

Sviluppo della gestione delle note di credito in un programma di fatturazione

Davide Fontana

Indice

1	Introduzione	5
2	Azienda ospitante	7
3	Il corso di formazione	9
3.1	Hibernate	9
3.2	Spring	9
3.2.1	Spring MVC	10
4	Il Programma di Fatturazione	13
4.1	Struttura dell' applicativo	13
5	Progetto delle Note di Credito	15
5.1	Analisi dei Requisiti	15
5.2	Database	16
5.2.1	Schema Concettuale	16
5.2.2	Schema Logico	17
5.2.3	Implementazione Fisica	17
5.3	Implementazione dell' applicativo.	17
5.3.1	Bean delle note di credito	17
5.3.2	Lista delle fatture	18
5.3.3	Inserimento e modifica della nota di credito	19
5.3.4	Lista delle note di credito	24

Capitolo 1

Introduzione

Questa relazione riguarda il tirocinio da me effettuato presso l'azienda Consoft Informatica S.r.l tra Febbraio e Aprile 2016 nella sede di Padova. Lo stage è cominciato con una prima parte di formazione sulle tecnologie usate all'interno dell'azienda tra cui Java J2EE, SQL, Hibernate e Spring MVC. In seguito, dopo la formazione, è seguita una parte pratica con attività di bug-fixing e poi la progettazione con successiva implementazione della gestione delle note di credito di un programma di fatturazione online che, attualmente, è ancora in fase di sviluppo all'interno dell'azienda.

Capitolo 2

Azienda ospitante

L'azienda in cui è stato svolto il tirocinio è la Consoft Informatica S.r.l operante nel settore Information & Communication Technology. Come riportato sul sito ufficiale [1], l'azienda è una società giovane, dinamica e innovativa la quale mette a disposizione competenze, servizi e strutture professionali tecnologicamente all'avanguardia a medie e grandi imprese al fine di potenziarne al massimo la competitività sul mercato. I professionisti che permettono a Consoft Informatica di perseguire il proprio obiettivo sono altamente qualificati e grazie ad un costante investimento nella formazione attraverso un addestramento legato alla metodologia del corpo docenti e all'utilizzo di strumenti software avanzati, sono in grado di rispondere alle esigenze del cliente garantendo affidabilità, convenienza ed efficienza. Consoft Informatica opera con i suoi oltre 150 dipendenti su tutto il territorio nazionale e ha 3 sedi ubicate a Padova, Firenze e infine Roma.

Capitolo 3

Il corso di formazione

Durante le prime due settimane sono stato inserito all'interno di un corso tenuto dalla mia tutor aziendale Giulia Paoli in cui ho potuto apprendere due framework indispensabili per la programmazione di software Enterprise di grandi dimensioni ovvero Hibernate e Spring.

3.1 Hibernate

Hibernate è un ORM (o anche object relational mapping) e ha la funzione di mappare le tabelle di un database in oggetti Java. Utilizzare questo framework conviene rispetto ad usare le normali API di Java per i seguenti motivi:

- Tutta la gestione della connessione al database viene gestita dal framework.
- Il reperimento dati viene fatto chiamando funzioni di Hibernate e non più scrivendo query. Questo permette quindi di cambiare database senza dover più preoccuparsi di riscrivere le query nel dialetto SQL del nuovo DBMS.
- Tutto quello che viene restituito dalle funzioni di Hibernate sono oggetti Java che rispecchiano negli attributi la tabella da cui sono stati presi.

Tutto questo ovviamente non viene fatto in automatico. Serve infatti un file di configurazione che specifica l'URI del database, il nome utente, la password e il dialetto SQL per il reperimento dati. Per la mappatura delle tabelle in oggetti Java bisogna inoltre scrivere delle classi Java apposite chiamate Bean e, tramite delle specifiche notazioni Java, indicare per ogni attributo della classe l'attributo della tabella corrispondente.

3.2 Spring

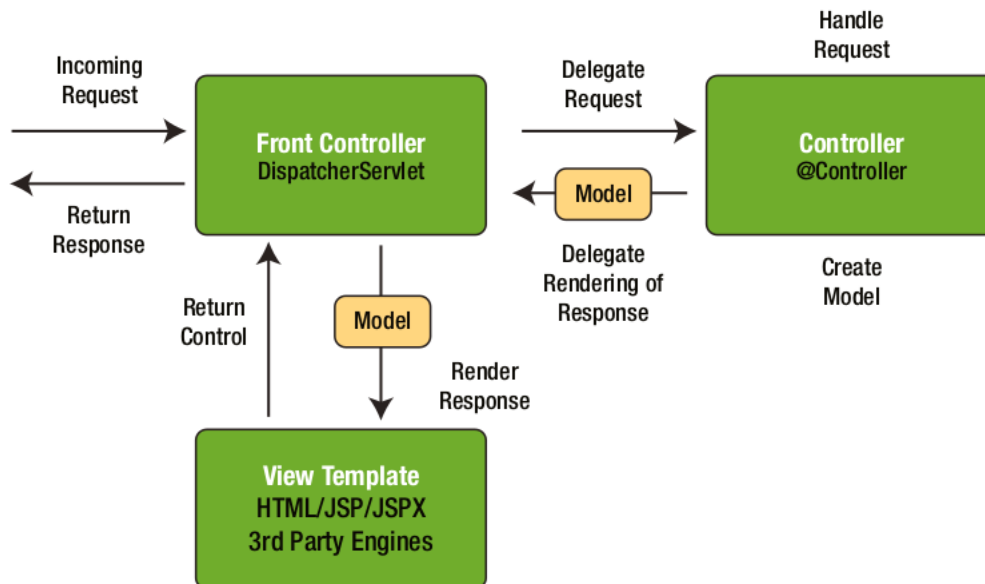
Spring è il framework su cui è stato costruito il programma di fatturazione. Questo framework è molto popolare e serve principalmente per semplificare lo sviluppo di applicazioni Java Enterprise dando al programmatore un modello di programmazione e delle API consistenti

per scrivere programmi complessi senza incorrere in errori producendo codice di alta qualità. Il framework inoltre si integra alla perfezione con la maggior parte degli ORM tra cui il famoso Hibernate. Il framework è stato creato basandosi sulla **Dependency Injection**. La DI è un design pattern che serve per invertire il controllo nella creazione degli oggetti. Infatti invece di creare le dipendenze tra le classi utilizzando il costrutto **new** si inietta la risorsa all'interno dell'oggetto destinazione. Questo ha i conseguenti vantaggi:

- Il client non si deve preoccupare delle diverse implementazioni
- È più facile eseguire procedure di Unit Testing e Mocking.
- La configurazione viene esternizzata. Infatti è possibile creare file xml che contengono le informazioni per la creazione dell'oggetto.

3.2.1 Spring MVC

In particolare per la creazione delle pagine web è stata usata un'estensione di Spring ovvero Spring MVC. Il design pattern MVC in Spring è spiegato nell'immagine (Si veda [2] per maggiori dettagli).



Per prima cosa abbiamo una richiesta web (con o senza parametri) inviata dall'utente la quale viene ridirezionata tramite un Front Controller ad una classe chiamata Controller da noi implementata. Infine dopo aver reperito i dati viene data come risposta una pagina web (html o jsp) che fa visualizzare la risposta. Un esempio di controller lo possiamo vedere in questo codice di esempio.

Esempio di controller

```
import org.springframework.beans.factory.annotation.Autowired;
```

```

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import com.apress.isf.spring.data.DocumentDAO;

@Controller
@RequestMapping("/search")
public class SearchController {

    @Autowired
    DocumentDAO documentDAO;

    @RequestMapping(value="/all",method=RequestMethod.GET)
    public String searchAll(Model model){
        model.addAttribute("docs", documentDAO.getAll());
        return "search/all";
    }
}

```

Il controller di per se è molto semplice ed è corredato di diverse notazioni.

- **@Controller:** Questo marca la classe come controller e quindi il Front Controller sa dove inviare la richiesta.
- **@RequestMapping:** Questa serve per come accedere al controller. In questo caso `path/search`. Poi per ogni funzione abbiamo un' ulteriore mappatura in cui è specificato il metodo di richiesta (GET o POST).
- **@Autowired:** Serve per iniettare l' implementazione dell' oggetto `DocumentDAO`.

Inoltre l' oggetto `model` serve per inserire le informazioni reperite dal controller per poter essere visualizzate nella pagina di risposta identificata con la stringa presente nel valore di ritorno della funzione. Le pagine web di risposta che vengono visualizzate dall' utente sono pagine `.jsp`. Queste pagine molte volte contengono codice Java ma se sono presenti errori possono essere pericolose in quanto l' intero codice della pagina apparirebbe sullo schermo e la sicurezza dell' applicativo sarebbe così compromessa. In questo caso quindi si sceglie di utilizzare la libreria `jstl` (o java standard tag library). Che permette di presentare i dati attraverso degli speciali tag come possiamo vedere nella pagina `jsp` di arrivo del nostro controller.

Esempio di view con i tag `jstl`

```

<html
  xmlns:jsp="http://java.sun.com/JSP/Page"
  xmlns:c="http://java.sun.com/jsp/jstl/core"
  xmlns:spring="http://www.springframework.org/tags"
  version="2.0">
<jsp:directive.page contentType="text/html; charset=UTF-8" />
<jsp:output omit-xml-declaration="yes" />
<head>
  <title>My Documents</title>
</head>
<body style="font-family: verdana;">
  <h2>My Documents - Search</h2>

```

```

<c:if test="${not empty docs}">
<table>
  <tbody>
    <c:forEach items="${docs}" var="doc" varStatus="status">
      <tr>
        <td>
          <table>
            <tbody>
              <tr>
                <td align="right">Name:</td>
                <td>${doc.name}</td>
              </tr>
              <tr>
                <td align="right">Type:</td>
                <td>${doc.type.name}</td>
              </tr>
              <tr>
                <td align="right">Location</td>
                <td>${doc.location}:</td>
              </tr>
            </tbody>
          </table>
        </td>
      </tr>
    </c:forEach>
  </tbody>
</table>
</c:if>
</body>
</html>

```

Dove `${...}` indica il contenuto all' interno dell' attributo di nome docs. Possiamo notare inoltre due tag che rappresentano i costrutti if e for che di solito troviamo in programmazione.

Capitolo 4

Il Programma di Fatturazione

Dopo aver finito di seguire il corso tenuto dalla mia responsabile aziendale per prendere confidenza con software di grandi dimensioni ho fatto circa due settimane di attività di bug fixing sul software di fatturazione in sviluppo internamente all'azienda. Questo applicativo è stato scritto con il framework Spring mentre i dati sono presenti sul DBMS MySQL e la loro gestione viene aiutata grazie all'uso di Hibernate. Per il log delle operazioni è stata usata la libreria Log4j mentre infine per la generazione dei pdf è stata usata la libreria JasperReport.

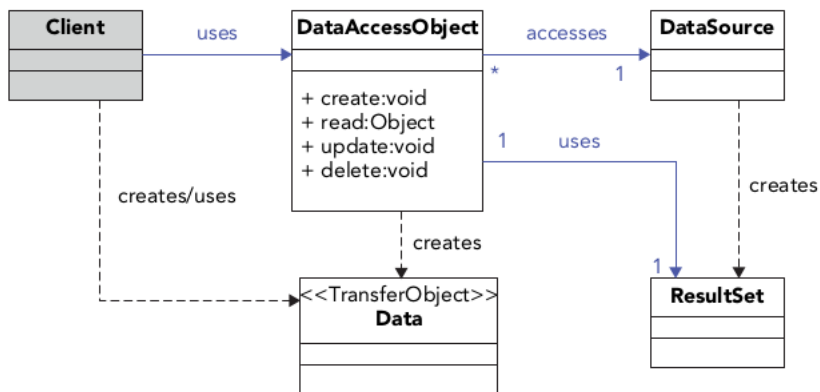
4.1 Struttura dell' applicativo

La struttura in package è organizzata in questo modo:

- `it.consoft.fatturazione.controller`: Contiene tutti i i Spring controller dell'applicativo.
- `it.consoft.fatturazione.form`: Contiene tutte le classi Java che servono a Spring per fare la validazione dei dati inviati da un form. Queste classi hanno delle annotazioni che permettono, appena il form viene sottomesso, di verificare se tutti i dati che sono inviati al controller corrispondono ai criteri di validazione. Per includere una classe per la validazione serve aggiungere l'attributo `modelAttribute` con il nome dell'oggetto della classe utilizzata nella validazione.
- `it.consoft.fatturazione.bean`: Contiene tutti i Bean per l' utilizzo in Hibernate. Un Bean è una classe Java che ha variabili di istanza private e solo metodi set e get.
- `it.consoft.fatturazione.utils`: Contiene una serie di classi utility come ad esempio per la formattazione delle date.
- `it.consoft.fatturazione.dao`: Contiene le classi DAO (Data Access Object). Le classi DAO sono delle classi che implementano le funzioni di inserimento, modifica e cancellazione dei dati di ogni tabella presente nel database gestendo inoltre la connessione del database. Tutto questo per fortuna ci viene semplificato grazie al framework

Hibernate e quindi la scrittura del codice all' interno delle funzioni risulta piuttosto semplice. Le funzioni dei DAO non vengono mai chiamate da tutte le classi ma solo da classi particolari chiamate **Service**.

- `it.consoft.fatturazione.service`: I service sono un' ulteriore astrazione dei DAO. Hanno le stesse funzioni dei DAO più ulteriori metodi che servono per reperire informazioni maggiormente specifiche. I Service infine sono le uniche classi che vengono chiamate nelle classi Java per interrogare il database e possono chiamare funzioni di altri Service. Questa distinzione tra DAO e Service non è casuale ma è data da un pattern specifico detto **Data Access Pattern** come specificato in questo class diagram.



Dove la classe Client è il Service. La sua importanza è data dal fatto che una strutturazione del codice in questa maniera permette non solo di rendere maggiormente testabile il software ma astrae ed incapsula tutti gli accessi ai dati.

Capitolo 5

Progetto delle Note di Credito

Dopo aver preso confidenza con la struttura dell' applicativo ho quindi potuto prendere l' incarico di progettare e implementare le note di credito.

5.1 Analisi dei Requisiti

La prima cosa da fare è stata intervistare la committente di tale progetto ovvero la Sig.na Anna Bellin responsabile amministrativa di Consoft Informatica per prendere nota delle specifiche del problema. Infatti la nota di credito viene applicata ad una fattura e ha due finalità:

1. Correttiva Parziale: Serve per correggere l'importo di una fattura nel caso in cui sia presente un errore per eccesso dell' imponibile. In questo caso l'importo della detrazione non è mai uguale o superiore al totale attuale della fattura.
2. Correttiva Totale: L'importo della detrazione è uguale a quello totale della fattura. Questo utilizzo è meno frequente e serve per annullare una fattura nel caso in cui non si riesca a risalire all' importo da correggere.

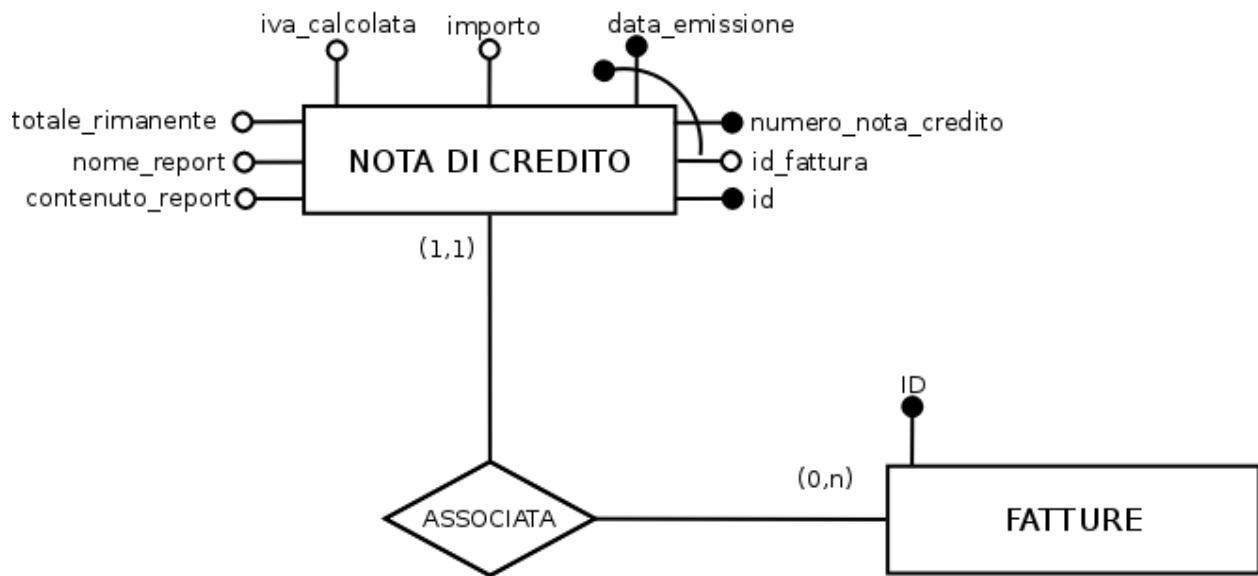
Ogni nota di credito è relativa ad un' unica fattura ma una fattura può avere più note di credito. E' composta dalle informazioni della azienda e il numero della fattura a cui questa nota di credito fa riferimento. Il numero della fattura è incrementale solo rispetto all' anno in corso e non alla fattura in cui la nota di credito è presente. La grafica deve essere in linea con quella attuale e deve consentire per ogni fattura (scaduta e non) di aggiungere note di credito e poter vedere, modificare, cancellare e stampare i report di quelle attualmente associate. La visualizzazione delle note di credito deve essere per data partendo dalla più recente facendo in modo che solo l'ultima nota credito sia modificabile/cancellabile mentre le altre devono avere solo la possibilità di essere visualizzate/stampate. Nel caso in cui l' ultima nota di credito venisse cancellata bisogna rendere disponibile la modifica di quella che c'era precedentemente. Infine l'imponibile nella lista delle fatture bisogna che sia compreso delle detrazioni rendendo poi impossibile la modifica della fattura stessa essendo che quindi la fattura è già stata emessa.

5.2 Database

Data quindi l'analisi dei requisiti si è cominciato a progettare la base di dati per salvare le informazioni. Qui sotto sono descritte tutte le fasi di progettazione.

5.2.1 Schema Concettuale

Essendo che le note di credito erano relative alle fatture si è deciso di creare una sola entità come mostrato nello schema sottostante.

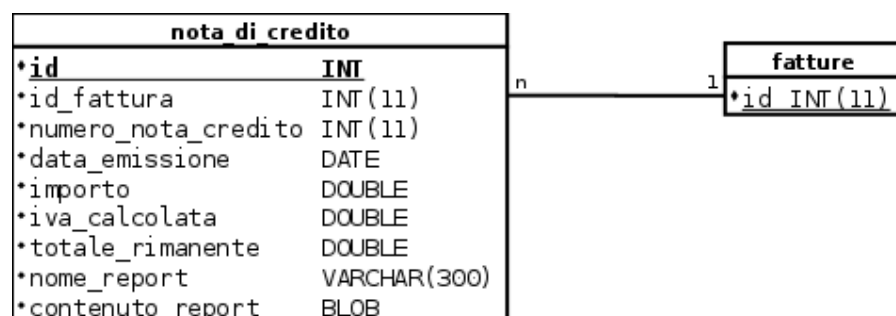


Sono state attuate le seguenti scelte sugli attributi:

1. L'attributo **totale_rimanente** anche se attributo derivato viene comunque messo per velocizzare le operazioni di visualizzazione delle fatture in modo che non si debba fare ogni volta i calcoli dell'imponibile con le note di credito.
2. Sono presenti due chiavi, la prima, chiave primaria, è un **id** numerico auto incrementante mentre la seconda, chiave candidata, è data dalla coppia **numero_nota_credito** e **data_emissione**. L'attributo **id_fattura** è una chiave esterna ed è associata con l'attributo **id** che è chiave dell'entità fattura.
3. Abbiamo due attributi, **nome_report** e **contenuto_report**, che contengono il nome e il contenuto del file **.pdf** della nota di credito appena generata.
4. Sono stati inseriti inoltre solo l'importo e l'IVA calcolata in quanto la percentuale dell'IVA utilizzata e l'importo totale della nota di credito possono essere ricavate dagli attributi importo e IVA calcolata.

5.2.2 Schema Logico

Si è quindi passato poi allo schema logico ricavato da quello concettuale.



Come numero di cifre intere è stato scelto undici per rimanere in linea con il numero di cifre dell' id della tabella fattura

5.2.3 Implementazione Fisica

Il codice della tabella in SQL è il seguente

Codice SQL della tabella

```
create table nota_di_credito(
    id bigint auto_increment check(id > 0),
    id_fattura int(11) not null, -- id della tabella fatture
    numero_nota_credito int(11) not null check(numero_nota_credito > 0),
    data_emissione date not null check(anno >= 0), -- anno di emissione della fattura
    importo double not null check(importo >= 0),
    iva_calcolata double not null check(iva_calcolata >= 0),
    totale_rimanente double not null check(totale_rimanente >= 0),
    nome_report varchar(300) not null,
    contenuto_report blob not null,
    unique (numero_nota_credito, data_emissione),
    primary key(id),
    foreign key (id_fattura) references fatture(id)
);
```

Per tutti i campi è stato deciso di renderli tutti non nulli perché alla generazione della nota di credito tutti i campi devono essere compilati.

5.3 Implementazione dell' applicativo.

Si è passati poi alla scrittura del codice per integrare la nuova funzionalità nell' applicativo. Per questa parte sono stato aiutato da due sviluppatori interni all' azienda di nome Mauro Veronese della sede di Padova e Rocco Spenza della sede di Roma.

5.3.1 Bean delle note di credito

Come prima cosa dopo la creazione del database si è dovuto scrivere il Bean per poter mappare la tabella in Hibernate.

Bean della tabella delle note di credito

```
@Entity
@Table(name = "nota_di_credito")
public class NotaDiCredito implements Serializable {

    private static final long serialVersionUID = 3217061096964979625L;

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "id", nullable = false)
    private Integer id;

    @Column(name = "numero_nota_credito", unique = true, nullable = false)
    @Min(value = 1)
    private Integer numeroNotaCredito;

    @Column(name = "data_emissione")
    private Calendar dataEmissione;

    @Column(name = "iva_calcolata", nullable = false)
    @Min(value = 0)
    private double ivaCalcolata;

    @Column(name = "totale_rimanente", nullable = false)
    @Min(value = 0)
    private double totaleRimanente;

    @Column(name = "importo", nullable = false)
    @Min(value = 0)
    private double importo;

    @Column(name = "nome_report")
    private String nomeReport;

    @Column(name = "contenuto_report")
    private Blob contenutoReport;

    @ManyToOne
    @JoinColumn(name = "id_fattura", nullable = false)
    private Fattura idFattura;

    /**
     * .... costruttori + set e get omessi.
     */
}
```

Si è scelto inoltre di indicare tramite le notazioni Java di effettuare un Join automatico con la tabella delle fatture essendo che nella maggioranza delle operazioni era necessario avere anche i dati della fattura associata.

5.3.2 Lista delle fatture

Successivamente si è passato a modificare la lista di visualizzazione delle fatture fatta precedentemente integrando l' inserimento delle note di credito all' interno dell' interfaccia di inserimento fatture.

Menu

Home

davide , Consoft Informatica s.r.l.

Logout

Consulenti

Clienti

Aziende

Ordini

Tariffa

Fattura

Scadenziario

Fornitori Attivi

Cronologia

Modifica tipi Fattura

PDF Manager

Oscura Rapporti

Filtro

Cliente:

Tutti i clienti

Ordine:

Tutti gli ordini

Data Inizio:

gg/mm/aaaa

Data Fine:

gg/mm/aaaa

Filtra

Totale fatture: 1,039,151.81€

Totale Ivato: 1,267,765.62€

Fatture

N° / Data	Cliente	Totale (Iva)						
198 del 30/04/2016	Insurance Online SpA	5171.58€(22%)	Attiva	+				
197 del 30/04/2016	Insurance Online SpA	3586.80€(22%)	Attiva	+				
196 del 30/04/2016	Insurance Online SpA	1673.72€(22%)	Attiva	+				
195 del 30/04/2016	Insurance Online SpA	2690.10€(22%)	Attiva	+				
194 del 30/04/2016	Sysdata Italia SpA	5268.69€(22%)	Attiva	+				
193 del 30/04/2016	Engineering Ingegneria Informatica SpA	1708.00€(22%)	Attiva	+				
192 del 30/04/2016	Engineering Ingegneria Informatica SpA	213.50€(22%)	Attiva	+				
191 del 30/04/2016	Engineering Tributi SpA	5947.50€(22%)	Attiva	+				
190 del 30/04/2016	Engineering Ingegneria Informatica SpA	317.20€(22%)	Attiva	+				
189 del 30/04/2016	Engineering Ingegneria Informatica SpA	8772.52€(22%)	Attiva	+				
188 del 30/04/2016	Engineering Ingegneria Informatica SpA	3904.00€(22%)	Attiva	+				

Sono stati inseriti i due bottoni verdi  e  i quali sono rispettivamente per l' inserimento e la visualizzazione della nota di credito per ogni fattura.

5.3.3 Inserimento e modifica della nota di credito

Si è passato poi alla scrittura dell' interfaccia di inserimento e modifica della nota di credito che possiamo vedere nell' immagine sottostante.

Menu

Home

davide , Consoft Informatica s.r.l. Logout

Consulenti

Clienti

Aziende

Ordini

Tariffa

Fattura

Scadenziario


Fornitori Attivi

Cronologia

Modifica tipi Fattura

PDF Manager

Oscura Rapporti



Consoft Informatica s.r.l.

Sede legale: Via Salvatore Negretti

Sede operativa: via Francesco Scipione Orologio

P.IVA: 09154571005 C.FISC: 09154571005

Tel. 049/8071490 Fax 049/8086723

Web: <http://www.consoftinformatica.it>

info@consoftinformatica.it

Intestatario:

Insurance Online SpA

Via del Fischione 19

56010 Migliarino Pisano

Destinatario:



Insurance Online SpA

Via del Fischione 19

56010 Migliarino Pisano

Tipologia Documento	Nota di Credito		
Della Fattura	198	Del	2016
Partita IVA	01548970506		
Codice Fiscale	01548970506		
Importo della nota di credito	% Iva	Data di emissione	
300	22	01/06/2016	
Totale Fattura	Totale Rimanente	Iva calcolata	
5171.58	4805.58	66	

Aggiungi Nota


Consoft Informatica

Optima Solutions

La cui struttura è data dal seguente codice html.

Codice Html della parte frontend

```

<form:form method="post" action="${azione}"
  modelAttribute="notaCreditoForm" id="aggiungiNotaCredito"
  name="aggiungiNotaCredito">
<table class="table table-bordered">
  <tr>
    <td colspan="1">Tipologia Documento</td>
    <td colspan="3">Nota di Credito</td>
  </tr>
  <tr>
    <td colspan="1">Della Fattura</td>
    <td colspan="1">${fattura.numeroFattura}</td>
    <td colspan="1">Del</td>
    <td colspan="1">
      ${fattura.annoFattura}
    </td>
  </tr>
  <tr>
    <td colspan="1">Partita IVA</td>
    <td colspan="3">${fattura.cliente.partitaIva}</td>
  </tr>
  <tr>
    <td colspan="1">Codice Fiscale</td>
    <td colspan="3">${fattura.cliente.codiceFiscale}</td>
  </tr>
  <tr>
    <td colspan="1">Importo della nota di credito</td>
    <td colspan="1">% Iva</td>
    <td colspan="2">Data di emissione</td>
  </tr>
  <tr>
    <td colspan="1">Totale Fattura</td>
    <td colspan="1">Totale Rimanente</td>
    <td colspan="1">Iva calcolata</td>
    <td colspan="1"></td>
  </tr>
  <tr>
    <td colspan="1">5171.58</td>
    <td colspan="1">4805.58</td>
    <td colspan="1">66</td>
    <td colspan="1"></td>
  </tr>
</table>
  <div>
    <button type="button">Aggiungi Nota</button>
  </div>
</form>

```

```

<form:input path="idFattura" type="hidden" name="idFattura"
    value="${idFattura}" />
<td colspan="1"><form:input type="number"
    class="form-control" min="0.01" required="required"
    step="0.01" path="importo" id="importo"
    name="importo" /></td>
<td colspan="1">
<form:input type="number" required="required"
    class="form-control" min="0" max="100" step="0.01" path="iva"
    id="iva" name="iva"/></td>
<td colspan="2">
<form:input path="data_emissione" id="data_emissione"
    name="data_emissione" required="required"
    cssClass="form-control datepicker" data-provide="datepicker"
    autocomplete="off"/></td>
</tr>
<tr>
    <td colspan="1">Totale Fattura</td>
    <td colspan="1">Totale Rimanente</td>
    <td colspan="1">Iva calcolata</td>
</tr>
<tr>
    <td colspan="1">${totaleFattura}</td>
    <td colspan="1"><span id="tot">${totale}</span></td>
    <td colspan="1"><span id="ivc"></span></td>
</tr>
</table>
</div>
<form:errors path="importo" />

<form:errors path="iva" />

<form:errors path="data_emissione" />

<c:if test="${variabile == 0}">
    <input type="submit" value="Aggiungi Nota">
</c:if>

<c:if test="${variabile == 1}">
    <input type="submit" value="Modifica Nota">
</c:if>
</form:form>

```

Dove `${azione}` indica il tipo di web request che deve essere fatta alla sottomissione della form. La variabile `azione` viene determinata attraverso questo codice presente nella pagina jsp.

Codice per la decisione dell' azione da compiere

```

<c:if test="${variabile == 0}">
    <c:url var="azione" value="/notecredito/inserisci"></c:url>
</c:if>

<c:if test="${variabile == 1}">
    <c:url var="azione" value="/notecredito/modifica"></c:url>
</c:if>

```

Questo perché la stessa pagina viene utilizzata sia per la fase di modifica sia in quella di inserimento di una nuova nota di credito. Il calcolo del totale che verrà detratto in front end viene fatto al momento grazie a codice Javascript/jQuery che esegue i calcoli ogni volta che l'importo o l'iva viene modificato come si può vedere nel codice sottostante.

Codice javascript per la parte frontend

```
<script>
// datepicker function
$(function() {
    $("#datepicker").datepicker();
});
$('#iva, #importo').bind(
    "keyup mouseup",
    function() {
        var $totale = $("#importo").val();
        var $ivc = ($("#importo").val() *
            ($("#iva").val() / 100));
        $("#ivc").text($ivc);
        $('#tot').text(
            ("${totale}" - $totale - $ivc).toFixed(2));
    });
</script>
```

Tutti i calcoli però vengono fatti nuovamente tramite Java dentro al controller per evitare manomissioni nell' inserimento semplicemente modificando l'html. Come prima cosa all' interno della form abbiamo specificato una classe che serve per validare i dati inseriti.

form di controllo delle note di credito

```
public class NotaCreditoForm {
    @NotNull(message="L'iva non puo' essere vuota")
    @DecimalMin(value = "0", message = "L' iva non deve essere minore di 0")
    private double iva;
    @Pattern(regexp = "([0-9]{1,2}\\/{1}[0-9]{1,2}\\/{1}[0-9]{4})",
        message = ConstUtility.DATEFORM)
    private String data_emissione;
    @Min(value = 1, message = "L' id della fattura deve essere >= 1")
    private int idFattura;
    @NotNull(message = "L'importo non puo' essere vuoto")
    @DecimalMin(value = "0.01",
        message = "L' importo totale non deve essere minore di 0")
    private double importo;

    /**
     * set e get omissi
     */
}
```

Il cui oggetto viene specificato nell' attributo `ModelAttribute` del form.

parte di reperimento dati e controllo del controller delle note di credito

```
@RequestMapping(value = "/notecredito/inserisci", method = RequestMethod.POST)
public String inserimentoNotaCredito(@Valid @ModelAttribute
    NotaCreditoForm notaCreditoForm, BindingResult result,
    Model model, HttpServletRequest request) {
    try {
        int idFattura = notaCreditoForm.getIdFattura();
        Fattura fattura = fatturaService.getElementById(notaCreditoForm.getIdFattura());
        Cliente cliente = fattura.getCliente();
        Azienda azienda = fattura.getAzienda();
        double importo = notaCreditoForm.getImporto();
        double ivaCalcolata = importo * (notaCreditoForm.getIva()/100);
        double totaleNotaCredito = importo + ivaCalcolata;
        // ivaCalcolata = NumUtil.formatDouble(ivaCalcolata, 2);
        int numeroNota = notaDiCreditoService.getNextNumeroNota();
```

```

double totaleRimanente = notaDiCreditoService.getLastTotRim(notaCreditoForm.getIdFattura())
    - NumUtil.dueDecimaliDouble(totaleNotaCredito);
Calendar calendar = Calendar.getInstance();
Calendar data = DataUtility.dateFromString(notaCreditoForm.getData_emissione());
int caso = 0;

if (result.hasErrors()) {
    caso = 1;
    model.addAttribute("message", "Errore nell inserimento dati");
}
if ((importo < 0) | (ivaCalcolata < 0) | (numeroNota < 0)) {
    caso = 1;
    model.addAttribute("message", "Errore, inseriti importi negativi ");
}
if (totaleRimanente < 0) {
    caso = 1;
    model.addAttribute("message",
        " Errore, importo nota di credito superiore al totale della Fattura ");
}
if (data.get(Calendar.YEAR) != calendar.get(Calendar.YEAR)) {
    caso = 1;
    model.addAttribute("message", "Errore, nota di credito non relativa a quest'anno");
}

```

Infine, se tutti i controlli vanno a buon fine, avviene la creazione della nota di credito in formato pdf e il caricamento di tutti i dati nel database come mostrato nel codice sottostante.

generazione del pdf e caricamento di tutti i dati

```
// Genero nota di credito
NotaDiCredito notaCredito = new NotaDiCredito(numeroNota,
    DataUtility.dateFromString(notaCreditoForm.getData_emissione()),
    NumUtil.dueDecimaliDouble(ivaCalcolata), NumUtil.dueDecimaliDouble(totaleRimanente),
    NumUtil.dueDecimaliDouble(importo), fattura);
// Genero pdf
byte[] pdf = PdfNotaDiCredito.generaPdfNotaCredito(fattura, azienda, cliente, notaCredito,
    sedeLegaleAzienda, sedeOperativaAzienda, datiBancariAzienda, indirizzoCliente);
notaCredito.setNomeReport("Nota di credito per la fattura n'" + fattura.getNumeroFattura()
    + fattura.getDescrizione() + ".pdf");
notaCredito.setContenutoReport(new SerialBlob(pdf));
// Commit
notaDiCreditoService.add(notaCredito);
List<NotaDiCredito> lista = this.notaDiCreditoService.getAllByNumFat(idFattura);

if (lista.size() > 0) {
    model.addAttribute("numeroFattura", fattura.getNumeroFattura());
    model.addAttribute("lista", lista);
    model.addAttribute("annoFattura", fattura.getAnnoFattura());
    model.addAttribute("idFattura", idFattura);
    model.addAttribute("modificabile", lista.get(0).getId());
    model.addAttribute("cancellabile", this.notaDiCreditoService.getLastId());
} else {
    model.addAttribute("listaVuota",
        "Non c'è nessuna nota per la fattura " + notaCreditoForm.getIdFattura());
}
} catch (Exception e) {
    logger.error("Errore nell' esecuzione delle operazioni di inserimento della nota di credito.", e);
    return "error";
}
model.addAttribute("message", "Nota di credito inserita correttamente!");

return "notacredito/visualizzaTutte";
}
```

5.3.4 Lista delle note di credito

La lista delle note di credito relativa ad una fattura può essere visualizzata in due modi

1. Attraverso la pressione del tasto nella lista delle fatture.
2. Appena finito di compilare e salvare nel database una nota di credito.

Front End

Un esempio di lista delle note di credito è la seguente

Menu

Home

davide , Consoft Informatica s.r.l.

Logout

Consulenti

Clienti

Aziende

Ordini

Tariffa

Fattura

Scadenziario

Fornitori Attivi

Cronologia

Modifica tipi Fattura

PDF Manager

Oscura Rapporti

Nota di credito inserita correttamente!

←

Note di Credito della Fattura 198 del 2016

Importo	TotaleRimanente			
300.0	4805.58			

Consoft

Consoft Informatica

Optima Solutions

Bibliografia

- [1] Sito ufficiale Consoft Informatica
<http://www.consoftinformatica.it>
- [2] Felipe Gutierrez, *Introducing Spring Framework: A Primer*, Apress.