

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

Davide Fontana



# Indice

<b>1</b>	<b>Introduzione</b>	<b>5</b>
<b>2</b>	<b>Azienda ospitante</b>	<b>7</b>
<b>3</b>	<b>Il corso di formazione</b>	<b>9</b>
3.1	Hibernate . . . . .	9
3.2	Spring . . . . .	9
3.2.1	Spring MVC . . . . .	10
<b>4</b>	<b>Il Programma di Fatturazione</b>	<b>15</b>
4.1	Struttura . . . . .	15
<b>5</b>	<b>Progetto delle Note di Credito</b>	<b>17</b>
5.1	Analisi dei Requisiti . . . . .	17
5.2	Database . . . . .	18
5.2.1	Schema Concettuale . . . . .	18
5.2.2	Schema Logico . . . . .	19
5.2.3	Codice SQL . . . . .	19
5.3	Implementazione della funzionalità . . . . .	19
5.3.1	Bean delle note di credito . . . . .	19
5.3.2	Lista delle fatture . . . . .	20
5.3.3	Inserimento e modifica della nota di credito . . . . .	21
5.3.4	Lista delle note di credito . . . . .	26
5.3.5	Eliminazione della nota di credito . . . . .	28
5.3.6	Visualizzazione del pdf della nota di credito . . . . .	29
<b>6</b>	<b>Conclusione del lavoro e commenti finali</b>	<b>31</b>



# 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 lo sviluppo di software 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 bisogna scrivere delle classi Java chiamate Bean con delle specifiche notazioni che indicano, per ogni variabile d'istanza della classe, l'attributo della tabella corrispondente. Un Bean è una classe Java che ha solo variabili di istanza private e metodi set e get.

### 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

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 con un metodo `set`. 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.

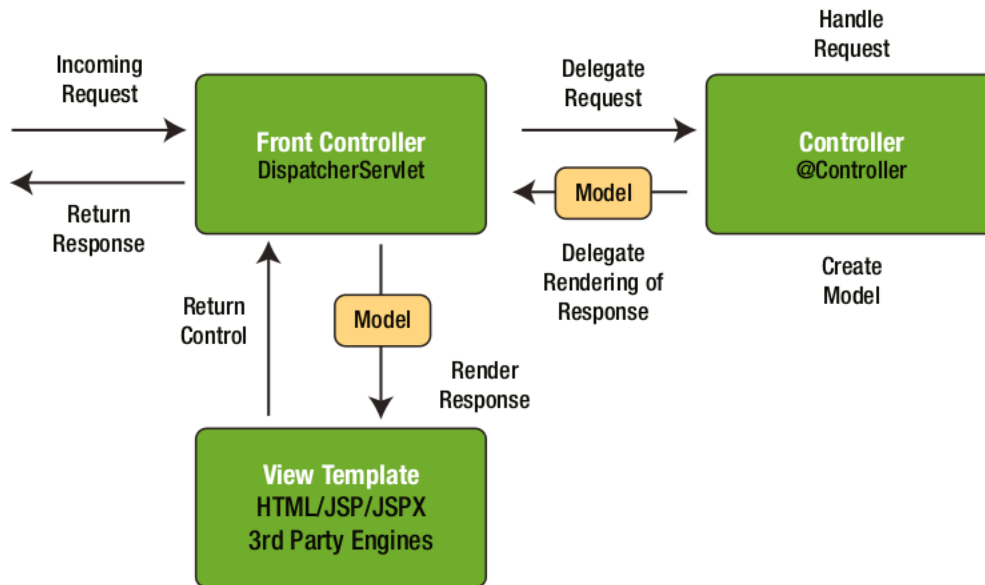
Spring al momento è un framework molto vasto e comprende diverse estensioni. Una delle più usate è Spring MVC.

### 3.2.1 Spring MVC

Questa estensione è usata per la creazione delle pagine web dove MVC è un design pattern di tipo strutturale e sta per Model View Controller. Impone di dividere i file all' interno del software in 3 parti con le seguenti funzioni:

- Model: Sono tutte le classi che hanno il compito di elaborare o reperire dati.
- View: Tutte le pagine web o classi che presentano i dati all' utente presi dalle classi Model.
- Controller: Sono il tramite tra le classi model e le classi view. Da una parte validano i dati delle view e li inviano alle classi model. Vice versa prendono i dati elaborati dal model e, dopo averli controllati, li passano alle pagine di visualizzazione.

Il funzionamento di Spring MVC è spiegato nell' immagine (Si veda [2] per maggiori dettagli).



Per prima cosa abbiamo una richiesta web inviata dall' utente la quale viene ridirezionata tramite un Front Controller ad una classe chiamata Controller da noi implementata. Infine dopo aver effettuato le operazioni richieste viene data come risposta una pagina web (html o jsp). 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;

    public void setDocumentDAO(DocumentDAO documentDAO) {
        this.documentDAO = documentDAO;
    }

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

Il controller 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 tutte le richieste all' url `<sito web>/search` verranno indirizzate a questa classe. Ogni funzione ha un' ulteriore notazione e serve per specificare quale funzione verrà chiamata in base all' url richiesto. Esempio: Nel caso in cui l' url sia `<sito web>/search/all` allora il front controller chiamerà la funzione `searchAll` della classe `SearchController`.
- **@Autowired:** Serve per iniettare l' implementazione dell' oggetto `DocumentDAO` chiamando il metodo `setDocumentDAO`.

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. Per poterli inserire bisogna prima di tutto importarli nel codice della pagina `jsp` con la seguente stringa

Stringa per importare i tag `jstl` nelle pagine `jstl`

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
```

Dove l' attributo `prefix` serve per distinguere lo schema dei tag da quelli standard dell' `html`. Alcune delle cose che è possibile fare con le `jstl` sono:

- verificare delle condizioni sui dati tramite dei tag che ricordano gli `if`
- stampare una lista di oggetti tramite cicli `for`

Un esempio di utilizzo possiamo vederlo nella pagina di visualizzazione in cui vengono mostrati i risultati passati dal controller visto in precedenza.

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  $\${\dots}$  indica il contenuto all' interno dell' attributo di nome docs.



# Capitolo 4

## Il Programma di Fatturazione

Dopo il periodo di formazione mi è stata assegnata un' attività di bugfixing su un programma di fatturazione online. Il programma è stato scritto in java e utilizza diversi componenti:

- Il framework Spring MVC per la parte web.
- Il framework Hibernate per interfacciarsi al database.
- Il DBMS (DataBase Management System) MySQL per salvare i dati.
- La libreria JasperReports per la generazione dei report in pdf.
- La libreria Log4j per salvare i log delle operazioni del programma.

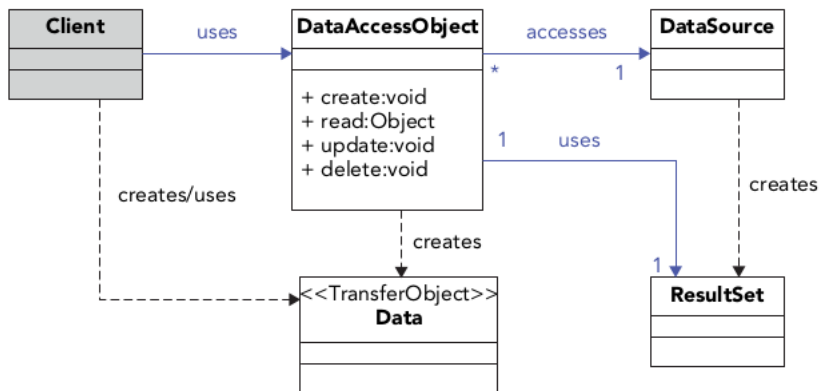
Tutto il lavoro è stato svolto su portatile munito di sistema operativo Microsoft Windows 10 e salvato in un repository aziendale con il CVS (Control Version System) Subversion.

### 4.1 Struttura

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.
- `it.consoft.fatturazione.bean`: Contiene tutti i Bean per l' utilizzo in Hibernate.
- `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 creazione, lettura, aggiornamento e cancellazione (in inglese per brevità viene chiamato con l'acronimo CRUD ovvero Create, Read, Update e Delete) di ogni tabella del database gestendone inoltre la connessione. Tutto questo viene semplificato grazie al framework Hibernate e quindi il codice da scrivere risulta contenuto. 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 per interrogare il database e possono chiamare funzioni di altri Service. Questa distinzione tra DAO e Service è stata fatta seguendo 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, rendendo possibile la simulazione degli oggetti, 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.

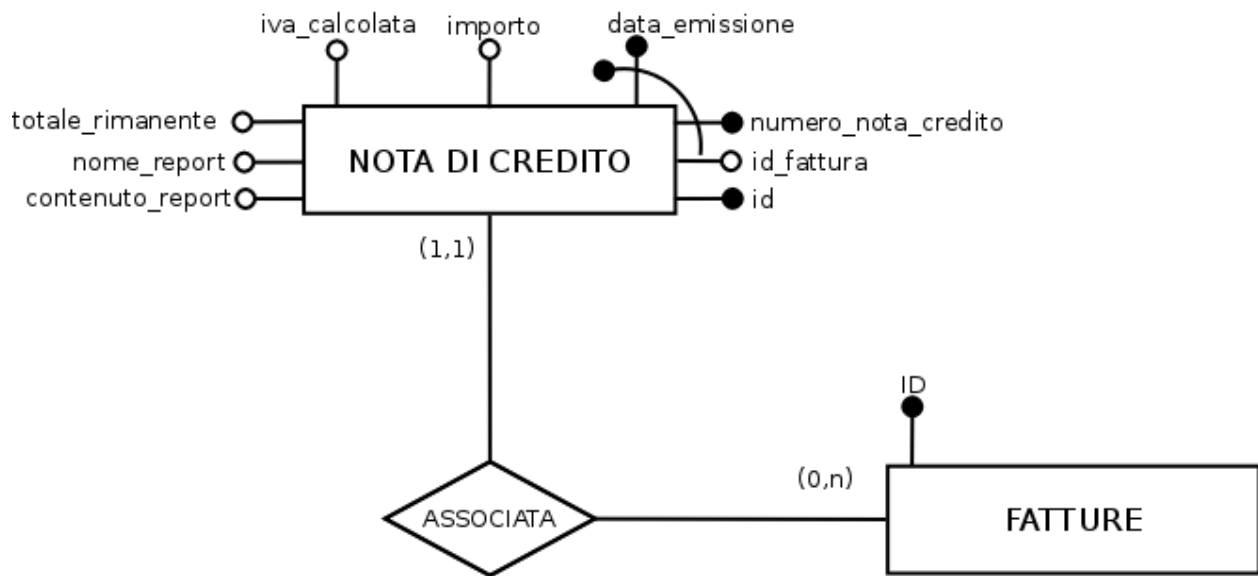
Ogni nota di credito è relativa ad un' unica fattura ma una fattura può avere più note di credito. E' composta dalle informazioni della azienda e il numero della fattura a cui questa nota di credito fa riferimento. Il numero della fattura è incrementale solo rispetto all' anno in corso e non alla fattura 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.

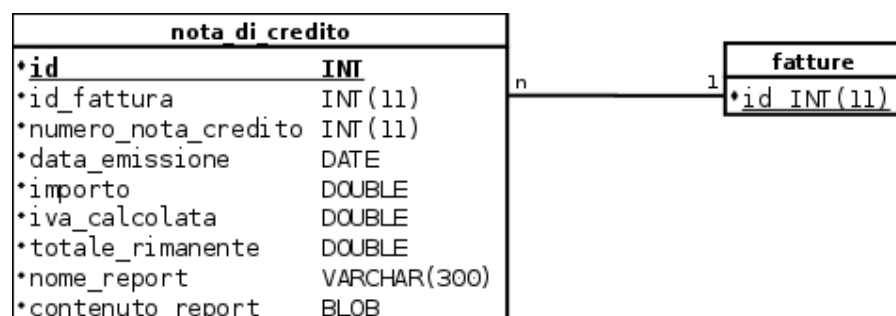


Sono state attuate le seguenti scelte sugli attributi:

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 fatture.
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 Codice SQL

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 della funzionalità

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

In seguito abbiamo cominciato a costruire l' interfaccia per l' inserimento e la modifica di una nota di credito. Si è reso indispensabile per una interfaccia coerente con entrambe le operazioni di utilizzare una sola pagina web. Per determinare il tipo di pagina si è deciso di verificare dentro alla pagina jsp, attraverso il valore di un attributo dentro alla web request, per poter poi inviare la form alla funzione del controller più adatta.

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>

```

La pagina web di inserimento finita si presenta così.

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](mailto: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>

```

```

<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 che viene determinata attraverso il codice visto precedentemente. Il calcolo del totale che verrà detratto viene fatto con codice Javascript/jQuery che esegue i calcoli ogni volta che l'importo o l'iva viene modificato come si può vedere qui sotto.

### 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>

```

## Inserimento

Prima dell' inserimento dei dati nel database tutti i calcoli però vengono fatti anche in Java dentro al controller per evitare manomissioni nell' inserimento semplicemente modificando l'html. Inoltre per si utilizza una funzione di Spring per la validazione dei dati. Consiste nell' associare al form di inserimento un oggetto di una classe simile ad un Bean con delle notazioni Java che verificano se i parametri inviati corrispondono al dominio richiesto. La classe è stata chiamata **NotaCreditoForm** e la sua implementazione è qui sotto.

### 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 omessi
     */
}

```

Per poter procedere alla vera e propria validazione bisogna, all'interno della funzione del controller che riceve i dati, specificare all' interno dei parametri il nome dell' oggetto da validare e un oggetto di tipo **BindingResult** già implementato Spring e che procede materialmente alla validazione tramite il metodo booleano **hasErrors**. Nel caso in cui la validazione presenti degli errori verranno caricati nell' oggetto model i messaggi di errore all' interno delle notazioni nella classe di validazione e che verranno automaticamente visualizzati dentro ai tag **form:errors** per ogni attributo in cui è stato trovato un errore. La fase di controlli è scritta in questo listato di codice.

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

```

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");
}
switch (caso) {

case 1:

    model.addAttribute("notaCreditoForm", notaCreditoForm);
    model.addAttribute("sedelegale", this.indirizzoService
        .getSedeLegaleByAzienda(this.fatturaService.getElementById(idFattura).getAzienda()));

```

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.

### generazione del pdf e caricamento di tutti i dati

```

        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'e' 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";
}

/**
 *
 * Questo metodo effettua la modifica di un dato all' interno di un

```

```

    * database.
    */
@RequestMapping(value = "/notecredito/modifica", method = RequestMethod.POST)

```

## Modifica

Per la parte di modifica è necessario prima di tutto caricare i dati esistenti dato l'id della nota di credito da modificare.

### Codice per il caricamento dei dati della nota di credito

```

@RequestMapping(value = "/notecredito/premodifica", method = RequestMethod.POST)
public String premodificaNotaCredito(@ModelAttribute("nota") int id, Model model) {
    try {
        NotaDiCredito nota = notaDiCreditoService.getElementById(id);
        Fattura fattura = nota.getFattura();
        int idFattura = fattura.getId();
        NotaDiCredito notaToLoad = notaDiCreditoService.getElementById(nota.getId());
        double importo = notaToLoad.getImporto();
        double totale = importo + notaToLoad.getIvaCalcolata();
        double iva = (totale / notaToLoad.getImporto()) - 1;
        iva = NumUtil.dueDecimaliDouble(iva);
        iva = iva * 100;
        model.addAttribute("notaCreditoForm", new NotaCreditoForm(importo, iva, notaToLoad.
            getDataEmissione()));
        model.addAttribute("idFattura", idFattura);
        model.addAttribute("variabile", 1);
        if(getTotRimPrec(idFattura) == -1) {
            model.addAttribute("totale", NumUtil.dueDecimaliDouble(this.fatturaService.
                getElementById(idFattura).getTotaleFattura()));
        } else {
            model.addAttribute("totale", NumUtil.dueDecimaliDouble(getTotRimPrec(idFattura))
            );
        }
    }
    /*
     * caricamento dei restanti dati omesso.
     */
}

```

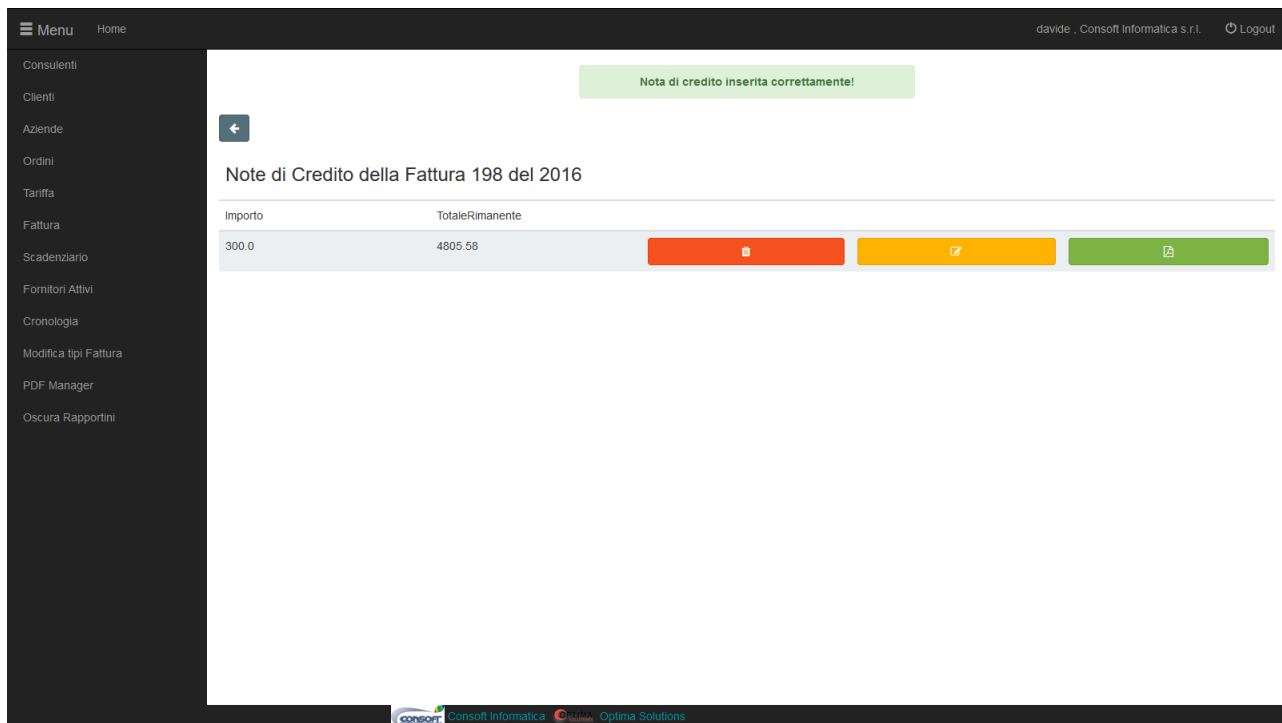
Il caricamento dei dati viene fatto utilizzando la classe `NotaCreditoForm` usata nella fase di inserimento per la validazione. Poi dopo aver effettuato le modifiche si usa la stessa procedura che abbiamo visto nella fase di inserimento della nuova nota di credito.

### 5.3.4 Lista delle note di credito

Dopo l'inserimento e la modifica si è poi passati a scrivere la pagina jsp che serve per visualizzare le note di credito precedentemente inserite. 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.

Un esempio di lista delle note di credito è la seguente immagine



che viene generata dalla seguente parte di codice della pagina `visualizzeTutte.jsp` La modifica e/o la cancellazione delle note di credito viene verificata grazie all' attributo modificabile e cancellabile che rappresenta l' id della nota di credito al quale è associata tale azione.

### Parte front end per la visualizzazione delle note di credito

```
<tbody>
  <c:forEach items="${lista }" var="note">
    <tr>
      <td class="col-sm-2">${note.importo }</td>
      <td class="col-sm-2"><fmt:formatNumber type="number" pattern="####0.00"
        maxIntegerDigits="6" maxFractionDigits="2"
        value="${note.totaleRimanente}" /></td>
      <c:if test="${note.id != cancellabile && cancellabile!=-1}">
        <td>Nota di credito non cancellabile</td>
      </c:if>
      <c:if test="${note.id == cancellabile && cancellabile!=-1}">
        <td><form:form rel="tooltip" data-title="Elimina nota" method="POST" action
          ="${eliminaNota }">
          <input type="hidden" name="idFattura" value="${idFattura}">
          <button
            class="btn btn-block btn-danger" type="submit">
            <i class="fa fa-trash" aria-hidden="true"></i>
          </button>
        </form:form></td>
      </c:if>
      <c:if test="${note.id == modificabile}">
        <td class="col-sm-2"><form:form rel="tooltip" data-title="Modifica nota"
          method="POST"
          action="${modifica }">
          <input type="hidden" name="variabile" value="1">
          <input type="hidden" name="nota" value="${note.id }" >
        </td>
      </c:if>
    </tr>
  </c:forEach>
</tbody>
```

```

                <button type="submit" class="btn btn-block btn-warning"><i class="
                    fa fa-pencil-square-o" aria-hidden="true"></i></button>
            </form:form></td>
        </c:if>
        <c:if test="${note.id != modificabile}">
            <td>Nota di credito non piu' modificabile</td>
        </c:if>
        <td><button rel="tooltip" data-title="Visualizza nota" type="button" class="btn
            btn-block btn-success"
            onclick="window.open('${vediPDF}/id=${note.id}', '_blank');return false"><i
                class="fa fa-file-pdf-o" aria-hidden="true"></i></button>
            <input type="hidden" name="id" value="${note.id}"></td>
        </tr>
    </c:forEach>
</tbody>

```

La parte del controller è piuttosto semplice infatti, dato l'id della fattura, vengono ritornate tutte le sue note di credito compresi gli id della nota di credito che può essere modificata e/o cancellata.

### Funzione del controller per ritornare la lista delle note di credito.

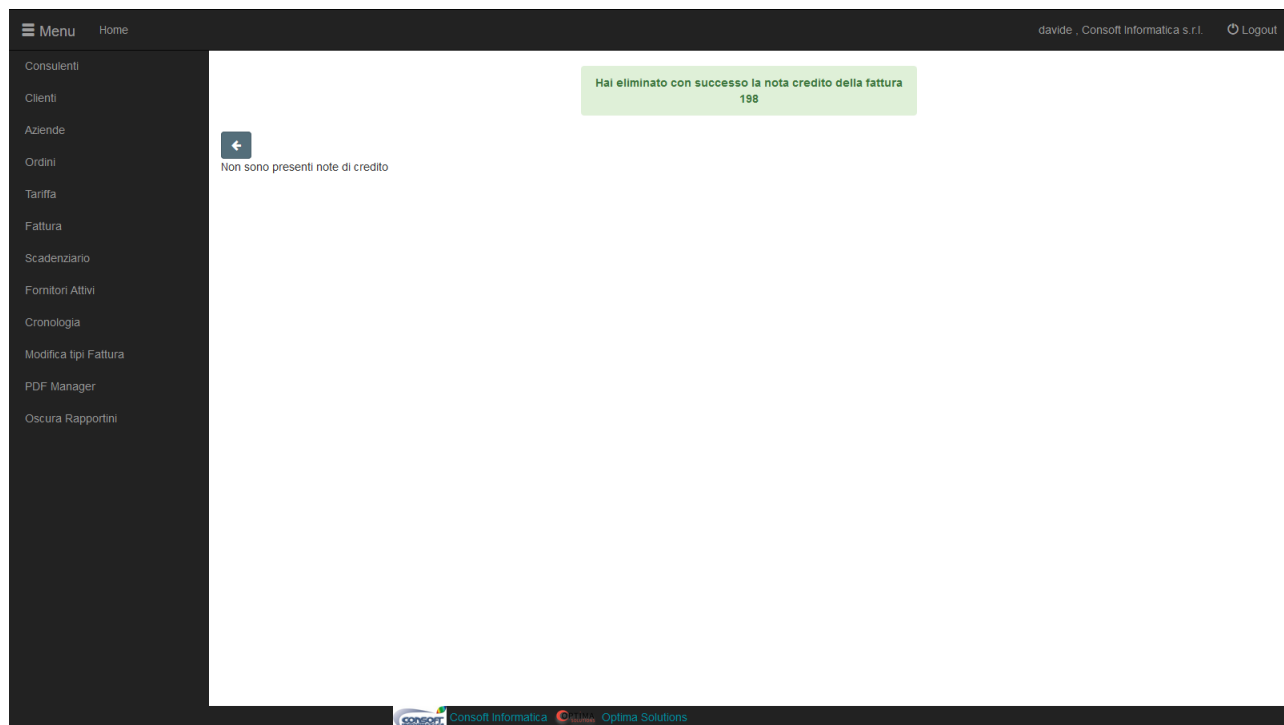
```

@RequestMapping(value = "/notecredito/elimina", method = RequestMethod.POST)
public String eliminaNota(@RequestParam("idFattura") int id, Model model) {
    try {
        List<NotaDiCredito> lista = this.notaDiCreditoService.getAllByNumFat(id);
        this.notaDiCreditoService.delete(lista.get(0));
        Fattura fattura = this.fatturaService.getElementById(id);
        model.addAttribute("message",
            "Hai eliminato con successo la nota credito della fattura " + fattura.
                getNumeroFattura());
        lista = this.notaDiCreditoService.getAllByNumFat(id);
        if (lista.size() > 0) {
            model.addAttribute("lista", lista);
            fattura = this.fatturaService.getElementById(id);
            model.addAttribute("annoFattura", fattura.getAnnoFattura());
            model.addAttribute("numeroFattura", fattura.getNumeroFattura());
            model.addAttribute("idFattura", fattura.getId());
            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 " + id);
        }
    } catch (Exception e) {
        model.addAttribute("errore", "Errore nell' eliminazione delle note della fattura " +
            id);
        logger.error("Errore nella eliminazione delle note della fattura " + id, e);
    }
    return "notacredito/visualizzaTutte";
}

```

### 5.3.5 Eliminazione della nota di credito

Dopodiché si è passati alla funzione di eliminazione della nota di credito. Per l'eliminazione nel controller, dopo aver premuto il tasto di cancellazione, si cancella dal database la nota selezionata e viene visualizzata nuovamente la lista delle rimanenti note di credito. Nel caso in cui non ci siano note da mostrare la pagina avrà il seguente messaggio per comunicare che non ci sono note credito presenti.



### 5.3.6 Visualizzazione del pdf della nota di credito

Infine per la visualizzazione della nota di credito viene generata, nel caso di un report non nullo, una response http di un documento in pdf settando nell' header di risposta il tipo di documento (in questo caso pdf), il nome del file ed infine il contenuto del file che viene inviato utilizzando la classe `OutputStream` per poter inviare i byte del file attraverso lo stream di risposta. Un esempio di nota di credito generate lo possiamo vedere nell' immagine qui sotto.



## Capitolo 6

# Conclusione del lavoro e commenti finali

Terminato lo sviluppo e dopo aver testato tutto nell' ambiente di prova tutto il lavoro svolto è stato caricato sul server di produzione senza la presenza di problemi in fase di post produzione. Durante tutto il lavoro io e gli altri due sviluppatori abbiamo collaborato attivamente tra di noi e con la nostra responsabile di progetto che, anche se non era presente fisicamente per motivi di lavoro, è stata sempre presente da remoto e ha sempre risposto ai nostri dubbi. Tutto il tirocinio si è svolto senza problemi e le persone con cui ho avuto il piacere di collaborare durante il mio periodo in azienda sono stati sempre cordiali e disponibili in ogni situazione.





# Bibliografia

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