Università degli Studi di Padova

Modulo di rilevamento interventi post-vendita in GWT

Relazione finale di Tirocinio

Laureando: Alberto Costa

Relatore: Prof. G.Clemente

Dipartimento di Ingegneria dell'Informazione

Anno Accademico 2012-2013

Indice

Introduzione

La tesi si propone di descrivere l'attività di tirocinio svolta presso l'azienda Sanmarco Informatica S.p.a. di Grisignano di Zocco (VI), software-house specializzata nello sviluppo di applicazioni gestionali per aziende manifatturiere.

Oggetto dello stage è stata la progettazione e lo sviluppo di una Web-application dedicata alla consuntivazione ed al monitoraggio dei progetti di assistenza tecnica, focalizzata sulla costificazione degli interventi delle risorse interne e in outsourcing.

La presentazione del lavoro di tesi è strutturata in due parti:

- *il primo capitolo presenta l'azienda Sanmarco Informatica S.p.a., le sue specializzazioni applicative e descrive il software gestionale GalileoERP ed il portale web GalileoWebPortal.*
- *il secondo capitolo illustra le fasi di Analisi, Progettazione e Sviluppo dell'applicazione.*
    - *la fase di Analisi definisce le specifiche e i vincoli cui l'applicativo dovrà rispondere, i dettagli implementativi e l'interfaccia grafico-funzionale.*
    - *la fase di Progettazione definisce il concetto di WEB Application, di pattern MVC e illustra le principali tecnologie utlizzate: JavaEE, GWT, Hibernate, con un breve cenno all'application server Jboss e al database DB2/AS400.*
    - *la fase di Sviluppo contiene l'implementazione dei pacchetti e delle classi sviluppate, e descrive le fasi di test e collaudo del prodotto finito.*

L'applicazione sviluppata grazie al lavoro di tesi, una volta alimentata dalla base dati di GalileoERP, è stata inserita nel contesto applicativo di Galileo Web Portal.

1 L'Azienda

- *Presentazione della Sanmarco Informatica S.p.a.*
- *Descrizione del software gestionale GalileoERP e del portale web GalileoWebPortal.*

1.1 Struttura dell'azienda

Sanmarco Informatica S.p.a. nasce a Vicenza negli anni '80 come software house specializzata nello sviluppo di applicazioni gestionali per aziende manifatturiere.

Si avvale di un team composto da circa 270 persone tra dipendenti e collaboratori.

Nel corso degli anni, l'azienda ha ampliato il proprio organico e l'offerta di servizi, specializzandosi nell'offerta di consulenza e supporto alla riorganizzazione dei processi in diversi ambiti e contesti aziendali:

- Project Management
- Amministrazione e finanziaria
- Archiviazione documentale e conservazione sostitutiva
- Fiscalità estere
- Web marketing e web engineering
- Rilevazione tempi e schedulazione
- Logistica di magazzino e spedizioni
- Controlli di Qualità sul prodotto
- Workflow management

Sanmarco Informatica S.p.a. è partner commerciale di diversi vendor, tra i quali spiccano IBM, Microsoft e recentemente anche Apple.

1.2 Specializzazioni applicative e software

1.2.1 Galileo ERP

I sistemi ERP (Enterprice Resource Planning) esistenti sul mercato, hanno lo scopo di gestire e integrare tutte le funzioni business aziendali, ovvero gli acquisti, le vendite, la gestione del magazzino, la contablità, ecc.

Inizialmente i sistemi ERP si preoccupavano di collegare funzioni di carattere contabile con funzioni di tipo logistico; solo in un secondo momento sono state sviluppate funzionalità in grado di integrare funzioni interne come produzione, vendite, distribuzione, manutenzione di impianti, fabbisogno di materiali.

Figura 1.1: Struttura di GalileoERP

In generale un sistema di gestione ERP si occupa di tutte le macro attività aziendali:

- FRM: Finance Resource Management, rappresenta la gestione dell'assetto finanziario dell'azienda, ovvero il controllo e successiva analisi del flusso monetario definito dalle entrate e dalle uscite;
- SCM: Supply Chain Management, rappresenta la gestione della catena di fornitura e distribuzione della produzione, tenendo traccia dei materiale dal momento dell'approvvigionamento alla consegna al cliente del prodotto finito;
- HRM: Human Resource Management, rappresenta l'area che si occupa della gestione delle risorse umane impiegate all'interno dell'azienda;

- CRM: Customer Relationship Management, rappresenta la gestione dei rapporti con i clienti, ovvero l'acquisizione di nuovi clienti, la fidelizzazione, la registrazione dei loro dati per poter aumentare le occasioni di relazione e per tener traccia delle attività al fine di analizzarne le interazioni;
- MRP: Manufacturing Resource Planning, rappresenta la gestione della pianificazione produttiva, ovvero delle risorse necessarie (umane e materiali).

Figura 1.2: Logo GalileoERP

GalileoERP è una soluzione gestionale di alto livello che trova largo utilizzo in diversi settori e categorie merceologiche. E' un'applicazione modulare in grado di integrare tra loro tutti i compartimenti aziendali. Estende inoltre le funzionalità dei tradizionali sistemi ERP offrendo specifiche soluzioni personalizzate per determinati settori economici.

1.2.2 Galileo Web Portal

É un portale web sviluppato da Sanmarco Informatica S.p.a. che possiede la caratteristica di essere multiambiente, multiprodotto e modulare.

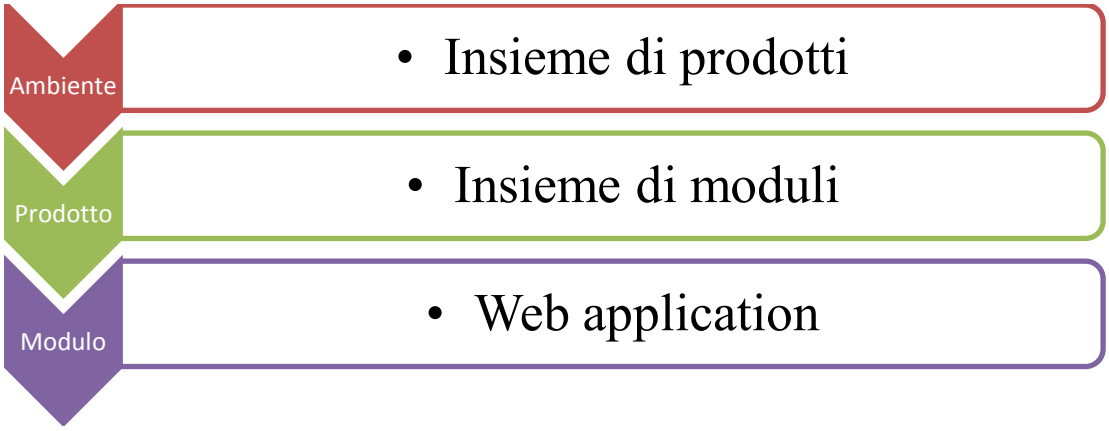


Figura 1.3: Schema Ambiente-Prodotto-Modulo

Per multiprodotto si intende che al suo interno possono coesistere applicazioni di varia tipologia. Il prodotto principale é GalileoERP ma contemporaneamente supporta anche altri applicativi che Sanmarco Informatica ha sviluppato e distribuisce.

Attualmente i prodotti che coesistono nel portale sono:

- GES (Gestionale GalileoERP)
- GBPM (Galileo Business Process Management)
- GPM (Galileo Project Management)
- GQ (Galileo Quality)

Per modulare si intende invece che ogni prodotto può contenere più moduli, ognuno con una specifica funzione specifica. Un modulo è una specifica web application che viene lanciata all'interno del portale web.

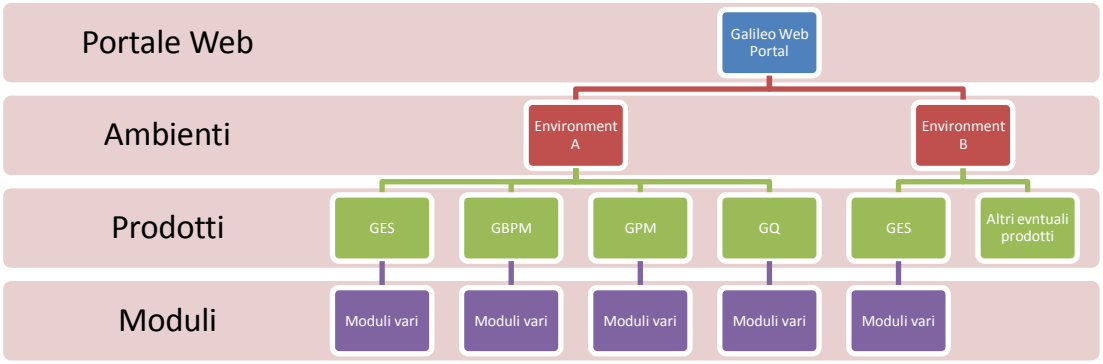


Figura 1.4: Struttura di Galileo Web Portal

L'applicazione sviluppata nella tesi fa parte dell'ambiente GES (Gestionale) essendo un modulo di appoggio di GalileoERP.

Il portale, sviluppato recentemente con le più moderne tecnologie disponibili sul mercato é accessibile dai principali browser e supporta anche i dispositivi mobili di ultima generazione.

2 Il Progetto

- *Presentazione del progetto sviluppato.*
- *Descrizione delle fasi di Analisi, Progettazione, Sviluppo e Collaudo del prodotto finito.*

2.1 Analisi

La fase di analisi è il processo che porta a definire le specifiche, stabilisce i servizi richiesti ed i vincoli del software. Di fronte alla richiesta di un cliente è sempre opportuno verificare quali risultati vorrebbe ottenere dall'utilizzo dello stesso e quali caratteristiche e qualità si aspetta di trovare una volta che l'applicativo sarà completato. Si deve riuscire a generare quindi una lista di requisiti completa che il software dovrà soddisfare.

Un'attenta e scrupolosa analisi consente al programmatore di risparmiare tempo ed energie durante la fase di produzione ed inoltre riduce notevolmente la possibilità di commettere errori. Raccogliendo tutte le informazioni in documenti, si evitano perdite di dettagli importanti necessari allo svolgimento delle fasi successive.

Nel capitolo è stata riassunta la documentazione di analisi.

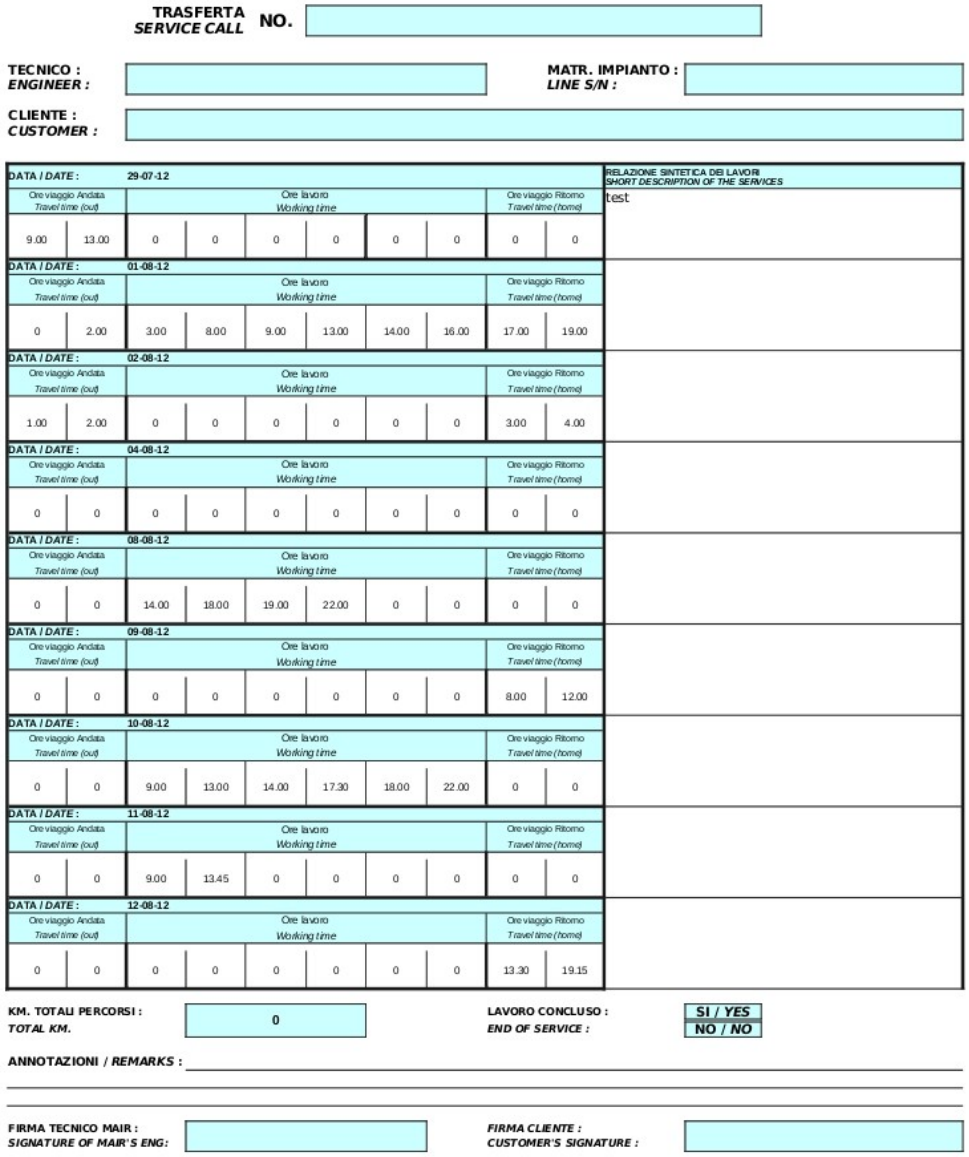
2.1.1 Scopo principale dell'applicazione

Lo scopo dell'applicazione è la consuntivazione ed il monitoraggio dei progetti di assistenza tecnica, focalizzata sulla contabilizzazione degli interventi delle risorse interne e in outsourcing. Per consuntivazione si initende la dichiarazione del lavoro svolto.

L'applicazione darà quindi ai tecnici interni ed ai fornitori la possibilità di consuntivare le attività di assistenza tecnica svolte attraverso un interfaccia WEB.

Inoltre darà la possibilità di esportare in formato Pdf ed eventualmente stampare un documento attestante il lavoro eseguito.

Questo documento sarà valido ai fini INAIL/INPS.



TRASFERTA **NO.**
SERVICE CALL

TECNICO : **MATR. IMPIANTO :**
ENGINEER : ***LINE S/N :***

CLIENTE :
CUSTOMER :

DATA / *DATE* :										RELAZIONE SINTETICA DEI LAVORI / *SHORT DESCRIPTION OF THE SERVICES*
Ore viaggio Andata *Travel time (out)*		Ore lavoro *Working time*						Ore viaggio Ritorno *Travel time (home)*		
29-07-12										test
9.00	13.00	0	0	0	0	0	0	0	0	
01-08-12										
0	2.00	3.00	8.00	9.00	13.00	14.00	16.00	17.00	19.00	
02-08-12										
1.00	2.00	0	0	0	0	0	0	3.00	4.00	
04-08-12										
0	0	0	0	0	0	0	0	0	0	
08-08-12										
0	0	14.00	18.00	19.00	22.00	0	0	0	0	
09-08-12										
0	0	0	0	0	0	0	0	8.00	12.00	
10-08-12										
0	0	9.00	13.00	14.00	17.30	18.00	22.00	0	0	
11-08-12										
0	0	9.00	13.45	0	0	0	0	0	0	
12-08-12										
0	0	0	0	0	0	0	0	13.30	19.15	

KM. TOTALI PERCORSI : 0
TOTAL KM.

LAVORO CONCLUSO : **SI / *YES***
END OF SERVICE : **NO / *NO***

ANNOTAZIONI / *REMARKS* :

FIRMA TECNICO MAIR :
SIGNATURE OF MAIR'S ENG:

FIRMA CLIENTE :
CUSTOMER'S SIGNATURE :

Figura 2.1: Documento INAIL/INPS

2.1.2 Condizioni di utilizzo

Login al portale

L'accesso al portale WEB deve essere effettuato distintamente da ogni singolo utente. Ad ognuno quindi viene fornito un proprio Account (Username,Password) per effettuare l'accesso all'applicativo.

Account = Username + Password

Visualizzazione e Modifica di interventi

Entrando nel portale l'utente deve vedere gli interventi aperti a lui assegnati. Data l'elevata numerosità delle rilevazioni che ci potrebbero essere, può essere logico prevedere nel portale una selezione da data a data.

Il prospetto deve prevedere la selezione dell'intervento tra quelli aperti e assegnati al tecnico dichiarante ma si deve prevedere anche che un utente possa registrare a fronte di un intervento al quale non è stato assegnato.

Tutte le attività che non vengono caricate direttamente dai tecnici sul portale, verrano caricate a consuntivo da personale interno, ma sempre con l'uso del portale web.

In questo caso si devono considerare i controlli che un utente diverso non vada a registrare interventi già consuntivati da un suo collega e/o viceversa.

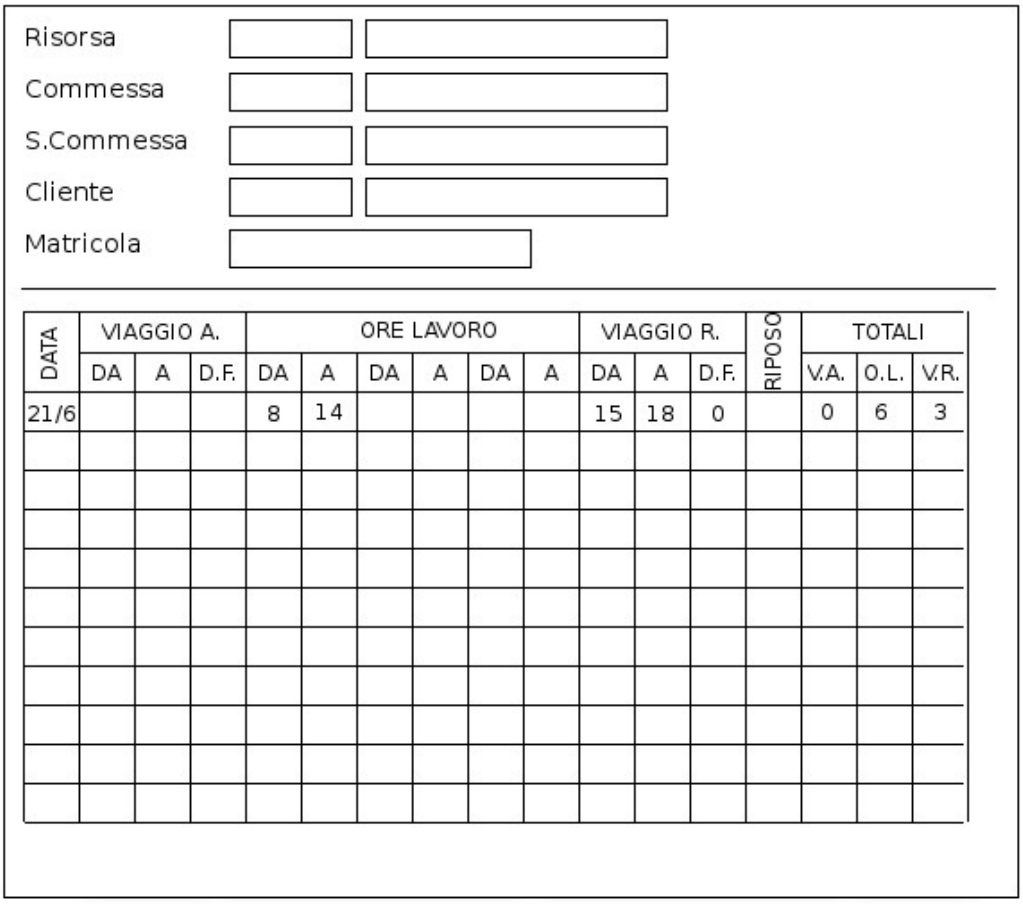
Viene così introdotto un concetto di utente compilatore e utente esecutore.

Si potranno specificare sia le ore lavorate sia le eventuali ore di viaggio sostenute.

Ore lavorate

Ore viaggio

La form deve essere organizzata in modo analogo all'attuale foglio trasferta utilizzato:



DATA	VIAGGIO A.			ORE LAVORO						VIAGGIO R.			RIPOSO	TOTALI		
	DA	A	D.F.	DA	A	DA	A	DA	A	DA	A	D.F.		V.A.	O.L.	V.R.
21/6				8	14					15	18	0		0	6	3

Figura 2.2: Foglio trasferta

Alcune di queste rilevazioni potrebbero essere già state contabilizzate, in questo caso non saranno modificabili.

L'utente si limiterà a inserire solo data, tempi di viaggio e lavoro, mentre il reperimento della causale da attribuire alla dichiarazione dovrebbe essere fatto automaticamente in base al paese relativo all'intervento (italia/europa/extra europa), al tecnico (interno/esterno) e dalla data/ora in cui si svolge l'attività.

Il reperimento della zona deve essere fatta analizzando per prima la diversa destinazione del cliente intestatario, poi lo spedire a, e poi il cliente intestatario dell'intervento.

Esempio di reperimento causale di fatturazione:

- se la data è festiva → causale festiva
- se l'ora è straordinaria → causale straordinaria
- se la fascia oraria copre sia ore ordinarie sia ore straordinarie, spacca le righe.

Occorre quindi definire una matrice collegamento rilevazione/causale/articolo in modo che essa possa essere reperita in automatico.

Flag di riposo

Ci sono anche i casi in cui ad esempio se il giorno dichiarato è domenica, e non sono dichiarate ore è inteso giorno di riposo (prevedere quindi un flag "giorno di riposo").

Le ore ordinarie sono intese = 8,00 sia per lavoro che per viaggio, le eccedenti sono straordinarie.

Le ore straordinarie sono riferite al viaggio (per alcuni clienti le ore ordinarie sono intese = 10, quindi le ore /giorno ordinarie potrebbero essere differenziate per cliente utilizzatore, da definirsi quindi in anagrafica del cliente).

Alcuni esempi:

- 6,5 ore lavoro + 2 ore viaggio =
  8 ore ordinarie e 0,5 ore straordinarie viaggio e 0 ore straordinarie lavoro
- 9,0 ore lavoro + 2 ore viaggio =
  8 ore ordinarie e 2 ore straordinarie viaggio e 1 ore straordinarie lavoro
- 1 ore lavoro + 10 viaggio =
  ore ordinarie fino a 8 ore come ore di lavoro ordinarie effettuate, oltre le 8 ore ordinarie fatturo le ore di viaggio pari a 3.

Da questo esempio si deduce che le prime 8 ore ordinarie di lavoro devono consumare le ore ordinarie di lavoro eseguite, (es. 1 ora) e poi fino a 8 ore le ore di viaggio, tutto il resto sono straordinario. In giorni festivi invece il viaggio anche se entro le 8 ore è sempre applicato a parte.

Esempio:

Ore viaggio 10 , Ore lavoro 1 → Totale 11 ore

Ore ordinarie = 8-1 = 7 , Ore straordinarie 10-7 = 3

Inoltre:

- Sabato e domenica e festivi sono sempre straordinari indipendentemente dalla fascia oraria.
- Da lunedi a venerdi sono ordinari (salvo i festivi) anche la diaria (nr. notti) cambia a seconda del paese.

Casistiche:

GIORNO FERIALE (Lun-Ven) su base ore ordinare/gg = 8

Ore Lavoro	Ore Viaggio	GG Lavoro Ordinari	Ore viaggio	Ore Lavoro straordniario	GG Riposo a Disposizione cliente	Lavoro Festivo	Ore Lavoro Straordinario festivo
6,5	2	1	0,5				
9	2	1	2	1			
1	10	1	3				
0	0	0	0	0	1		

Tabella 1.1 Distribuzione ore in un giorno feriale

GIORNO FESTIVO (Sab-Dom o Festivo) su base ore ordinare/gg = 8

Ore Lavoro	Ore Viaggio	GG Lavoro Ordinari	Ore viaggio	Ore Lavoro straordniario	GG Riposo a Disposizione cliente	Lavoro Festivo	Ore Lavoro Straordinario festivo
6,5	2		2	0		0,81	
9	2		2			1	1
1	10		10			0,13	
4	2		2			0,5	

Tabella 1.2: Distribuzione ore in un giorno festivo

Per ovviare al coefficiente di conversioneore/gg 8=0,125 e 10=0,100 è indispensabile codificare articoli diversi.

- Le ore festive sono ricavabili da calendario aziendale.
- Le ore straordinarie sono ricavabili da oltre > ore ordinarie.
- Definire in anagrafica cliente nr. ore ordinare per assistenza (es. Alcuni clienti 10, std 8).

REGOLE DI IDENTIFICAZIONE CAUSALE

1. Identificazione data (festiva o feriale).
2. Tipo attività (1 riposo, 2 lavoro, 3 viaggio).
3. Classifica cliente per stabilire la base di calcolo ore ordinarie/giornata.

Calcolo tabella riassunto

Aggancio causale/articolo

Con tecnico/fornitore

Nella stampa del consuntivo spese di trasferta le ore sono esposte anche in rapporto a gg. pertanto gli articoli codificati ai fini attività dovranno avere UM ore e UMA gg.

Esempio: 8 ore = 1 gg, 4 ore = 0,5 gg. quindi coeff. 0,125 (8x0,125=1)

2.1.3 Anteprima interfaccia grafica

Con l'aiuto di un semplice editor grafico è stata realizzata un anteprima del prodotto che si andrà a realizzare. Questa operazione, all'apparenza banale, in realtà è molto utile ai fini dello sviluppo del prodotto.

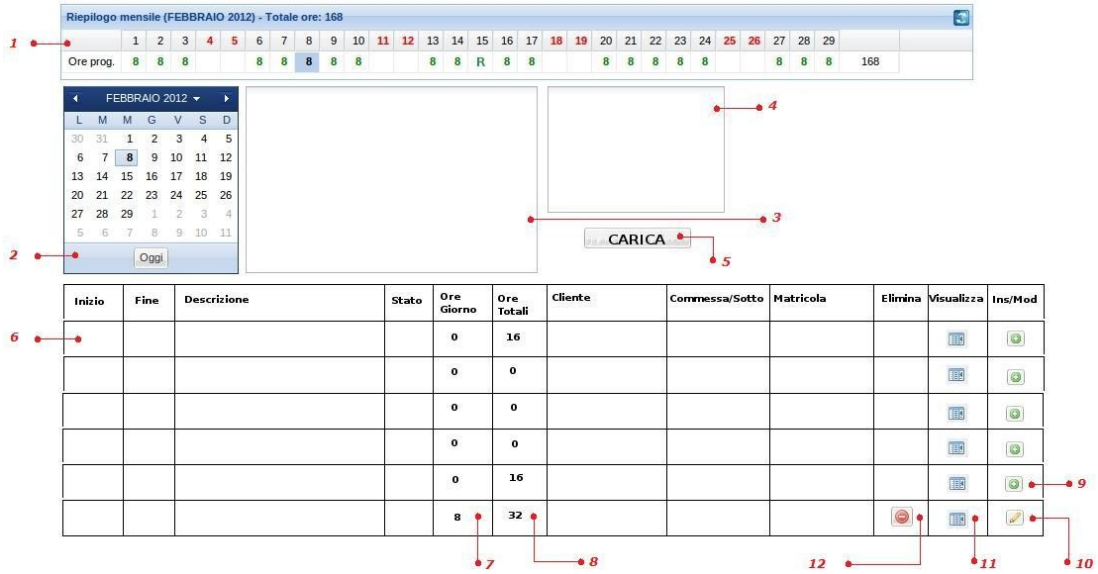


Figura 2.3: Anteprima applicazione

Nella schermata principale del modulo saranno presenti:

Riepilogo mensile per risorsa selezionata (1)

Mostrerà, per ogni giorno, il totale delle ore inserite, oppure la lettera "R" nel caso di riposo.

Calendario (2):

servirà per indicare la data di filtro degli interventi e di inserimento del consuntivo.

Al cambio del mese nel calendario, verrà eseguito il refresh del riepilogo mensile.

Lista operatori (3):

in base alla tipologia di accesso, la lista conterrà:

- Elenco delle risorse aziendali (nel caso di utente che può consuntivare per tutti).
- Operatore che ha effettuato l'accesso (nel caso di tecnico).
- Descrizione del fornitore (nel caso di accesso da parte di fornitori).

Al cambio della risorsa (nel primo caso), verrà eseguito il refresh del riepilogo mensile.

Lista viste (4)

Nella lista saranno presenti le tipologie di viste cioè:

- TODO : interventi pianificati nella data selezionata per l'operatore selezionato.
- Pianificati : interventi assegnati all'operatore selezionato, indipendentemente dalla data.

Pulsante di caricamento (5)

In base alla selezione di operatore, data e vista caricherà la griglia degli interventi.

Lista degli interventi (6)

Ogni riga della griglia conterrà le seguenti colonne:

- Data inizio intervento
- Data fine intervento
- Descrizione intervento
- Stato intervento
- **Ore giorno** (7) : Totale ore inserite per quel giorno, dall'operatore selezionato, a fronte del relativo intervento.
- **Ore totali** (8) : Totale ore inserite dall'operatore selezionato, a fronte del relativo intervento, indipendentemente dalla data.
- Cliente
- Commessa/Sottocommessa
- Matricola
- **Pulsante di Inserimento** (9) o **Modifica** (10)

I pulsante permetteranno l'apertura della form di inserimento consuntivo per l'intervento relativo, o l'apertura della form in modifica nel caso di consuntivo già inserito (se il consuntivo è già stato contabilizzato, tutti i dati saranno in sola lettura).

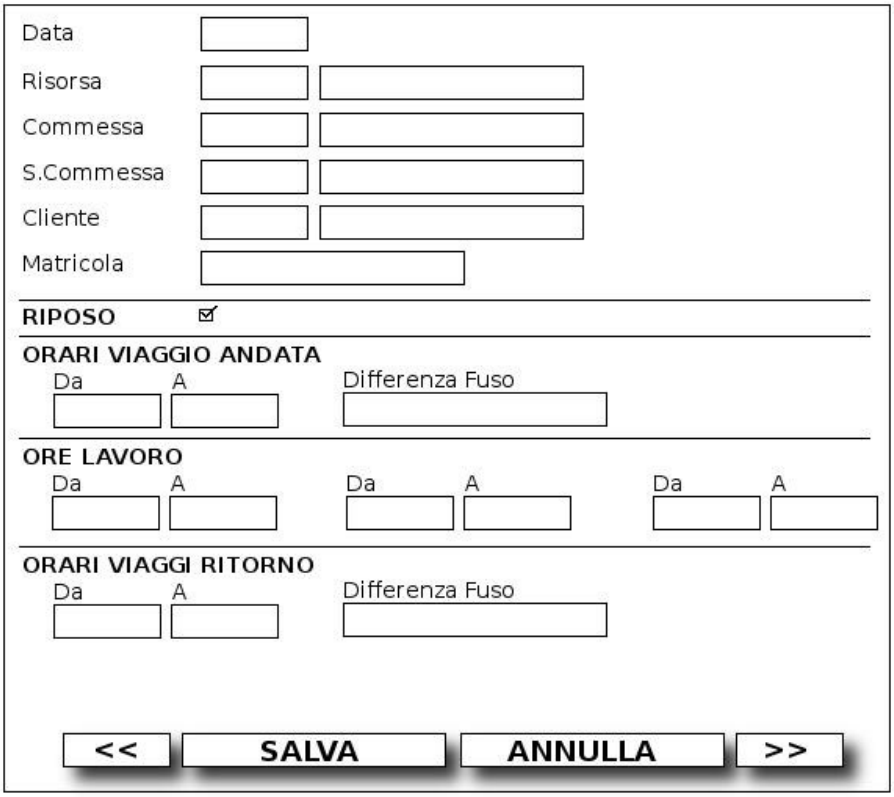


Figura 2.4: Anteprima form inserimento/modifica

Nella form saranno presenti:

- Dati relativi all'intervento.
- Flag riposo.
- Orari viaggi / lavoro.

- Pulsanti per salvataggio / annullamento.
- Pulsanti per passare al giorno precedente / successivo, mantenendo lo stesso intervento.
- **Pulsante di riepilogo consuntivi per intervento** (11)

Il pulsante aprirà un popup col dettaglio di tutti i consuntivi inseriti dall'operatore per l'intervento selezionato, ordinati per data (Figura).

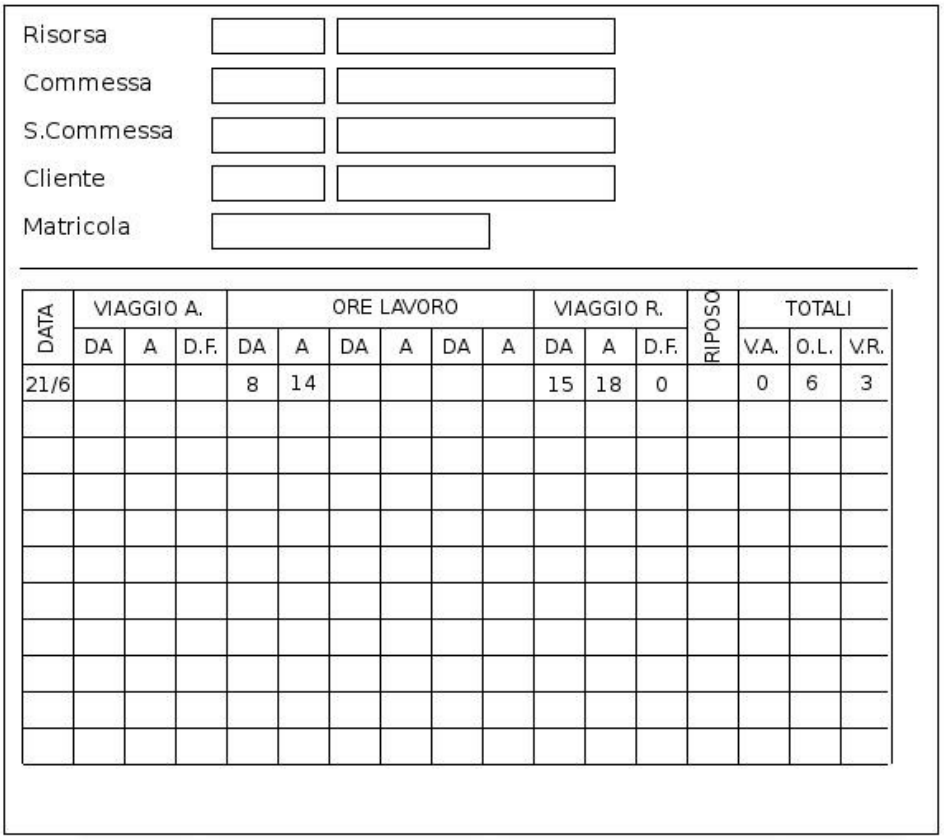


Figura 2.5: Anteprima form di riepilogo

Nella schermata di riepilogo per intervento saranno presenti:

- Dati relativi all'intervento.
- Riepilogo tabellare di tutti i consuntivi inseriti dalla risorsa a fronte dell'intervento selezionato.
- **Pulsante di eliminazione** (12):

Il pulsante permette la cancellazione di un consuntivo nel caso in cui questo sia stato inserito e non sia ancora stato contabilizzato.

Per l'utente che potrà inserire consuntivi per tutti gli operatori, sarà inoltre disponibile il riepilogo mensile di tutti gli operatori (Figura 2.6), che conterrà i totali delle ore per giorno o la lettera 'R' in caso di riposo.

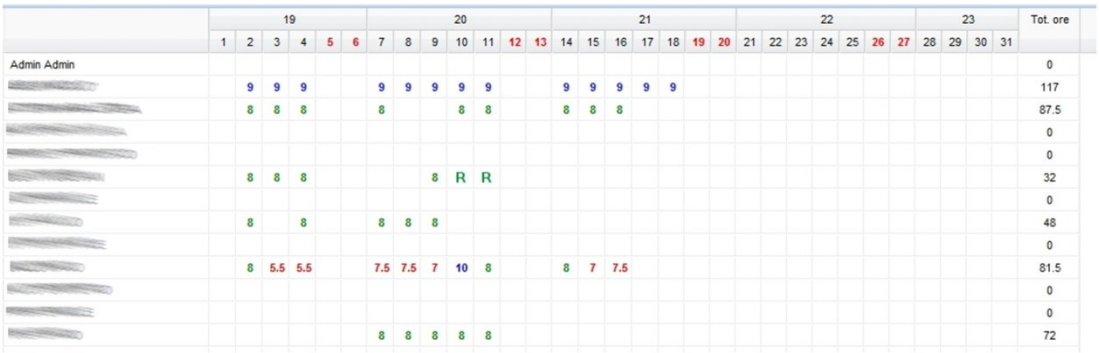


Figura 2.6: Anteprima riepilogo mensile

2.2 Progettazione

Viene stabilita la struttura generale (hardware e software) del sistema.

La progettazione del modulo, inserendosi in un sistema preesistente, non ha seguito le normali fasi necessarie alla progettazione di un'applicazione nativa. Il modulo, nonostante funzioni separatamente dagli altri, non deve essere ricreato da zero: possono infatti essere riutilizzate tecnologie, classi e funzionalità precedentemente sviluppate e collaudate.

Questa condizione impone quindi delle limitazioni in ambito di progettazione architetturale ma anche numerosi vantaggi in termini di tempo ed affidabilità delle tecnologie utilizzate.

Oltre alla descrizione del pattern MVC si illustrano le principali tecnologie utlizzate: JavaEE, GWT, Hibernate con un breve cenno all'application server Jboss e al database DB2/AS400.

2.2.1 WEB Application

Un Fat Client è un'architettura di tipo client/server, in cui è il client ad eseguire la maggior parte del lavoro e delle elaborazioni.

Questa architettura può essere usata in reti di computer molto potenti anche se questo termine, per lo più, viene usato per riferirsi al software.

Una Web-application invece, è un software che non necessita di essere installato nel computer in quanto esso si rende disponibile su un server in rete e può essere fatto funzionare attraverso un normale Web browser (Internet Explorer, Mozilla Firefox, Chrome, ecc.) in posizione di client.

Risulta evidente l'architettura client-server sulla quale si basano gli applicativi Web: il client, dopo aver instaurato una connessione con il server, invia la richiesta per un servizio; il server dopo aver elaborato i dati necessari rende disponibile al client il servizio richiesto.

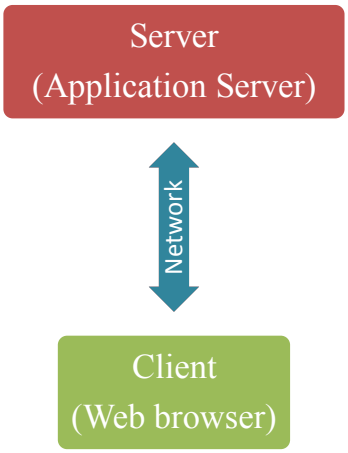


Figura 2.7: Architettura Client-Server

I vantaggi di utilizzare una Web-Application sono:

1. Non necessitano di alcuna installazione.
   L'unico strumento necessario ad un client è un web browser compatibile con l'applicazione.
2. Funzionano su qualsiasi sistema operativo.
3. Le applicazioni sono disponibili ovunque vi sia un dispositivo connesso ad Internet.

Tra i difetti di una WEB-application va segnalato che una connessione Internet poco performante può incidere negativamente sul loro funzionamento. E' indispensabile quindi utilizzare connessioni con una buona quantità di banda e con connessione permanente.

2.2.2 Pattern : MVC

Definire uno schema generale di riferimento (pattern architetturale) per la progettazione e strutturazione di applicazioni di tipo interattivo è una fase importante nella progettazione di applicazioni software.

Un'opportuna architettura, che definisce le linee guida allo sviluppo del progetto è utile sia per standardizzare il modello di sviluppo del progetto corrente, sia per le applicazioni future. Una progettazione architetturale che si basa sulla separazione dei ruoli dei componenti software rende strutturalmente indipendenti moduli con funzionalità differenti e favorisce qualità del software.

MVC Permette una maggiore strutturazione del codice, con un aumento della manutenibilità software e una suddivisione dell' applicazione in sottosistemi.

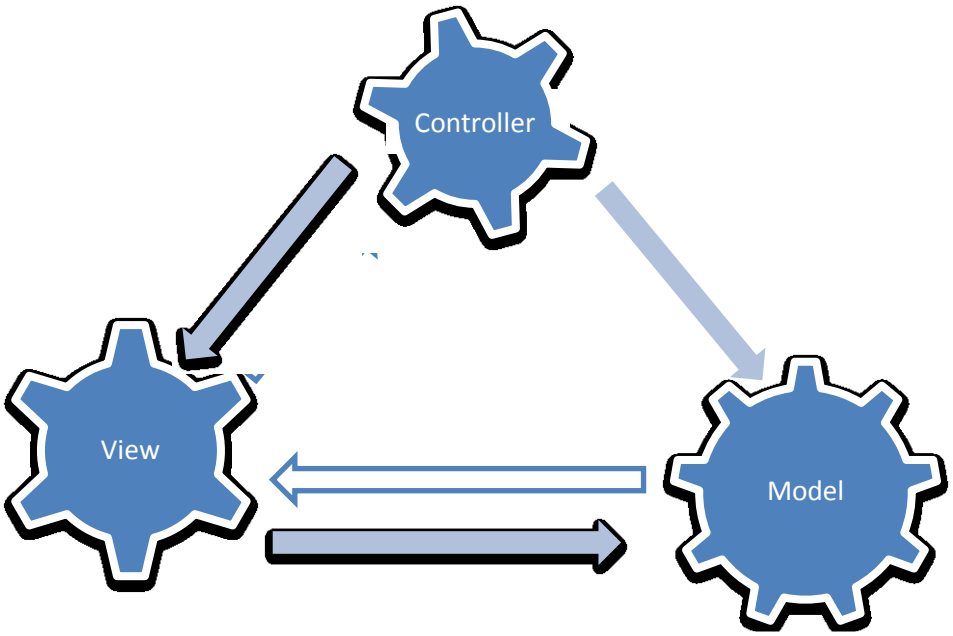


Figura 2.8:Pattern MVC

Per comprendere la filosofia del modello, in Figura, è riportata la schematizzazione delle tre componenti in cui si divide.

Le componenti sono:

1. la VIEW dove viene gestita la presentazione dei dati. E' rappresentata da tutto quel codice di presentazione che permette all'utente, ad esempio tramite un browser, di operare le richieste. All' interno di questo livello lavorano sia programmatori che grafici che curano la parte estetica della presentazione;
2. il MODEL che rappresenta e gestisce i dati, tipicamente persistenti su database;
3. il CONTROLLER che dirige i flussi di interazione tra vista e modello. Nello specifico, intercetta le richieste HTTP del client e traduce ogni singola richiesta in una specifica operazione per il model; in seguito può eseguire lui stesso l' operazione oppure delegare il compito ad un'altra componente. Inoltre, seleziona la corretta vista da mostrare al client ed inserisce, eventualmente, i risultati ottenuti.

Il modello più utilizzato in ambito delle applicazioni orientate al Web e non solo è il pattern Model-View-Controller (MVC).Il modello del sistema GWP adotta il pattern architetturale MVC (Model View Controller).

Figura 2.9: Qualità acquisite dal software grazie al pattern MVC

Estendibilità

Semplicità di progetto e decentralizzazione dell'architettura.

Software facilmente estendibile agendo su moduli specifici.

Agevolazione del lavoro in team e della pianificazione del progetto, dividendo quest'ultimo in componenti indipendenti delegabili a gruppi di lavoro differenti.

La manutenzione del codice risulta molto più semplice.

Riusabilità:

possibilità di estrarre e riutilizzare componenti per altri progetti e applicazioni.

Interoperabilità

Interazione tra moduli con ruoli differenti con la possibilità di creare gerarchie tra componenti.

Aumenta la flessibilità delle applicazioni e incrementa la scalabilità.

2.2.3 Java EE

Java EE (Enterprise Edition) è una piattaforma software di calcolo Java.

E' costituita da un insieme di specifiche per lo sviluppo di applicazioni per la parte di codice lato server e, negli anni è diventata sinonimo di sviluppo di applicazioni aziendali robuste, sicure ed efficienti.

Queste caratteristiche la rendono tra le più importanti piattaforme tecnologiche di sviluppo, soprattutto in ambiti in cui la sicurezza e la robustezza sono vincoli imprescindibili (ad esempio applicazioni bancarie).

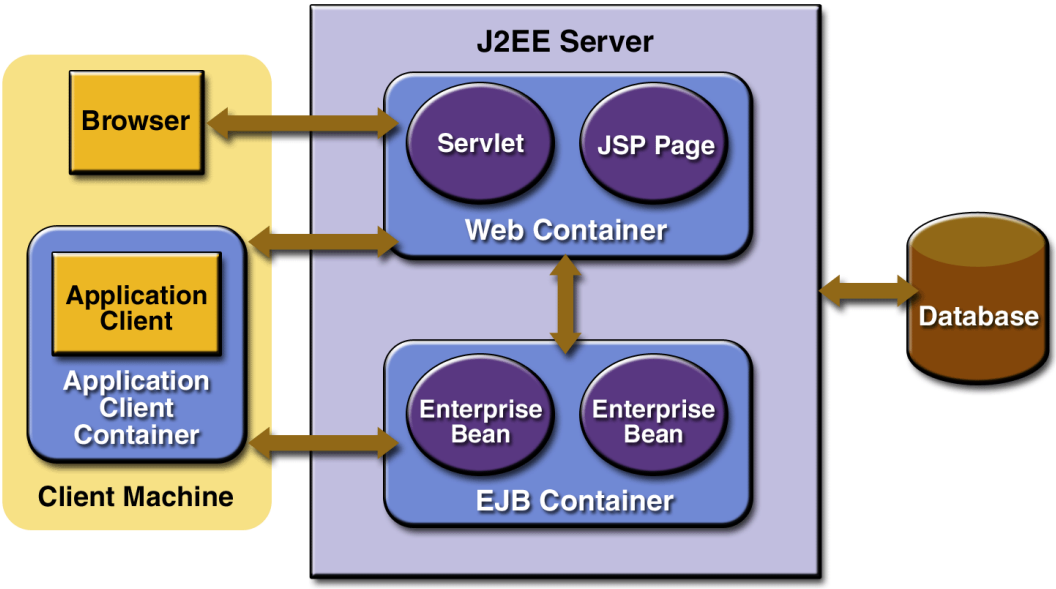


Figura 2.10: Architettura JavaEE

La piattaforma fornisce API e 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 versione JavaEE estende la versione Java2PlatformStandardEdition (Java SE).

La piattaforma incorpora un design basato in gran parte su componenti modulari in esecuzione su un application server. Il software per JavaEE è principalmente sviluppato in linguaggio di programmazione Java. Include molte tecnologie che estendono le funzionalità di base della piattaforma Java.

La specifica descrive i seguenti componenti:

- EJB definisce un sistema a componenti distribuito che rappresenta il cuore della specifica Java EE. Tale sistema, infatti, fornisce le tipiche caratteristiche richieste dalle applicazioni enterprise, come scalabilità, sicurezza, persistenza dei dati e altro.
- JNDI definisce un sistema per identificare e elencare risorse generiche, come componenti software o sorgenti di dati.
- JDBC è un'interfaccia per l'accesso a qualsiasi tipo di basi di dati (compresa anche nella standard edition).
- JTA è un sistema per il supporto delle transazioni distribuite.
- JPA è l'API per la gestione della persistenza dei dati.
- JAXP è un API per la gestione di file in formato XML.
- JMS (Java Message Service) descrive un sistema per l'invio e la gestione di messaggi.

Gli EJB (Enterprise Java Bean) sono i componenti che implementano, lato server, la logica di business all'interno dell'architettura Java EE. Le loro specifiche intendono fornire una metodologia standard per implementare la logica di funzionamento delle applicazioni di tipo enterprise, applicazioni cioè che forniscono servizi via Internet.

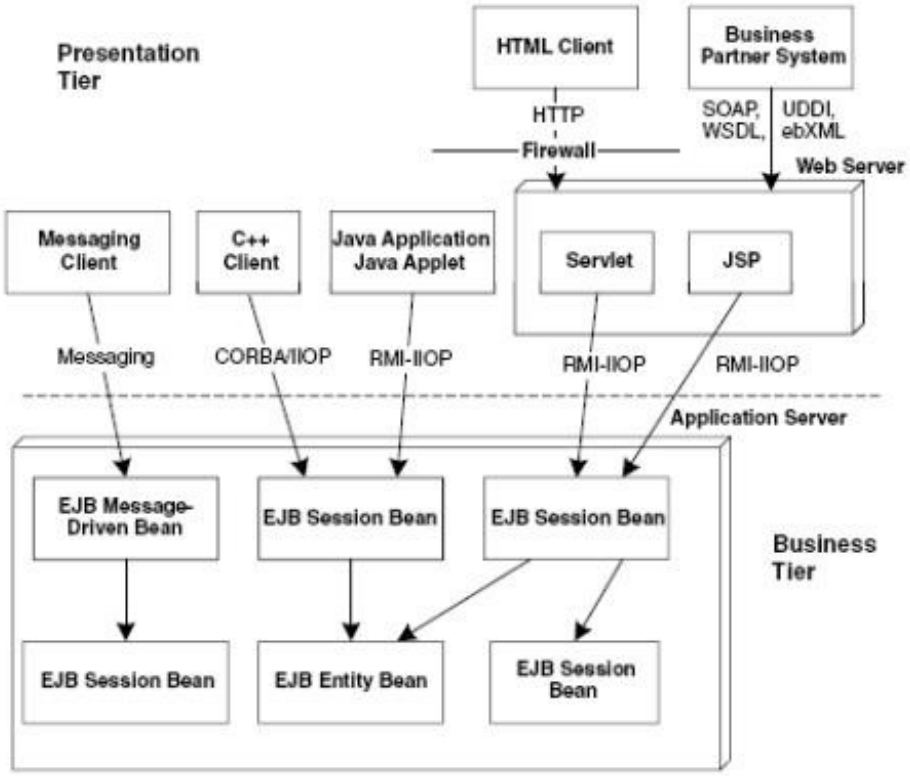


Figura 2.11: EJB schema

Per realizzare applicazioni di questo tipo è necessario affrontare una serie di problematiche e tecniche che possono rivelarsi molto complesse e laboriose da risolvere. Gli Enterprise Java Bean forniscono una soluzione a questi problemi e semplificano lo sviluppo di applicazioni di questo tipo.

Le specifiche EJB descrivono in dettaglio come realizzare un application server che fornisca le seguenti funzionalità:

- Persistenza
- Elaborazione delle transazioni
- Controllo della concorrenzialità
- Programmazione ad eventi tramite il Java Message Service
- Servizio di directory per elencare e nominare gli EJB (JNDI)
- Sicurezza (JCE e Java Authentication and Authorization Service)
- Installazione di componenti software in un application server
- Invocazione di procedure remote tramite l'utilizzo di RMI-IIOP o CORBA
- Fornire servizi web

Gli EJB offrono dunque una serie di soluzioni per affrontare lo sviluppo di applicazioni Internet attraverso una metodologia standard.

2.2.4 Hibernate

Hibernate è una piattaforma middleware open source per lo sviluppo di applicazioni Java.

Viene distribuito in licenza LGPL. La GNU Lesser General Public License (abbreviata in GNU LGPL o solo LGPL) è una licenza di software libero creata dalla Free Software Foundation, studiata come compromesso tra la GNU General Public License e altre licenze non-copyleft come la Licenza BSD, la Licenza X11 e la Licenza Apache. Fu scritta nel 1991 (aggiornata nel 1999 e nel 2007) da Richard Stallman, con l'ausilio legale di Eben Moglen.

La LGPL è una licenza di tipo copyleft ma, a differenza della licenza GNU GPL, non richiede che eventuale software "linkato" al programma sia rilasciato sotto la medesima licenza.

La LGPL è principalmente usata per le librerie software; talvolta è utilizzata anche da applicativi, come Mozilla Firefox, OpenOffice.org o LibreOffice.

Lo scopo principale di Hibernate è quello di fornire un mapping delle classi Java in tabelle di un database relazionale; sulla base di questo mapping Hibernate gestisce il salvataggio degli oggetti di tali classi su database. Si occupa inoltre del reperimento degli oggetti da database, producendo ed eseguendo automaticamente le query SQL necessarie al recupero delle informazioni e la successiva reistanziazione dell'oggetto precedentemente "ibernato" (mappato su database).

Fornisce il servizio Object Relational Mapping (ORM) che semplifica e automatizza la gestione della corrispondenza tra il modello ad oggetti delle nostre applicazioni e quello relazionale dei database server.

L'ORM è una tecnica di programmazione che favorisce l'integrazione di sistemi software aderenti al paradigma della programmazione orientata agli oggetti con sistemi RDBMS.

Un prodotto ORM fornisce, mediante un'interfaccia orientata agli oggetti, tutti i servizi inerenti alla persistenza dei dati, astraendo nel contempo le caratteristiche implementative dello specifico RDBMS utilizzato.

Hibernate supporta numerosi Database come:

- Oracle
- DB2
- PostgreSQL
- MySQL

ed il suo progetto è legato al famoso Application Server JBOSS.

Si pone tra il modello ad oggetti che si ricava dalla logica della nostra applicazione e l'insieme delle tabelle, chiavi primarie e vincoli relazionali presenti nel database relazionale.

Permette di rendere facilmente persistenti le classe Java senza eccessivi vincoli ed uno o più file di configurazione, la cui estensione è hbm, stabiliscono la corrispondenza tra oggetti della nostra applicazione e le tabelle del database. Ogni oggetto, descritto dalla classe Java, viene progettato in fase di modellazione del

dominio applicativo e per poter essere gestito da Hibernate deve semplicemente aderire alle fondamentali regole dei più pratici javabeans: metodi pubblici getXXX
e setXXX per le proprietà persistenti.

La Figura rappresenta il modo con cui Hibernate usa il database e i dati di configurazione per fornire servizi di persistenza (e oggetti persistenti) all'applicazione.

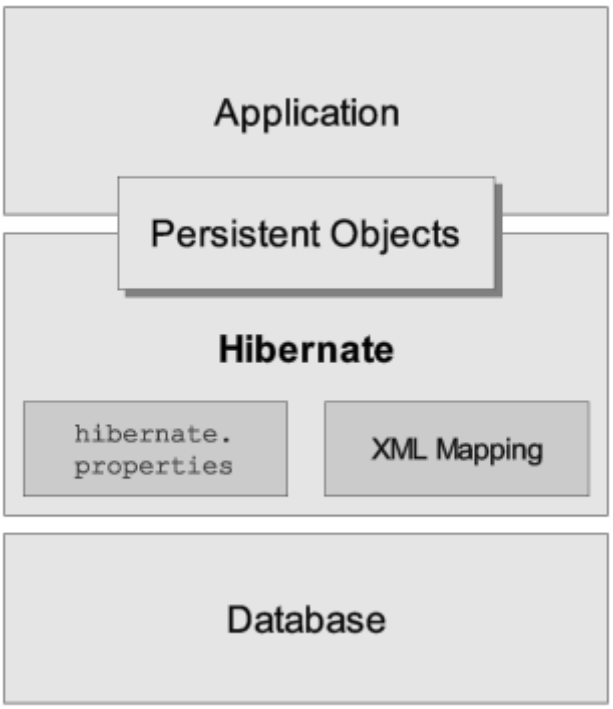


Figura 2.12: Hibernate schema

Uno dei concetti fondamentali dell'architettura di Hibernate è quello di classi persistenti, ovvero quelle classi che in un'applicazione implementano le entità del database (ad esempio Customer e Order in una applicazione di e-commerce). Hibernate funziona meglio se queste classi seguono alcune semplici regole, conosciute anche come il modello di programmazione dei cari vecchi oggetti java (in inglese e nel seguito si usa l'acronimo POJO che sta per Plain Old Java Object). Un POJO è più o meno come un JavaBean, con proprietà accessibili tramite metodi getter e setter (rispettivamente per recuperare e impostare gli attributi).

Un esempio di POJO:

```
package eg;
import java.util.Set;
import java.util.Date;

public class Cat {
    private Long id; // identifier

    private Date birthdate;
    private Color color;
```

```
private char sex;
private float weight;
private int litterId;

private Cat mother;
private Set kittens = new HashSet();

private void setId(Long id) {
    this.id=id;
}
public Long getId() {
    return id;
}

void setBirthdate(Date date) {
    birthdate = date;
}
public Date getBirthdate() {
    return birthdate;
}

void setWeight(float weight) {
    this.weight = weight;
}
public float getWeight() {
    return weight;
}

public Color getColor() {
    return color;
}
void setColor(Color color) {
    this.color = color;
}

void setSex(char sex) {
    this.sex=sex;
}
public char getSex() {
    return sex;
}

void setLitterId(int id) {
    this.litterId = id;
}
public int getLitterId() {
    return litterId;
}

void setMother(Cat mother) {
    this.mother = mother;
}
public Cat getMother() {
    return mother;
}
void setKittens(Set kittens) {
    this.kittens = kittens;
}
public Set getKittens() {
    return kittens;
}
```

```
        // addKitten not needed by Hibernate
        public void addKitten(Cat kitten) {
            kitten.setMother(this);
        kitten.setLitterId( kittens.size() );
            kittens.add(kitten);
        }
    }
```

Il mapping, a differenza di Jpa dove vengono usate annotazioni, viene definito in un documento XML progettato per essere leggibile e modificabile a mano; esistono un certo numero di strumenti per generare il documento di mapping ma è possibile effettuarlo anche attraverso un semplice editor di testo.

Ecco il file XML di mapping della classe descritta precedentemente:

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
      "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
          "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping package="eg">

        <class name="Cat"
            table="cats"
            discriminator-value="C">

                <id name="id">
                        <generator class="native"/>
                </id>

                <discriminator column="subclass"
                     type="character"/>

                <property name="weight"/>

                <property name="birthdate"
                    type="date"
                    not-null="true"
                    update="false"/>

                <property name="color"
                    type="eg.types.ColorUserType"
                    not-null="true"
                    update="false"/>

                <property name="sex"
                    not-null="true"
                    update="false"/>

                <property name="litterId"
                    column="litterId"
                    update="false"/>

                <many-to-one name="mother"
                    column="mother_id"
                    update="false"/>

                <set name="kittens"
                    inverse="true"
                    order-by="litter_id">
```

```
                <key column="mother_id"/>
                <one-to-many class="Cat"/>
            </set>

            <subclass name="DomesticCat"
                discriminator-value="D">

                <property name="name"
                    type="string"/>

            </subclass>

        </class>

        <class name="Dog">
            <!-- mapping for Dog could go here -->
        </class>

</hibernate-mapping>
```

I tag principali di un file XML sono:

1. <!DOCTYPE…>:

Indica l'effettivo Document Type Definition (DTD) utilizzato, specificato nell'URL indicato.

Il DTD è uno strumento utilizzato dai programmatori il cui scopo è quello di definire le componenti ammesse nella costruzione di un documento XML.

Tutti gli XML mappings devono dichiarare il DTD.

2. Hibernate-Mapping:

```
<hibernate-mapping
         schema="schemaName"
         catalog="catalogName"
         default-cascade="cascade_style"
         default-access="field|property|ClassName"
         default-lazy="true|false"
         auto-import="true|false"
         package="package.name"
 />
```

Si compone di vari sottoelementi e attributi opzionali.

E' il tag che contiene l'intera definizione della tabella mappata.

Consente di definire:

- tabella
- chiavi primarire
- campi e loro proprietà

2.2.5 Framework : GWT e GXT

Figura 2.13: GWT logo

Google Web Toolkit (GWT) è uno strumento di sviluppo che permette la creazione di applicazioni web e risponde perfettamente all'esigenza di creare applicazioni in stile J2EE. Facilita la codifica delle pagine web poichè produce automanticamente codice JavaScript.

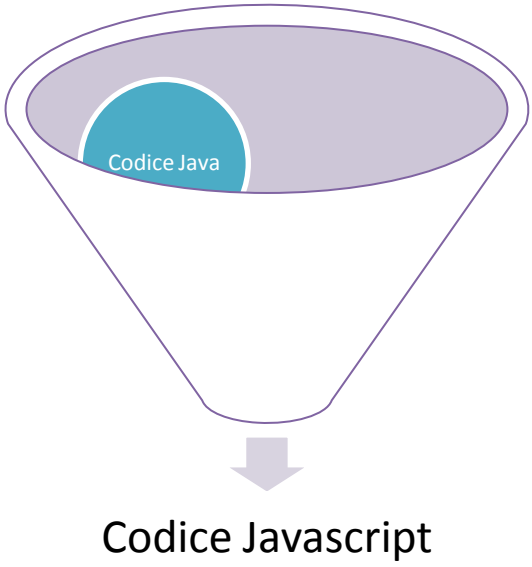


Figura 2.14: Meccanismo di compilazione GWT

Lo sviluppatore non deve necessariamente conoscere questi linguaggi poichè la programmazione è in linguaggio Java.

Viene distribuito con Apache License, ovvero una licenza di software libero non copyleft scritta dalla Apache Software Foundation (ASF) che obbliga gli utenti a preservare l'informativa di diritto d'autore e d'esclusione di responsabilità nelle versioni modificate. La Licenza Apache consente agli utenti di usare il software per ogni scopo, di distribuirlo, modificarlo e di distribuire versioni modificate del software. Quindi GWT è completamente open source..

Le librerie fornite con GWT risultano piuttosto troppo rudimentali. I widget sono essenziali e mancano di alcune funzionalità utili per una web application di alto livello. A compensare questa "mancanza" ci pensa quindi la libreria GXT sviluppata dalla software-house Sencha.

Figura 2.15: Sencha GXT logo

La libreria GXT è risultata essere una libreria molto utile per creare una GUI moderna e dinamica. Permette di arricchire la grafica offrendo widget, layout, template grafici e molto altro.

Si basa su un interessante e potente meccanismo a eventi in stile MVC che consente di creare applicazioni interattive e di creare il flusso applicativo in modo efficace.

Ecco alcuni dei componenti che la libreria mette a disposizione:

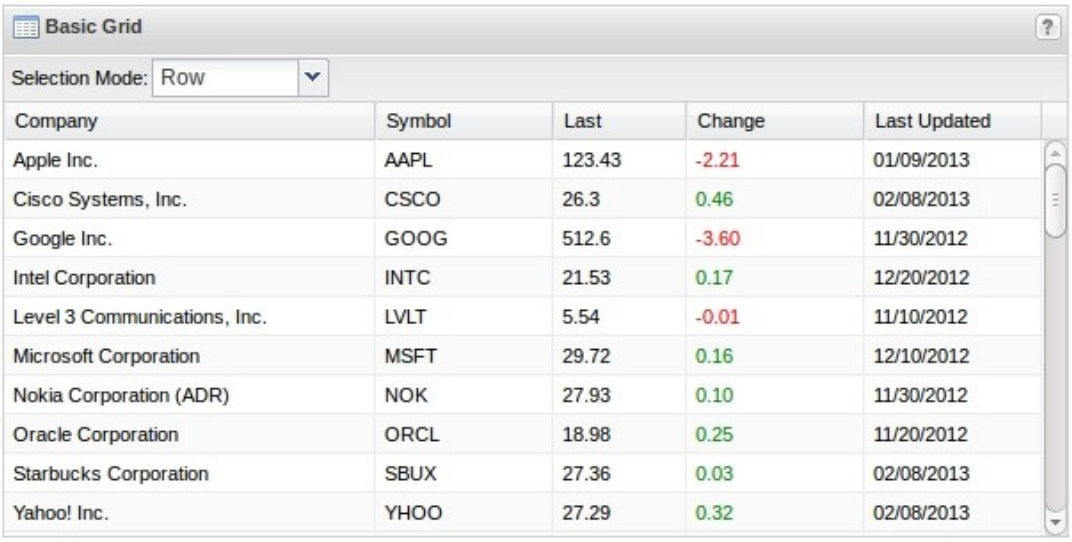


Figura 2.16 : Sencha GXT – Basic grid widget

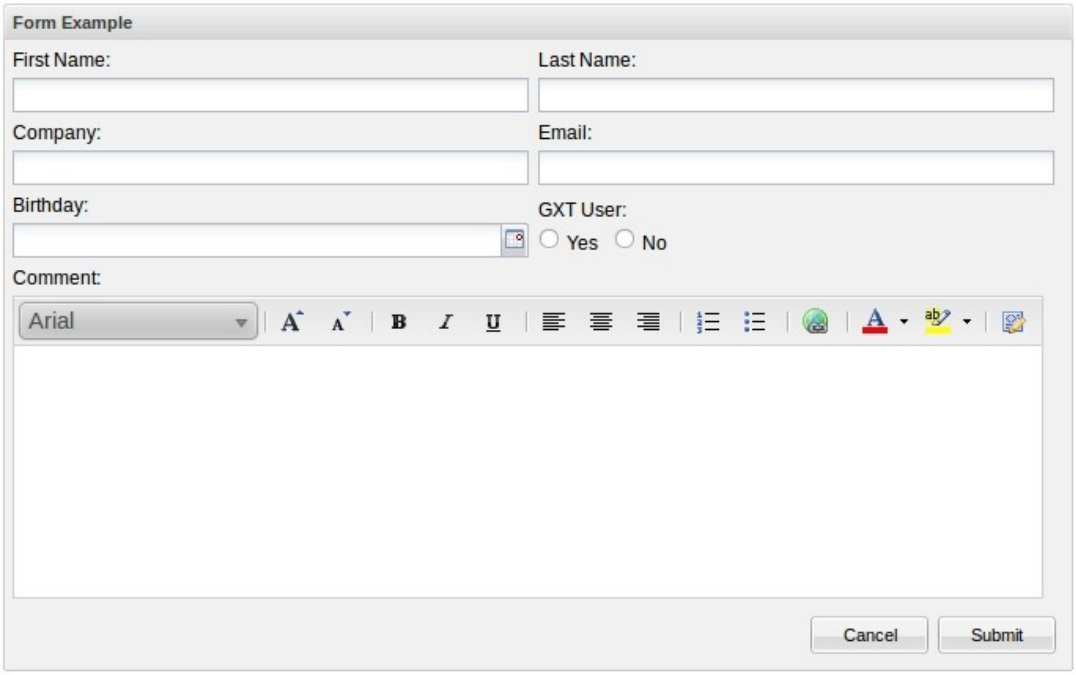


Figura 2.17 : Sencha GXT – Forn widget

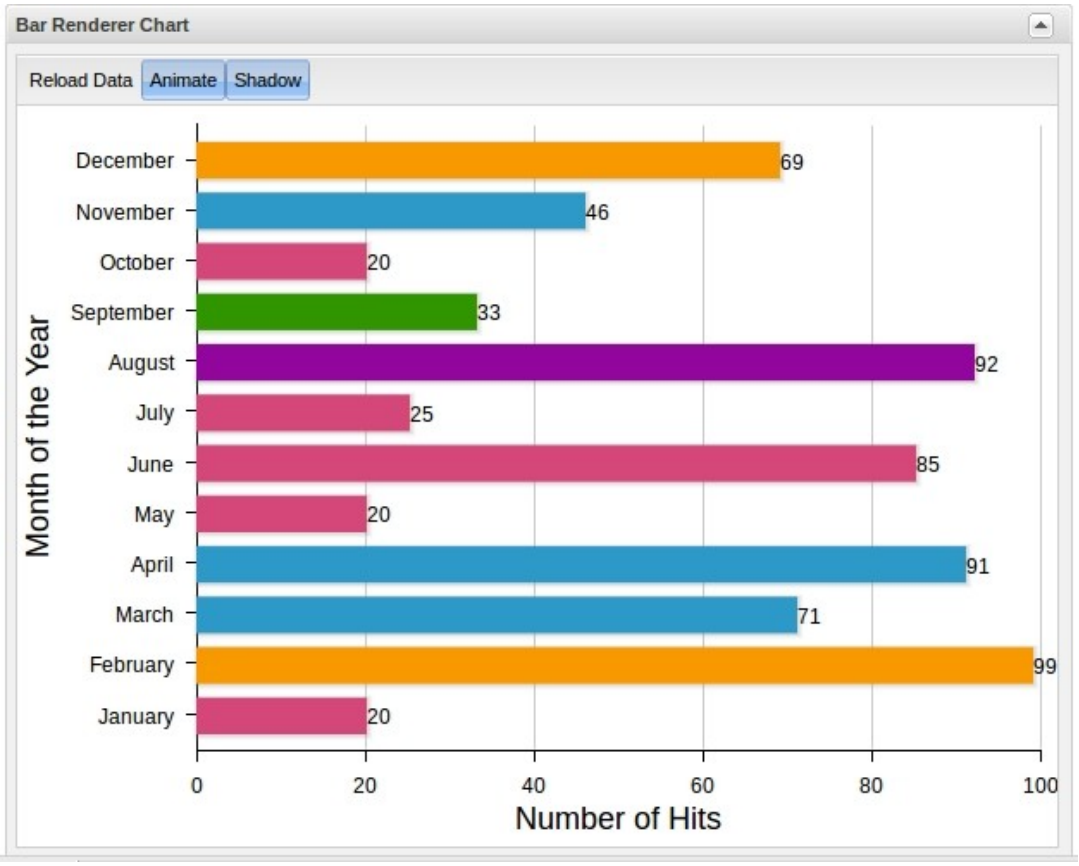


Figura 2.18 : Sencha GXT – Chart widget

GWT RPC

GWT-RPC (Remote Procedure Call) è il meccanismo utilizzato da GWT per la comunicazione con il server. Favorisce lo scambio di oggetti Java attraverso il protocollo http ed è basato sula ben nota architettura servlet Java. Prevede la generazione di efficiente codice lato client e lato server per serializzare gli oggetti attraverso la rete utilizzado il deferred binding.

Permette di spostare tutta la logica UI (User Interface) sul client, conseguendo un notevole miglioramento delle prestazioni (si riduce la largezza di banda necessaria e il carico sul Web server).

E' possibile utilizzare il framework GWT-RPC per rendere trasparenti le chiamate alle servlet Java e lasciare a GWT il compito di prendersi cura dei dettagli di basso livello come la serializzazione degli oggetti.

GWT-RPC facilita il passaggio di oggetti Java tra client e server (e anche viceversa) attraverso il protocollo HTTP.

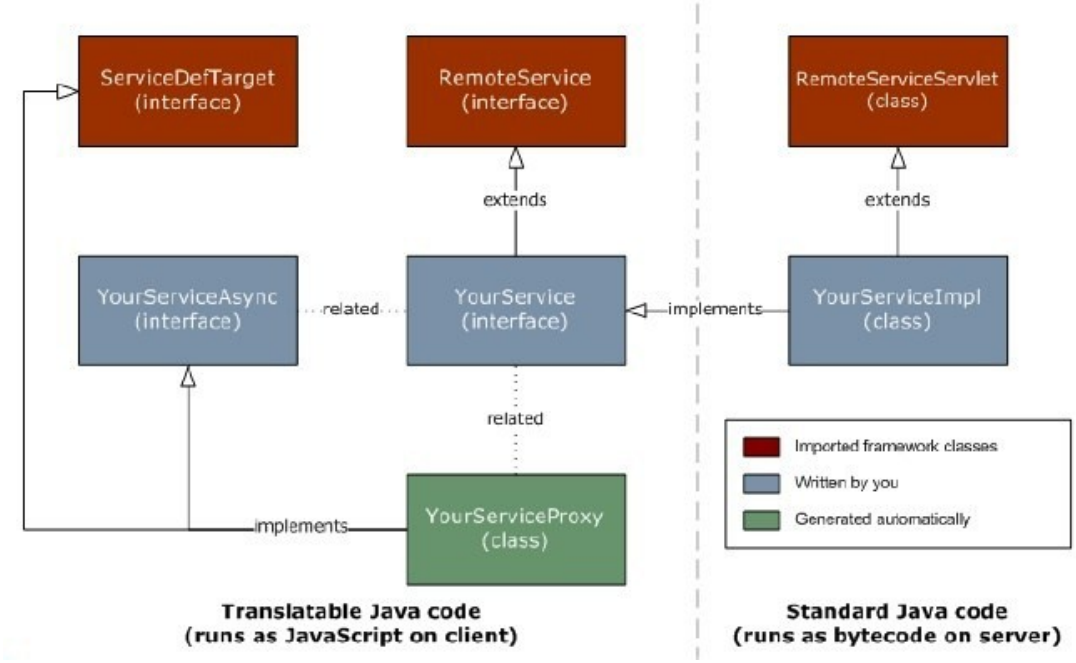


Figura 2.19: GWT RCP

Utilizzando GWT-RPC tutte le chiamate effettuate dalla pagina HTML al server sono asincrone. Questo significa che le chiamate non bloccano il client mentre attende una risposta dal server, ma viene eseguito il codice immediatamente successivo. I vantaggi di effettuare chiamate asincrone rispetto alle chiamate sincrone, si riscontrano in una migliore esperienza per gli utenti finali:

1. Interfaccia utente più reattiva, infatti, a causa del fatto che nei browserWeb il motore JavaScript è generalmente di tipo single-thread, una chiamata sincrona al server genera un "blocco" fino alla conclusione della stessa, rovinando così l'esperienza dell'utente finale.

2. Risulta possibile eseguire altri lavori in attesa della risposta da parte del server; per esempio, è possibile costruire l'interfaccia utente e contemporaneamente recuperare i dati dal server per popolarla, riducendo così il tempo complessivo necessario all'utente per visualizzare i dati sulla pagina.
3. E' possibile effettuare chiamate multiple al server nello stesso tempo.

2.2.6 Application Server : JBOSS AS 5

Figura 2.20: Logo JBOSS

JBoss è un'applicazione server multipiattaforma, utilizzabile da qualsiasi sistema operativo che supporti Java. Implementa tutti i servizi di Java EE (J2EE) ed è completamente open source.

JBOSS: pioniere degli Application Server OpenSource, funziona sia da EJB-Container che da Web-Container grazie all'integrazione di Apache Tomcat.

GWP (Galileo Web Portal) sviluppato dalla SANMARCO INFORMATICA S.P.A. utilizza questa tipologia di server per le proprie applicazioni.

2.2.7 Database: IBM DB2 e il sistema AS400

DB2 è un RDBMS (Relational DataBase Management System) creato da IBM e utilizzato dalle applicazioni sviluppate presso SANMARCO INFORMATICA S.p.a..

Utilizza SQL.

Figura 2.21: Logo IBM DB2

La sua prima versione risale al 1983, e secondo molti è stato il primo prodotto a utilizzare il linguaggio SQL ed è stato scritto in C (linguaggio) e C++.

Il sistema AS/400 (Application System/400) è un minicomputer sviluppato dall'IBM per usi prevalentemente aziendali, come supporto del sistema informativo gestionale.

Nasce nel giugno 1988 come successore del system/38 e dopo oltre 20 anni è ancora in produzione con il nome commerciale di iSeries e dal 2004 di System i. Oggi si chiama semplicemente I.

Il suo successo è stato determinato dai suoi numerosi vantaggi:

- costo relativamente limitato (anche 20.000 euro)
- più di 2 500 applicazioni software disponibili
- una grande stabilità sia in termini di sistema operativo che di hardware.

Viene pubblicizzato per la sua sicurezza e riservatezza dei dati, velocità e trattamento di grandi quantità di record e capacità di gestire centinaia di terminali connessi contemporaneamente.

L'architettura del sistema può essere rappresentata col classico modello a strati tipico dei computer:

1. Al livello più basso troviamo l'hardware. Il sistema AS/400 utilizza processori Power, mentre in precedenza usava processori RISC (dopo il 1998) e ancora prima processori CISC.
2. Al livello immediatamente superiore, si trova uno strato software chiamato Machine Interface (MI), che collega l'hardware al vero e proprio Sistema Operativo. La Machine Interface ha lo scopo di permettere al produttore l'aggiornamento dell'hardware senza per questo dover modificare il Sistema operativo. Ciò è tornato utile quando si è passati dal processore tipo CISC a quello tipo RISC: alcune versioni del Sistema Operativo giravano su entrambi gli hardware.
3. Al livello superiore troviamo il sistema operativo chiamato in origine os/400, ed attualmente (dal 2004) i5/OS.
4. Al di sopra del Sistema Operativo ci sono i cosiddetti "Prodotti programma" forniti da IBM, ovvero tutte le utilità e gli strumenti per la gestione e l'utilizzo del sistema, ad esempio i compilatori dei linguaggi implementati (RPG/RPG ILE, C, C++, Cobol, Fortran, Pascal, Java, etc.), gli strumenti per la programmazione (SDA, SEU, RLU, PDM), il database integrato nel sistema operativo (caratteristica unica di questa macchina), gli strumenti per la gestione dei dati (DFU, SQL, QUERY / QUERY MANAGER) ed altro.
5. Ancora sopra c'è lo strato finale, quello dei programmi applicativi.
6. Attualmente l'ultimo livello di sistema operativo è V7R1M0

2.3 Sviluppo

Consiste nella programmazione ovvero nella traduzione dei modelli del progetto in un programma (codice).

La fase di implementazione delle componenti del sistema ha previsto la stesura del codice di tutte le classi in base al disegno architetturale illustrato nelle sezioni precedenti.

Per la fase di test e collaudo è stato utilizzata la tecnica di tipo alfa-beta.

2.3.1 Software development environment : Eclipse

Eclipse è un ambiente di sviluppo integrato multilinguaggio e multipiattaforma. Ideato da un consorzio di grandi società quali Ericsson, HP, IBM, Intel, MontaVista Software, QNX, SAP e Serena Software, chiamato Eclipse Foundation sullo stile dell'open source.

Eclipse può essere utilizzato per la produzione di software di vario genere, si passa infatti da un completo IDE per il linguaggio Java (JDT, "Java Development Tools") a un ambiente di sviluppo per il linguaggio C++ (CDT, "C/C++ Development Tools") e a plug-in che permettono di gestire XML, Javascript, PHP e persino di progettare graficamente una GUI per un'applicazione JAVA (Eclipse VE, "Visual Editor"), rendendo di fatto Eclipse un ambiente RAD.

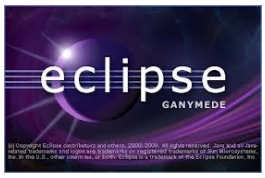


Figura 2.22: Eclipse

La versione utilizzata è la 3.4-Ganymede.

2.3.2 Implementazione delle classi

In questa sezione viene descritta l'architettura delle classi suddivisa per parte client e parte server. Per ciascuna classe vengono descritte le funzionalità principali, omettendo attributi, operazioni e classi interne non rilevanti ai fini della comprensione del ruolo della classe.

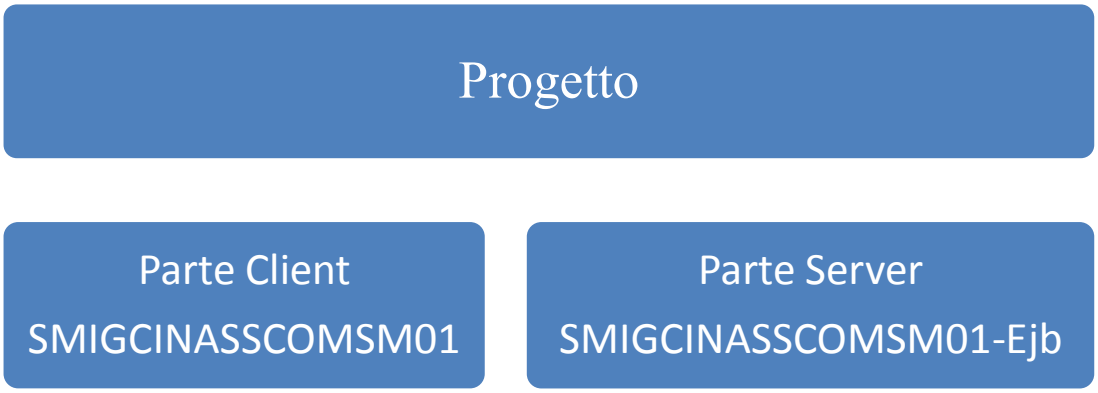


Figura 2.23 : Struttura principale del progetto

Come si può notare dalla figura il progetto è stato suddiviso in due cartelle principali, una cartella contiene la parte di codice eseguita lato client dal browser web, mentre l'altra contiene la parte di codice eseguita lato server dall'application server Jboss.

Parte Client : SMIGCINASSCOMSM01

Figura 2.24: File html e xml della parte client

SMIGCINASSCOSM01.html

Pagina html del modulo GWT. Tutti i componenti disegnati dagli script javascript verranno inseriti dentro questa pagina.

Contiene inoltre alcuni script e css utili all'applicazione. Ad esempio gli script JS e i fogli di stile CSS per il caricamento dei widget del framework GXT.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<html>
    <head>
        <title>Consuntivazione interventi assistenza tecnica</title>
        <meta http-equiv="Pragma" content="no-cache">
        <meta http-equiv="expires" content="0">
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
        <meta name="gwt:property" content="locale=it">
        <link rel="stylesheet" type="text/css" href="gxt/css/gxt-all.css"/>
       <link rel="stylesheet" type="text/css" href="resources.css"/>
    </head>
    <body style="overflow: hidden">
        <script type="text/javascript" language="javascript"
src="it.smi.gpm.SMIGCINASSCOSM01.nocache.js"></script>
        <iframe src="javascript:''" id="__gwt_historyFrame" style="width: 0;
height: 0; border: 0"></iframe>
    </body>
</html>
```

SMIGCINASSCOSM01.gwt.xml

Indica la configurazione di base del modulo GWT sviluppato.

Nel nostro modulo abbiamo specificato i tag:

`<inherits name="logical-module-name"/>`

Permette di specificare i moduli ereditati.

Nel nostro caso il pacchetto "*it.smi.common.Common*" che contiene classi di base e widget standardizzati.

`<entry-point class="classname"/>`

Specifica la classe che avrà la funzione di punto di ingresso per il programma.

Nel nostro caso la classe "*it.smi.gpm.client.SMIGCINASSCOSM01*".

`<extend-property name="client-property-name" values="comma-separated-values"/>`

Permette di settare delle proprietà generali del modulo.

Nel nostro caso viene settata la lingua "*it_IT*"

```
<module>
    <inherits name="it.smi.common.Common" />
    <entry-point class='it.smi.gpm.client.SMIGCINASSCOSM01'/>
    <extend-property name="locale" values="it_IT"/>
    <set-property name="locale" value="it_IT"/>
</module>
```

web.xml

Xml di configurazione in cui viene specificato:

- `<welcome-file-list>` → il file html iniziale dell'applicazione
- `<servlet>` → la servlet della nostra applicazione
- `<servlet-mapping>` → la mappatura della servlet

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4"
xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
    <display-name>SMIGCINASSCOSM01</display-name>
    <welcome-file-list>
        <welcome-file>SMIGCINASSCOSM01.html</welcome-file>
    </welcome-file-list>
    <servlet>
        <servlet-name>SMIGCINASSCOSM01</servlet-name>
        <servlet-
class>it.smi.gpm.server.SMIGCINASSCOSM01ServiceServlet
        </servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>SMIGCINASSCOSM01</servlet-name>
        <url-pattern>/SMIGCINASSCOSM01</url-pattern>
    </servlet-mapping>
</web-app>
```

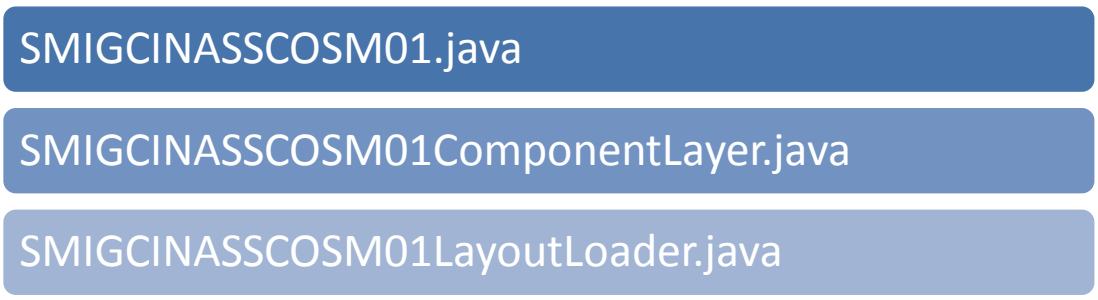


Figura 2.25: Classi Java della parte client

SMIGCINASSCOSM01.java

E' il punto di ingresso della nostra applicazione.

Oltre all'inizializzazione del modulo e delle strutture dati implementa anche tutte le principali funzionalità come ad esempio l'inserimento, la modifica o la cancellazione di un consuntivo.

```
public class SMIGCINASSCOSM01 extends SMIGCINASSCOSM01ComponentLayer {

    public SMIGCINASSCOSM01 () {…} // SMIGCINASSCOSM01

    public void loadParamsBeforeRender () {…}

    protected void init () {…}

    public void print () {…}

    public void loadData () {…}

    public void delete (String cdDitta, String tpRich, String nrRich, String
    nrRiga, String nrSotRiga) {…}

    public void loadMonthTable () {…}

    public void loadMonthMultiTable () {…}

    public String replacePuntoZero (String str) {…}

    public void loadServices () {…}

} // SMIGCINASSCOSM01
```

SMIGCINASSCOSM01ComponentLayer.java

Si occupa del disegno di tutti i componenti grafici e della loro interazione con l'utente. Crea e aggiunge i vari widget come ad esempio il calendario, la griglia e le varie form presenti nell'applicazione.

```
public abstract class SMIGCINASSCOSM01ComponentLayer extends BaseEntryPoint
{

    // Module service.
    public SMIGCINASSCOSM01ServiceAsync service;
    // Hidden components
    public SmiQuery smiQuery;
    public DateTimeConstants constants;
    public String weekOfTheYear = "";
    // Components
    public SmiGcLineCalendar lineCalendar;
    public SmiGcLineCalendar multiLineCalendar;
    public SmiDatePicker dtpCalendar;
    public ComboBox<RowData<String>> cmbOperators;
    public ComboBox<RowData<String>> cmbCustomers;
    public SmiGrid grdFilter;
    public SmiGrid grdMain;
    // Datas
    public ArrayList<RowData<String>> operatorList;
    public ArrayList<RowData<String>> customerList;
    public String operatorDescription;
    // Listeners
    public DeleteListener deleteListener = new DeleteListener ();
    public ShowListener showListener = new ShowListener ();
    public EditListener editListener = new EditListener ();
    public ClearGridSelectionChange clearGridSelectionChange = new
    ClearGridSelectionChange ();
    // Windows
    public SMIGCINASSCOSM01EditBox editBox;
```

```
    public SMIGCINASSCOSM01ListBox listBox;

    public void drawHeader () {…}

    public void drawParameters () {…}

    public void drawRiepilogo () {…}

    public void drawMainGrid () {…}

    public void manageSelectionCombos () {…}

    public void drawFilterGrid () {…}

/*********************************************************************/
/** RENDERERS ********************************************************/
/*********************************************************************/

    GridCellRenderer<RowData<String>> oreGgRenderer = new
    GridCellRenderer<RowData<String>> () {…}; // oreGgRenderer

    GridCellRenderer<RowData<String>> oreIntRenderer = new
    GridCellRenderer<RowData<String>> () {…}; // oreIntRenderer

    GridCellRenderer<RowData<String>> dataRenderer = new
    GridCellRenderer<RowData<String>> () {…}; // dataRenderer

    GridCellRenderer<RowData<String>> deleteCellRenderer = new
    GridCellRenderer<RowData<String>> () {…}; // deleteCellrenderer

    GridCellRenderer<RowData<String>> showCellRenderer = new
    GridCellRenderer<RowData<String>> () {…}; // showCellrenderer

    GridCellRenderer<RowData<String>> editCellRenderer = new
    GridCellRenderer<RowData<String>> () {…}; // editCellRenderer

/*********************************************************************/
/** LISTENERS ********************************************************/
/*********************************************************************/

    BlockCode reloadMonthTable = new BlockCode () {…};

    BlockCode reloadMonthMultiTable = new BlockCode () {…};

    public class DeleteListener extends SelectionListener<ButtonEvent> {…}

    public class ShowListener extends SelectionListener<ButtonEvent> {…}

    public class EditListener extends SelectionListener<ButtonEvent> {…}

    public class CmbOperatorsSelectionChange implements
    Listener<SelectionChangedEvent<ModelData>> {…}

    public class CmbCustomersSelectionChange implements
    Listener<SelectionChangedEvent<ModelData>> {…}

    public class ClearGridSelectionChange implements
    Listener<SelectionChangedEvent<ModelData>> {…}

    public class BtnLoadSelectionListener extends
    SelectionListener<ButtonEvent> {…}
```

```
} // SMIGCINASSCOSM01ComponentLayer
```

SMIGCINASSCOSM01LayoutLoader.java

Si occupa del posizionamento dei componenti grafici nell'interfaccia utente.

In essa per ogni componente vengono specificati gli attributi di posizionamento, visibiltà, abilitazione alla modifica.

```
public class SMIGCINASSCOSM01LayoutLoader extends BaseLayoutLoader {

    public HashMap<String, HashMap<String, String>> getStandardDesign () {

        HashMap<String, HashMap<String, String>> toReturn;
        toReturn = new HashMap<String, HashMap<String, String>> ();

        toReturn.put("lineCalendar",get(5,5,105,980,true,false,"", true));
        toReturn.put("dtpCalendar",get(5,110,190,177,true,false,"",true));
        toReturn.put("txtOperator",get(210,110,17,54,true,false,"",true));
        toReturn.put("cmbOperators",get(225,130,22,290,true,false,"",true));
        toReturn.put("txtCustomer",get(210,165,17,48,true,false,"",true));
        toReturn.put("cmbCustomers",get(225,185,22,290,true,false,"",true));
        toReturn.put("grdFilter",get(560,110,140,200,true,false,"",true));
        toReturn.put("btnLoad",get(610,263,24,100,true,false,"",true));
        toReturn.put("grdMain",get(5,310,400,980,true,false,"",true));
        toReturn.put("boxMain",get(0,0,720,1000,true,false,"",true));

        return toReturn;
    } // getStandardDesign
}
```

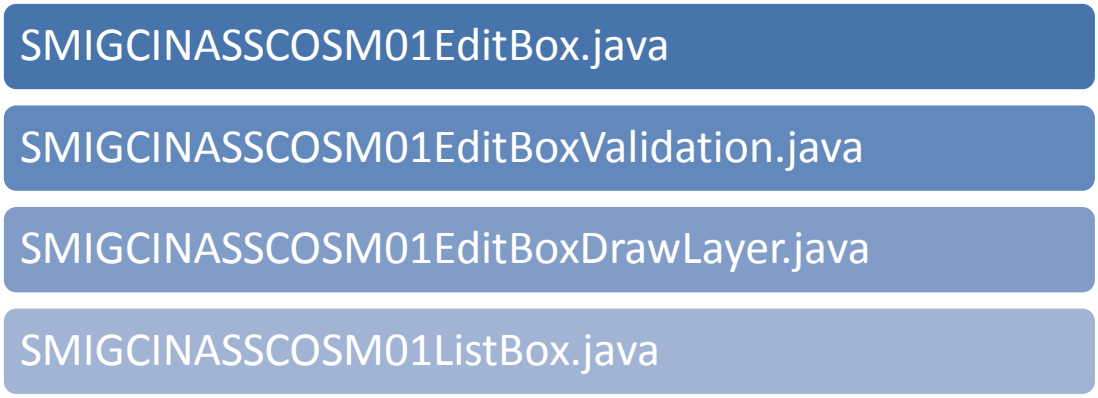


Figura 2.26: Classi Java della parte client

SMIGCINASSCOSM01EditBox.java

E' la form di Inserimento/Modifica del consuntivo. Gran parte del lavoro svolto dall'applicazione viene delegato a questa classe. Contiene e funzionalità di caricamento, salvataggio, copia e incolla consuntivo.

Per semplificare la stesura del codice si è deciso però di suddividere alcuni compiti di questa classe in altre 2 classi

- SMIGCINASSCOSM01EditBoxValidation.java
- SMIGCINASSCOSM01EditBoxDrawLayer.java

```
public class SMIGCINASSCOSM01EditBox extends
SMIGCINASSCOSM01EditBoxValidationLayer {

    public String tecnicoAV = "";

    public SMIGCINASSCOSM01EditBox (SMIGCINASSCOSM01ServiceAsync service,
    long idEnvironment, String cdOperatorLogin, String user, String descOpe)
    {
        super (service, idEnvironment, cdOperatorLogin, user, descOpe);
    } // SMIGCINASSCOSM01EditBox

    public void load (String cdDitta, String tpRich, String nrRich, String
    nrRiga, String nrSotRiga, String codOperator, String codCustomer, Date
    data, String description) {…}

    public void loadAvwra00f () {…}

    public void save () {…}

    final AsyncCallback<String> cbSaveAvwra00f = new AsyncCallback<String>
    () {…};

    final AsyncCallback<HashMap<String, String>> cbLoadAvwra00f = new
    AsyncCallback<HashMap<String, String>> () {…};

    public void copy () {…}

    public void paste () {…}

    public void showMask (boolean visible) {…}

    public void valorize (SmiTextField txt, HashMap<String, String> row,
    String name) {…}

    public void valorize (ComboBox<RowData<String>> cmb, HashMap<String,
    String> row, String name, boolean force) {…}

    public void valorizeCombo (ComboBox<RowData<String>> cmb, String val)
    {…}

    final AsyncCallback<HashMap<String, String>> cbLoadAvint00f = new
    AsyncCallback<HashMap<String, String>> () {…};

    public void changeDay (int offset) {…}

}
```

SMIGCINASSCOSM01EditBoxValidation.java

Si occupa della validazione della form. Dato che la form necessita di una validazione piuttosto complessa come si è visto in fase di Analisi si è deciso di raggruppare l'intera validazione in questa classe.

SMIGCINASSCOSM01EditBoxDrawLayer.java

Si occupa del disegno dei componenti grafici della form con i relativi pulsanti.

SMIGCINASSCOSM01ListBox.java

Gestisce la form di riepilogo consuntivi per intervento.

Figura 2.27: Classi Java della parte client

SMIGCINASSCOSM01Service.java

Interfaccia per il servizio che estende RemoteService.

```
public interface SMIGCINASSCOSM01Service extends RemoteService {…}
```

SMIGCINASSCOSM01ServiceAsync.java

Interfaccia asincrona che sarà chiamata dal codice clientside.

```
public interface SMIGCINASSCOSM01ServiceAsync {…}
```

SMIGCINASSCOSM01ServiceServlet.java

Classe per implementare il codice server-side che estende la classe RemoteServiceServlet ed implementa l'interfaccia: SMIGCINASSCOSM01Service.java.

```
public class SMIGCINASSCOSM01ServiceServlet extends RemoteServiceServlet
implements SMIGCINASSCOSM01Service {}
```

Parte Client : SMIGCINASSCOMSM01

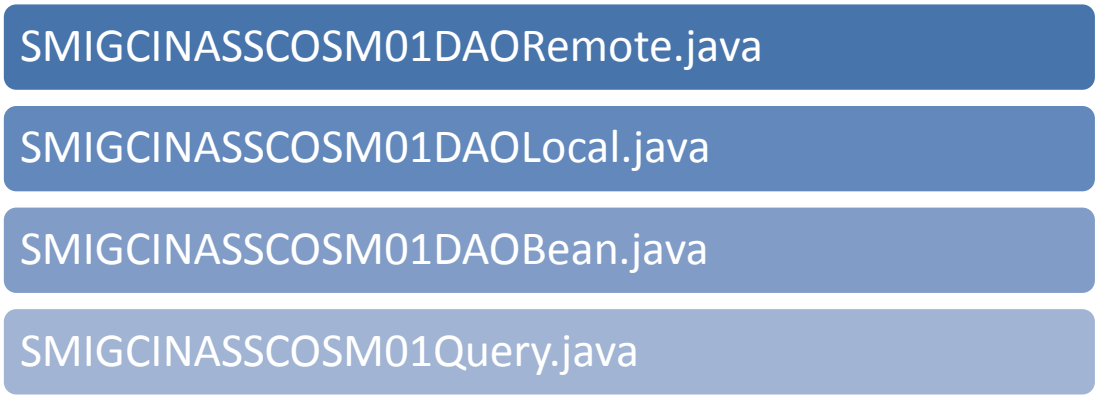


Figura 2.28: Classi Java della parte server

SMIGCINASSCOSM01DAORemote.java

Questa 'interfaccia espone i metodi chiamati per accedere alle operazioni sui dati lato server.

```
public interface ISMIGCINASSCOSM01DAORemote {…}
```

SMIGCINASSCOSM01DAOLocal.java

Questa ’interfaccia espone i metodi chiamati localmente per accedere alle operazioni sui dati lato server.

```
public interface ISMIGCINASSCOSM01DAOLocal {…}
```

I metodi potranno essere invocati solamente da un altro bean.

Analogamente alla remote, l'interfaccia local è pensata per esporre i metodi di business di un bean, con l'importante differenza che in questo caso i metodi esposti potranno essere invocati solamente da un altro bean co-located e non da un client remoto.

SMIGCINASSCOSM01DAOBean.java

Implementa l’interfaccia locale e ne implementa i metodi.

```
@Stateless
@Remote (ISMIGCINASSCOSM01DAORemote.class)
@Local (ISMIGCINASSCOSM01DAOLocal.class)
@RemoteBinding (jndiBinding = "SMIGCINASSCOSM01DAO/remote")
@LocalBinding (jndiBinding = "SMIGCINASSCOSM01DAO/local")
public class SMIGCINASSCOSM01DAOBean extends GpmBaseEjb implements
ISMIGCINASSCOSM01DAOLocal {

    private int getRecordStatus {…} // getRecordStatus

    @Override
    public ArrayList<HashMap<String, Object>> loadListAvwra00f {…}

    @Override
    public String deleteAvwra00f {…}

    @Override
    public ArrayList<HashMap<String, Object>> loadList {…}

    @Override
    public HashMap<String, String> loadAvint00f {…} // loadAvint00f

    @Override
    public ArrayList<ArrayList<HashMap<String, Object>>> loadLoginType {…}

    protected Connection getConnection {…}

    @Override
    public HashMap<String, String> loadAvwra00f {…}

    @Override
    public String saveAvwra00f {…}

    public HashMap<String, String> loadMonthTable {…}

    public ArrayList<HashMap<String, String>> loadMonthMultiTable {…}

    @Override
    public HashMap<String, String> loadAvint00fPrint {…}

    @Override
```

```
    public ArrayList<HashMap<String, String>> loadAvwra00fPrint {…}

    @Override
    public ArrayList<String> loadRiferimenti {…}

} // SMIGPMPRATCOSM02DAOBean
```

SMIGCINASSCOSM01Query.java

Contiene dei metodi che in base ai parametri passati e un oggetto di tipo StringBuilder costruiscono le query da effettuare.

```
public class SMIGCINASSCOSM01Query {
public class SMIGCINASSCOSM01Query {

    public static String listSql {…}

    public static String detailsAvint00f () {…}

    public static String saveAvwra00f (boolean useOperator) {…}

    public static String updateAvwra00f (boolean useOperator) {…}

    public static String listAvwra00fSql (boolean useOperator) {…}

    public static String monthListAvwra00f (boolean useOperator) {
        StringBuilder toReturn = new StringBuilder ();
        toReturn.append ("SELECT  DTITWA, COUNT (DISTINCT FRIPWA) AS
NUM_DIFF, ");
        toReturn.append ("SUM ((OVAAWA - FVAAWA) - (OVADWA - FVADWA) +
(OVRAWA - FVRAWA) - (OVRDWA - FVRDWA)) AS OV, ");
        toReturn.append ("SUM ((OL1AWA - OL1DWA) + (OL2AWA - OL2DWA) +
(OL3AWA - OL3DWA)) AS OL ");
        toReturn.append ("FROM AVWRA00F WHERE ");
        toReturn.append ("DTITWA >= ? AND DTITWA < ? AND ");
        if (useOperator) {
            toReturn.append ("CTEAWA = ? ");
        } else {
            toReturn.append ("FOMNWA = ? ");
        } // if - else
        toReturn.append ("GROUP BY DTITWA");
        return toReturn.toString ();
    } // monthListAvwra00f

}
```

2.3.3 Test e collaudo

L'attività di test non ha previsto l'utilizzo di sistemi di testing automatici ma si è svolta grazie alla collaborazione di personale addetto dell'azienda SanmarcoInformatica ed alla collaborazione con il cliente stesso.

E' stato utilizzato un collaudo di tipo Alfa-Beta.

La fase di test e collaudo è stata suddivisa in due fasi:

a) Alfa test
b) *Beta test*

Gli Alfa test consistono in un primo collaudo interno da parte di personale dell'azienda produttrice del software.

Terminata questa prima fese di collaudo e una volta avvenuta la correzione delle eventuali segnalazioni si passa alla fase successiva.

I Beta test consistono in un rilascio controllato a pochi utenti selezionati. Nel nostro caso l'azienda cliente del software realizzato ha svolto tale funzione per un breve periodo.

Sono emerse numerose imperfezioni nel programma sviluppato dovute soprattutto alla mia inesperienza nel settore, nessuna di queste però gravi o bloccanti.

Ho apportato le correzioni in tempi relativamente brevi senza quindi compromenttere i tempi di consegna previsti.

Inoltre il cliente nei primi giorni di utilizzo del programma si è dimostrato molto disponibile nel segnalare alcune imperfezioni che sono state corrette anch'esse in breve tempo.

2.3.4 Il prodotto finito

In questa sezione possiamo visualizzare i principali print screen dell'applicazione. L'applicazione ha completamente raggiunto gli obiettivi proposti con considerevole soddisfazione da parte del cliente risultando flessibile, configurabile e facilmente estendibile con nuove funzionalità.

Di seguito vengono mostrati i principali print-screen realizzati durante la fase di test del prodotto..

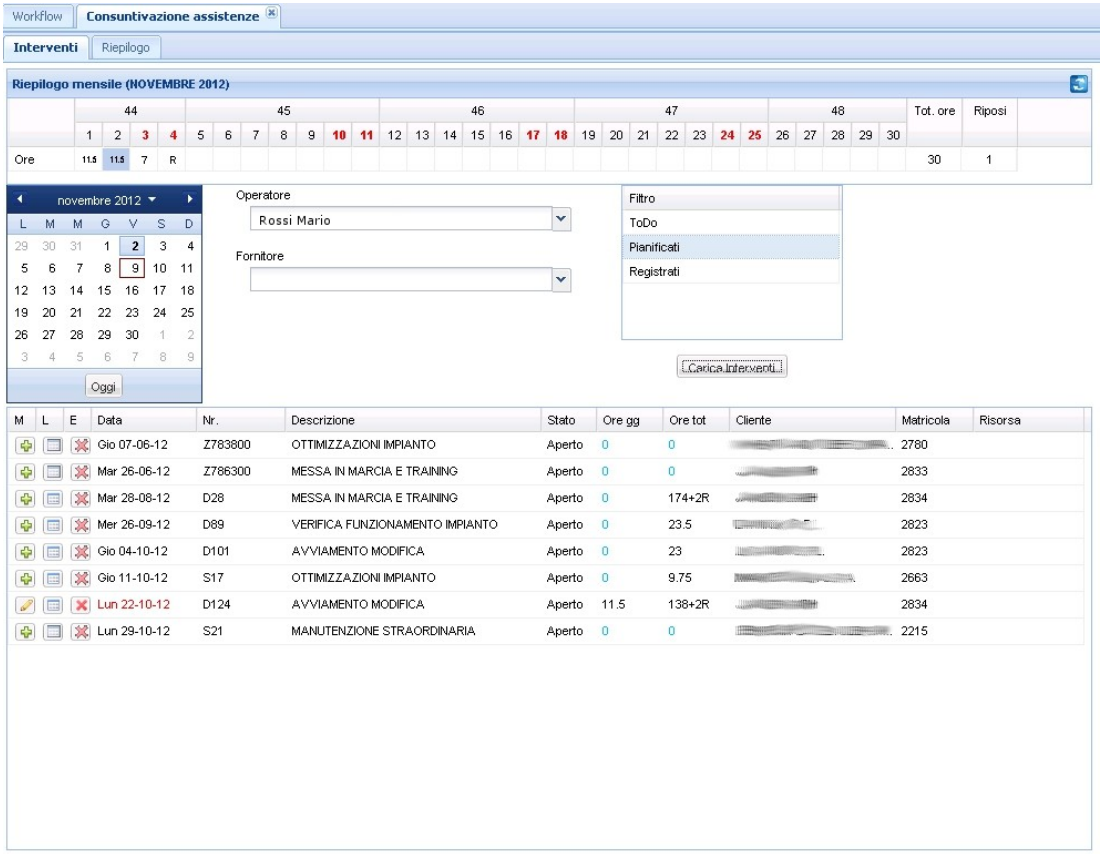


Figura 2.29: Home Page

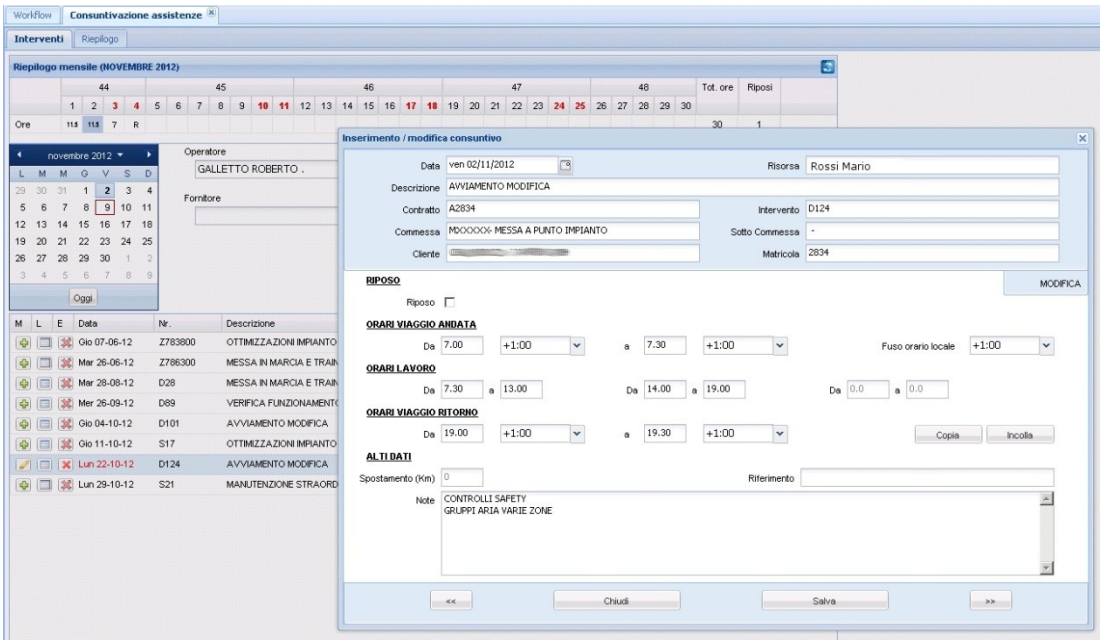


Figura 2.30: Modifica Consuntivo

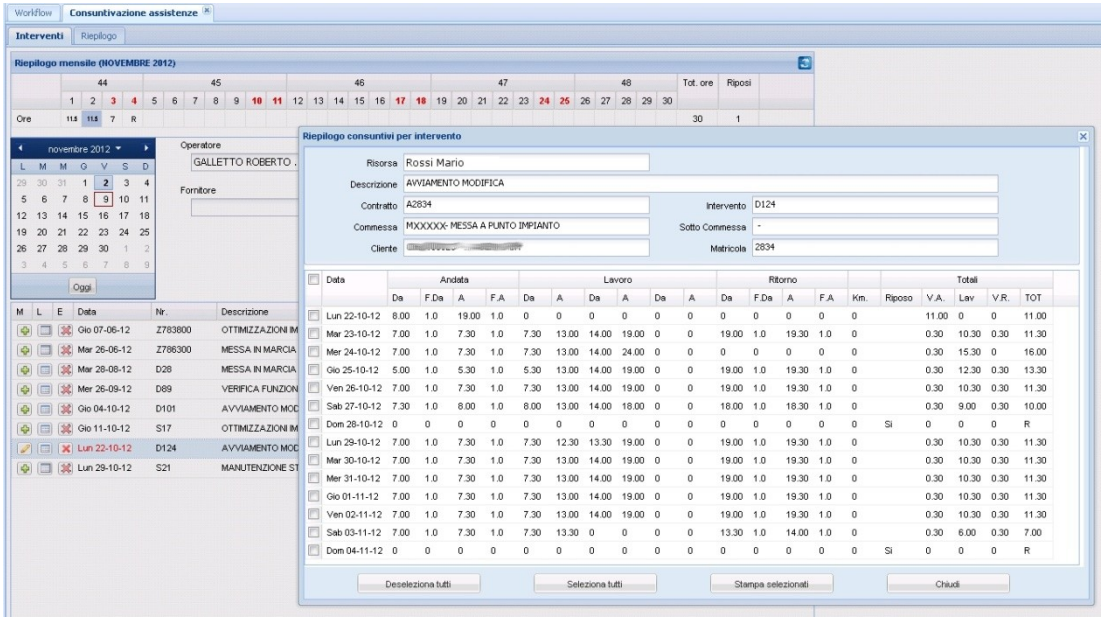


Figura 2.31: Riepilogo per intervento

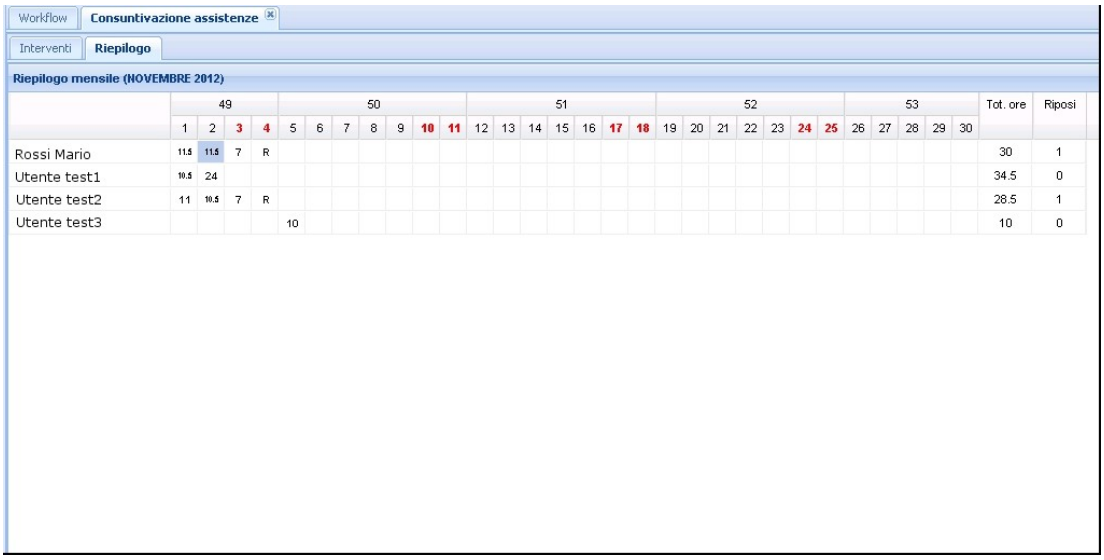


Figura 2.32: Riepilogo mensile

Bibliografia

[1] Cay Horstmann: "Concetti di informatica e fondamenti di Java", Apogeo.
[2] Robert Cooper Robert, Charles Collins: "GWT in Practice", Manning Publications Co. .
[3] Robert Hanson, Adam Tacy: "GWT in Action: Easy Ajax with the Google Web Toolkit", Manning Publication Co. .
[4] Ryan Dewsbury: "Google Web Toolkit Application", Prentice Hall.

Sitografia

[1] Oracle: http://www.oracle.com/index.html
[2] JBoss server: http://www.jboss.org/
[3] IBM DB2: http://www-01.ibm.com/software/data/db2/
[4] EJB: http://www.oracle.com/technetwork/java/javaee/ejb/index.html
[5] GWT Google Web Toolkit: https://developers.google.com/web-toolkit/
[6] GWT-Gxt: http://www.sencha.com/