione dei movimenti generati dalle allocazioni, l’analisi di tali movimenti e del profitto.
- *Manager*: si occupa della gestione e dell'analisi delle allocazioni, dell'analisi dei movimenti e del profitto.

Per schematizzare tutto ciò facciamo riferimento allo Use Case Diagram in Figura 2.1.

2.3.1 Use Case Diagram

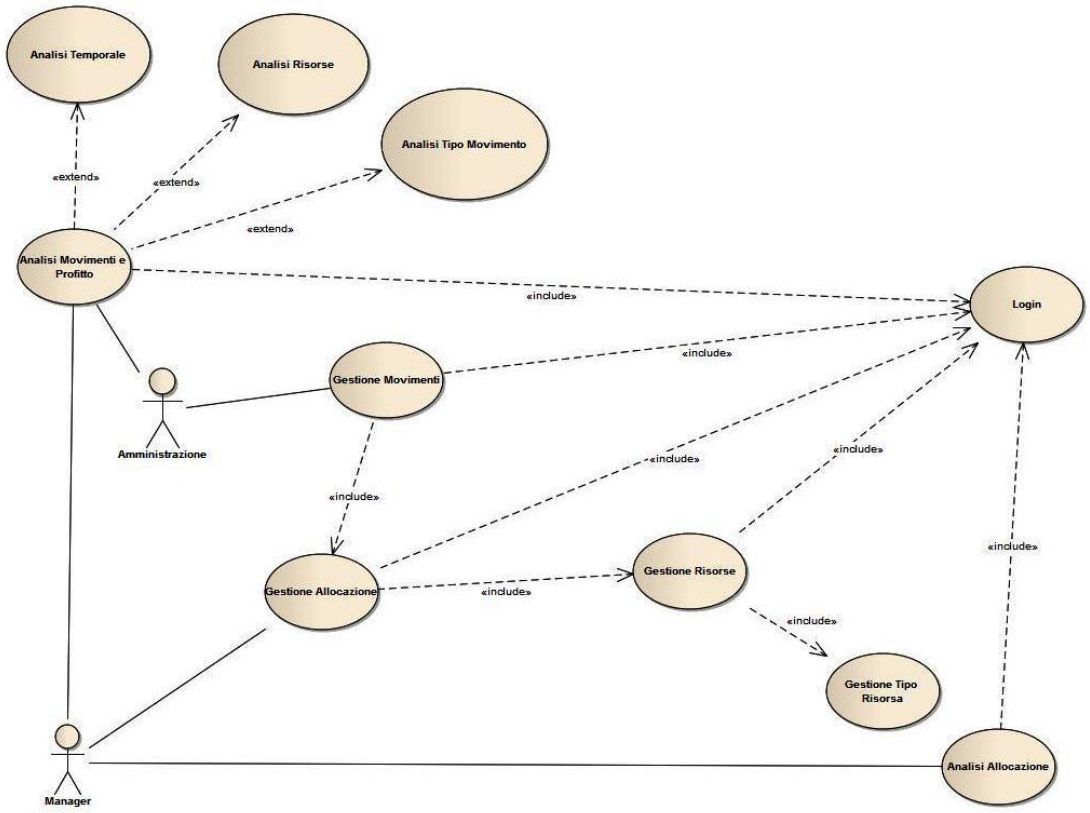

Figura 2.1 Use Case Diagram Progetto A.R.E.M.

Si sceglie, per brevità, di omettere l'uso del formalismo dei diagrammi di cockburn, ma si riporta di seguito una breve descrizione dei loro scenari:

- **Login** – accesso al software A.R.E.M;
- **Gestione Allocazione** – il Manager è il solo in grado di poter creare e/o aggiornare le allocazioni tra le risorse. Per la corretta esecuzione di tali funzioni è necessario che egli abbia creato, se non sono già presenti in database, le opportune risorse da allocare;
- **Gestione Risorse** – funzione che permette al Manager la creazione di nuove risorse che verranno inserite in database;
- **Gestione Tipo Risorse** – ogni risorsa può essere di un solo tipo. La creazione dei tipi da associare alle risorse è a carico del Manager;

- **Gestione Movimenti** – questa funzione permette all'Amministratore di creare e/o aggiornare i movimenti finanziari dovuti alle allocazioni delle risorse;
- **Analisi Allocazioni** – il Manager può effettuare un'analisi sulle allocazioni che sono state create durante l'intero periodo di vita dell'azienda che usa il software;
- **Analisi Movimenti e Profitto** – il Manager e l'Amministratore possono analizzare i movimenti finanziari scaturiti a seguito delle allocazioni tra le risorse. Tale analisi può esser effettuata su base temporale, in funzione di una singola risorsa o del tipo di movimento che si vuole analizzare.

2.4 Dominio del problema

Stabilito che le Risorse sono collegate tra loro tramite delle Allocazioni e che da queste vengono generati Movimenti che sono di tipo Costo o Ricavo, abbiamo costruito il modello Entità-Relazioni riportato in Figura 2.2.

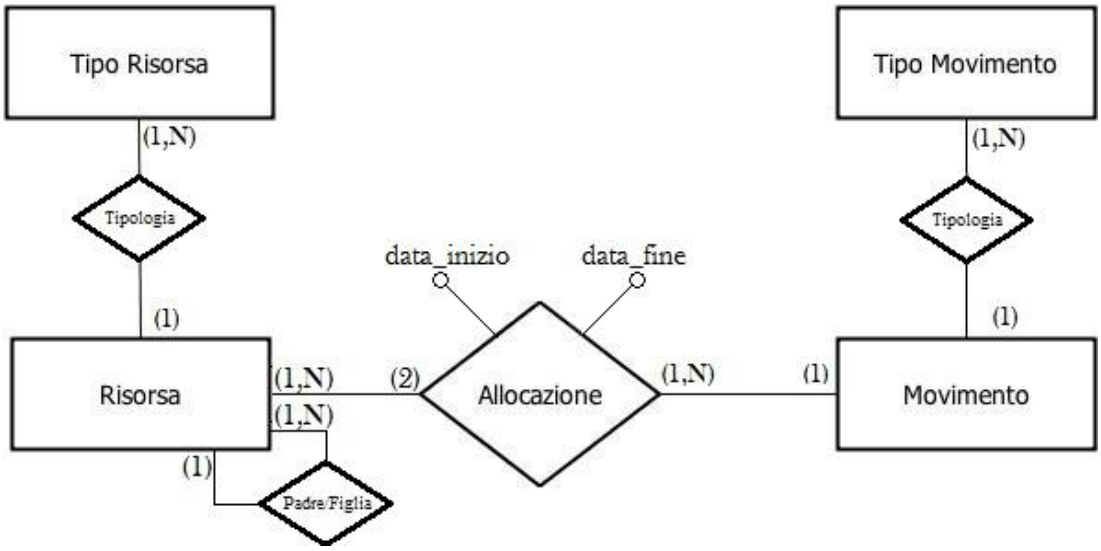

Figura 2.2 Modello ER del progetto

Di seguito riportiamo il dizionario delle entità e delle relazioni in forma tabellare.

Nome	Descrizione	Attributi	Identificatore
Risorsa	Vengono memorizzati i nominativi delle risorse da allocare.	id_risorsa nominativo	id_risorsa
Tipo Risorsa	Contiene i tipi da associare alle risorse.	id_tipo_risorsa descrizione	id_tipo_risorsa
Allocazione	Tabella relativa alle allocazioni tra risorse.	id_allocazione data_inizio data_fine risorsa_padre risorsa_figlia	id_allocazione

Movimento	Tabella relativa ai movimenti finanziari.	id_movimento data_inizio data_fine id_tipo_movimento peso ripartito id_allocazione	id_movimento
Tipo Movimento	Tabella relativa ai tipi di movimenti.	id_tipo_movimento causale importo importo_orario	id_tipo_movimento

Tabella 2.1 Tabella delle entità

Nome	Descrizione	Attributi	Entità
Tipologia	Tale relazione associa Movimento a Tipo Movimento e Risorsa a Tipo Risorsa	nessuno	Movimento Tipo Movimento Risorsa Tipo Risorsa
Padre/Figlia	Mette in relazione due risorse secondo la gerarchia padre-figlia	nessuno	Risorsa

Tabella 2.2 Tabella delle relazioni

Per il caricamento in memoria della struttura dati memorizzata nel database si ricorre all'uso di una struttura ad albero come quella esposta al cliente in fase di analisi, Figura 2.1.

In tal modo l'accesso alle risorse allocate avviene in un tempo pari a *O(log N)* dove N è il numero dei nodi dell'albero, quindi il numero delle risorse allocate.

É importante precisare che un movimento non è legato alla singola risorsa ma all'allocazione della stessa ad un'altra, per esempio il costo dello stipendio di un medico non è associato alla risorsa, ma all'allocazione tra la stanza e il medico.

Detto questo, per facilitare la comprensione, accosteremo il movimento alla risorsa figlia dell'allocazione così da parlare di ricavo o costo di una risorsa.

Abbiamo poi concentrato l'attenzione sui movimenti che, come detto in precedenza, possono essere costi o ricavi ed hanno una durata che deve essere compresa nell'arco temporale dell'allocazione corrispondente.

Un Costo può essere:

- diretto: legato strettamente alla risorsa;
- ereditato: legato a risorse discendenti;
- ripartito: ereditato dalle risorse discendenti in base ad un peso, espresso in percentuale, stabilito dall'utente. Ad esempio il costo della bolletta della luce del Poliambulatorio può essere ripartito ai vari studi con un peso che varia a seconda delle prese di corrente presenti. Nel prosieguo della tesi questo concetto verrà ripreso e reso più chiaro.

Un Ricavo può essere:

- diretto: legato strettamente alla risorsa;
- ereditato: legato a risorse discendenti.

Notevole importanza è stata data anche all'interfaccia per rendere l'utilizzo del software più semplice ed immediato possibile.

2.4.1 La struttura architetturale del software

Quando si progetta un software che sfrutta una base di dati per memorizzare le informazioni, l'approccio più diffuso, specialmente nel caso di applicazioni complesse, è quello di scomporre il sistema in diversi livelli (*layer*).

Ogni livello ha dei compiti specifici ed è in grado di comunicare con gli altri. Questa tecnica è utile per suddividere le funzionalità migliorando le prestazioni e facilitando gli eventuali cambiamenti futuri.

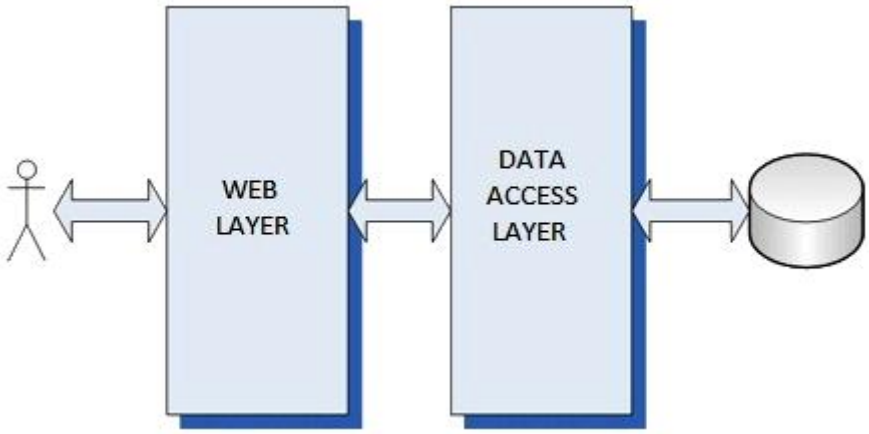

Figura 2.4 Bozza dell'architettura del progetto

Nel nostro caso abbiamo strutturato il software in due parti:

1. ***Data Access Layer*** (DAL) che è il livello di interazione con la base di dati, dove si effettuano le connessioni ed i controlli, si dichiarano le entità e si eseguono *query* di selezione, inserimento, aggiornamento o cancellazione.
2. ***Web Layer*** che racchiude due livelli
    - Il livello di presentazione costituisce l'interfaccia utente, ed ha lo scopo di gestire l'interazione del sistema con il mondo esterno. Include le maschere per la visualizzazione e l'inserimento dei dati, i controlli, dai più semplici ai più complessi, e i meccanismi per intercettare e trattare opportunamente gli eventi generati dalle azioni svolte dagli utenti;
    - Il livello intermedio contiene la logica elaborativa dell'applicazione, intercetta le richieste provenienti dallo strato di presentazione e le gestisce opportunamente.

Capitolo 3

Data Access Layer

In questo capitolo parleremo della logica di accesso alle informazioni ed il loro salvataggio nel database. Per rendere queste operazioni, rapide, semplici e sicure, utilizzeremo un Object Relational Mapping (ORM).

Nello specifico, un ORM definisce il mapping tra gli oggetti della propria applicazione e quelli presenti all'interno della base di dati, fornendo anche tutti i metodi per l'accesso e la modifica delle informazioni, garantendo un'elevata portabilità rispetto al DBMS utilizzato.

Il tipo di ORM che abbiamo usato è Hibernate, che successivamente andremo ad integrare al framework Spring.

Per la memorizzazione dei dati abbiamo scelto come supporto un database Oracle.

Paragrafi presenti nel capitolo:

- 3.1 Class Diagram;
- 3.2 Hibernate:
- 3.2.1 Configurazione;
- 3.2.2 Mapping tramite annotation;
- 3.3 Spring Core:
- 3.3.1 Configurazione;
- 3.3.2 Integrazione con Hibernate;
- 3.4 Struttura A.R.E.M. DAL:
- 3.4.1 Entity;
- 3.4.2 DAO;
- 3.4.3 Service;
- 3.4.4 Controller;
- 3.5 Punti critici e soluzioni.

3.1 Class Diagram

Sulla base delle informazioni in nostro possesso e dell'analisi svolta, abbiamo progettato il Class Diagram presente in Figura 3.1, esso è la traduzione del modello ER in un dominio ad oggetti, frutto dell'uso del framework ORM Hibernate. Ovviamente con gli opportuni *impedance mismatch*.

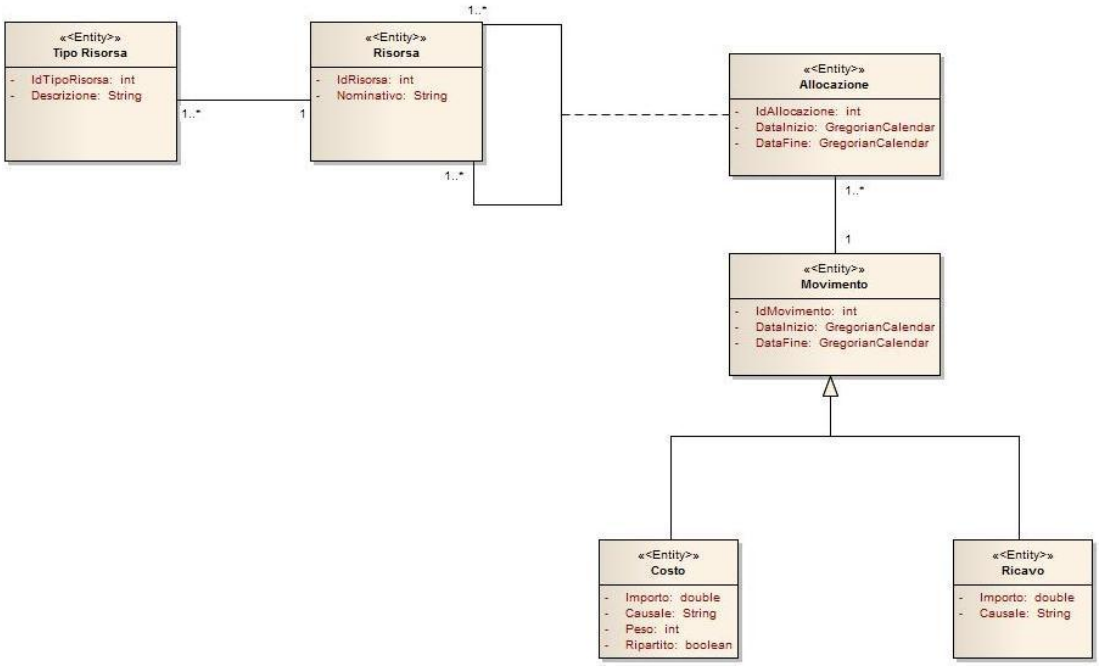

Figura 3.1 Class Diagram A.R.E.M. DAL

In figura vengono mostrate le Entity che dovranno essere create per poter soddisfare le richieste del committente. Tali entity saranno le tabelle del database.

Nei paragrafi successivi analizzeremo nel dettaglio le tabelle create per lo sviluppo del progetto.

3.2 Hibernate

Come già detto in precedenza Hibernate è un ORM che offre servizi di query per le Java Application, inoltre mappa le classi Java in tabelle di database e i tipi di dati Java in tipi di dati SQL garantendo così un'elevata persistenza e facilitando molto il lavoro dello sviluppatore.

I vantaggi di Hibernate sono molteplici :

- Si occupa di mappare le classi Java utilizzando i file XML senza scrivere alcuna riga di codice in più;
- Fornisce semplici Application Programming Interface (API) per l'archiviazione e il recupero di oggetti Java direttamente da e verso il database;
- Se non vi sono cambiamenti nel database o in qualsiasi tabella allora l'unica necessità è quella di cambiare le proprietà del file XML;
- Riduce al minimo l'accesso al database con strategie intelligenti di caricamento diretto;
- Fornisce una semplice interrogazione alla base di dati;
- Supporta la maggior parte dei Relational Data Base Management System (RDBMS).

Hibernate usa la Java API Java DataBase Connectivity (JDBC) che fornisce un livello base di astrazione di funzionalità comuni di database relazionali, permettendo a quasi tutti i database con un driver JDBC di essere sostenuti da Hibernate.

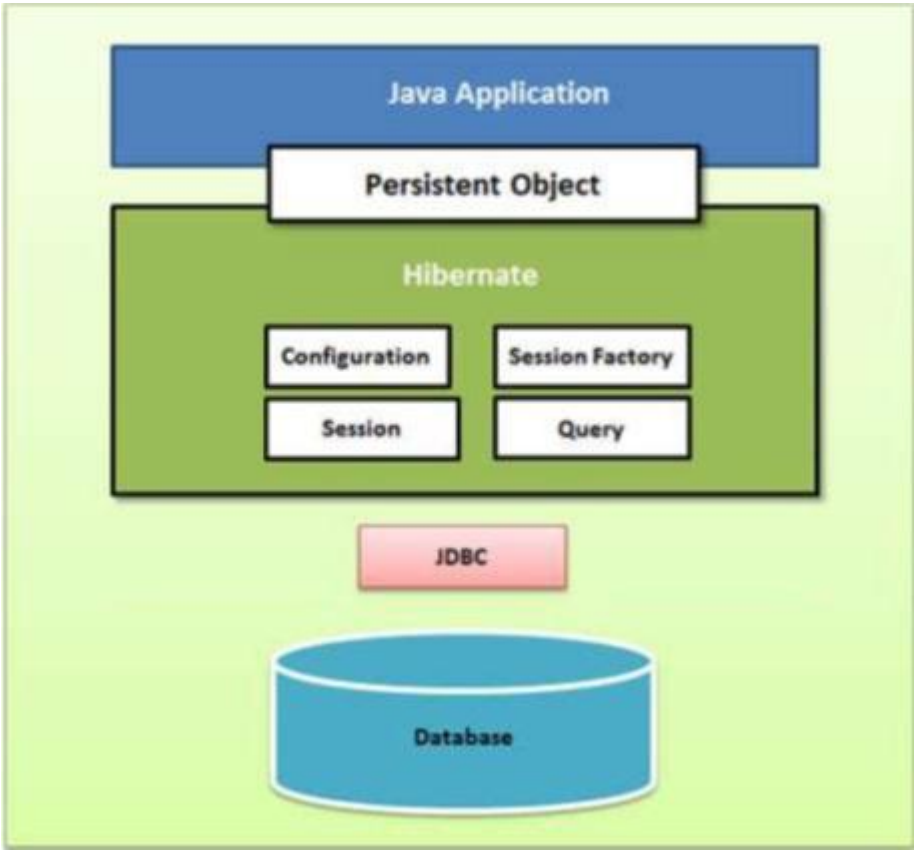

Figura 3.2 Struttura di Hibernate

In figura 3.2 è rappresentata la struttura di Hibernate che andiamo ad analizzarla nel dettaglio.

Configuration.

L'oggetto di configurazione è il primo che si crea, una sola volta, in qualsiasi applicazione Hibernate.

Rappresenta un file di configurazione con le proprietà richieste e prevede due componenti chiave:

- *Connessione al database*

  questo viene gestito attraverso un file di configurazione supportato da Hibernate;
- *Configurazione Class Mapping*

  questo componente crea il collegamento tra le classi Java e le tabelle del database.

SessionFactory.

La configurazione viene utilizzata per creare un oggetto SessionFactory che è un oggetto thread-safe ed è utilizzato da tutti i processi di un'applicazione.

SessionFactory è un oggetto complesso e pesante per cui, di solito, si crea quando l'applicazione viene avviata e conservato per un uso successivo.

Session.

L'oggetto Session viene utilizzato per ottenere una connessione fisica con un database mediante la SessionFactory.

È progettato per essere istanziato ogni volta che è necessaria una interazione con il database.

Gli oggetti persistenti vengono salvati e recuperati tramite l'oggetto Session, perciò questo viene creato e distrutto quando necessario.

Query.

Gli oggetti Query utilizzano stringhe Structured Query Language (SQL) o Hibernate Query Language (HQL) per recuperare i dati dal database e creare oggetti.

3.2.1 Configurazione Hibernate

La cosa più importante quando si desidera usare Hibernate, oltre ad importare le librerie necessarie, è creare un file di configurazione chiamato *hibernate.cfg.xml* dove settare le proprietà fondamentali per la comunicazione col database e precisamente:

- **hibernate.dialect**
  questa proprietà permette ad Hibernate di generare l'SQL appropriato per il database selezionato;
- **hibernate.connection.driver_class**
  la classe del driver JDBC;
- **hibernate.connection.url**

URL JDBC per l'istanza del database;

- **hibernate.connection.username**
  nome utente per accedere al database;
- **hibernate.connection.password**
  password per accedere al database;
- **hibernate. hbm2ddl.auto**
  che può assumere i valori:
  - **create**
    Lo schema relativo alla mappatura viene creato ad ogni inizio sessione sovrascrivendo quello precedente.
    I contenuti andranno persi;
  - **create-drop**
    Come create, ma a fine sessione lo schema creato viene cancellato;
  - **validate**
    Verifica se c'è corrispondenza tra la mappatura e lo schema in memoria, se non coincidono viene lanciata un'eccezione;
  - **update**
    Se lo schema in memoria e la mappatura coincidono viene lasciato lo schema così com'è con i suoi contenuti, altrimenti viene aggiornato alle eventuali modifiche.

3.2.2 Mapping tramite annotation

Il modo più semplice per mappare un database con hibernate è tramite annotazioni inserite opportunamente nel codice.

Vediamo quelle che abbiamo utilizzato nel nostro progetto:

- @Entity
  Comunica ad Hibernate che la classe sottostante è un'entità;

- @Table

  Tramite l'attributo name, permette di definire il nome che avrà la tabella corrispondente nel database. Se questo attributo non è presente, il nome della tabella sarà quello della classe;

- @Id

  Comunica che l'attributo sottostante sarà chiave primaria nella tabella;

- @Column

  Assegna il nome di una colonna tramite l'attributo name.

- @GeneratedValue

  Definisce la generazione della chiave primaria, esso accetta una strategia e un generatore.

  Le strategie sono diverse ma quella che usiamo è la strategia di default GenerationType.AUTO e non ha bisogno di un generatore, il numero assegnato è progressivo.

- @JoinColumn

  Stabilisce il nome della chiave esterna e la chiave al quale essa è associata, l'attributo unique serve a far rispettare o meno il vincolo di unicità.

Con Hibernate, ovviamente è possibile mappare anche le associazioni tra classi sempre attraverso annotation.

Abbiamo utilizzato @OneToMany per l'associazione uno a molti monodirezionale e bidirezionale, tra le due va fatta una distinzione:

- Monodirezionale

  Il riferimento alla classe associata è presente solo nell'entità "debole", cioè quella che nell'associazione ha cardinalità "molti".

  In questo caso l'annotazione viene inserita nella classe "forte", cioè quella che ha cardinalità uno, e viene aggiunta l'annotation @JoinColumn.

- Bidirezionale

  Il riferimento alla classe associata è presente in entrambe le entità.

Le annotazioni @OneToMany e @JoinColumn vengono poste nell'entità "debole", mentre in quella "forte" viene messa l'annotazione @ManyToOne che fa riferimento alla classe associata attraverso l'attributo mappedBy opportunamente valorizzato.

3.3 Spring Core

Spring è un framework opensource per lo sviluppo di applicazioni su piattaforma Java creato per facilitare lo sviluppo dei programmatori.

La sua struttura è mostrata in Figura 3.3.

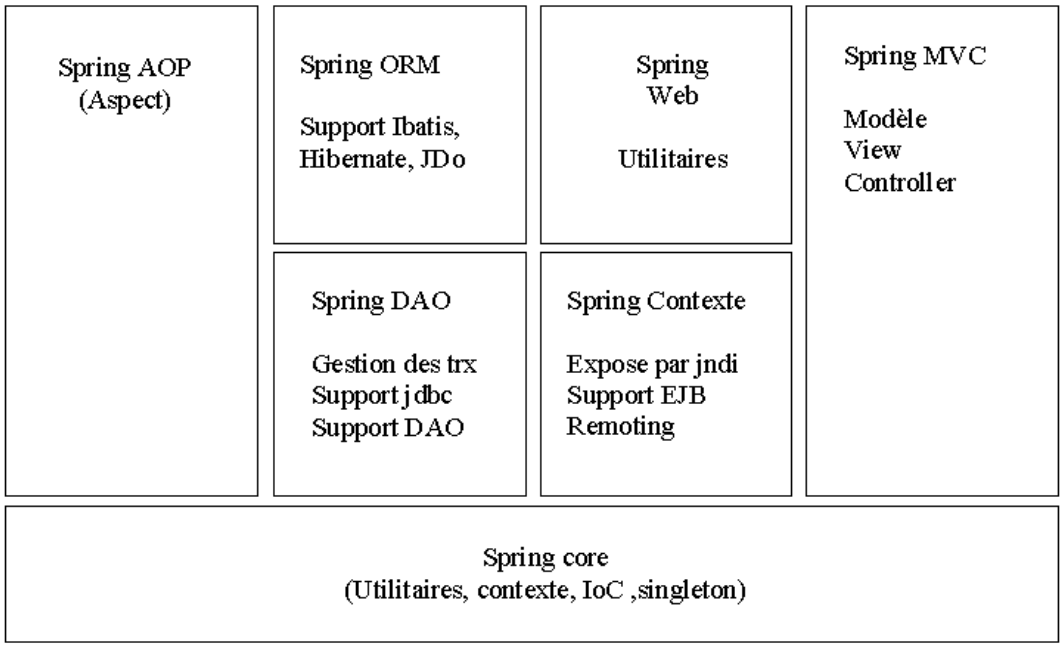

Figura 3.3 Struttura del framework Spring

Il kernel base di Spring viene definito Spring Core ed è quello utilizzato in questa fase di progettazione.

Questo framework introduce un concetto fondamentale per il corretto funzionamento di Spring, il quale rende anche l'idea delle potenzialità del framewrok, quello dell'Inversion of Control (IoC).

IoC è un pattern di programmazione basato sul concetto di "invertire il controllo del flusso di sistema rispetto alla programmazione procedurale".

Si tende, in pratica, a tenere disaccoppiati i singoli componenti di sistema.

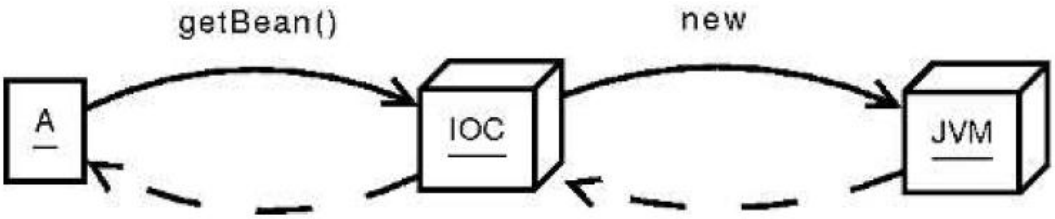

Figura 3.4 Inversion of Control

Le funzionalità di istanza e richiamo degli oggetti sono fornite da un ambiente esterno, questo porta a rendere gli oggetti riusabili e, se ci sono modifiche da fare, non bisogna farle per tutti i punti in cui gli oggetti sono creati e inizializzati.

Una particolare forma di IoC è la Dipendence Injection con la quale le classi non sono responsabili delle proprie dipendenze.

Ci sono due tipi di injection:

- Setter Injection

  Dipendenze fornite attraverso i metodi di configurazione dei componenti (i metodi set);
- Constructor Injection

  Dipendenze fornite attraverso i costruttori dei componenti.

In Sping quindi mappiamo le associazioni tra le classi, l'oggetto che gestisce i bean e le loro dipendenze è l'ApplicationContext, che provvede alla fase di injection leggendo un file di configurazione.

3.3.1 Configurazione Spring

Una cosa importante da fare, dopo aver importato le librerie necessarie, è creare il file di configurazione spring.xml:

- Definire i bean attraverso i tag <beans> e <bean>.

In Java li recuperiamo con il metodo *getBean dell'ApplicationContext* che restituisce un Object.

Ogni *bean* deve avere un nome unico, se il tag ha un attributo di nome *id*, il valore di questo viene usato come nome.

Se non c'è l'attributo *id*, Spring cerca un attributo *name*, se nessuno dei due è definito allora il nome usato sarà quello della classe;

- Effettuare il Setter Injection mediante il tag <property> con il nome dell'attributo e il valore da settare;
- Effettuare il Constructor Injection attraverso il tag <contructor-arg>, che assegna il valore al costruttore;

Se <property> e <constructor-arg> sono inseriti insieme allora il costruttore avrà la precedenza.

3.3.2 Integrazione con Hibernate

L'integrazione di Spring con Hibernate è la fase più importante di questa parte del progetto.

Partendo dai file hibernate.cfg.xml e spring.xml bisogna creare un nuovo file, configurazione.xml, con i seguenti *bean*:

- dataSource.

  Dove settare le proprietà fondamentali per la comunicazione col database come si faceva nel file di configurazione di hibernate;
- sessionFactory.

  Per la creazione della Session Factory, l'oggetto responsabile dell'apertura delle sessioni verso il database.

  I parametri fondamentali da inserire sono:
  - Il datasource da utilizzare per la connessione al database;
  - Dialetto SQL utilizzato da Hibernate per l'ottimizzazione delle query;
  - Lista di file di mapping.

3.4 Struttura A.R.E.M. DAL

La parte DAL del progetto A.R.E.M, come mostrato in figura 3.5, è strutturato su vari livelli che permettono di suddividere il lavoro e facilitare le modifiche future :

- **DAO**
- **Service**
- **Controller**

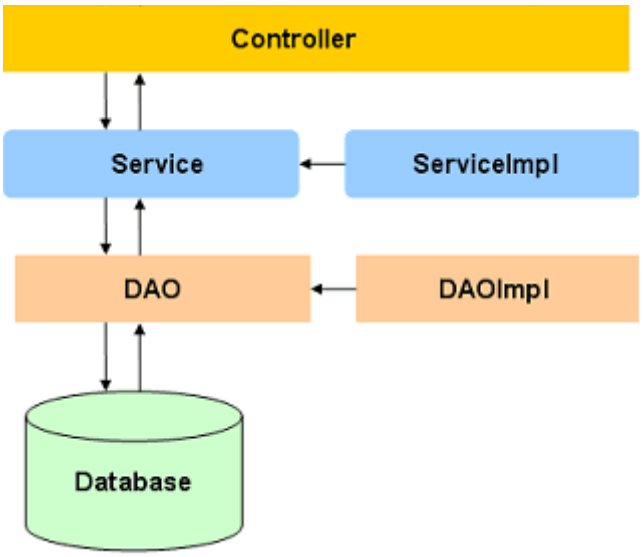

Figura 3.5 Struttura di A.R.E.M DAL

Prima di entrare nel dettaglio è bene introdurre un altro livello attraverso il quale viene fatto il mapping del database tramite le annotation viste nel capitolo 3.2.2, il livello Entity.

3.4.1 Entity

Le entità altro non sono che le classi create e mappate sul database attraverso l'annotazione @Entity.

Le entity del Progetto A.R.E.M. sono:

- Risorsa

  In questa tabella vengono memorizzate tutte le risorse che si possono allocare. Essa presenta i seguenti attributi:

  - IdRisorsa.

    È il codice identificativo della risorsa, la chiave primaria nel database, ed è un intero auto generato con l'annotazione @GeneratedValue;
  - Nominativo.

    È il nominativo della risorsa. Ad esso è associata l'annotazione @Column con attributo unique settato a "true" per evitare duplicati, in questo modo una risorsa può essere identificata sia dall'id che dal nominativo;
  - TipoRisorsa.

    È una chiave esterna all'entità TipoRisorsa dato che c'è una relazione Molti a Uno.

    Per rappresentare questo legame usiamo in questa classe l'annotazine @ManyToOne e in TipoRisorsa l'annotazione @OneToMany, dato che l'associazione è bidirezionale;
  - èPadre.

    Indica l'allocazione nella quale la risorsa è padre. È una chiave esterna all'entità Allocazione dato che c'è una relazione Uno a Molti;
  - èFiglia.

Indica l'allocazione nella quale la risorsa è figlia, ed è una chiave esterna all'entità Allocazione dato che c'è una relazione Uno a Molti.

- TipoRisorsa

  Descrive il tipo della risorsa. Gli attributi sono:

  - IdTipoRisorsa.

    È il codice identificativo del tipo di risorsa, è un "intero" auto generato con l'annotazione @GeneratedValue;

  - Descrizione.

    È la descrizione del tipo di risorsa corrispondente ad una stringa col nome del tipo di risorsa;

  - Risorsa.

    Indica la risorsa alla quale questo tipo è associato. È una chiave esterna in quanto vi è un legame Uno a Molti tra i tipi e le risorse.

- Allocazione

  Rappresenta un'associazione tra due risorse, con attributi :

  - idAllocazione.

    Identificativo dell'allocazione, è un "intero" auto generato con l'annotazione @GeneratedValue;

  - idRisorsaPadre.

    Identificativo della risorsa padre nell'allocazione, è una chiave esterna alla tabella Risorsa e dato che c'è una relazione Molti a Uno, su tale attributo andrà l'annotazione @ManyToOne e, come detto in precedenza, in Risorsa ci sarà una @OneToMany corrispondente, dato che l'associazione è bidirezionale.

  - idRisorsaFiglia.

    Identificativo della risorsa figlia nell'allocazione, è una chiave esterna alla tabella Risorsa e dato che c'è una relazione Molti a Uno, su tale attributo andrà l'annotazione @ManyToOne e,

come detto in precedenza, in Risorsa ci sarà una @OneToMany corrispondente, dato che l'associazione è bidirezionale.

- Movimento.

  Identificativo del movimento generato dall'allocazione, è una lista di chiavi esterne alla tabella Movimento dato che c'è una relazione Uno a Molti.

  Per dare questo tipo di istruzione al database, usiamo l'annotazione @OneToMany e in Movimento ci sarà una @ManyToOne corrispondente, dato che l'associazione è bidirezionale.
- dataInizio.

  Data iniziale dell'allocazione, è un GregorianCalendar.
- dataFine.

  Data finale dell'allocazione, è un GregorianCalendar.

- Movimento

  Rappresenta un movimento economico, che può essere di tipo Costo o Ricavo e di conseguenza è legato a queste due entità tramite l'annotazione @Inerheritance con strategia SIGLE_TABLE.

  Gli attributi di questa entità sono:

  - idMovimento.

    Identificativo del movimento, è un intero auto generato con l'annotazione @GeneratedValue;
  - Allocazione

    Identificativo dell'allocazione che genera il movimento, è una chiave esterna alla tabella Allocazione dato che c'è una relazione Molti a Uno.

    Usiamo l'annotazione @ManyToOne e, come detto in precedenza, in Allocazione ci sarà una @OneToMany corrispondente, dato che l'associazione è bidirezionale.
  - dataInizio.

Data iniziale del movimento, è un GregorianCalendar;

- dataFine.

  Data finale del movimento, è un Gregorian Calendar;

- Costo

  Estensione della classe Movimento, rappresenta un movimento economico in uscita per l'azienda, gli attributi sono :

  - Importo.

    L'importo del movimento;

  - importoOrario.

    Dato che le allocazioni possono anche essere orarie, è utile avere informazioni sul costo orario delle allocazioni. Tale costo è calcolato automaticamente dal sistema in funzione dell'importo e delle date;

  - Causale.

    Motivazione di questo spostamento di denaro;

  - Peso.

    quanto influisce la risorsa su un movimento ripartito, in percentuale;

  - Ripartito.

    Valore booleano che indica se il movimento deve essere ripartito alle allocazioni discendenti o meno;

- Ricavo

  Estensione della classe Movimento, rappresenta un movimento economico in entrata per l'azienda, gli attributi sono:

  - Importo.

    L'importo del movimento;

  - importoOrario.

    Dato che le allocazioni possono anche essere orarie, è utile avere informazioni sul ricavo orario delle allocazioni. Tale costo è

calcolato automaticamente dal sistema in funzione dell'importo e delle date;

- Causale.

  Motivazione di questo spostamento di denaro;

- Account

  Serve per memorizzare gli account che avranno accesso all'applicazione, gli attributi sono:

  - Username.

    Username dell'utente che accederà all'applicazione, è la chiave primaria della tabella;
  - Password.

    Password che permette all'utente di accedere all'applicazione;
  - Nome e Cognome.

    Nome e Cognome dell'utente a cui è associato l'accout;
  - tipoAccount.

    Tipologia di account assegnato, nel nostro caso sarà Manager o Amministratore.

3.4.2 DAO

Le classi Data Access Objects hanno il compito di interagire con il database. Possiedono quindi i metodi utili per effettuare le operazioni CRUD sul database, dove CRUD sta per Create, Read, Update e Delete.

Ogni operazione ha un'equivalente istruzione nel linguaggio SQL

- Create (*INSERT*)
- Read (*SELECT*)
- Update (*UPDATE*)
- Delete (*DELETE*)

Ad ogni classe Entity corrispondono due classi DAO, l'interfaccia e l'implementazione.

Questa distinzione serve a creare una separazione tra l'implementazione e l'effettivo utilizzo della classe, cioè con l'interfaccia si dice cosa si deve fare, mentre con l'implementazione si definisce come farla.

L'implementazione della classe DAO è preceduta dall'annotazione di Spring @Repository, cioè l'interfaccia DAO che esegue le operazioni CRUD sulla base di dati.

3.4.3 Service

Le classi service vengono utilizzate per evitare ai controller che utilizzeranno i DAO l'uso diretto di questi. Il motivo di tale scelta ricade nella ormai consueta metodologia di divisione in livelli per rendere ogni strato del software indipendente, quanto più possibile, dagli altri. Difatti, utilizzando un livello service sarà più semplice in futuro sganciare la base di dati corrente ed agganciarne un'altra.

Per ogni classe DAO c'è la corrispondente classe Service. Questo significa che, per ogni Entity, si avranno due classi service, cioè una interfaccia e una implementazione.

Come nelle classi DAO, l'interfaccia indica cosa si deve fare, mentre l'implementazione come farla.

In questo caso sopra il nome delle classi d'implementazione si inserisce l'annotazione @Service e al suo interno, sopra ogni metodo, l'annotazione @Transactional, la quale è necessaria per richiamare il corrispondente metodo dell'implementazione della classe DAO.

Le due parti possono essere sviluppate e compilate separatamente, ma ovviamente verrà utilizzata sempre l'interfaccia che richiamerà l'implementazione.

3.4.4 Controller

Il livello Controllore si prende carico di effettuare i controlli sui dati che si vuole inserire nel database in modo da rispettare tutti i vincoli richiesti. Per identificare questa classe inseriamo l'annotazione di Spring @Controller.

Citiamo alcuni dei controlli effettuati:

- controllo Date.

  Controlla che la data iniziale non superi la data finale;

- controlloDateMovimento.

  Controlla che il movimento venga generato durante il periodo di vita dell'allocazione;

- contolloDateAllocazione.

  Controlla le date di allocazione.

  Non dovrà mai verificarsi un'allocazione del tipo figlia-->padre dove i padri sono due contemporaneamente.

  È invece possibile avere un'allocazione figlia-->padre con due padri differenti in tempi differenti;

- controlloRipartito.

  Controllo incrociato tra la somma dei pesi dei movimenti a cui si ripartisce il costo e lista delle risorse.

  Per tutti gli elementi presenti in lista, che saranno i figli della risorsa figlia dov'è avvenuta l'allocazione che ha dato luogo al movimento da ripartire, viene ripartito il costo del movimento in funzione del peso che viene attribuito (per testare il controllo abbiamo imposto che i pesi siano tutti uguali).

  Durante il controllo si effettua una verifica sul flag "ripartito" del movimento che si sta creando, il quale se settato effettua una ripartizione anche sui figli del nodo al quale già è stato ripartito il movimento estendendo questo controllo a macchia d'olio fino a quando si desidera (in modo ricorsivo).

Ultima fase consiste nel controllare la somma dei pesi che dovrà essere sempre uguale a 100 in quanto i pesi sono delle percentuali attribuite ai movimenti per ripartire l'importo;

- controlloPeso.

  Controlla il valore del peso settato al movimento, che dovrà essere sempre un valore compreso tra 0 e 100.

3.5 Punti critici e soluzioni

Tra i punti critici incontrati, durante lo sviluppo del livello DAL, il più importante è stato il controllo delle date, in quanto sia allocazioni che movimenti ne hanno una iniziale e una finale.

In particolare è stato necessario gestire i periodi di vita delle allocazioni nelle ripartizioni dei movimenti su più livelli. Quando un Costo o un Ricavo vengono ripartiti, viene assegnato un movimento alle allocazioni figlie di quella di partenza nel periodo scelto, questo viene fatto ricorsivamente.

Date due allocazioni, la seconda è figlia della prima se la risorsa padre nella seconda allocazione è figlia nella prima allocazione, cioè le allocazioni discendenti devono avere un periodo di vita incluso in quello dell'allocazione genitrice.

Alle discendenti viene ripartito il movimento se soddisfano il controllo schematizzato in Figura 3.6.

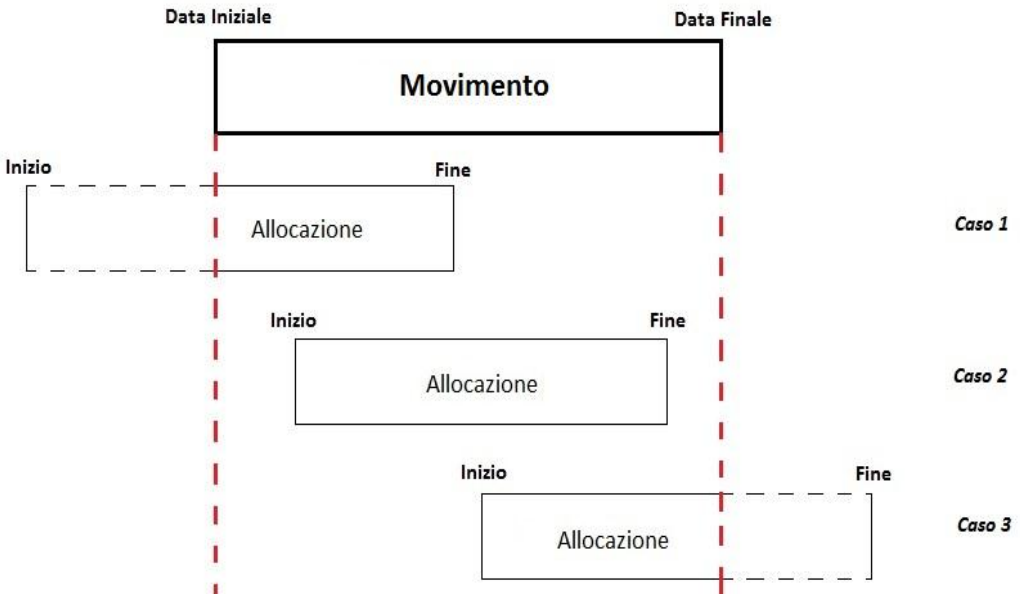

Figura 3.6 Schema del vincolo riguardante le ripartizioni di un movimento

L'importo delle ripartizioni varia in base al peso e al caso in cui ci troviamo:

- **caso 1** (in figura 3.6):

  Il movimento inizia durante il periodo di validità dell'allocazione e finisce dopo, quindi viene assegnato all'allocazione un costo o un ricavo che ha come data iniziale quella del movimento e data finale quella dell'allocazione, con un importo proporzionato al peso e alla durata in ore;

- **caso2** (in figura 3.6):

  Il movimento inizia prima dell'allocazione e finisce dopo, quindi viene assegnato all'allocazione un costo o un ricavo che ha come data iniziale e data finale quelle dell'allocazione, con importo proporzionato al peso e alla durata in ore;

- **caso 3** (in figura 3.6):

  Il movimento inizia prima del periodo di validità dell'allocazione e finisce prima, quindi viene assegnato all'allocazione un costo o un ricavo che ha come data iniziale quella dell'allocazione e data finale quella del movimento, con un importo proporzionato al peso e alla durata in ore;

- **caso 4**:

  Il movimento inizia e finisce nello stesso momento dell'allocazione, quindi viene assegnato all'allocazione un costo o un ricavo che ha data iniziale e finale del movimento, con un importo proporzionato al peso e alla durata in ore.

Capitolo 4

Web Layer

In questo capito parleremo delle scelte di progettazione del lato web del progetto A.R.E.M e di come esso è stato strutturato e sviluppato.

Prima di procedere nel capitolo è bene chiarire la scelta di un software web based a dispetto di uno stand alone. Sviluppare un'applicazione basata sul web dà la possibilità, a chi la commissiona, di poter interagire con il software in qualunque punto del mondo esso si trovi, con la sola necessità di avere a disposizione una connessione ad internet. Per far ciò abbiamo creato un Dynamic Web Project in Java denominato AremWEB, legato al livello DAL presentato nel capitolo precedente, ed inglobato nella cartella “lib” di WEB-INF tutte le librerie che ci serviranno per il suo sviluppo.

Di seguito sono elencati i paragrafi presenti in questo capitolo:

- 4.1 Class Diagram;
- 4.2 Apache Tiles:
- 4.2.1 Struttura e configurazione;
- 4.3 Spring MVC:
- 4.3.1 Il pattern Model-View-Controller;
- 4.3.2 Configurazione Spring MVC;
- 4.3.3 Integrazione Apache Tiles;
- 4.4 Struttura A.R.E.M. Web:
- 4.4.1 Jboss;
- 4.4.2 HTML5 e CSS3;
- 4.4.3 JSP e JSTL;
- 4.4.4 View di A.R.E.M. Web;
- 4.4.5 Controller di A.R.E.M. Web;
- 4.5 Punti critici e soluzioni.

4.1 Class Diagram

Il livello web del progetto presenta il seguente diagramma, nella quale sono evidenziate le pagine web, *Boundary*, ed i controllori, *Controller*, che le richiamano.

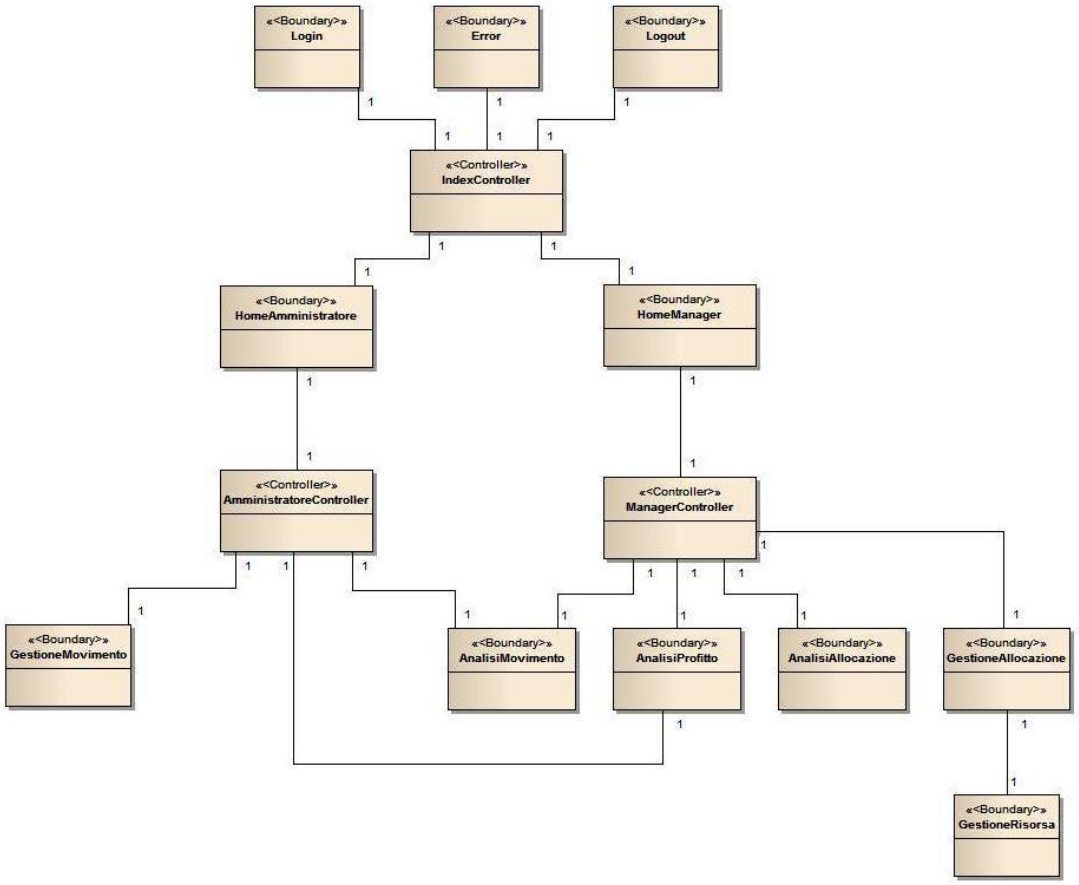

Figura 4.1 Class Diagram A.R.E.M. Web

Nei paragrafi successivi sarà più chiaro il funzionamento del livello web e la sua strutturazione, per ora ci limitiamo a dire che il web layer è strutturato in modo da rimandare al browser la pagina corretta a seguito di una particolare richiesta.

Per soddisfare le richieste del committente e per poter rendere il software riutilizzabile e migliorabile, ci siamo avvalsi di alcune tra le migliori tecnologie, ad oggi, adoperabili in conformità con Java Enterpriese Edition.

Gli strumenti che abbiamo utilizzato per sviluppare il web layer sono:

- Apache Tiles;
- Spring MVC;
- Jboss;
- Html5;
- Css3;
- JSP;
- JSTL;

Come abbiamo appena detto, la piattaforma di sviluppo scelta per il nostro progetto è Java Enterprise Edition.

La Java Platform Enterprise Edition è una piattaforma software di programmazione Java. Essa fornisce API ed un ambiente di runtime per lo sviluppo e l'esecuzione di software per le imprese, compresi i servizi di rete e web, e di altre grandi applicazioni di rete a più livelli, scalabile, affidabile e sicuro.

La Java EE estende la Java 2 Platform, Standard Edition (Java SE). La piattaforma incorpora un design basato in gran parte su componenti modulari in esecuzione su un server di applicazioni. Il software per Java EE è principalmente sviluppato in linguaggio di programmazione Java.

4.2 Apache Tiles

Apache Tiles è un templating framework costruito per semplificare lo sviluppo di interfacce utente delle applicazioni web. Tale strumento offre la possibilità di creare un template standard per tutte le pagine del proprio progetto web e, se necessario, personalizzare tutto o parte del template di alcune pagine particolari.

4.2.1 Struttura e configurazione

Apache Tiles è organizzato in una serie di pagine JSP, della quale parleremo in dettaglio nei prossimi paragrafi, che andranno a comporre il template base con una struttura del tipo in Figura 4.2:

Figura 4.2 Template base di Apache Tiles

Ogni sezione del template è, come detto in precedenza, una pagina JSP che verrà richiamata all'interno di un'altra pagina JSP "contenitore". Questo contenitore è il template delle pagine del nostro progetto.

Definita la struttura di Tiles, passiamo alla configurazione di questo

strumento.

Innanzitutto nel nostro progetto andiamo a caricare le librerie necessarie al funzionamento di Tiles:

com.springsource.org.apache.tiles.core-2.1.2.osgi.jar

com.springsource.org.apache.tiles.core-sources-2.1.2.osgi.jar

com.springsource.org.apache.tiles.jsp-2.1.2.jar

com.springsource.org.apache.tiles.jsp-sources-2.1.2.jar

com.springsource.org.apache.tiles.servlet-2.1.2.jar

com.springsource.org.apache.tiles.servlet-sources-2.1.2.jar

com.springsource.org.apache.tiles-2.1.2.osgi.jar

com.springsource.org.apache.tiles-sources-2.1.2.osgi.jar

Successivamente creiamo un file XML all'interno della cartella WEB-INF del nostro progetto, ad esempio tiles.xml, dove andremo a specificare quale pagina JSP fungerà da template base, il "contenitore" definito in precedenza, quali JSP lo compongono e quali pagine del nostro progetto lo utilizzeranno. Di seguito vi è uno stralcio del file configurazione creato per AremWEB:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE tiles-definitions PUBLIC
    "-//Apache Software Foundation//DTD Tiles Configuration 2.0//EN"
    "http://tiles.apache.org/dtds/tiles-config_2_0.dtd">
<tiles-definitions>
    <definition name="baseLayout" template="/WEB-INF/jsp/layout.jsp">
        <put-attribute name="title" value="" />
        <put-attribute name="header" value="/WEB-INF/jsp/header.jsp"
/>
        <put-attribute name="menu" value="" />
        <put-attribute name="body" value="" />
        <put-attribute name="footer" value="/WEB-INF/jsp/footer.jsp"
/>
```

```
</definition>

<definition name="index" extends="baseLayout">
    <put-attribute name="title" value="Homepage" />
    <put-attribute name="body" value="/WEB-INF/view/index.jsp"
/>
</definition>
```

La parte interessante del codice di configurazione è quella dei tag <definition> e <put-attribute>. Nel primo andiamo a dichiarare quale pagina JSP sarà il template base e quali pagine del progetto utilizzeranno questo template. Nei tag <put-attribute> invece andiamo a dichiarare con quali JSP è composto il template.

Nel caso specifico del nostro progetto, per il baseLayout abbiamo deciso di utilizzare solo header e footer per comporre il template base. Tale scelta è dovuta alla presenza di due tipologie di utenze che avranno permessi di accesso diversi al web layer e quindi avranno dei menù differenti per le operazioni a loro concesse.

Il body ed il title non vengono inizializzati in quanto ogni pagina del progetto avrà un proprio body ed un proprio title.

4.3 Spring MVC

L'uso combinato di Spring Core e HTML e/o JSP, ha dato vita al framework Spring MVC, il quale sfrutta le capacità di Spring Core associate al design pattern M-V-C per generare e gestire pagine web basate su HTML o JSP.

Nel prosieguo del capitolo analizzeremo il design pattern MVC, come utilizzare la filosofia di tale pattern con Spring Core e infine come sfruttare le potenzialità di Apache Tiles in Spring MVC.

4.3.1 Il pattern Model-View-Controller

Il pattern MVC viene utilizzato per la suddivisione dell'interfaccia utente e dei gestori di evento in tre elementi, rappresentati in Figura 4.3:

- la parte di Business Logic (Model);
- la parte di Interfaccia Utente (View);
- la parte di Program Progression (Controller).

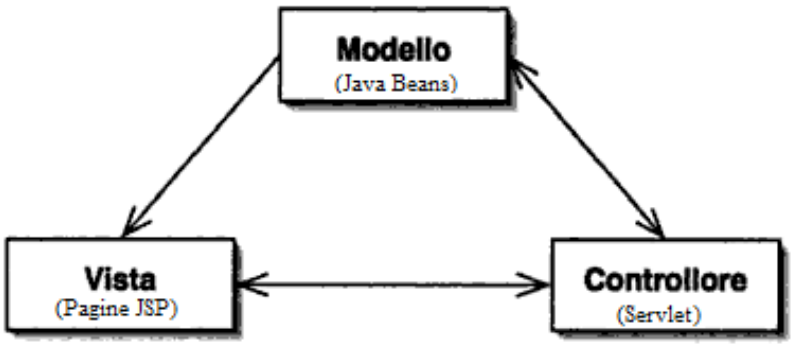

Figura 4.3 Pattern MVC

Ciascun elemento ha uno scopo ben definito:

- **Model**: incapsula lo stato dell'applicazione memorizzando tutti i dati relativi al programma. Interagisce con gli altri elementi per fornire lo

stato del programma. Quindi il *Model* rappresenta lo strato di business dell'applicazione ed è, solitamente, implementato attraverso l'uso dei *JavaBeans*, che in Spring prendono il nome di *Beans*.

- **View**: la vista interpreta il modello al fine di fornire una rappresentazione all'utente. Prende in input gli eventi generati dall'interazione dell'utente con l'interfaccia che passa al controllore. Rappresenta l'interfaccia utente gestita attraverso l'utilizzo delle pagine HTML e/o JSP.
- **Controller**: determina il comportamento dell'applicazione implementando le logiche di controllo. Esegue le modifiche al modello in funzione dell'operatività dell'utente. Eventualmente seleziona quale vista presentare all'utente, in funzione delle sue scelte. Favorisce lo scambio di dati tra Model e View. Il componente Web più indicato a svolgere tale funzione è sicuramente la servlet che, interagendo direttamente con il Model, è in grado di aggiornare una o più View corrispondenti. Lo schema generale del pattern MVC è presentato in Figura 4.4.

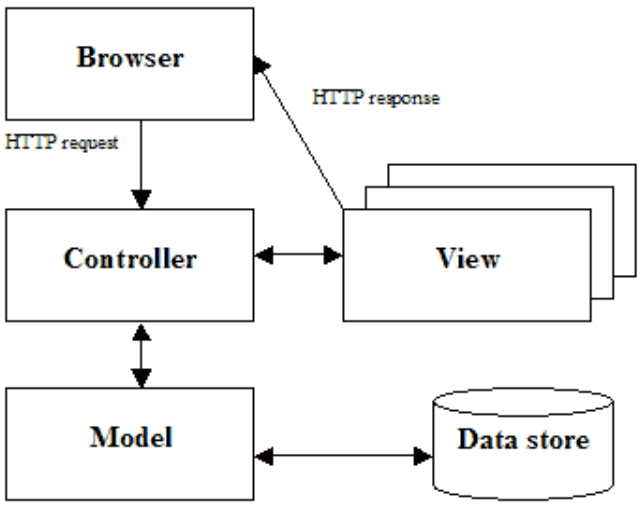

Figura 4. 4 Schema del pattern MVC

Quando si utilizza questo tipo di approccio è molto importante cercare di tenere i tre strati quanto più separati ed indipendenti possibile.

Agendo in tal modo, infatti, si renderà di gran lunga più facile ed agevole la manutenzione del codice. Se, per esempio, si decidesse di apportare delle modifiche ad un'applicazione andando a modificare la visualizzazione dei dati rispetto alla precedente implementazione, sarà sufficiente modificare soltanto il presentation layer (View) senza intaccare gli altri due componenti del pattern MVC. Ancor meglio, sarà possibile aggiungere ulteriori viste a quelle già implementate sfruttando al meglio i concetti della programmazione orientata agli oggetti.

I vantaggi offerti dal pattern MVC sono numerosi:

- Consente di separare la parte di business rules dalla parte di presentation, favorendo così la modularità dei componenti;
- Chi sviluppa il front-end (View), non ha bisogno di sapere come queste pagine verranno popolate (mediante il Model);
- È possibile, mediante il Controller, gestire la visualizzazione dei dati in maniera dinamica. View diverse per utenti diversi, più View per lo stesso utente;
- E' possibile centralizzare la sicurezza dell'applicazione all'interno del Controller. Con il pattern MVC, l'interfaccia verso l'utente viene fatta passare solo attraverso l'oggetto controller, questo rappresenta quindi l'unico punto d'accesso e come tale l'unico punto in cui vanno effettuati tutti i controlli di sicurezza.

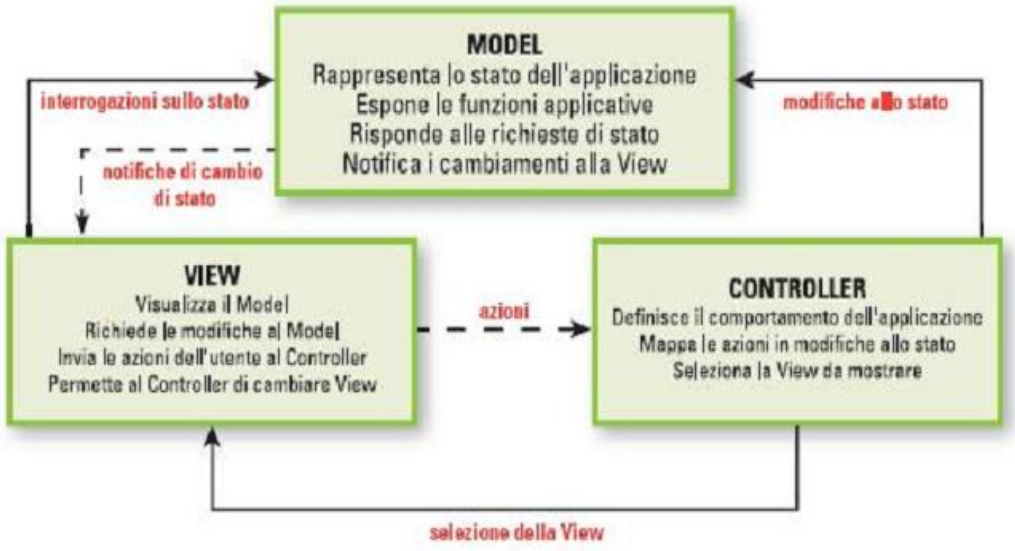

Figura 4.5 Pattern MVC e responsabilità dei tre componenti

4.3.2 Configurazione Spring MVC

Prima di passare alla configurazione del framework è bene introdurre un aspetto fondamentale del funzionamento di Spring MVC, ovvero l'uso del DispatcherServlet.

Spring MVC è stato progettato per lavorare intorno ad un DispatcherServlet il quale gestisce tutte le richieste e le risposte HTTP. Il suo funzionamento è illustrato in modo molto semplice nella Figura 4.6:

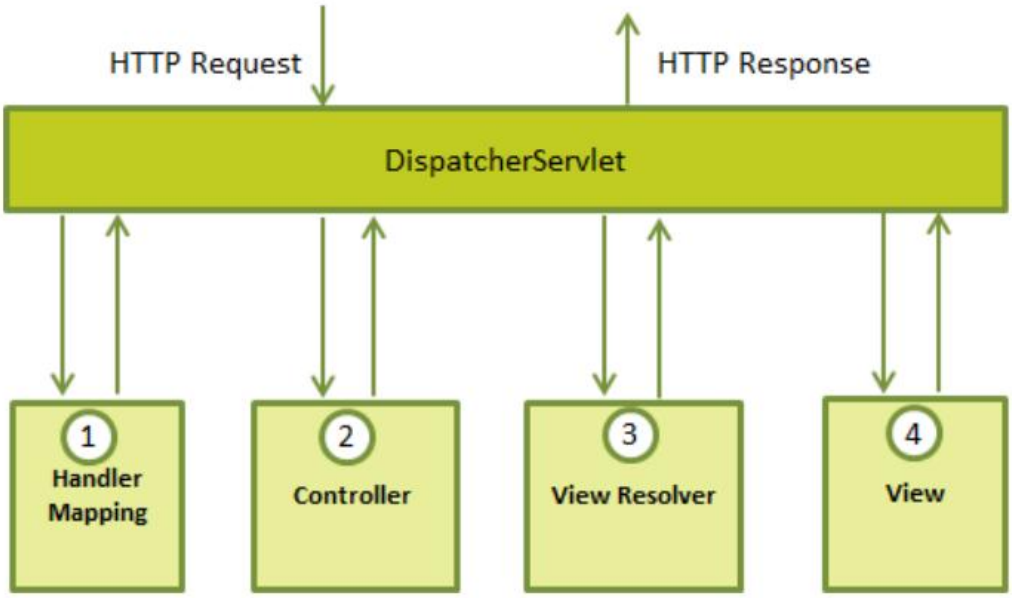

Figura 4.6 Funzionamento DispatcherServlet

Nella figura vengono illustrate, nell'ordine in cui vengono eseguite, le funzioni che il DispatcherServlet svolge nel momento in cui arriva una richiesta HTTP:

1. Dopo aver ricevuto una HTTP Request, il DispatcherServlet consulta la HandlerMapping per chiamare il Controller appropriato;
2. Il Controller cattura la richiesta e richiama i metodi di servizio appropriati in base alla ricezione di una HTTP Request di tipo GET o POST. Il metodo di servizio imposta i dati del Model sulla base della logica di business definita e restituisce il nome logico della View, e il Model, al DispatcherServlet;
3. Il DispatcherServlet invia il nome logico della View al ViewResolver il quale cercherà la pagina corrispondente e se esiste la invia al DispatcherServlet. Qualora la pagina non fosse presente, verrà visualizzata la pagina di errore 404 del protocollo HTTP;
4. Una volta ottenuta la pagina appropriata, il DispatcherServlet passa i dati del Model a tale View, la pagina viene popolata con tali dati e

restituita al DispatcherServlet, il quale inoltra una HTTP Response per poter visualizzare nel browser il risultato della HTTP Request.

I componenti citati, HandlerMapping, Controller, ViewResolver, fanno parte del WebApplicationContext, il quale è un'estensione dell'ApplicationContext con alcune caratteristiche extra, necessarie al funzionamento di un'applicazione web.

Chiarito il funzionamento del DispatcherServlet, passiamo alla sua configurazione.

Per configurare il DispatcherServlet è necessario creare un file XML nel quale andremo a specificare il nome del DispatcherServlet e quali tipo di pagine dovrà mappare. Tutto questo lo andiamo a dichiarare in un file web.xml, che andremo a posizionare nella cartella WEB-INF del nostro progetto. Di seguito riportiamo alcune righe di codice, quelle più rilevanti, del file web.xml creato per AremWEB:

- In questi tag andiamo a specificare il nome del DispatcherServlet e lo abilitiamo all'avvio dell'applicazione (load-on-startup).

  ```
  <servlet>
  <servlet-name>dispacciatore</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet
  </servlet-class>
  <load-on-startup>1</load-on-startup>
  </servlet>
  ```

- Qui andiamo a specificare quali tipi di pagine il nostro DispatcherServlet dovrà mappare, nel nostro caso abbiamo scelto di far mappare tutte le pagine del progetto "/"

  ```
  <servlet-mapping>
  <servlet-name>dispacciatore</servlet-name>
  <url-pattern>/</url-pattern>
  </servlet-mapping>
  ```

- Qui andiamo a dichiarare il contesto del nostro progetto, ovvero dove

sono presenti tutte le informazioni necessarie per recuperare i dati necessari al Model, i beans ed i parametri di connessione al database

<context-param>
<param-name>contextConfigLocation</param-name>
<param-value>/WEB-INF/dataSource-context.xml</param-value>
</context-param>

Creato il file web.xml, passiamo alla creazione ed alla configurazione del DispatcherServlet. Per creare il DispatcherServlet è necessario creare un file XML del tipo nome-servlet.xml, dove il "nome" è a scelta, mentre il resto è obbligatorio. Nel nostro progetto abbiamo scelto di creare il DispatcherServlet come dispacciatore-servlet.xml e guardando ai punti sopra citati nella configurazione del file web.xml, il nome "dispacciatore" compare tra i tag <servlet-name>, in questo modo abbiamo dichiarato quale DispatcherServlet stiamo utilizzando. Anche il DispatcherServlet và posizionato nella cartella WEB-INF.

A questo punto passiamo alla configurazione del DispatcherServlet mostrando alcune righe di codice del file dispacciatore-servlet.xml di AremWEB:

- In questi tag andiamo ad abilitare l'uso delle annotatione di Spring, e specifichiamo in quale package si trovano le classi java che faranno da Controller per il nostro progetto.

  <context:annotation-config />
  <context:component-scan base-package="controller" />

- Qui andiamo ad abilitare le annotation di Spring MVC ed indichiamo il path dove è possibile recuperare i file di tipo immagine, css e JavaScript, fornendo a tale path un alias in modo da evitare di riscriverlo per intero ogni volta che vogliamo recuperare ed associare un file particolare alle nostre View.

  <mvc:annotation-driven />
  *<mvc:resources location="/resources/" mapping="/resources/**" />*

- Tra questi tag settiamo la parte relativa al ViewResolver. Da notare che esso è in realtà un bean, proprio come avevamo già detto ad inizio capitolo, per il quale vengono dichiarate due property importanti quali il percorso dove andare a cercare le View, ed il tipo di View da restituire. Con Spring MVC è possibile utilizzare solo pagine di tipo JSP.

  <bean id="viewResolver"
  class="org.springframework.web.servlet.view.UrlBasedViewResolver"
  >
  <property name="viewClass"
  value="org.springframework.web.servlet.view.JstlView" />
  <property name="prefix" value="/WEB-INF/view/" />
  <property name="suffix" value=".jsp" />
  </bean >

Definito il DispatcherServlet non ci resta che settare il contesto per il corretto funzionamento del nostro progetto. In realtà tale contesto lo abbiamo già creato, si tratta dello stesso file di contesto del livello DAL, quindi non dobbiamo fare altro che ricrearlo nella cartella WEB-INF di AremWEB, con la dovuta accortezza nel settare adeguatamente il nome del file XML il quale dovrà coincidere con quello presente tra i tag <param-name> di <context-param> presenti nel file web.xml e che nel nostro caso è dataSource-context.xml.

A questo punto abbiamo creato e configurato a dovere i file necessari al corretto funzionamento della nostra applicazione web, AremWEB, usando il framework Spring MVC.

4.3.3 Integrazione Apache Tiles

Nel paragrafo 4.2 abbiamo parlato di Apache Tiles, adesso vedremo come integrare le funzionalità di tale templating framework con Spring MVC.

Il passo necessario all'integrazione di Tiles con Spring MVC è la corretta configurazione del ViewResolver.

Nel paragrafo precedente abbiamo visto come settare tale componente in modo da recuperare pagine JSP da un path ben preciso. Adesso andremo a modificare questo ViewResolver in modo da poter sfruttare Apache Tiles:

- Rispetto al precedente abbiamo dovuto cambiare la stringa del value del viewClass.

  ```
  <bean id="viewResolver"
  class="org.springframework.web.servlet.view.UrlBasedViewResolver"
  >
  <property name="viewClass"
  value="org.springframework.web.servlet.view.tiles2.TilesView" />
  </bean>
  ```

- Abbiamo aggiunto questo bean che collega Spring con Tiles. Quando il DispatcherServlet invoca il ViewResolver questi controlla il file tiles.xml e restituisce la pagina richiesta.

  ```
  <bean id="tilesConfigurer"
  class="org.springframework.web.servlet.view.tiles2.TilesConfigurer">
  <property name="definitions">
  <list>
  <value>/WEB-INF/tiles.xml</value>
  </list>
  </property>
  </bean>
  ```

A questo punto possiamo combinare il lavoro di Apache Tiles e Spring MVC ed iniziare a creare il web layer del progetto A.R.E.M.

4.4 Struttura A.R.E.M. Web

In questo paragrafo andiamo ad illustrare nel dettaglio la struttura ed il funzionamento del livello web del progetto A.R.E.M.

Prima di addentrarci nel dettaglio soffermiamoci un attimo sulle tecnologie usate per sviluppare il web layer.

Per adesso abbiamo parlato solo di due tecnologie utilizzate, Apache Tiles e Spring MVC, ma è d'obbligo descrivere anche gli altri strumenti utilizzati, per i quali prenderemo in prestito le definizioni di Wikipedia.

4.4.1 JBoss

Nella costruzione del progetto A.R.E.M. abbiamo scelto di utilizzare come application server JBoss AS 7.

In informatica JBoss è un application server open source che implementa l'intera suite di servizi Java EE. Essendo basato su Java, JBoss è un application server multipiattaforma, utilizzabile su qualsiasi sistema operativo che supporti Java. Il gruppo di sviluppo di JBoss era originariamente assunto presso la "JBoss Inc." fondata da Marc Fleury. Nell'aprile del 2006, Red Hat ha comprato JBoss per 420 milioni di dollari. Come progetto open source, JBoss è supportato e migliorato da una enorme rete di sviluppatori. JBoss è stato pioniere nel mondo dell'open source professionale, dove gli sviluppatori iniziali del progetto creano un prodotto e offrono i loro servizi. Legati a JBoss esistono inoltre molti altri progetti, tra cui JBoss AS, Hibernate, Tomcat, JBoss ESB, jBPM, JBoss Rules (ex Drools), JBoss Cache, JGroups, JBoss Portal, SEAM, JBoss Transaction, JBoss Messaging tutti sotto il marchio della JBoss Enterprise Middleware Suite (JEMS).

Le caratteristiche di JBoss AS 7 sono:

- **Velocità** - il processo di boot è molto ben ottimizzato: i servizi sono avviati in modo concorrente per ridurre notevolmente i tempi di attesa e

per sfruttare al meglio le caratteristiche multi-core ormai presenti da tempo nei processori. I servizi non critici vengono attivati in modalità lazy, al loro primo utilizzo.

- **Modulare** - i classloaders gerarchici sono spesso problematici, e possono causare il fallimento di deploy e in generale comportamenti inaspettati. **JBoss AS 7** abbandona questa modalità di tipo parent delegation model, per passare invece ad una struttura modulare. I moduli JBoss forniscono quindi una vero isolamento delle applicazioni, nascondono le classi che fanno parte dell'implementazione del server e caricano esclusivamente le classi di cui necessita la nostra applicazione. I moduli realizzati come package di classi rimangono pertanto isolati se non esplicitamente indicati come dipendenze verso altri moduli.
- **Leggerezza - è** stato implementato un approccio alla gestione della memoria "aggressivo", e al fine di minimizzare le pause del **garbage collector** vengono caricati soltanto i jar di cui si ha bisogno. Tutto questo mantiene la quantità della memoria utilizzata eccezionalmente ridotta.
- **Amministrazione** - l'interfaccia di amministrazione è semplice ed intuitiva, focalizzata sull'utente.
- **Portabilità -** quanto alla portabilità: **JBoss AS 7** raggiunge la Java EE 6 Full Profile certified.

4.4.2 HTML5 e CSS3

HTML e CSS sono due strumenti fondamentali per costruire delle pagine web in modo rapido, chiaro e stilisticamente gradevole.

HTML è un linguaggio di markup per la progettazione delle pagine web attualmente in fase di definizione presso il World Wide Web Consortium (W3C).

Le novità introdotte dall'HTML5 rispetto all'HTML4 sono finalizzate

soprattutto a migliorare il disaccoppiamento tra struttura, definita dal markup, caratteristiche di resa (tipo di carattere, colori, eccetera), definite dalle direttive di stile, e contenuti di una pagina web, definiti dal testo vero e proprio.

Inoltre HTML5 prevede il supporto per la memorizzazione locale di grosse quantità di dati scaricati dal web browser, per consentire l'utilizzo di applicazioni basate su web (come per esempio le caselle di posta di Google o altri servizi analoghi) anche in assenza di collegamento a Internet.

In particolare:

- vengono rese più stringenti le regole per la strutturazione del testo in capitoli, paragrafi e sezioni;
- vengono introdotti elementi di controllo per i menu di navigazione;
- vengono migliorati ed estesi gli elementi di controllo per i moduli elettronici;
- vengono introdotti elementi specifici per il controllo di contenuti multimediali (tag <video> e <audio>);
- vengono deprecati o eliminati alcuni elementi che hanno dimostrato scarso o nessun utilizzo effettivo;
- vengono estesi a tutti i tag una serie di attributi, specialmente quelli finalizzati all'accessibilità, finora previsti solo per alcuni tag;
- viene supportato Canvas che permette di utilizzare JavaScript per creare animazioni e grafica bitmap;
- introduzione della geolocalizzazione, dovuta ad una forte espansione di sistemi operativi mobili (quali Android e iOS, tra i più diffusi);
- sistema alternativo ai normali cookie, chiamato Web Storage, più efficiente, il quale consente un notevole risparmio di banda;
- standardizzazione di programmi JavaScript, chiamati Web Workers e possibilità di utilizzare alcuni siti offline;
- sostituzione del lungo e complesso doctype, con uno semplice.

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

CSS3 presenta tre caratteristiche fondamentali:

- Lascia inalterato il core del linguaggio: tutto quello che è valido e funzionante dei CSS 2 continua ad esserlo nei CSS3;
- Aggiunge un gran numero di nuove proprietà;
- E' organizzato in moduli. Piuttosto che riproporre un'unica specifica, il W3C ha organizzato la specifica in moduli, ciascuno dei quali copre una determinata area dei CSS.

I contesti in cui CSS3 rivela già oggi tutte le sue potenzialità sono:

- I selettori;
- Le nuove proprietà e i nuovi metodi per la definizione del colore**;**
- Le nuove proprietà dedicate alla gestione di bordi e sfondi**;**
- Funzionalità legate al testo e ai font come la possibilità di usare caratteri tipografici non presenti sul computer dell'utente (web fonts con @-fontface);
- Nuovi modi per impostare il layout (layout multi-colonna, flexible box model);
- La possibilità di servire fogli di stile ad hoc in base alle caratteristiche dei dispositivi (media queries);
- Metodi e tecniche per dare dinamicità alla pagina (transizioni**,** trasformazioni**,** animazioni).

4.4.3 JSP e JSTL

JavaServer Pages, di solito indicato con l'acronimo JSP è una tecnologia di programmazione Web in Java per lo sviluppo di applicazioni Web che forniscono contenuti dinamici in formato HTML o XML.

Si basa su un insieme di speciali tag con cui possono essere invocate funzioni predefinite o codice Java (JSTL). In aggiunta, permette di creare librerie di nuovi tag che estendono l'insieme dei tag standard (JSP Custom Tag Library). Le librerie di tag JSP si possono considerare estensioni indipendenti dalla piattaforma delle funzionalità di un Web server.

Nel contesto della piattaforma Java, la tecnologia JSP è correlata con quella delle servlet: all'atto della prima invocazione, le pagine JSP vengono infatti tradotte automaticamente da un compilatore JSP in servlet. Una pagina JSP può quindi essere vista come una rappresentazione ad alto livello di un servlet. Per via di questa dipendenza concettuale, anche l'uso della tecnologia JSP richiede la presenza, sul Web server, di un servlet container, oltre che di un server specifico JSP detto motore JSP (che include il compilatore JSP); in genere, servlet container e motore JSP sono integrati in un unico prodotto (per esempio: Tomcat svolge entrambe le funzioni).

JSP è una tecnologia alternativa rispetto a numerosi altri approcci alla generazione di pagine Web dinamiche, per esempio PHP, o ASP o la più tradizionale CGI. Differisce da queste tecnologie non tanto per il tipo di contenuti dinamici che si possono produrre, quanto per l'architettura interna del software che costituisce l'applicazione Web.

Una pagina JSP è un documento di testo, scritto con una sintassi specifica, che rappresenta una pagina Web di contenuto parzialmente o totalmente dinamico. Elaborando la pagina JSP, il motore JSP produce dinamicamente la pagina HTML finale che verrà presentata al web browser dell'utente. La pagina JSP può contenere tre tipi di elementi, a cui

corrispondono tre diversi modi di elaborazione: contenuti statici, direttive e script.

JavaServer Pages Standard Tag Library (JSTL) è una libreria inclusa come componente della piattaforma software di sviluppo per applicazioni Web Java EE. È un'estensione di JSP ed incorpora un insieme di tag HTML definiti tramite file XML e programmati in linguaggio Java.

Questa libreria è stata rilasciata da società di sviluppo software quali la Sun Microsystems, utilizzabili per la creazione di JavaServer Pages. In alternativa alle librerie di tag standard si possono creare librerie di tag personalizzati, chiamate Custom Tag Library.

Il vantaggio della creazione di tag standard, analogamente a quanto avviene nelle funzioni di libreria, è quello di definire dei comportamenti univoci dello stesso tag in contesti diversi eliminando la necessità di definire degli scriptlet (applet fatte con un linguaggio di scripting) all'interno delle pagine, ottenendo anche l'ulteriore vantaggio di distinguere il compito dello sviluppatore Java da quello del web designer.

La libreria JSTL permette di operare direttamente sulla pagina JSP in diversi modi quali ad esempio: funzionalità base come settare una variabile o stamparne a video il contenuto, ricreare delle strutture di selezione come IF, dei cicli FOR, formattare secondo i gusti desiderati l'output di date o caratteri speciali, interrogare il database tramite linguaggio SQL oppure integrare del codice XML.

4.4.4 View di A.R.E.M. Web

Definite e descritte le tecnologie che abbiamo utilizzato per la realizzazione del web layer di progetto A.R.E.M., possiamo iniziare a descrivere le View più importanti della web application, demandando al paragrafo successivo i metodi di accesso a tali View tramite Controller.

Ovviamente il punto di partenza, osservando attentamente anche il class

diagram ad inizio capitolo, è la pagina di index.

Essendo un'applicazione di back-end, quindi utilizzabile solo da chi ha i privilegi dovuti, la pagina di index che compare è una semplice pagina di login:

Figura 4.7 View di Login

Tramite una form, l'utente accederà al sito e potrà sfruttare gli strumenti messi a disposizione.

Come visto nello Use Case Diagram al capitolo due, le tipologie di utenze che possono accedere agli strumenti di progetto A.R.E.M. sono Amministratore e Manager, i quali avranno dei privilegi ovviamente differenti.

Una volta effettuato il login, si viene reindirizzati alla pagina relativa al tipo di utente loggato dalla quale sarà possibile accedere agli strumenti di cui si ha il privilegio di disporre. Nel prosieguo analizzeremo dapprima le View di competenza del Manager e successivamente quelle dell'Amministratore.

Dalla homepage del Manager è possibile accedere ai seguenti strumenti:

- Gestione Allocazioni;
- Analisi Allocazioni;
- Analisi Movimenti;
- Analisi Profitto;
- Logout.

Gestione Allocazioni.

Il Manager ha la possibilità di creare e/o modificare le allocazioni tra le Risorse tramite una semplice ed intuitiva View.

Come già accennato nel capitolo due, per memorizzare la struttura dati abbiamo utilizzato delle strutture ad albero, quindi ci è sembrato logico mostrare a video tale albero, il quale prende il nome di *albero delle allocazioni*, esso raccoglie tutte le allocazioni di uno specifico arco temporale e rispecchia la struttura dell'azienda. Quando il periodo cambia, anche l'albero cambia, ed avremo quindi tanti alberi quanti sono gli archi temporali.

Per poter memorizzare questi cambiamenti abbiamo pensato di utilizzare una barra di navigazione per dare la possibilità all'utente di scegliere il periodo d'interesse.

Questa barra, visibile in Figura 4.8, prende il nome di *Timeline* ed associa ad ogni periodo l'albero di allocazioni corrispondente.

Figura 4.8 Bozza Timeline

Quando si accede alla pagina di Gestione Allocazione la prima cosa che viene visualizzata a video è questa Timeline:

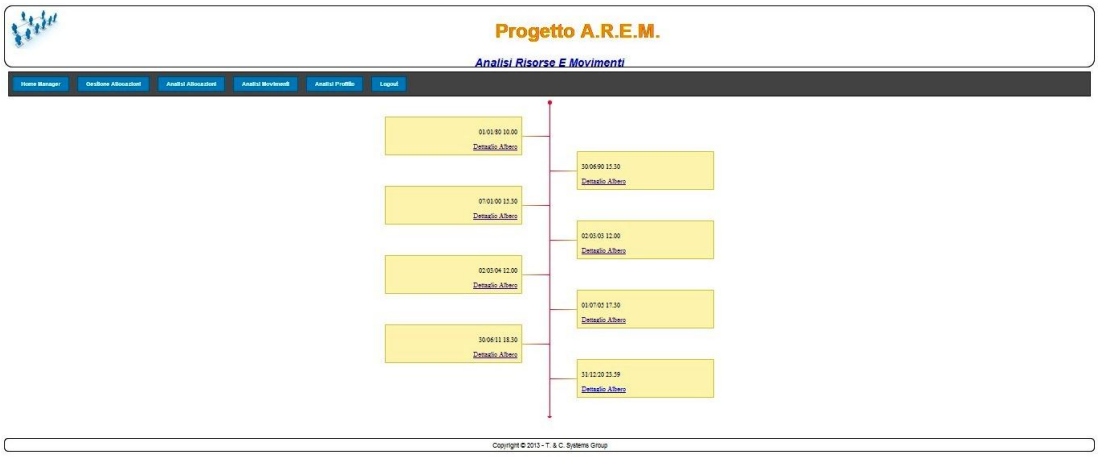

Figura 4.9 Timeline alberi delle allocazioni

Compongono questa Timeline tutti gli alberi delle allocazioni che sono presenti nello stesso periodo, il concetto sarà più chiaro guardando le prossime View.

Prima di andare avanti è doveroso precisare che in fase di setup del progetto verrà creata un'allocazione primaria tra il committente, che viene considerato a sua volta una Risorsa, e l'ambiente esterno che chiameremo Home, anch'esso comunque una Risorsa. Tale setup è necessario affinché l'applicazione possa avere un punto di partenza, infatti in fase di creazione dell'allocazione verranno settate la data di inizio e fine allocazione, e tutte le allocazioni successive non potranno essere antecedenti o successive a questa.

Cliccando su uno dei tag della Timeline si avrà accesso all'albero delle allocazioni:

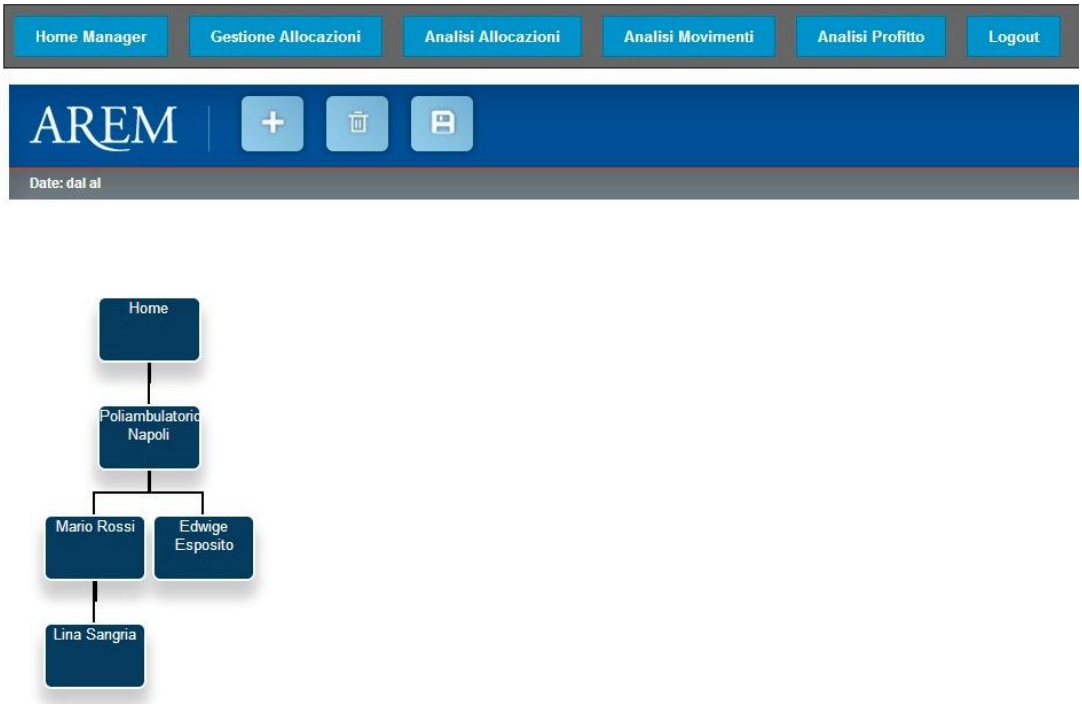

Figura 4.10 Albero delle allocazioni valide nel periodo dal-al

Risalta subito all'occhio la presenza di due date, dal/al, le quali rappresentano l'intervallo di validità dell'albero, ovvero tutte le allocazioni che lo compongono sono valide in questo periodo, e ad ogni mutamento di data corrisponderà un altro albero presente sulla Timeline.

I rettangoli presenti sul desk sono le Risorse, in essi viene evidenziato il nominativo. I riquadri sono draggabili, ovvero è possibile trascinarli ed agganciarli ad altre Risorse in modo da creare una nuova allocazione, oppure trascinarli sul cestino per poterli eliminare dal foglio di lavoro.

Nel momento in cui le Risorse vengono eliminate dal foglio, viene eseguita una operazione di update nel database, dove viene aggiornata l'allocazione in cui è avvenuta la modifica impostando la data di fine a quella corrente, generando quindi un nuovo albero e lasciando inalterato quello precedente, in modo da avere sempre uno storico degli alberi precedenti.

Stesso concetto quando si trascina una Risorsa presente sul foglio sotto un altro rettangolo, oppure quando si crea una nuova Risorsa e la si alloca ad un altra, con la sola differenza che in questo caso viene eseguita una create e

bisognerà impostare a mano le date di inizio e fine allocazione. Tali operazioni vengono fatte solo nel momento in cui si clicca sul tasto Salva.

Abbiamo accennato alla creazione di un Risorsa, questa operazione viene eseguita quando si clicca sull'apposito bottone, per maggiori dettagli vedere la Figura 4.12.

Comparirà un pop-up, Figura 4.11, contente una form di creazione dove andremo a settare il nominativo ed il tipo di risorsa. Se il nominativo è già presente in database, la risorsa verrà recuperata se il tipo di risorsa corrisponde a tale risorsa, altrimenti verrà creata nel database. Se il nominativo è già presente ma il tipo risorsa impostato è differente, la risorsa verrà aggiornata ai dati inseriti nella form.

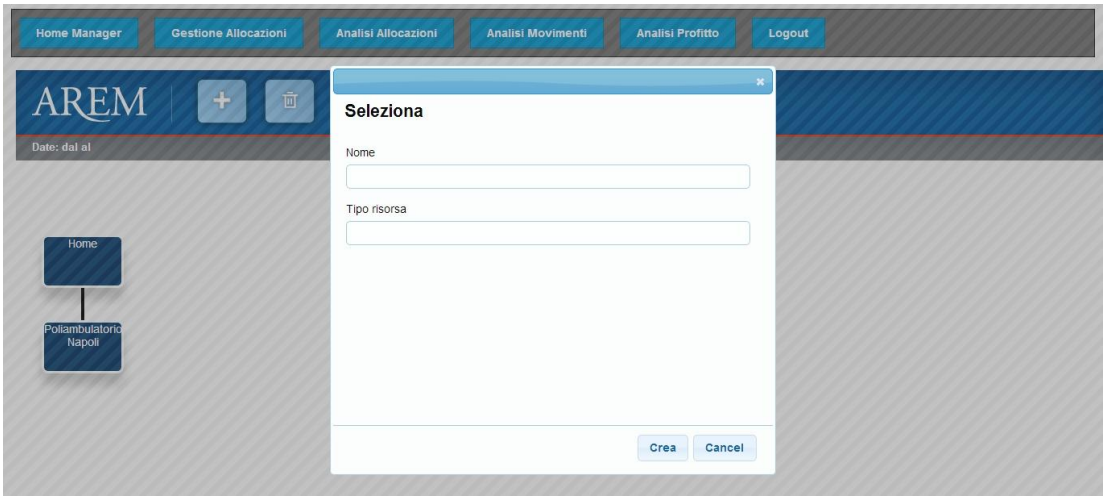

Figura 4.11 Form per la creazione di una Risorsa

Chiusa la form di creazione Risorsa, questa sarà presente sul foglio di lavoro e pronta ad essere allocata.

Prima di chiudere con questa View, c'è da notare la prima risorsa in alto, Home. Come abbiamo detto precedentemente questa è la Risorsa d'ambiente e non è draggabile, quindi non la si potrà cancellare o allocare altrove.

Figura 4.12 Menù strumenti di gestione dell'albero

Analisi Allocazioni.

La View per l'analisi Allocazioni presenta la stessa struttura della gestione, con la sola differenza che una volta avuto accesso alla Timeline e cliccato su uno qualunque dei tag, l'albero potrà solo essere visualizzato e non modificato. Questo ci permetterà di fare una stampa di report senza correre il rischio di modificare inavvertitamente l'albero delle allocazioni.

Analisi Movimenti.

Per ogni Allocazione creata esistono da zero a enne Movimenti di tipo Costo o Ricavo. Tali Movimenti sono gestiti dall'Amministratore, al Manager è concessa solo la loro analisi. L'analisi dei Movimenti si divide in:

- **Analisi Movimenti Risorsa** – si vogliono analizzare i movimenti di una particolare Risorsa in un determinato periodo. Una volta impostati i termini della ricerca compare una Timeline con i risultati e cliccando sui tag sarà possibile vedere in dettaglio l'albero delle Allocazioni con le relative Risorse ad essa allocate e Movimenti che ne scaturiscono.

Figura 4.13 Albero Movimenti Risorsa Root

- **Analisi temporale dei Movimenti** – in quest'analisi si ricercano tutti i Movimenti in un determinato periodo, quindi una volta impostate le date di inizio e fine periodo di ricerca si viene reindirizzati alla Timeline con i risultati della ricerca e cliccando su uno dei tag è possibile vedere l'albero delle Allocazioni con le relative Risorse e Movimenti.

Figura 4.14 Albero di tutti i movimenti di un dato periodo

- **Analisi tipologia Movimenti** – con quest'analisi si vuole ricercare tutti i Movimenti di natura Costo o Ricavo. Basterà scegliere una tipologia di Movimento e la Timeline darà come risultato tutti gli alberi delle Allocazioni dov'è presente tale Movimento, e cliccando su un tag è possibile vedere in dettaglio l'albero.

Figura 4.15 Costi Novembre 2000

Figura 4.16 Ricavi Novembre 2000

Analisi Profitto.

Uno strumento di analisi non richiesto dal committente, ma secondo noi necessario per un software che verrà utilizzato per la gestione e l'analisi, è quello di Analisi del Profitto. Avendo a disposizione dati relativi a Costi e Ricavi, e sfruttando la formula

$$\Pi = RT - CT$$

ovvero Profitto uguale a Ricavo Totale meno Costo Totale, abbiamo realizzato due tipi di ispezioni:

- **Analisi Profitto Temporale** – in questo tipo di analisi basta inserire il periodo che si vuole analizzare e si verrà reindirizzati ad una View dove abbiamo realizzato un grafico a torta, usando i grafici messi a disposizione da Google, dove si evincono i dati di Costi e Ricavi totali nel periodo selezionato.

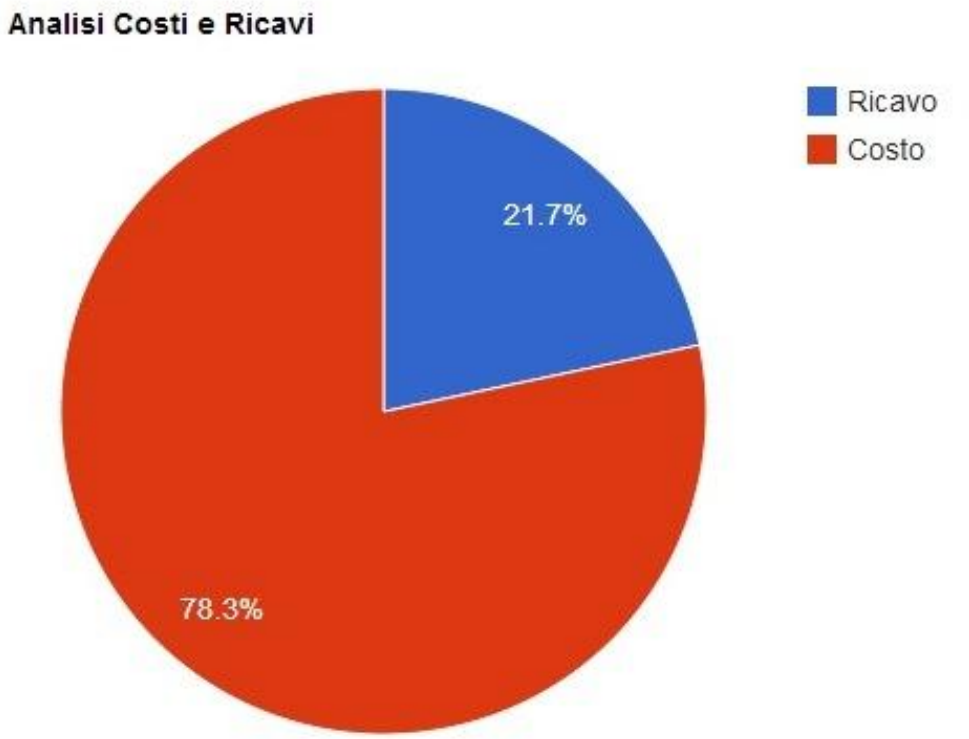

Figura 4.17 Grafico per l'analisi del Profitto in un dato periodo

- **Analisi Profitto Annuale** – con questa interrogazione vogliamo analizzare il profitto dell'ultimo anno di lavoro, partendo dalla data corrente. In questo tipo di analisi diamo la possibilità di poter scegliere se visualizzare i dati su base trimestrale o semestrale.

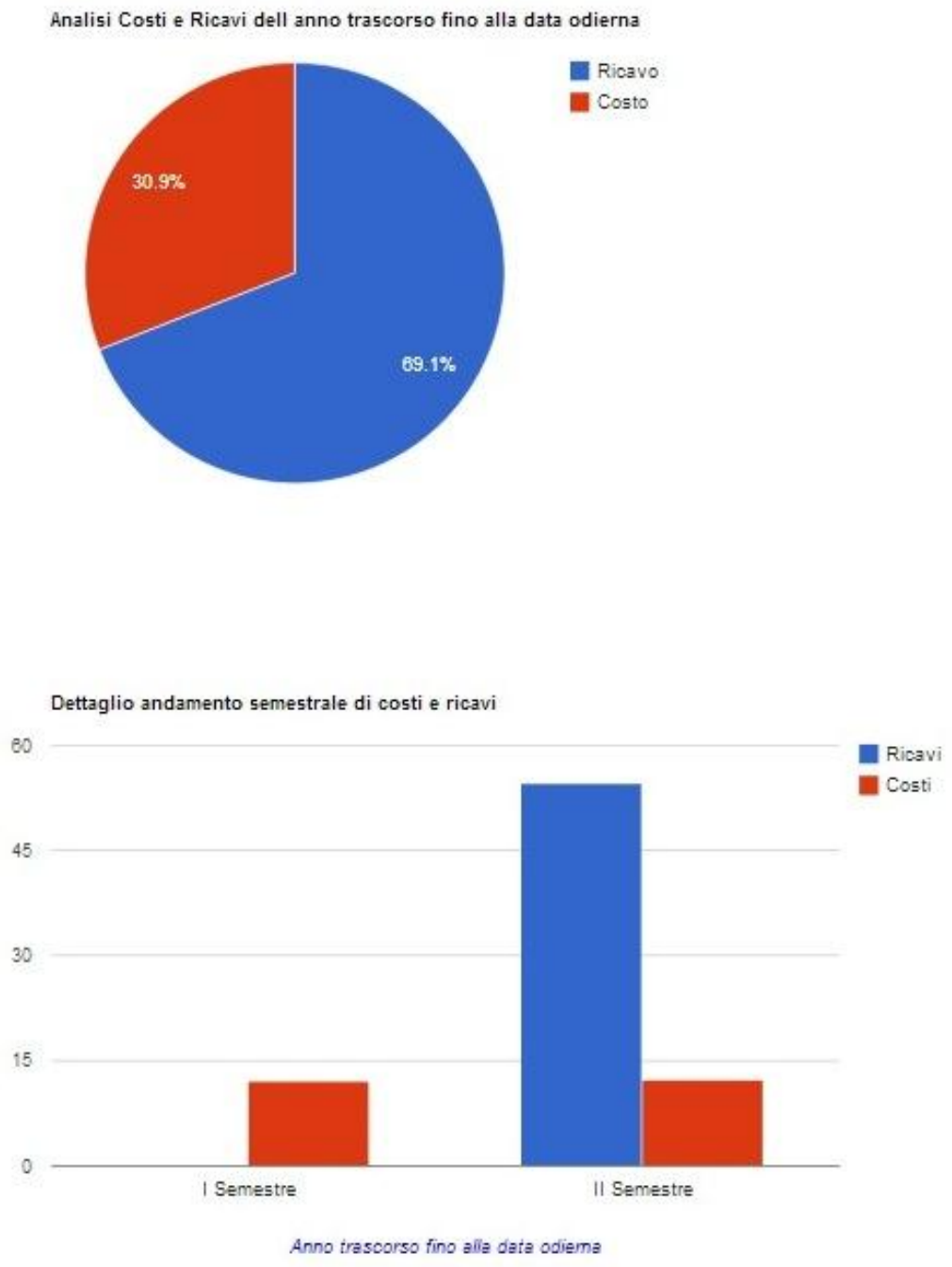

Figura 4.18 Analisi del profitto annuale con dettaglio semestrale

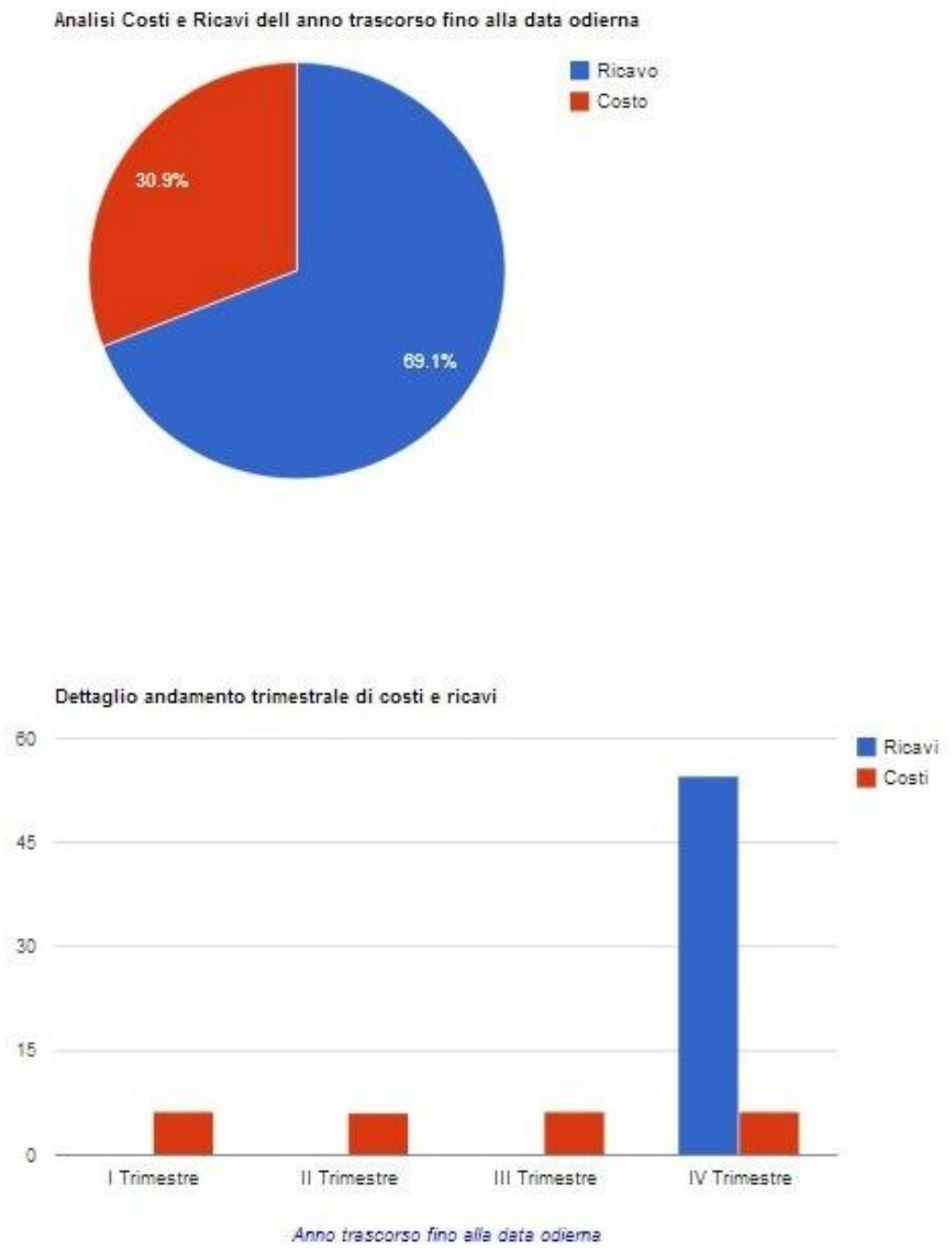

Figura 4.19 Analisi del profitto annuale con dettaglio trimestrale

Logout.

Questa è una semplice pagina che indica che l'utente ha effettuato un logout e la sessione sulla quale stava lavorando è stata distrutta. La distruzione della sessione è necessaria per la sicurezza in quanto chiunque potrebbe sfruttare una sessione ancora aperta. Per ragioni di sicurezza abbiamo anche previsto un controllo sull'utente loggato in ogni pagina.

Terminata l'analisi delle View relative al Manager, passiamo ad analizzare quelle relative all'Amministratore.

I privilegi di Amministratore permettono le seguenti operazioni:

- Gestione Movimenti;
- Analisi Movimenti;
- Analisi Profitto;
- Logout.

Per quanto riguarda l'analisi dei Movimenti, del Profitto ed il Logout, non cambia nulla rispetto a quanto già è stato detto durante la descrizioni delle View del Manager. Quello che c'è di nuovo è la gestione dei Movimenti.

Come già detto in precedenza ad ogni Allocazione corrispondono da zero ad enne Movimenti, quindi per poterli gestire è necessario che l'Allocazione ad esso associato sia già presente in database.

Per poter inserire i Movimenti, basterà cliccare sulla linea di congiunzione di due Risorse, ovvero l’Allocazione, per far apparire una form simile a quella utilizzata per la creazione della Risorsa. Nella form che compare vi sono due campi sulla quale ci dobbiamo soffermare, “Ripartito” e “Peso”.

Il committente ha richiesto, per un analisi scrupolosa, che ci fosse la possibilità di poter ripartire i Costi, no i Ricavi, fino ad un desiderato livello sottostante e che tale ripartizione avesse un peso differente a seconda dell’allocazione sulla quale far ricadere questo costo ripartito.

Per poter soddisfare tale richiesta abbiamo dovuto implementare un metodo che prendesse l’importo del Costo che si vuole ripartire, tramite una

query sul database individuare tutte le Risorse che al momento dell'accensione di tale Movimento avessero un allocazione che per Risorsa Padre avesse la Risorsa Figlia dell'allocazione sulla quale viene a verificarsi un Costo Ripartito.

Ottenute queste informazioni, nella form che si apre per la creazione dei Movimenti, compariranno, per tutte le Allocazioni sulla quale bisogna ripartire, i campi "Peso", espresso in percentuale, e "Ripartito" da poter settare. Settando il Peso, il software in automatico riempie il campo "Importo" con il dovuto valore, frutto del calcolo percentuale tra importo totale e peso dell'allocazione, e se viene valorizzato anche il campo Ripartito, le operazioni descritte verranno ripetute in maniera ricorsiva fino a quando il campo Ripartito non sarà uguale a "false".

4.4.5 Controller di A.R.E.M. Web

Dopo aver illustrato le View più significative della web application, possiamo analizzare nel dettaglio come queste vengono richiamate al momento opportuno.

Dietro al redirect di ogni pagina vi è un controllo, il quale adotta un metodo di servizio particolare in funzione della richiesta HTTP ricevuta. Come abbiamo ampiamente spiegato nel paragrafo relativo al funzionamento del pattern MVC, il DispatcherServlet cattura la richiesta HTTP proveniente dall'esterno e demanda all'opportuno controller la responsabilità di gestire Model e View.

I controllori di cui dispone il progetto A.R.E.M. sono:

- IndexController;
- ManagerController;
- AmministratoreController;

tutti presenti nel package controller del progetto. Come per il capitolo precedente, andiamo ad analizzare le parti più importanti di ogni Controller.

IndexController.

Tale controller è incaricato di gestire le pagine di accesso al sito e di errore. Inserendo nel browser l'URL per accedere al software, il DispatcherServlet inoltra la richiesta ai Controller, l'IndexController la intercetta ed esegue il metodo di servizio adatto alla richiesta. Vediamo come nel seguente pezzo di codice:

```
@Controller
public class IndexController {
    @Inject
    private AccountService accountSERV;
    @RequestMapping(value= {"/", "/home"})
    public String home(
                HttpSession sessione){
        sessione.removeAttribute("tipoUtente");
        sessione.invalidate();
        System.out.println("ti reindirizzo alla index per la login");
        return "index";
    }
```

La prima istruzione che risalta all'occhio è @Controller, questa è una delle annotation di Spring MVC ed indica che il pezzo di codice che segue identifica un Controller. Nel prosieguo del codice abbiamo utilizzato anche altre annotation tra cui @Inject, che si occupa di iniettare la dipendenza con i service del livello DAL ed è nativa di Java, @RequestMapping, che identifica la richiesta fatta tramite l'URL del browser.

L'ultima annotation vista è di fondamentale importanza, senza di essa il controller non potrebbe prendere in consegna la richiesta fatta, ed essa quindi non sarebbe servita dal DispatcherServlet.

Intercettata la richiesta, il controller applica il metodo ad essa collegato e ritorna il nome della View alla quale fare il redirect.

Al metodo viene passata una variabile HttpSession che, come detto nei paragrafi precedenti, sarà utile ai fini della sicurezza, garantendo l'accesso al solo titolare dei permessi per quella pagina.

Di seguito analizziamo il metodo incaricato del login:

```
@RequestMapping(value = "/login", method = RequestMethod.POST)
        public String loginForm(
                        @RequestParam("username")
                        String username,
                        @RequestParam("password")
                        String password,
                        Model modelIndex,
                        HttpSession sessione){
                System.out.println("Ti reindirizzo alla pagina di login");
                ArrayList<Account> account =
accountSERV.selectRicercaAccount(username, password);
                if (account.isEmpty()){
                        System.out.println("campi nome utente e password
errati");
                        String errore = "errore nell'inserimento di nome
utente e password";
                        modelIndex.addAttribute("modelIndex", errore);
                        return "index";
                        }
                if(account.get(0).getTipoAccount()==1){
                        int tipo = account.get(0).getTipoAccount();
                        sessione.setAttribute("tipoUtente", tipo);

        System.out.println(sessione.getAttribute("tipoUtente"));
                        return "manager";
                }
```

```
            else{
                int tipo = account.get(0).getTipoAccount();
                sessione.setAttribute("tipoUtente", tipo);

System.out.println(sessione.getAttribute("tipoUtente"));
                return "amministratore";
            }
        }
```

È importante notare nel codice sovrastante l'indicazione del metodo utilizzato nella richiesta di login, RequestMethod.POST. Trattandosi di una pagina di login, l'utente dovrà inserire dei valori che andranno poi controllati, per questo motivo la richiesta deve essere di tipo POST, se il metodo viene omesso la richiesta è di tipo GET. Al metodo vengono passati i parametri inseriti nella form di login tramite la annotation @RequestParam, che indica tra parentesi la variabile che verrà valorizzata nella form.

L'IndexController è responsabile anche della gestione della pagina di logout e di errore, nel caso si verificasse una violazione delle norme di sicurezza della sessione HTTP.

ManagerController e AmministratoreController

Data la mole di codice scritto per i due controllori incaricati della gestione delle operazioni per Manager ed Amministratore, abbiamo preferito omettere la parte relativa ad esso in quanto avrebbe richiesto un numero eccessivo di pagine.

Le annotation illustrate precedentemente sono quelle utilizzate anche per i due controller in esame, la differenza sostanziale tra i tre controllori è nei service del livello DAL richiamati e nella verifica e nel controllo dei dati necessari a popolare la pagina.

4.5 Punti critici e soluzioni

I punti delicati, le criticità incontrate durante lo sviluppo del web layer del progetto sono state varie e di diversa natura.

Tra le più rilevanti e degne di nota sono le difficoltà nel gestire le date, via crucis di ogni programmatore, in quanto il tipo utilizzato, GregorianCalendar, si è rivelato di difficile gestione ed adattamento alle nostre esigenze. Ad esempio, nelle View dove era necessario inserire delle date per poter fare delle analisi, non è stato possibile passare il valore come tipo GregorianCalendar, in quanto non ci permetteva di effettuare i controlli dovuti oppure restituiva dei risultati inattesi. Ciò è dovuto al formato in cui vengono scritte le date in GregorianCalendar. La soluzione a tale problema è stata quella di adottare un "parse" delle date, in modo da dare un format unico agli oggetti GregorianCalendar e poter quindi assicurarci un corretto controllo e dei risultati di ricerca esatti. Senza questa soluzione le diverse Timeline non sarebbero state visualizzate e non sarebbe stato possibile nemmeno avere dei corretti risultati sull'analisi dei Movimenti e del Profitto.

Un altro punto ostico è stato quello della creazione dell'albero delle allocazioni. Gli alberi vengono costruiti tramite una serie di tag <ul> e <li>, istruzioni di HTML.

Per ottenere un albero abbiamo dovuto fare una serie di query, da richiamare in modo ricorsivo per ottenere una struttura completa e corretta. Fin qui nessun problema, le difficoltà sono giunte nel momento in cui bisognava passare i risultati di queste query alla View.

Per risolvere il problema abbiamo pensato di costruire la struttura <ul> direttamente nelle query, ovvero restituire come risultato della query una struttura di questo tipo:

```
<ul>Padre
    <li>Figlio1</li>
    <li>Figlio2
```

```
            <ul>
                <li>Nipote1</li>
                <li>Nipote2</li>
            </ul>
        </li>
    </ul>
```

Questo è un albero che ha un padre, due figli di cui uno ha a sua volta due figli che saranno nipoti per la prima Risorsa padre. Questo tipo di struttura viene restituita come Stringa dalla query e tramite le tag library JSTL siamo riusciti a passarla alla View tramite Model e a parserizzarla in modo da poterla usare come codice HTML.

Di notevole difficoltà anche la parte relativa alla ripartizione dei costi. Come già evidenziato nel paragrafo relativo alla View di Gestione Movimenti, il committente ha richiesto la possibilità di ripartire alcuni costi per le allocazioni sottostanti.

Ad esempio, per una corretta analisi sui costi di gestione, l'Amministratore ha ritenuto opportuno ripartire il Costo dell'utenza relativa al consumo di energia elettrica in modo da poter fare un'analisi sull'efficienza delle Risorse allocate. Tale Movimento viene acceso dall'allocazione Home-Poliambulatorio e lo si vuole ripartire per tutte le Allocazioni sottostanti. Per poter effettuare tale operazione, l'utente in fase di immissione del costo da ripartire, una volta valorizzato il campo "Ripartito", vedrà comparire una form aggiuntiva dove saranno presenti tutte le allocazioni sottostanti sulla quale dover creare il Movimento che avrà come importo la quota parte relativa al peso, in percentuale, settato.

Tutto ciò è reso possibile da una serie di query e chiamate ricorsive dei metodi necessari alla ripartizione dei costi.

Capitolo 5

Conclusioni e sviluppi futuri

In questo capitolo, ultimo della tesi, trarremo le dovute conclusioni del progetto, analizzando i punti di forza e gli obiettivi che siamo riusciti a soddisfare.

Daremo inoltre alcune indicazioni sugli sviluppi futuri che coinvolgono l'introduzione di nuovi strumenti e la migrazione di tale software in un portale, trasformando tale web application in una portlet.

Paragrafi presenti nel capitolo sono:

- 5.1 Conclusioni;
- 5.2 Sviluppi futuri.

5.1 Conclusioni

Giunti al termine dello sviluppo software di A.R.E.M. possiamo finalmente trarre alcune osservazioni su cosa siamo riusciti ad ottenere e cosa vogliamo che questo software diventi.

Quando il committente si è presentato con le sue richieste, a primo impatto abbiamo pensato che il software fosse un banale gestionale. Studiando meglio quello che richiedeva ed analizzando i requisiti, ci siamo resi conto che avremmo dovuto sviluppare un software in grado di monitorare ogni dimensione di un'azienda e/o organizzazione, non solo la pura gestione del personale ma anche, e soprattutto, quella riguardante tutti gli elementi fisici che compongono tali realtà.

Guidati dal tutor aziendale, dai suoi colleghi e adottando uno sviluppo agile, siamo in tre mesi, tra corsi per apprendere le tecnologie e sviluppo software, riusciti a mettere in piedi un application web in grado di soddisfare, ed ampliare, le richieste del committente.

Ad oggi A.R.E.M. è in grado di poter:

- Creare ed aggiornare Risorse di ogni genere;
- Allocare in modo rapido ed intuitivo le Risorse, senza incorrere in alcun tipo di errore logico;
- Gestire Costi e Ricavi di tutte le Allocazioni;
- Ripartire fino ai livelli necessari i costi di gestione;
- Analizzare lo storico delle Allocazioni e dei Movimenti;
- Analizzare i profitti di qualunque periodo.

Tutto questo indipendentemente da chi richiede i servigi di tale software.

Proprio questa capacità rappresenta un punto di forza di questo progetto, in quanto, rispetto ad altri presenti sul mercato, è stato pensato e sviluppato per essere riutilizzabile.

Adottando la tecnica dello sviluppo agile siamo riusciti a presentare di volta in volta parti di progetto finite e funzionanti, risposto in modo rapido a nuove richieste di natura grafica e funzionale, riguardanti soprattutto la Gestione delle Allocazioni e dei Movimenti, e a completare il progetto nei tempi previsti.

5.2 Sviluppi futuri

Ultimato lo sviluppo del software, abbiamo pensato ad alcune integrazioni da poter fare sul progetto.

Una delle più immediate, e già in corso d'opera, è la trasformazione di questa web application in una portlet utilizzabile in un portale.

Anzitutto un portale è un sito web che racchiude un gruppo consistente di risorse proveniente dal mondo Internet. Molti portali sono costruiti e mantenuti con componenti software chiamati portlets.

I portlets sono moduli web riusabili all'interno di un portale Web.

Tipicamente, una pagina di un portale è suddivisa in una collezione di finestre, il contenuto di ciascuna delle quali viene definito da un diverso portlet.

Ciascun portlet è destinato ad una semplice applicazione, ad esempio servizi di news, previsioni meteo, o funzionalità legate a forum o email.

In quanto finestre, i portlet possono essere chiusi o ridotti o spostati. L'utente che accede al portale può così personalizzare la sua pagina personale, adattando i contenuti della stessa alle proprie esigenze.

L'idea dunque è quella di creare un portale dove mettere a disposizione una serie di strumenti, portlets, tra cui il nostro software.

Per poter realizzare tutto ciò, si dovrà trasformare il nostro progetto in una portlet e integrarlo in un portale, in via di risoluzione, che sarà basato su Liferay.

Liferay è un web portal framework open source scritto in java, che consiste in tre parti:

- kernel (Liferay Portal), che funge da base per le applicazioni e i contentuti;
- Content management system (Liferay CMS);
- Suite di applicazioni per realizzare collaboration e social networks (Liferay Collaboration).

Liferay è basato su un'architettura service-oriented (SOA), che rende possibile lo sviluppo di applicazione a partire dai servizi web esistenti, le portlets.

Le portlets sono gestite da un portlet container ed un portal server, che è responsabile di servire le pagine al browser. Insieme questi due sistemi costituiscono l'infrastruttura del portale necessaria al deployment delle applicazioni.

Ad oggi lo sviluppo di tale aggiornamento del progetto è giunto alla fase di trasformazione del software in una portlet. Abbiamo creato il portale Liferay su una macchina Linux e configurato il database. Terminata la fase di trasformazione saremo in grado di utilizzare la "Portlet A.R.E.M.".'

Nella Figura 5.1 è possibile osservare il portale Liferay. In esso abbiamo creato una portlet di prova, Liferay Test, la quale verrà poi sostituita con la futura "Portlet A.R.E.M.".

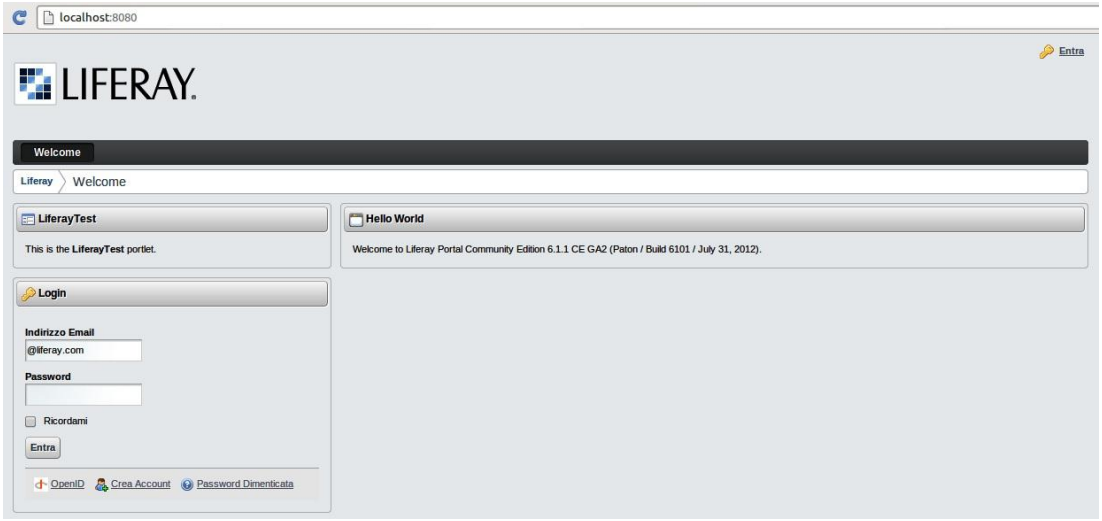

Figura 5.1 Portale Liferay

Una funzionalità che verrà integrata successivamente è la capacità di predire i profitti legati ad allocazioni future.

Per poter integrare tale strumento ci siamo avvalsi di alcune idee sfruttate nel campo della genetica.

Il concetto è abbastanza semplice, analizzando il profitto corrente, tutte le allocazioni che vanno a modificare tale profitto in positivo vengono considerate, altrimenti no.

Per poter sviluppare questa capacità, sfrutteremo degli algoritmi che andranno a calcolare quali allocazioni, tra quelle che il Manager vorrà creare, saranno convenienti, in funzione dei loro Movimenti. Per rendere ancora più efficiente tale strumento, gli algoritmi che sfrutteremo saranno in grado di fornire in quale ordine creare certe allocazioni, anche se controproducenti, con l'obiettivo di massimizzare sempre il profitto, in modo da poter sfruttare anche le allocazioni da scartare, se presentano dei punti favorevoli (ad esempio un macchinario che oggi mi costa tanto ma che domani lo potrò allocare ad un medico che mi darà un ricavo in grado di compensare i costi del macchinario).

La filosofia adottata da questi algoritmi rispecchia in parte quella implementata dall'algoritmo di Dijkstra, adoperato nell'ambito delle reti di calcolatori per creare l'albero a costi minimi per poter raggiungere, da un determinato nodo, tutti i nodi della rete.

Bibliografia

- Massimiliano Tarquini e Alessandro Ligi, 2001.
  Java mattone dopo mattone;
- Bruce Eckel.
  Thinking in Java 4th Editio;.
- Ciro Arciprete.
  Html5, Css3, Javascript e JQuery;
- Sue Spielman, 2004.
  JSTL – practical guide for JSP programmer;
- Antonio Agliata.
  Hibernate, Spring, Spring MVC;
- Simone Traversi, 2010.
  Spring 3;
- Richard Sezov, 2012.
  Liferay in action;
- http://www.wikipedia.com;
- http://www.jboss.org;
- http://www.html.it;
- http://www.springsource.org;
- http://www.hibernate.org;
- http://www.mkyong.com;
- http://www.tutorialspoint.com;
- http://stackoverflow.com;

Ringraziamenti

Giunto alla fine di quest'avventura, dopo aver pensato, studiato e scritto tanto queste righe sono quelle che aspettavo da tempo e che tanto temevo perché risultano quelle più difficili da scrivere.

Di prassi si scrive "è doveroso rivolgere un ringraziamento…". Per me non è doveroso scrivere quello che leggerete perché i ringraziamenti non devono mai essere doverosi ma sentiti e dunque sento di ringraziare i miei genitori senza i quali oggi non mi troverei davanti al pc a scrivere queste righe. Mi hanno dato la possibilità non solo di studiare ma di crescere in una famiglia sana, trasmettendomi quei valori di amore, rispetto ed umiltà che mi hanno dato la forza per raggiungere numerosi traguardi e ringrazio Dio perché ha reso possibile tutto questo.

Ringrazio chi mi ha seguito durante le ultime fasi di questa carriera Universitaria, il prof. Porfirio Tramontana, sempre prodigo di consigli e presente in qualunque momento.

Grazie al dott. Antonio Agliata, che nell'elenco delle società convenzionate con la Federico II è indicato come tutor aziendale della T.&.C. Systems Group, ma per me è stato qualcosa che col passare del tempo ha assunto fattezze ben diverse passando da tutor ad insegnante, collega ed infine amico, se così mi permetti di definirti.

Grazie a tutte le persone che ho incontrato durante i mesi di tirocinio presso la T.&.C. Systems Group, grazie a voi ho imparato molto ed in molti campi, dalla programmazione al mondo dei sistemisti fino al calcio balilla.

Grazie a tutti gli amici che in questi anni ho incontrato nel mondo accademico, ne siete veramente tanti ma sappiate che mi ricorderò di voi per sempre perché quello che abbiamo vissuto insieme in questo periodo è indimenticabile.

Grazie alle persone che mi sono state vicine nella vita, a chi è partito ma non fa mai mancare la sua presenza, come fai tu Umberto, e agli amici della Curia Arcivescovile di Napoli.

Grazie anche a chi in me non ha mai creduto veramente perché mi ha dato la forza, qualora non fosse già abbastanza, di proseguire per la mia strada, dritto in meta.

Grazie a voi tre, Aldo, Alessandro e Giovanni, perché ci siete sempre e perché siete così come vi vedo, spontanei, e non mi avete mai fatto mancare i vostri consigli e i vostri sorrisi.

Aldo sempre pronto a farci sbellicare dalle risate ma restando serio, quando gli riesce.

Alessandro impulsivo ed imprevedibile, trasmetti una carica ed una forza impressionante.

Giovanni saggio, pragmatico e teorico quanto basta per dare sempre il suo apporto con consigli utili al momento giusto, ultimo della serie "scegli il tirocinio esterno".

In ultimo, perché gli ultimi saranno i primi e tu lo sei sempre stata, Roberta. In questi anni se sono riuscito a sorridere nei momenti difficili è stato anche grazie a te, mi hai "sopportato" nei momenti difficili prima di ogni esame ed in quelli bui quando questi andavano male. Tu e la tua famiglia siete per me un altro porto sicuro dove potermi rifugiare durante la tempesta. Ringrazio Dio perché ha fatto in modo che le nostre strade si incrociassero e mi dispiace per chi non ha avuto la mia stessa fortuna o per chi in te non vede quello che vedo io. Sono felice di averti al mio fianco in questo giorno e per i giorni che verranno.

Spero di non aver dimenticato nessuno, se l'ho fatto è stato solo per distrazione, oppure perché non erano ringraziamenti sentiti…☺

Grazie