Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione
Corso di Laurea in Ingegneria Informatica

Realizzazione di un applicativo web tramite utilizzo di JSF 2.0 e libreria PRIMEFACES

Laureando:
Thien Kim Ly

Relatore:
Ch.mo Prof. Sergio Congiu

Anno accademico 2012/2013

Sommario

L'obiettivo di questa tesi è la realizzazione di un applicativo web destinato ai dispositivi mobile, usando JavaServer Faces 2.0 e PrimeFaces. L'applicativo è stato realizzato durante un tirocinio di laurea triennale della durata di 250 ore. All'inizio verrà fatta una breve introduzione sull'azienda in cui è stato svolto il tirocinio, per poi passare alla presentazione dell'applicativo realizzato. Verranno successivamente presentate le principali tecnologie usate per realizzare tale applicativo, soffermandosi sui framework più importanti ovvero JSF, PrimeFaces e jQuery Mobile. Verrà quindi illustrato il lavoro svolto partendo da una fase di analisi e studio, per poi passare alla realizzazione concreta. Dopo aver illustrato la struttura Model-View-Controller dell'applicativo, sarà presentata la composizione delle sue pagine. Verranno successivamente mostrati alcuni particolari componenti, quali l'integrazione della mappa di Google Maps e i grafici. Verranno quindi discussi i principali problemi incontrati e le possibili soluzioni, per arrivare alla fase di test dell'applicativo mediante dispositivi mobile. Infine verrà fatta una breve conclusione contenente i risultati ottenuti e una valutazione critica sul lavoro svolto.

Indice

Elenco delle figure

Capitolo 1

Introduzione

Tra il 23 luglio 2012 e il 17 ottobre 2012 ho svolto un tirocinio universitario presso l'azienda Sanmarco Informatica di Grisignano di Zocco, a Vicenza. L'obiettivo del tirocinio è stato quello di realizzare una nuova versione dell'applicativo web iGalileo, un software creato precedentemente dall'azienda, che permette agli agenti di commercio di consultare in qualsiasi momento i dati della propria azienda tramite tablet o smartphone. La motivazione principale che ha spinto alla realizzazione di questa nuova versione è stata la possibilità di sfruttare le potenzialità di nuove tecnologie, quali JavaServer Faces 2.0, un framework Java realizzato per applicazioni web, e PrimeFaces, la distribuzione di JSF attualmente più usata al mondo. Quest'ultima in particolare, grazie ai nuovi componenti creati appositamente per i dispositivi mobile, ha permesso una ristrutturazione grafica dell'applicativo. Infatti il nuovo iGalileo avrebbe dovuto presentare una nuova interfaccia utente, mantenendo però la maggior parte delle funzionalità del vecchio applicativo.

1.1 L'azienda

Sono venuto a conoscenza dell'azienda grazie allo STAGE-IT, un evento promosso dall'Università di Padova per dare la possibilità agli studenti di venire a contatto con più di quaranta aziende operanti nel settore IT (Information Technology) e di sostenere dei colloqui con queste.

Nata negli anni '80 come software house specializzata nello sviluppo di applicazioni gestionali per aziende manifatturiere, Sanmarco Informatica si è evoluta in un crescendo di esperienze di successo e di scelte imprenditoriali che individuano nella specializzazione del proprio capitale umano l'elemento centrale. Oggi, con gli oltre 280 dipendenti e collaboratori diretti, 4 sedi presso Grisignano di Zocco, Reggio Emilia (RE), Tavagnacco (UD) e Vimercate (MB) e 12 distributori sul territorio nazionale, è il partner ideale per la consulenza e la fornitura di soluzioni specializzate a supporto della riorganizzazione di processi in numerosi ambiti aziendali e professionali. Il principale applicativo sviluppato è Galileo Erp, soluzione gestionale destinata alle seguenti categorie merceologiche:

- Industria metalmeccanica con produzione per lotti ripetitivi, su commessa, terzista o impiantista;
- Produzione settore chimico e stampaggio plastico;

- Produzione di mobili, serramenti, porte, cancelli, parquet;
- Produzione vitivinicola, distillerie, liquorifici;
- Produzione alimentare, dolciaria, lattiero casearia, salumifici, prosciuttifici;
- Produzione e commercializzazione confezioni;
- Imprese edili, general contractor;
- Commerciale in genere, distribuzione, vendita al banco, vendita verso GDO, tentata vendita, vendita via web.

Sanmarco Informatica offre inoltre una serie di corsi per la formazione professionale. Svolgere attività con i principali software operativi, gestire componenti hardware, conoscere i linguaggi di programmazione, saper realizzare siti web, conoscere le reti e diventarne tecnici sono alcune delle competenze che è possibile acquisire con tali corsi.

Per ulteriori informazioni si visiti il sito ufficiale [1].

1.2 Progetto e sue funzionalità

Oggigiorno la maggior parte delle persone possiede uno smartphone o un tablet. Chi li usa per lavoro, chi per svago, chi per comunicare in modo rapido con amici e conoscenti; ormai nessuno riesce a farne a meno. Questi dispositivi infatti sono pratici, semplici da utilizzare, ma soprattutto comodi da portare sempre con sé. L'esigenza degli agenti di commercio di avere sottomano i dati della propria azienda in qualsiasi posto e in qualsiasi momento, ha portato quindi alla creazione di iGalileo, che dà la possibilità di consultare le informazioni di cui si ha bisogno in pochi secondi, senza la scocciatura di dover accendere un personal computer o di dover sfogliare carte su carte. La possibilità di tenere sotto controllo l'andamento del proprio fatturato, di controllare la situazione dei pagamenti dei propri clienti, di visionare lo stato degli ordini effettuati e di visualizzare una lista degli articoli in vendita sono solo alcune delle funzionalità di questo applicativo. Inoltre questi dati sono costantemente aggiornati grazie al gestionale di Sanmarco Informatica.

Per poter utilizzare questo software un agente deve innanzitutto possedere un proprio account. Tramite le proprie credenziali di accesso (nickname e password) da immettere nella schermata di login, l'agente potrà quindi entrare nel menu principale e da qui iniziare la navigazione attraverso il sito. Nel menu è presente una lista composta da cinque voci (vedi Fig. 1.1), ciascuna delle quali apre una diversa schermata in base alle funzionalità richieste. Le voci sono le seguenti:

1. Comunicazioni: mostra le ultime news (ad esempio relative alle vendite) ordinate per categoria;
2. Schede Clienti: permette di selezionare un cliente, dopo aver eseguito una ricerca, e di mostrarne i relativi dati anagrafici, amministrativi, ecc.;
3. Stato Ordini: permette di consultare gli ordini effettuati dai propri clienti;

Figura 1.1: Menu principale di iGalileo

4. Scadenzario: permette di controllare lo stato dei pagamenti, divisi in scadenze (pagamenti da effettuare entro una data futura) e scaduti (pagamenti insoluti);

5. Fatture: permette di controllare il fatturato.

Nel menu principale è inoltre possibile accedere alla schermata opzioni, che dà la possibilità di cambiare la lingua, di mostrare il nickname dell'utente loggato e di cambiare eventualmente la password. Sempre nel menu principale è presente un grafico che rappresenta, mese per mese, il trend del fatturato dell'anno attuale confrontato con quello dell'anno precedente.

La sezione schede clienti è la più importante poiché offre svariate funzionalità. Permette infatti di visualizzare le informazioni su ordini, fatture e scadenze relative al solo cliente selezionato. È possibile inoltre fare una ricerca degli articoli venduti per poi visualizzarne in dettaglio le caratteristiche.

Grazie all'integrazione della mappa di Google Maps, c'è infine la possibilità di geolocalizzare la posizione in cui ci si trova e di tracciare il percorso da tale punto all'indirizzo del cliente selezionato, in modo da semplificare notevolmente all'agente il ritrovamento dell'abitazione del cliente.

1.3 Il mio ruolo

Durante il mio tirocinio ho rivestito il ruolo di programmatore/sviluppatore web. Pertanto fin da subito sono stato inserito all'interno del gruppo web, un gruppo di colleghi addetti appunto allo sviluppo di applicativi web.

Durante lo svolgimento del tirocinio non sono stato solo, poiché ho lavorato sotto la supervisione del mio tutor aziendale, Francesco Zorzan, e con l'aiuto del collega Daniele Lovato. Il loro aiuto è stato essenziale per la buona riuscita del progetto, poiché le mie conoscenze sul settore web erano veramente basilari. Infatti ho dovuto dedicare i primi giorni alla lettura di alcune guide sulla realizzazione di tali applicativi e allo studio delle documentazioni delle nuove librerie da usare.

Dopo aver imparato le nozioni base su PrimeFaces ho creato un piccolo e semplice progetto per testare il funzionamento di alcuni componenti di tale libreria. Successivamente i miei compiti sono stati quelli di analizzare la struttura del vecchio applicativo, poiché la struttura del nuovo iGalileo sarebbe rimasta quasi identica, e di testare il funzionamento del prodotto finito per capire le funzionalità che avrebbe dovuto avere anche il nuovo prodotto. Da qui in poi inizia il lavoro vero e proprio, cioè la realizzazione del nuovo applicativo.

Per quanto riguarda l'organizzazione, il tutor mi assegnava di giorno in giorno vari obiettivi da portare a termine entro una data prestabilita e varie indicazioni su come svolgere i miei compiti. I primi giorni sono stati sicuramente i più duri, poiché era la prima volta che affrontavo un lavoro del genere ed ero ancora inesperto. D'altra parte sono sempre riuscito a portare a termine il mio lavoro grazie anche ai consigli e ai suggerimenti dei miei colleghi.

Capitolo 2

Tecnologie

2.1 Tecnologie usate

Vengono di seguito presentate le varie tecnologie adottate per lo sviluppo del progetto:

- Eclipse Juno [2]: ambiente di sviluppo integrato (IDE) multi-linguaggio e multi-piattaforma;
- Java: linguaggio di programmazione orientato agli oggetti;
- Apache Tomcat 7 [3]: contenitore servlet open source che implementa le specifiche JavaServer Pages (JSP) e Servlet di Sun Microsystems, fornendo una piattaforma per l'esecuzione di applicazioni Web sviluppate nel linguaggio Java;
- JavaServer Faces (JSF) 2.0: tecnologia Java basata sul design pattern architetturale Model-View-Controller (MVC);
- PrimeFaces: framework di componenti JSF;
- jQuery Mobile: framework cross-browser per le applicazioni web.

2.2 JSF

JavaServer Faces è un framework Java EE (Enterprise Edition) per lo sviluppo del layer presentazione delle applicazioni web lato-server. Il suo scopo è quello di semplificare la realizzazione dell'interfaccia utente (UI) delle applicazioni web. Le caratteristiche principali di JSF sono le seguenti:

- Paradigma MVC: JSF è basato sul pattern architetturale Model-View-Controller;
- Component oriented: viene messo a disposizione un insieme di componenti predefiniti da utilizzare per sviluppare l'interfaccia web, seguendo così la logica RAD (Rapid Application Development);
- Event driven: JSF è basato sugli eventi, ovvero azioni che l'utente compie sui componenti di interfaccia. Registrando dei listener su questi eventi è possibile la loro gestione lato-server;

- RenderKits: JSF permette di separare il comportamento dei componenti di interfaccia dalla loro rappresentazione grafica, utilizzando classi diverse. Una classe Renderer si occupa di renderizzare i componenti, ovvero di rappresentare il loro aspetto grafico. È possibile pertanto utilizzare un Render kit diverso senza modificare il funzionamento di tali componenti;

- Validazione: JSF mette a disposizione una serie di validatori standard per validare automaticamente i dati inseriti dall'utente nell'UI (ad esempio nelle form) e memorizzare il proprio stato;

- Binding e JavaBean: i dati immessi dall'utente nell'interfaccia web lato-client devono poi essere contenuti in oggetti lato-server che modellano le entità sulle quali devono andare ad agire le elaborazioni del livello di Model (del pattern MVC). Pertanto i dati devono andare a popolare opportuni attributi di JavaBean. JSF fornisce gli strumenti per effettuare questa operazione di "binding" in modo automatico e dichiarativo;

- Facelets: JSF offre un'integrazione standard con Facelets, un linguaggio di dichiarazione delle pagine XHTML;

- Ajax: JSF (2.0+) supporta pienamente Ajax (Asynchronous JavaScript and XML), una tecnica di sviluppo per la realizzazione di applicazioni web.

Uno dei punti di forza di JSF è sicuramente la standardizzazione. Infatti a differenza di molti altri framework dedicati allo sviluppo di applicativi web come Struts e Spring, JSF fa parte della specifica Java EE, ed è quindi il framework standard per lo sviluppo dello strato web.

Come già detto, per contenere i dati immessi nell'interfaccia utente vengono utilizzati degli oggetti incapsulati in classi JavaBean chiamate backing-bean. Queste classi devono avere un costruttore senza argomenti e devono essere serializzabili, ovvero capaci di salvare e ripristinare il proprio stato in modo persistente. Questi backing-bean definiscono una serie di proprietà, cioè degli attributi privati accessibili tramite metodi get e set. Tramite queste proprietà è possibile eseguire le funzioni più comuni, come la validazione dei dati dei componenti, la gestione degli eventi e le elaborazioni del Model. Per collegare il valore di un componente alla proprietà di un bean o per fare riferimento ai metodi di un backing-bean si utilizza l'Expression Language (EL), un linguaggio di scripting che permette di accedere ai JavaBeans, tramite espressioni valore o espressioni metodo. Poiché il ciclo di vita di una pagina JSF è suddivisa in diverse fasi è necessario ritardare la valutazione delle espressioni finché viene raggiunta la fase appropriata. Per fare ciò gli attributi dei tag devono utilizzare la sintassi differita attraverso il delimitatore #{}.

Infine per quanto riguarda il ciclo vita dell'elaborazione di una richiesta JSF prevede 6 fasi (vedi Fig. 2.1):

1. Restore View: durante questa fase si procede alla costruzione della vista della pagina, dei gestori degli eventi, dei validatori e dei componenti presenti nella vista;

2. Apply Request Values: durante questa fase vengono memorizzati i dati associati alla richiesta nei componenti associati ai vari elementi della pagina;

3. Process Validation: è la fase di validazione dei valori inseriti dall'utente secondo le logiche di validazione definite;

4. Update Model Values: vengono aggiornate le proprietà dei backing-bean con i valori inseriti dall'utente se questi hanno superato la fase di validazione;

5. Invoke Application: in questa fase vengono gestiti gli eventi ed eseguite le relative action, dovuti alle azioni effettuate dall'utente sull'interfaccia;

6. Render Response: viene memorizzato lo stato della vista e generata la risposta per il client.

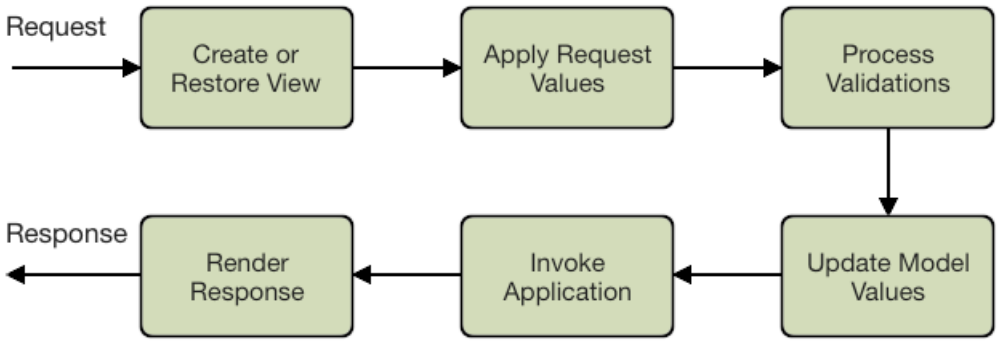

Figura 2.1: Le 6 fasi del ciclo di vita di JSF

La versione precedente di iGalileo utilizzava JSF 1.1, mentre per sviluppare la nuova versione abbiamo utilizzato JSF 2.0, poiché questa release presenta diversi vantaggi rispetto alla precedente.

Innanzitutto con JSF 2.0 è stata introdotta la tecnologia Facelets, sebbene questa sia compatibile anche con le versioni precedenti. Sono stati risolti i problemi di compatibilità con Ajax che ora è supportato pienamente. È stata introdotta la configurazione basata sulle annotazioni, che semplifica notevolmente la configurazione dei Bean non richiedendo più l'utilizzo di file XML. Sono stati infine introdotti nuovi scope, ovvero i cicli di vita delle istanze dei Bean, nuovi tipi di navigazione, il supporto alla richiesta GET e molte altre funzioni.

Per una trattazione esaustiva si rimanda al sito ufficiale [4] e all'articolo [5].

2.3 Facelets

Spesso le pagine di una applicazione web hanno un layout molto simile con elementi che rimangono invariati, come ad esempio gli header e i footer. Da qui nasce il desiderio di non dover riscrivere più volte porzioni identiche di codice ma di poter invece riutilizzare le parti di codice già scritte, in modo da far risparmiare tempo prezioso ai programmatori. La soluzione che JSF ci offre si chiama Facelets, un linguaggio di dichiarazione delle pagine che viene usato per costruire viste JSF. Questa tecnologia è compatibile con qualsiasi versione di JSF, è indipendente dal web container (o servlet container) e ha un tempo

di compilazione molto veloce. Inoltre supporta il templating, ovvero l'utilizzo di modelli preimpostati per costruire le pagine, e l'Expression Language.

Come visto in precedenza anche Facelets dispone di una libreria di tag. Tra i tag più importanti troviamo innanzitutto ui:composition che permette di incapsulare il contenuto da utilizzare per popolare un'altra pagina chiamata template, ovvero un modello che descrive la struttura che tutte le pagine che ne faranno riferimento dovranno seguire. Per fare ciò bisogna specificare il suo path nell'attributo template di ui:composition. Altri due componenti molto utili sono ui:insert e ui:define. Il primo va inserito all'interno di un template, mentre il secondo in un'altra pagina all'interno di ui:composition. Entrambi possiedono un attributo name e il contenuto di ui:insert sarà sostituito dal contenuto del corrispondente ui:define avente lo stesso name. Il componente ui:include ha una funzione analoga a ui:insert con la differenza che va inserito in una pagina che non sia un template. Infine il tag ui:param viene utilizzato per il passaggio di oggetti tra componenti Facelets, dopo aver definito per ciascun oggetto un nome e un valore.

Qui sono stati presentati solo i tag usati in iGalileo, ma ne esistono molti altri. Per approfondire si veda [6].

2.4 PrimeFaces

PrimeFaces è una suite open source di componenti JSF con varie estensioni, creata da Prime Teknoloji, una compagnia turca di sviluppo software. Questa libreria presenta un ricco set di componenti per le applicazioni web, tra cui editor HTML, finestre di dialogo, meccanismi per l'auto-completamento, grafici, calendari e molto altro. Grazie all'integrazione con jQuery, PrimeFaces supporta Ajax e il rendering parziale delle pagine. Può vantare inoltre un'ampia documentazione e una community grande, attiva e sempre pronta a dare supporto agli utenti. Altre note di merito sono la possibilità di creare dei temi grafici personalizzati e di utilizzare un UI kit mobile per realizzare applicazioni web orientate ai dispositivi mobili. PrimeFaces, anche grazie alla sua semplicità di utilizzo e alle sue prestazioni, ha saputo abbattere in poco tempo la concorrenza di rivali come RichFaces ed IceFaces (vedi Fig. 2.2).

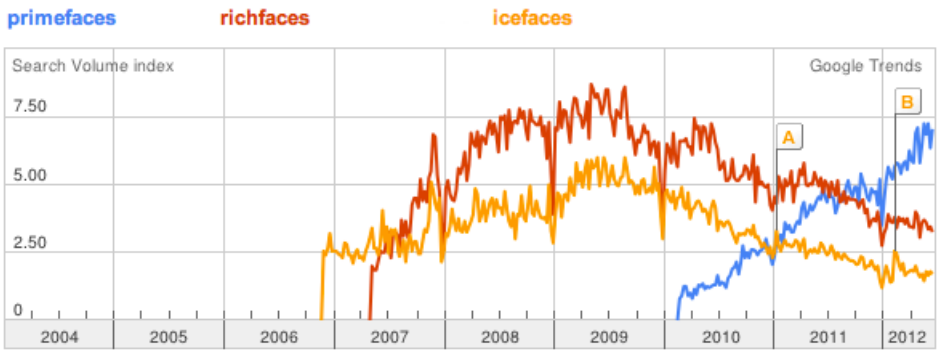

Figura 2.2: Grafico che paragona la popolarità di PrimeFaces con i suoi concorrenti

Tutti questi motivi hanno portato ad optare per l'uso di PrimeFaces nella nuova versione di iGalileo, al posto di RichFaces, il framework usato invece nella versione precedente.

Un altro framework utilizzato nel nuovo iGalileo è PrimeFaces Mobile, un UI kit basato a sua volta su jQuery Mobile, pensato per implementare le pagine JSF che sono ottimizzate per i dispositivi mobile mantenendo un aspetto nativo.

Per maggiori dettagli e per la documentazione si rimanda al sito ufficiale [7].

2.5 jQuery Mobile

jQuery è una libreria di funzioni javascript multi-browser, che si propone come obiettivo quello di semplificare la programmazione lato client delle pagine HTML. Fornisce un codice sintetico, limitando al minimo l'estensione degli oggetti in modo da mantenere la massima compatibilità con altre librerie. "Write less, do more" è infatti il motto di jQuery che sottolinea allo stesso tempo la semplicità e la potenza di questo framework.

jQuery Mobile invece è un framework web basato su jQuery e ottimizzato per smartphone e tablet. La motivazione che ha portato alla creazione di jQuery Mobile è stato il bisogno di poter creare un'unica applicazione che supporti più sistemi operativi, mantenendo nel contempo un'aspetto nativo. Chiaramente questo riduce notevolmente i tempi di realizzazione di un software, in quanto non c'è bisogno di creare diversi applicativi per diversi sistemi operativi.

I principali sistemi operativi supportati da jQuery Mobile sono iOS, Android, BlackBerry, Windows Phone, Symbian e Meego. Supporta quindi i dispositivi mobili attualmente più diffusi, in particolare iPad, iPhone e Samsung Galaxy Tab. Proprio questi sono infatti i dispositivi che abbiamo usato per testare l'iGalileo durante il mio tirocinio. Infine come già accennato jQuery è un framework multi-browser, cioè è compatibile con più browser, in particolare con le ultime versioni di Firefox, Chrome, Safari, Internet Explorer e Opera.

Per maggiori dettagli e per la documentazione si rimanda al sito ufficiale [8].

Capitolo 3

Lavoro svolto

3.1 Analisi

Lo sviluppo vero e proprio del progetto è stato preceduto da una sessione di studio della documentazione delle nuove librerie e da un lavoro di analisi del vecchio applicativo preesistente. Lo studio è stato affiancato da una spiegazione del tutor aziendale, che ha illustrato la struttura del vecchio iGalileo, facendo una panoramica delle classi principali di cui l'applicativo è composto, soffermandosi in particolar modo sulla struttura MVC. Al termine dell'analisi del vecchio iGalileo il tutor ha preparato l'ambiente di lavoro e creato la struttura del nuovo progetto. La scelta dell'ambiente di lavoro è caduta su Eclipse Juno, ultima release del noto IDE per programmare in Java, uno dei migliori grazie alla sua efficienza, stabilità e alle sue numerose funzionalità.

3.2 Struttura applicativo

Poiché lo scopo del nuovo iGalileo è quello di presentare una nuova interfaccia grafica usando le nuove librerie ma mantenendo lo stesso funzionamento del vecchio progetto, esso presenta gli stessi package e le stesse classi, anche se successivamente è stata apportata qualche piccola modifica.

Come già accennato, la struttura dell'applicativo è basata sul paradigma Model-View-Controller (MVC), un pattern architetturale molto diffuso nello sviluppo di interfacce grafiche di sistemi software orientati agli oggetti. Questo pattern è basato sulla separazione dei compiti fra i componenti software che interpretano tre ruoli principali (vedi Fig. 3.1):

- Model: fornisce i metodi per accedere ai dati utili all'applicazione;
- View: gestisce l'interfaccia utente e visualizza i dati contenuti nel Model;
- Controller: coordina l'applicazione interagendo con il View e il Model; in particolare riceve i comandi dell'utente (attraverso il View) e li attua modificando lo stato degli altri due componenti.

Questa struttura implica diversi vantaggi, tra cui la separazione tra logica applicativa e interfaccia utente e la possibilità di riutilizzare gli stessi componenti per più applicazioni. Proprio quest'ultimo fatto ha permesso il riutilizzo delle classi del vecchio iGalileo,

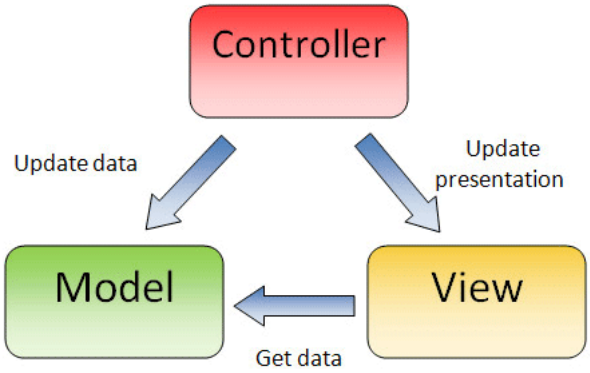

Figura 3.1: Struttura del paradigma Model-View-Controller

dandomi modo di concentrarmi fin da subito sulla realizzazione dell'interfaccia utente, senza preoccuparmi del funzionamento dell'applicativo. Ciò sottolinea come sia importante tenere i tre strati il più possibile separati e indipendenti fra loro, facilitando la scalabilità e la manutenzione dell'applicazione.

Il progetto è formato innanzitutto dalla cartella Java Resources, contenente a sua volta le cartelle src e Libraries. La prima contiene una serie di package, ciascuno formato da un gruppo di classi Java. Queste classi costituiscono il Model. Tra i package più importanti troviamo quelli dedicati ai backing-bean, alle entità, ai manager e ai data provider. La seconda invece, come è intuibile dal nome, contiene tutte le librerie che saranno usate, tra cui PrimeFaces.

Poi è presente la cartella WebContent, formata dalle cartelle WEB-INF e mobile. La prima contiene una serie di file di configurazione, mentre la seconda contiene vari file XHTML, che corrispondono alle pagine che vanno a formare l'interfaccia utente dell'applicativo. Queste pagine rappresentano il View, e proprio su queste pagine è stata fatta la maggior parte del mio lavoro. Sono infine presenti altre cartelle di minore importanza, perlopiù file di configurazione.

Il ruolo di Controller invece è assegnato a una servlet, ovvero un programma residente in un server in grado di gestire le richieste generate da uno o più client attraverso uno scambio di messaggi. Nel nostro caso la servlet è contenuta all'interno di Apache Tomcat, un web server che fa anche da contenitore servlet.

3.3 Primi passi con PrimeFaces

Prima di addentrarmi con la realizzazione di iGalileo però, ho dovuto verificare quanto appreso creando un piccolo progetto molto semplice (che chiameremo Test), un applicativo web composto da poche pagine, in cui inserire alcuni componenti di PrimeFaces Mobile.

Questa libreria prevede una struttura a pagine XHTML basate su viste (view) JSF. Una pagina (page) è formata da elementi XHTML, composti a loro volta da un tag di apertura, da eventuali attributi coi corrispondenti valori, da un contenuto e da un tag di chiusura. I tag di apertura e di chiusura definiscono rispettivamente l'inizio e la fine dell'elemento, mentre gli attributi forniscono informazioni aggiuntive.

Ogni pagina deve avere un'intestazione in cui definire i namespace che specificano le librerie di tag da utilizzare. L'intestazione usata in tutte le pagine di iGalileo (e di Test) è la seguente:

```
<f:view xmlns="http://www.w3.org/1999/xhtml"
        xmlns:h="http://java.sun.com/jsf/html"
        xmlns:ui="http://java.sun.com/jsf/facelets"
        xmlns:f="http://java.sun.com/jsf/core"
        xmlns:p="http://primefaces.org/ui"
        xmlns:pm="http://primefaces.org/mobile"
        contentType="text/html" renderKitId="PRIMEFACES_MOBILE">
```

Questa intestazione specifica che possono essere usate le seguenti sei librerie di tag:

- XHTML (senza prefisso);
- HTML con prefisso h;
- Facelets con prefisso ui;
- JSF con prefisso f;
- PrimeFaces con prefisso p;
- PrimeFaces Mobile con prefisso pm.

La voce renderKitId invece dichiara l'identificatore (ID) dell'UI kit che verrà usato, in questo caso PRIMEFACES_MOBILE.

Ogni pagina, introdotta dal tag pm:page, è formata da una o più viste. Ciascuna vista, introdotta dal tag pm:view, corrisponde a una schermata dell'interfaccia utente. A sua volta ogni vista può essere composta da un header, un content e un footer, individuati rispettivamente dai tag pm:header, pm:content e pm:footer. L'header è l'intestazione ed è graficamente composto da una barra orizzontale situata nella parte superiore della schermata, dove solitamente si inserisce un titolo ed eventualmente dei pulsanti per navigare tra le varie schermate. Il content, ovvero il contenuto, rappresenta invece la parte principale della pagina, dove inserire i vari componenti. Infine abbiamo il footer, ovvero il piede, che può essere considerato l'opposto dell'header, poiché è composto da una barra orizzontale che si trova però nella parte inferiore della schermata. Solitamente questo viene utilizzato per mostrare delle informazioni a piè di pagina oppure, se integrato ad un componente chiamato pm:navbar, viene usato come menu di navigazione attraverso le pagine. Il seguente esempio mostra come creare una pagina contenente una vista.

```
<pm:page title="Test">
    <pm:view id="view1" swatch="a">
        <pm:header title="Hello" />
        <pm:content>
            Content
        </pm:content>
        <pm:footer />
    </pm:view>
</pm:page>
```

View, header e footer dispongono inoltre di un attributo swatch, con il quale è possibile cambiare il loro tema, ovvero lo schema che definisce i colori dei loro componenti. I temi principali forniti da jQuery Mobile sono cinque e sono individuati dalle prime cinque lettere dell'alfabeto. Esiste però la possibilità di utilizzare dei temi personalizzati grazie a ThemeRoller, un tool che permette di creare facilmente dei temi direttamente nel sito di jQuery Mobile, con la possibilità di vedere in anteprima il risultato finale, da scaricare e includere nella propria applicazione.

Infine PrimeFaces permette una navigazione locale, ovvero dà la possibilità di navigare tra le viste di una stessa pagina, senza effettuare alcuna richiesta HTTP. Per fare ciò è possibile usare ad esempio dei componenti chiamati h:outputLink, dando all'attributo value l'identificatore della vista destinazione preceduta da un cancelletto (#).

```
<h:outputLink value="#view2">Go to view2</h:outputLink>
```

Se prima di passare alla vista destinazione c'è la necessità di fare una richiesta AJAX, bisogna allora utilizzare i p:commandButton (o altri componenti con funzionamento analogo, come i p:commandLink) passando all'attributo action il nome del metodo (di un backing-bean) da eseguire. Questo deve restituire una stringa contenente l'identificatore della vista destinazione preceduta da pm:.

```
<p:commandButton value="Go to view2" action="#{bean.method}" />

public String method() {
    ... // do something
     return "pm:view2";
}
```

Durante la navigazione tra una vista a un'altra è possibile inoltre visualizzare un effetto di transizione. Slide, ad esempio, fa sì che la nuova schermata, corrispondente alla vista destinazione, compaia con un effetto scivolata da destra a sinistra. Per utilizzare questo effetto basta aggiungere ?transition=slide all'identificatore della vista destinazione.

```
<h:outputLink value="#view2?transition=slide">
     Go to view2 with transition effect
</h:outputLink>
```

Altri tipi di transizione sono fade, pop, flip, turn, flow, slideup, slidedown e none.

Tornando al progetto Test, ho creato una pagina con più viste. Ciascuna vista aveva un header e un footer personalizzati, mentre all'interno dei vari content ho inserito degli h:outputLink per navigare da una vista a un'altra visualizzando un effetto di transizione. Ho successivamente creato una seconda pagina dove ho inserito altri componenti PrimeFaces, tra cui il p:button (un bottone), il p:inputText (una casella di testo), il pm:switch (un pulsante che permette di scegliere tra due stati, ad esempio tra si e no), il p:accordionPanel (un pannello che raggruppa il contenuto in più tab) e il p:dataList (una lista di altri componenti). Infine ho testato i vari temi dando degli swatch diversi ai vari view, header e footer.

3.4 Prime pagine di iGalileo

Veniamo ora alla realizzazione di iGalileo. In iGalileo il template definisce alcune impostazioni di configurazione come ad esempio il path del CSS (Cascading Style Sheets o Fogli di stile) da utilizzare, ovvero un linguaggio per definire la formattazione di documenti HTML, XHTML e XML. Al suo interno troviamo anche il tag ui:insert con attributo name pari a content. Le pagine principali contengono tutte il tag ui:composition, specificando lo stesso template, e al suo interno il tag ui:define sempre con attributo name pari a content. In tal modo la pagina template sarà di volta in volta popolata dal contenuto di ui:define della pagina corrispondente. Ogni pagina di iGalileo contiene una o più viste. Ogni vista contiene al suo interno tre componenti ui:include, in modo da includere un header, un content e un footer. In questo modo è possibile fare riferimento in viste diverse allo stesso header e allo stesso footer.

Dopo aver costruito il template ho iniziato a creare la prima pagina, ovvero la pagina di login. Questa pagina contiene due viste, una principale in cui l'utente può inserire nickname e password per loggarsi e una secondaria accessibile tramite un pulsante sull'header che permette di impostare la lingua. Successivamente ho creato la pagina di menu con l'aiuto del tutor. Per costruire la lista del menu abbiamo utilizzato il componente p:datalist contenente al suo interno cinque p:commandLink corrispondenti alle voci Comunicazioni, Schede Clienti, Stato Ordini, Scadenzario e Fatture. Nell'header di questa pagina sono presenti due pulsanti, uno per tornare alla schermata di login e l'altro per accedere alle opzioni.

Le settimane successive sono state dedicate alla realizzazione delle pagine Comunicazioni e Schede Clienti. La pagina delle comunicazioni contiene due viste: vwList e vwDetail. La prima corrisponde a una schermata che mostra la lista di tutte le news divise per categoria. La lista è nuovamente formata da un p:dataList composto da p:commandLink. Cliccando su una di queste voci si seleziona una news passando così nella schermata dei suoi dettagli, ovvero la seconda vista. Qui sarà possibile leggere la notizia e scaricare eventuali allegati presenti.

La pagina delle Schede Clienti è molto simile a quella delle Comunicazioni. Anch'essa infatti dispone delle viste vwList e vwDetail, con funzionamento analogo a quelle relative alle Comunicazioni. In aggiunta però presenta una terza vista chiamata vwParams, che dà la possibilità di fare una ricerca del cliente, secondo vari parametri, per poi visualizzarne i risultati in una lista. Infine dopo aver selezionato il cliente dalla lista sarà possibile consultarne i dettagli, ovvero una serie di informazioni generali, tra cui l'indirizzo, la partita IVA, il codice fiscale, il numero di telefono ecc., dati amministrativi come il fido, il codice IBAN e il conto corrente e dati logistici, come le modalità di spedizione o i giorni in cui effettuare le consegne.

Dopo essere entrato nei dettagli di un cliente comparirà un menu per accedere alla pagina degli articoli. Questa pagina segue lo stesso schema delle schede clienti, ed è pertanto composta da tre viste che permettono rispettivamente di fare una ricerca dell'articolo, listare i risultati e visualizzare i dettagli dell'articolo selezionato, mostrandone alcune informazioni come il prezzo, la quantità, le dimensioni e un'immagine di anteprima.

Infine per quanto riguarda le pagine Stato Ordini, Scadenzario e Fatture, seguono tutte ancora una volta lo schema ricerca-lista risultati-dettaglio. Queste danno inoltre la possibilità di scaricare come allegati documenti relativi agli ordini o alle fatture.

Per aiutarmi nella stesura delle pagine ho fatto spesso riferimento non solo alla documentazione di PrimeFaces, ma anche al vecchio iGalileo per vedere la corrispondenza tra i componenti di RichFaces e quelli di PrimeFaces e per capire di volta in volta quali proprietà di backing-bean richiamare.

3.5 Protocolli e mappa di Google

Nella schermata dei dettagli delle Schede Clienti sono presenti alcuni elementi interattivi. Ad esempio, se si utilizza un dispositivo mobile, cliccando sul numero di telefono del cliente selezionato, automaticamente verrà aperta l'applicazione predefinita del dispositivo per chiamare quel numero o per mandargli un SMS. Questo è reso possibile grazie ai protocolli, ovvero dei link con prefissi speciali, che invece di rimandare a un'altra pagina aprono un programma. Ad esempio i prefissi tel: e sms: permettono rispettivamente l'apertura del programma per telefonare e del programma per mandare SMS. Il prefisso mailto: invece permette di aprire il programma predefinito di posta elettronica.

Cliccando invece sull'indirizzo del cliente si aprirà una nuova schermata con integrata la mappa di Google (vedi Fig. 3.2). Il dispositivo mediante geolocalizzazione troverà la posizione geografica in cui ci si trova (o per essere più precisi, in cui si trova il dispositivo stesso) e calcolerà il percorso migliore da tale punto all'indirizzo del cliente, visualizzando inoltre le indicazioni stradali per arrivarci.

Figura 3.2: Risultato della geolocalizzazione e della ricerca del percorso migliore

Per utilizzare questa mappa ho usato l'elemento p:gmap, specificando la posizione iniziale (latitudine e longitudine), il tipo e le dimensioni della mappa e il livello di zoom. Questo elemento per funzionare necessita inoltre del seguente script:

```
<script src="http://maps.google.com/maps/api/js?sensor=false" />
```

che dichiara l'utilizzo dell'API di Google Maps. Il parametro sensor impostato a false specifica che l'applicazione non richiede l'utilizzo di un sensore di localizzazione GPS. In questo caso verranno invece raccolte informazioni sui punti di accesso wireless nelle vicinanze e sull'indirizzo IP del dispositivo usato, per calcolare una stima della posizione geografica.

Infine per eseguire la funzione di geolocalizzazione e per mostrare le indicazioni stradali bisogna servirsi di alcune funzioni javascript di cui si è occupato il mio collega Daniele.

3.6 Inserimento grafici

Una volta terminate le pagine principali, abbiamo pensato di aggiungere alcune nuove funzionalità ad iGalileo, sfruttando alcuni componenti di PrimeFaces che rappresentano grafici. Questi si chiamano chart e in particolare abbiamo usato il Line Chart, il Bar Chart e il MeterGauge Chart. Il primo visualizza una o più serie di dati in un grafico cartesiano mediante linee rette, il secondo è analogo ma utilizza delle barre mentre il terzo visualizza i dati in un display semicircolare a scartamento metrico, ovvero un display con un aspetto molto simile a un tachimetro con una lancetta che indica un valore preimpostato. È possibile personalizzare questi componenti in svariati modi, dando ad esempio un effetto di ombreggiatura o modificando i colori delle serie, o suddividendo il display del MeterGauge in più intervalli di colori diversi.

Per poter utilizzare il Line Chart bisogna innanzitutto creare una classe backing-bean contenente un oggetto di tipo CartesianChartModel, che va a costituire il modello del grafico. Successivamente bisogna aggiungere al modello una o più serie utilizzando il metodo addSeries(), a cui va passato come parametro un oggetto di tipo ChartSeries. Infine per impostare i dati da visualizzare su ciascuna serie è necessario utilizzare il metodo set(), passandogli come parametri una coppia chiave/valore. La chiave è una stringa che identifica un dato, mentre il valore è una variabile che identifica il valore assegnato al dato. È possibile inoltre dare un nome alle serie passando come parametro al metodo setLabel() una stringa contenente tale nome. Una volta realizzata la classe bisogna aggiungere il componente p:lineChart nella vista in cui si vuole visualizzare il grafico.

```
<p:lineChart value="#{chartBck.animateChartFatturatoTrend.model}"
legendPosition="se" xaxisLabel="mesi" yaxisLabel="fatturato"
animate="true" />
```

L'attributo value specifica la fonte dei dati che riempiranno il grafico, ovvero il modello. Gli attributi legendPosition, xaxisLabel e yaxisLabel specificano rispettivamente la posizione della legenda rispetto al grafico (utilizzando i punti cardinali) e il nome degli assi x e y. L'attributo animate di tipo booleano invece specifica se il grafico presenta un'animazione o meno.

Il Line Chart è stato utile per rappresentare il fatturato mensile dell'azienda (vedi Fig. 3.3). Questo grafico mostra due serie, una relativa al trend del fatturato dell'anno precedente e l'altra relativa a quello dell'anno attuale. Per ciascuna serie vengono rappresentati i valori mensili del fatturato. L'asse x mostra i mesi dell'anno, mentre l'asse y

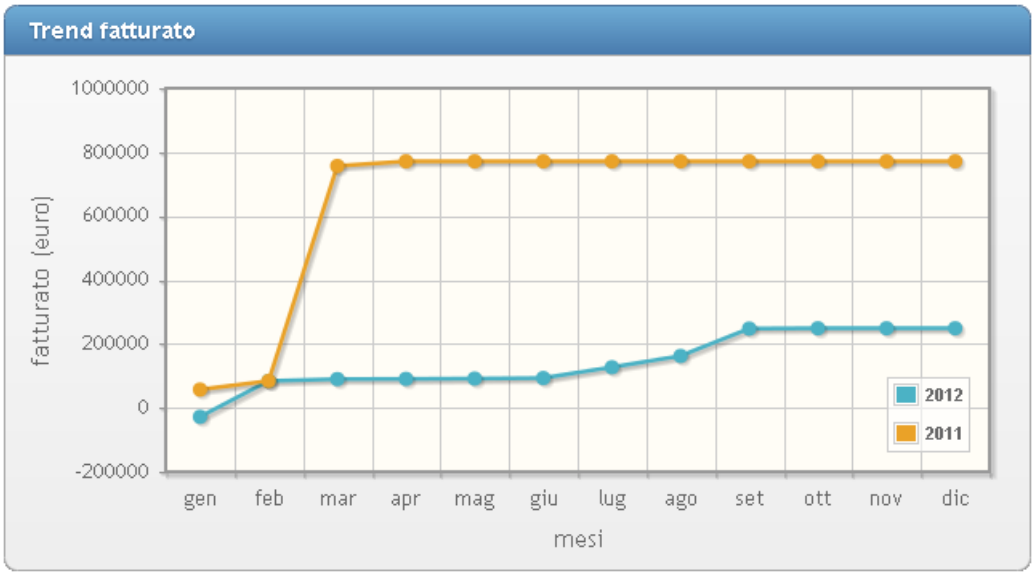

Figura 3.3: Grafico del fatturato costruito tramite lineChart

mostra i relativi valori del fatturato. Questi valori vengono uniti da una retta spezzata che viene costruita con un'animazione.

Il Bar Chart è un componente molto simile. Esso strutturalmente è composto dalla stessa classe del Line Chart e per utilizzarlo basta servirsi del componente p:barChart.

```
<p:barChart value="#{chartBck.animateChartCliente.model}"
legendPosition="ne" xaxisLabel="anni" yaxisLabel="euro" animate="true" />
```

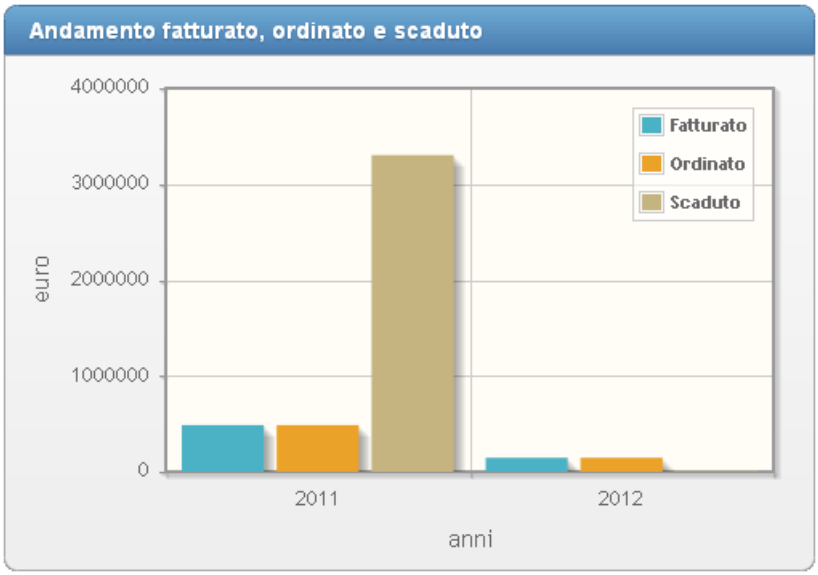

Figura 3.4: Grafico fatturato, ordinato e scaduto costruito tramite barChart

Il Bar Chart è stato usato per rappresentare il valore degli ordini effettuati nell'anno corrente e nell'anno precedente. Dalle schede clienti invece è possibile visualizzare un Bar Chart che rappresenta il valore totale del fatturato, degli ordini e degli scaduti dell'anno attuale sempre paragonato all'anno precedente, relativi al cliente selezionato (vedi Fig. 3.4).

Anche il MeterGauge Chart richiede una classe backing-bean contenente un modello. Questo modello è composto da un oggetto di tipo MeterGaugeChartModel che richiede come parametri una variabile numerica, che indica un valore preimpostato, e un oggetto di tipo ArrayList<Number>, ovvero una lista di numeri che individuano gli intervalli con cui è possibile suddividere il display. Infine bisogna aggiungere il componente p:meterGaugeChart nella pagina in cui si vuole visualizzare il display.

```
<p:meterGaugeChart value="#{chartBck.meterGaugeChartNuoviClienti.model}"
label="n. clienti" seriesColors="FF0000,00FF00" />
```

L'attributo label specifica una stringa da visualizzare mentre seriesColors indica la lista di colori (in formato esadecimale) che devono assumere gli intervalli del display.

Abbiamo utilizzato il p:meterGaugeChart nelle schede clienti per mostrare il numero di nuovi clienti acquisiti dall'agente in un anno (vedi Fig. 3.5). Il display è suddiviso in due intervalli, uno di colore rosso e uno di colore verde. La lancetta indica il numero di clienti acquisiti durante l'anno attuale: se si trova sul verde significa che quest'anno l'agente ha acquisito più clienti dell'anno passato, mentre se è sul rosso significa che ne ha di meno. Questo componente è semplice e allo stesso tempo molto utile, poiché dà subito un'idea intuitiva sulla situazione.

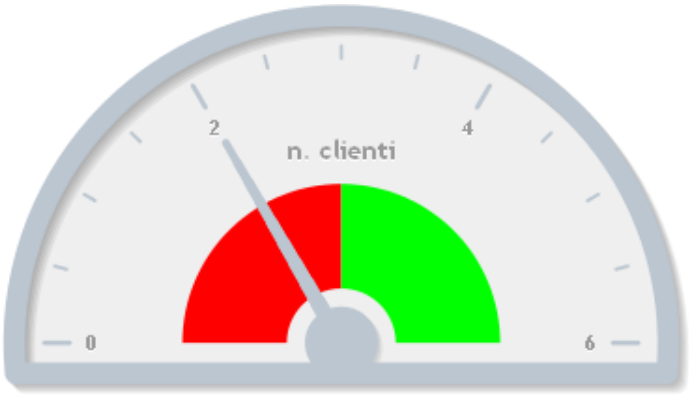

Figura 3.5: Grafico dei nuovi clienti costruito tramite meterGaugeChart

Anche nella pagina delle scadenze è presente questo display, che indica però l'ammontare degli scaduti. In questo caso i colori sono invertiti poiché avere pochi pagamenti insoluti rappresenta una situazione positiva.

Per semplificare lo sviluppo di iGalileo, invece di creare una classe backing-bean per ogni tipo di grafico ho creato un'unica classe backing-bean chiamata ChartBck, che a sua volta istanzia le varie classi (entità) corrispondenti ai grafici. Ciascuna istanza di entità possiede un metodo per chiamare una classe manager, che a sua volta chiama una classe data-provider. In quest'ultima un metodo apposito estrae i dati da un database

dopo aver eseguito le istruzioni SQL adeguate, e li inserisce all'interno di un oggetto di tipo DataTable, il quale viene successivamente passato tramite il manager alla classe ChartBck. Qui infine i dati vengono estratti dalla DataTable e inseriti all'interno del modello della corrispondente entità.

3.7 Risoluzione problemi

Purtroppo la realizzazione di iGalileo non è stata esente da problemi ed ostacoli che di settimana in settimana abbiamo dovuto superare od aggirare.

La struttura a più viste di PrimeFaces Mobile, se da un lato permette di navigare localmente e di usare degli effetti di transizione durante il passaggio da una vista a un'altra, d'altro canto limita notevolmente il modo in cui strutturare le pagine. Infatti alcuni componenti come i grafici e la mappa funzionano solamente se si trovano sulla prima vista di una pagina. Siamo pertanto stati obbligati a volte a seguire questa logica, altre volte a creare una nuova pagina apposita per inserire tali componenti, perdendo però così gli effetti di transizione.

Non è possibile nemmeno navigare da una pagina verso una vista di un'altra pagina, se questa vista non è la prima. Per far ciò abbiamo dovuto utilizzare un trucchetto, ovvero utilizzare l'attributo rendered in modo da non renderizzare le viste precedenti della pagina destinazione in determinate condizioni, per mostrare soltanto quella di nostro interesse.

Altro problema è dovuto alla compatibilità di alcuni componenti con dispositivi diversi. Ad esempio quando si vuole inserire una data in una casella di testo utilizzando un dispositivo Apple, come l'iPhone o l'iPad, dopo aver cliccato su tale casella compare in automatico un simpatico ingranaggio girevole chiamato viewpicker (vedi Fig. 3.6), che permette di scegliere una data semplicemente facendo ruotare l'ingranaggio con il dito.

Figura 3.6: Componente viewpicker per scegliere una data

Se invece si utilizza un dispositivo Android abbiamo notato che non solo non compare nessun componente (come potrebbe essere ad esempio un calendario) ma compare la tastierina numerica che obbliga l'inserimento di numeri vietando però l'uso di simboli, come ad esempio il trattino-meno (-) e la barra (/). In questo modo non è possibile rispettare le formattazioni più classiche per le date, che consistono nel dividere giorno, mese e anno con trattini o barre (. . . -. . . -. . . e . . . /.../. . .). Per risolvere questo problema abbiamo dovuto creare una diversa configurazione relativa alle date per dispositivi Apple

e Android, in modo da conservare da una parte l'uso del viewpicker e permettere dall'altra una formattazione diversa.

Alcune pagine permettono di scaricare degli allegati in formato PDF. Purtroppo, terminato il download e aperto l'allegato nella stessa scheda del browser, viene persa la sessione corrente impedendo il proseguirsi della navigazione in iGalileo. Abbiamo aggirato il problema assegnando il valore blank all'attributo target del componente commandLink che attiva il download, in modo da aprire l'allegato in una nuova scheda del browser e mantenere così la sessione di iGalileo.

```
<h:commandLink value="Download" target="blank"
action="#{queryFattureBck.downloadFatturaGDAction}" />
```

Come già accennato solitamente si inseriscono dei pulsanti nell'header che permettono di navigare da una vista a un'altra. Questi pulsanti sono costituiti dal componente p:button, che però non possiede l'attributo action. Nel caso si voglia richiamare un backing-bean si è pertanto costretti ad utilizzare il p:commandButton, un pulsante graficamente più grande del normale button e troppo grande per essere contenuto nell'header. Per fortuna è possibile dare al commandButton l'aspetto di un button semplicemente facendo ricorso al CSS.

Per quanto riguarda la mappa inizialmente dava alcuni problemi di funzionamento, che abbiamo risolto assegnando il valore TRUE all'attributo fitBounds di p:gmap, sebbene questo fosse il valore già impostato di default. Inoltre l'idea iniziale era che la mappa fosse caricata in automatico subito dopo aver caricato la pagina, mostrando immediatamente la posizione in cui ci si trova. La geolocalizzazione però richiede il permesso da parte dell'utente di comunicare la propria posizione, che viene automaticamente negato nel caso in cui si imposti il caricamento automatico della mappa.

Infine un altro problema riscontrato nei dispositivi mobile è che quando si cambia l'orientamento di tali dispositivi, ad esempio da verticale a orizzontale o viceversa, i grafici non cambiano la propria dimensione risultando così sproporzionati rispetto alle dimensioni dello schermo. Per risolvere ho fatto uso di una funzione javascript che, quando rileva un cambiamento nell'orientamento del dispositivo usato, chiama il componente p:remoteCommand, che a sua volta chiama un metodo backing-bean per aggiornare i dati del grafico, ricostruendolo così con una dimensione appropriata.

```
<script type="text/javascript">
window.addEventListener('orientationchange', handleOrientation, false);
function handleOrientation() { updateGrafico(); }
</script>

<p:remoteCommand name="updateGrafico" update="grafico"
action="#{chartBck.fillChartCliente}" />
```

La maggior parte dei problemi comunque è dovuta alla giovinezza di PrimeFaces, essendo un progetto nato soltanto un paio di anni fa: basti pensare che PrimeFaces Mobile non è arrivato nemmeno alla versione 1.0 (la versione attuale è 0.9.3). Esso manca ancora di molti componenti utili, come ad esempio la finestra di dialogo che avremmo voluto usare in iGalileo, e probabilmente molti bug verranno risolti con le release future.

3.8 Test finale su dispositivi mobile

Una volta conclusa la realizzazione di iGalileo abbiamo testato il suo funzionamento. Sebbene sia stato chiaramente realizzato su PC abbiamo dovuto testarlo sui dispositivi mobile, poiché quelli sono il mezzo con cui gli utenti potranno usufruire di iGalileo. I dispositivi che abbiamo utilizzato per il test sono l'iPad (1, 2 e 3), il Samsung Galaxy Tab 8.9 e l'iPhone 4S.

Ci sono molte differenze tra un PC e un dispositivo mobile di cui bisogna tenere conto durante lo sviluppo di un applicativo web. Innanzitutto mentre su PC si può interagire tramite il mouse, i moderni smartphone e tablet dispongono ormai tutti di uno schermo tattile (touch screen), che permette all'utente di interagire con l'interfaccia grafica mediante le dita o particolari oggetti. Pertanto risulta inutile l'utilizzo di alcune funzionalità non destinate al mobile come ad esempio lo zoom dei grafici, azione non eseguibile con le dita.

Altra cosa di cui tenere conto è che nei dispositivi mobile, a differenza dei PC, esistono due modalità di visualizzazione: portrait e landscape. La prima consiste nel tenere il dispositivo in una posizione verticale, mentre il secondo nel tenerlo in una posizione orizzontale (vedi Fig. 3.7).

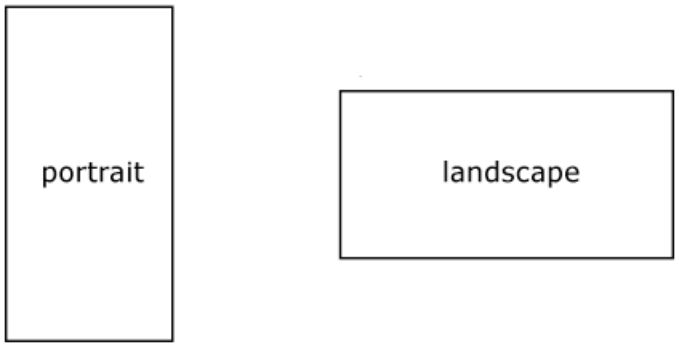

Figura 3.7: Differenza tra gli orientamenti dei dispositivi mobile: portrait e landscape

Per nostra fortuna il layout si adatta in automatico alle dimensioni dello schermo disponendo i componenti in modo ottimale. Unica eccezione la fanno i grafici, che non aggiornano la loro dimensione dopo aver ruotato il dispositivo, e per i quali abbiamo dovuto porre rimedio. Utilizzando invece uno schermo piccolo come quello di uno smartphone i componenti vengono automaticamente rimpiccioliti per permettere una visualizzazione gradevole (vedi Fig. 3.8).

Altra considerazione va fatta sulle prestazioni. Qualsiasi persona utilizzi un browser per navigare in Internet sa quanto sia fastidioso dover aspettare che una pagina web venga caricata. L'accoppiata JSF-PrimeFaces però permette di renderizzare le pagine con una velocità più elevata rispetto ai framework concorrenti, risultando così la scelta più adatta sia per applicativi desktop che per applicativi mobile.

Un'ultima osservazione va fatta infine sui browser. Infatti abbiamo dovuto controllare che non ci fossero problemi con i principali browser per mobile. Lo sviluppo del software è stato fatto principalmente utilizzando Firefox e Chrome, poiché entrambi permettono di analizzare i vari componenti di cui sono composte le pagine e di mostrarne il codice

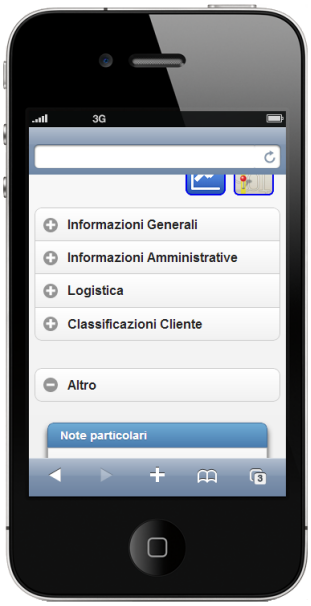

Figura 3.8: Dettaglio delle schede clienti di iGalileo su iPhone

sorgente. I dispositivi Apple invece utilizzano Safari come browser di default. Tuttavia grazie al supporto che jQuery Mobile ci offre non abbiamo riscontrato problemi di compatibilità.

Capitolo 4

Conclusione

Dopo sei settimane di lavoro sono riuscito a portare a termine il progetto, grazie anche all'aiuto dei miei colleghi di lavoro. iGalileo è finito e perfettamente funzionante sui principali dispositivi mobile sul mercato. L'azienda ha inoltre già trovato numerosi clienti desiderosi di acquistarlo.

Grazie a questa esperienza ho imparato molte cose nuove. Partendo da una conoscenza abbastanza scarsa delle applicazioni web sono arrivato a realizzarne una (non da solo, ma è già un buon inizio). Ho constatato la potenza di JSF e del pattern MVC, conosciuto pregi e difetti di PrimeFaces e appreso familiarità con jQuery. Ho acquisito molte nozioni utili nel campo e fatto molta esperienza. Ho inoltre testato con mano le funzionalità di iGalileo grazie ai dispositivi mobile messi a disposizione dall'azienda. Ho appreso come sia importante organizzare il proprio lavoro e strutturare il progetto in modo da renderlo scalabile e manutenibile. Se inizialmente chiedevo spesso aiuto e consigli ai colleghi, nelle ultime settimane sono riuscito a cavarmela da solo e a risolvere i problemi che mi si presentavano in autonomia. Ho imparato inoltre a debuggare il software utilizzando Eclipse per scoprire eventuali malfunzionamenti.

Con questo progetto abbiamo inoltre dimostrato come sia vantaggioso per un programmatore utilizzare framework che danno la possibilità di creare applicazioni compatibili con più sistemi operativi e più dispositivi mobile, piuttosto che dover creare un'applicazione nativa per ciascun dispositivo.

Questa esperienza è stata utile anche perché sono entrato in contatto per la prima volta con il mondo del lavoro. Fin da subito mi sono sentito a mio agio nell'ambiente lavorativo, grazie soprattutto al mio tutor e ai miei colleghi, che si sono sempre dimostrati simpatici e molto disponibili, e con i quali ho instaurato un rapporto di amicizia. Insieme abbiamo collaborato alla realizzazione di iGalileo aiutandoci nei momenti di bisogno e imparando ad affrontare problemi e avversità. Sono infine contento di aver svolto questo tirocinio, riuscendo a raggiungere gli obiettivi imposti e vedendo così realizzato un progetto concreto.

Bibliografia

[1] Sito ufficiale Sanmarco informatica: *http://www.sanmarcoinformatica.it/home.pag*

[2] Sito ufficiale Eclipse Juno: *http://eclipse.org/juno/*

[3] Sito ufficiale Apache Tomcat: *http://tomcat.apache.org/*

[4] Sito ufficiale JavaServer Faces: *http://javaserverfaces.java.net/*

[5] Alfredo Larotonda. Articolo su JavaServer Faces: *http://www2.mokabyte.it/*

[6] Giuseppe Sicari. Articolo su Facelets: *http://www.giuseppesicari.it/articoli/facelets/*

[7] Sito ufficiale PrimeFaces e PrimeFaces Mobile: *http://www.primefaces.org/*

[8] Sito ufficiale jQuery Mobile: *http://jquerymobile.com/*