ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA
SCUOLA DI SCIENZE
CORSO DI LAUREA IN SCIENZE E TECNOLOGIE INFORMATICHE

PROGETTAZIONE E REALIZZAZIONE DI UN SOFTWARE GESTIONALE WEB-BASED SU PIATTAFORMA CLOUD

Relazione finale in
BASI DI DATI

Relatore
Chiar.mo Prof. Dario Maio

Correlatore
Dott.ssa Annalisa Franco

Presentata da
Matteo Zanuccoli

Terza Sessione
Anno Accademico 2013 - 2014

Sommario

Indice delle tabelle

Indice delle figure

1 – Introduzione

La realizzazione di un software gestionale web-based su piattaforma cloud nasce dalla necessità, da parte di alcuni clienti dell'azienda *Librasoft snc,* di poter disporre di un sistema per la gestione delle varie anagrafiche, del magazzino, delle vendite e degli acquisti, i cui dati siano accessibili da remoto e che non richieda alcuna installazione preliminare.

La necessità, mostrata da questi clienti, di poter accedere ai dati da remoto, deriva dal trend attuale definito "Always On" ("Sempre collegato"). Grazie allo sviluppo esponenziale che hanno avuto, negli ultimi anni, le infrastrutture di telecomunicazioni e grazie anche alla presenza di dispositivi mobili sempre più potenti e reattivi, l'utente è perennemente collegato alla rete. (Figure 1 e 2)

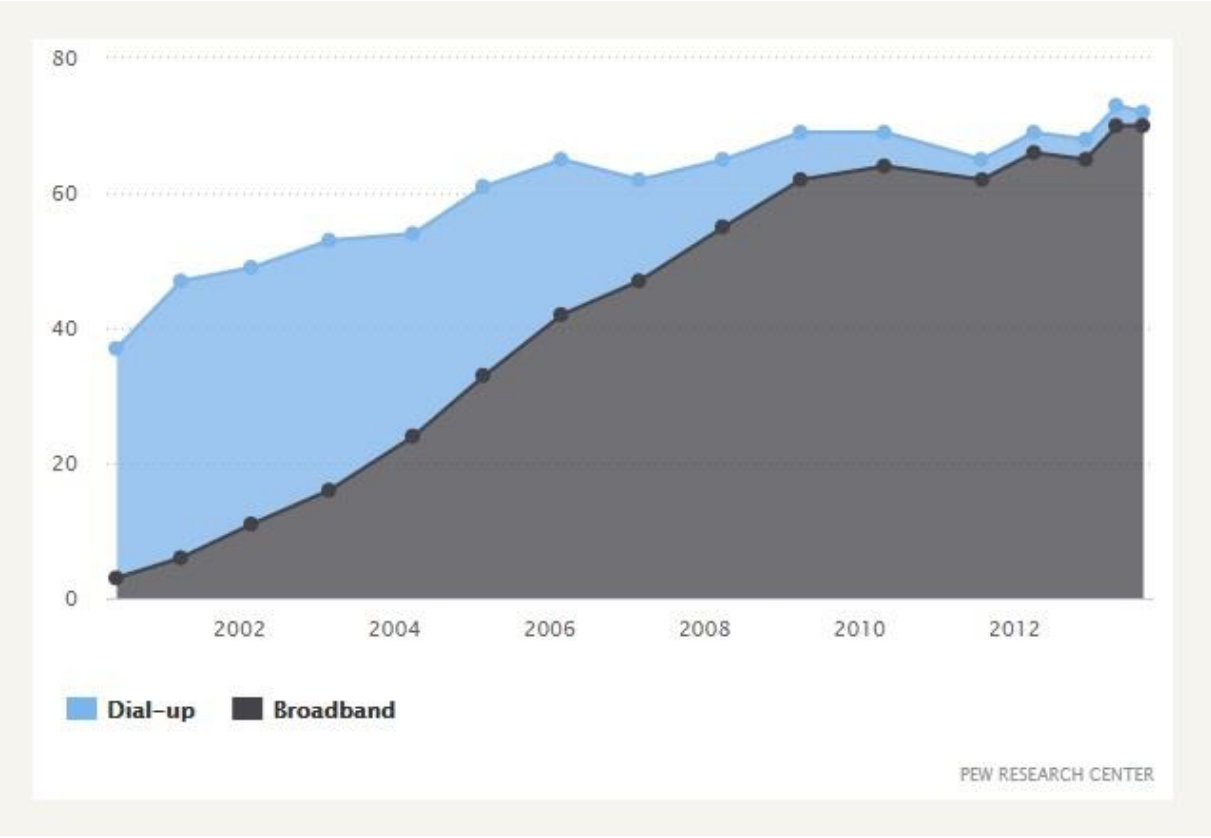


Figura 1 – Crescita della banda larga

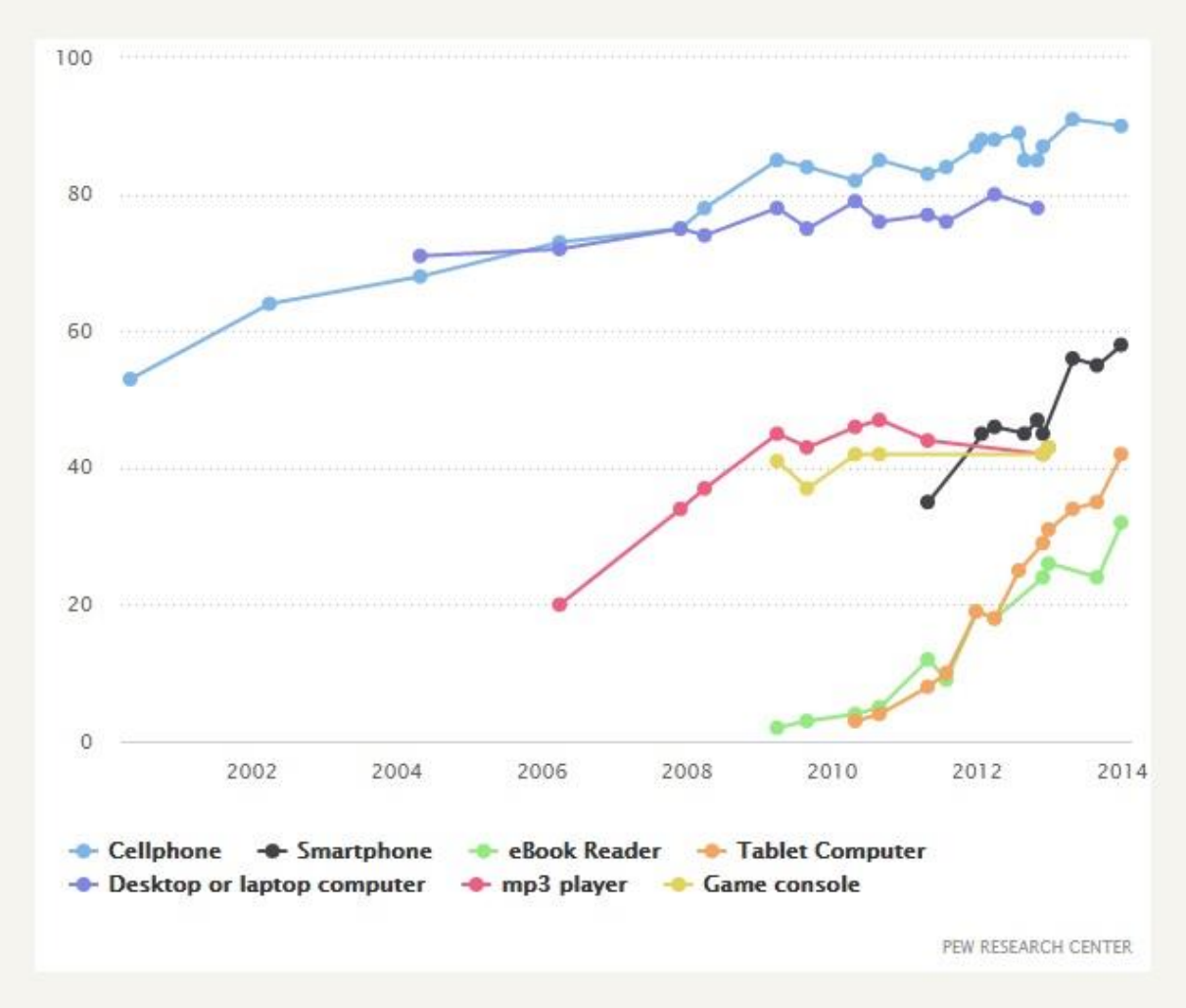


Figura 2 – Crescita dei dispositivi mobili

"First, the rise of the internet changed the way that people got information and shared it with each other, affecting everything from users' basic social relationships to the way that they work, learn, and take care of themselves." **[PEWINT]**

Analizzando la citazione riportata sopra ci si accorge che, in effetti, questo sviluppo tecnologico, oltre ad aver modificato il modo in cui le persone si relazionano fra di loro, ha anche cambiato il modo in cui le persone **lavorano**. Fino a pochi anni fa, lo scambio di informazioni lavorative avveniva, perlopiù, per via telefonica. Nella società odierna invece, il sempre maggiore utilizzo della tecnologia ha sviluppato la necessità di sostituire le comunicazioni telefoniche con un sistema online in grado di sincronizzare il lavoro di più utenti. Questa necessità non è del tutto infondata.

Una soluzione web-based, rispetto ad una soluzione locale porta innumerevoli vantaggi:

- Non è richiesta installazione di software da parte dell'utente**.** Essendo una applicazione web, è sufficiente collegarsi all'opportuno URL per poter usufruire del servizio.
- I dati sono accessibili da qualsiasi postazione collegata ad internet**.** È possibile accedere al sistema anche con smartphone e tablet.
- I dati sono al sicuro. Se la piattaforma cloud implementa meccanismi di RAID e di backup, la possibilità di perdere dati è molto remota.
- Le nuove feature non richiedono aggiornamenti da parte dell'utente. Non sempre gli utenti sono disposti ad aggiornare il proprio software e questo provoca una frammentazione delle versioni.

Viceversa, uno dei maggiori limiti che deriva dall'uso di questa tecnologia è l'impossibilità di lavorare in assenza di connessione. Nell'ultimo capitolo di questa relazione, sarà presentata una soluzione per ovviare a questo problema.

L'obiettivo di questo lavoro di tesi è quindi la realizzazione del sistema introdotto a inizio capitolo: si presterà particolare attenzione alla progettazione della base di dati e all'implementazione delle funzionalità CRUD (Create Read Update Delete) su di essa. Sarà infine presentato l'aspetto grafico dell'applicazione che, grazie all'utilizzo delle più moderne tecnologie web, presenterà un layout moderno in linea con i dettami del Material e del Flat Design. **[MATER] [FLAT]**

La realizzazione di questo sistema prevedrà alcune fasi del ciclo di vita di un software. **[IS]**

La prima fase sarà quella di studio del dominio applicativo, di analisi dei requisiti e di progettazione concettuale della base di dati. Successivamente, si procederà con la progettazione logica, che porterà alla definizione dello schema della base di dati relazionale e concluderà la fase di progettazione.
Una volta definita una solida base di dati si procederà alla realizzazione della Web-Application, che offrirà all'utente una interfaccia grafica per eseguire le varie operazioni richieste.

A fronte di quanto detto la relazione si divide in sette capitoli:

- Il primo capitolo è una introduzione al progetto e alle problematiche affrontate.
- Nel secondo capitolo si condurrà una veloce analisi di un prodotto concorrente per poi, in seguito, procedere con l'intervista col committente. In seguito all'intervista si procede con l'analisi dei requisiti che descrive in linguaggio naturale ciò che sarà richiesto al sistema.
- Nel terzo capitolo si passa alla progettazione graduale dello schema concettuale, rappresentato tramite il formalismo Entity/Relationship **[ER01] [ER02]**, partendo da uno schema scheletro e raffinandolo sempre di più fino ad ottenere le viste di ogni sottoinsieme del sistema. Integrando le viste si ottiene lo schema concettuale finale.
- Nel quarto capitolo si procede con la progettazione logica. Essa deve tener conto di specifiche caratteristiche richieste dall'ORM (Object-relational Mapping) e dal framework utilizzato per l'implementazione **[CAKE02]**. Al termine della progettazione logica si arriva infine alla definizione dello schema relazionale raffinato.
- Nel quinto capitolo si parla delle tecnologie utilizzate per l'implementazione.
- Nel sesto capitolo si parla del sistema realizzato, ponendo l'attenzione sull'interfaccia grafica.
- Nel settimo capitolo vengono presentati alcuni sviluppi futuri di questo progetto, già in corso d'opera al momento della stesura di questa relazione, e presentate infine le conclusioni.

2 – Analisi dei requisiti

2.1 – Premessa

Il sistema richiesto dal committente parte da un progetto precedentemente sviluppato in collaborazione con l'azienda *Librasoft snc*.
Da questo progetto è nato un sistema informativo web-based finalizzato alla gestione di:

- Agenti
- Clienti / Gruppi di Clienti
- Prodotti / Categorie di Prodotti
- Ordini in uscita
- Listini di prezzi
- Utenti

Tuttavia, analizzando le esigenze dei committenti dell'azienda e studiando le funzionalità di altri prodotti similari, ci si è accorti che le funzionalità esposte erano insufficienti per poter diventare un prodotto commercialmente apprezzabile.

2.2 – Case Study: Giobby.

Prima di intraprendere la classica strada di analisi dei requisiti, si analizzano le funzionalità di un sistema già affermato nel mondo dei gestionali in cloud: Giobby. **[GIOBBY].**
Questo software è sviluppato dalla *Digitalsoft SRL*, un'azienda italiana che si occupa di sviluppo software e consulenza IT.
Accedendo al sistema, si può notare che questo offre numerose funzioni di gestione, presentando inoltre un aspetto grafico professionale che ricorda la Metro UI di Microsoft. (Figura 3)

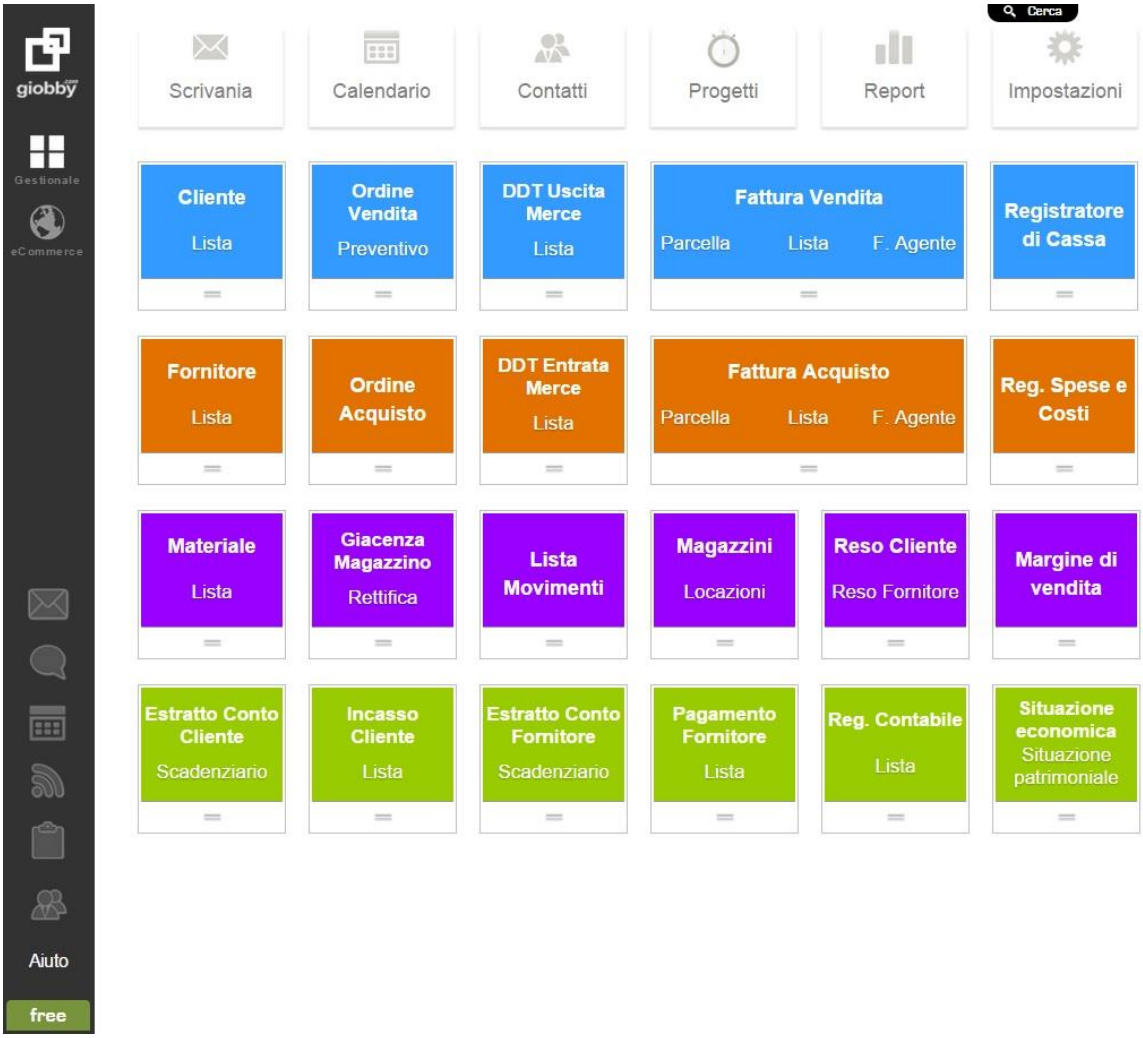


Figura 3 – Homepage di Giobby

Dopo qualche ora di utilizzo si è potuto constatare che il prodotto analizzato risulta però poco intuitivo a livello di User Experience (UX). Le funzionalità offerte sono talmente tante che risulta difficile individuare la sezione giusta nella quale trovare la funzionalità ricercata. Questa problema è probabilmente dovuto alla generalità che il software si prefigge di avere. Soluzioni embedded o **modulari** potrebbero risultare più intuitive e produttive.

2.3 – Funzionalità aggiuntive richieste

La prima funzionalità aggiuntiva richiesta è quella della gestione dei magazzini. Mentre nella precedente versione del sistema era previsto un solo magazzino (E quindi le quantità dei prodotti e altre caratteristiche rilevanti erano gestite direttamente nell'anagrafica dei prodotti), viene adesso richiesta la possibilità di una gestione multi-magazzino.

La seconda funzionalità aggiuntiva è quella della gestione degli ordini in entrata e, di conseguenza, anche dei fornitori e di tutte le anagrafiche che ne conseguono.
La terza funzionalità aggiuntiva è quella di poter interfacciare il gestionale con un sistema e-commerce.

La complessità del problema presentato è tale da dover riprogettare interamente il sistema partendo dalla base di dati.

2.4 – Specifiche in linguaggio naturale

In questo paragrafo viene trascritta la descrizione in linguaggio naturale dei requisiti che il sistema dovrà soddisfare. Essendo un colloquio preliminare, la descrizione riguarderà solo una parte del sistema finale, e sarà in seguito necessario analizzare le specifiche richieste per estrarre i concetti fondamentali ed eliminare le eventuali ambiguità derivanti dall'utilizzo di un linguaggio naturale e dal diverso tipo di conoscenze del committente rispetto al progettista.

Di seguito la trascrizione dell'intervista.

"La figura centrale del sistema[1] sono i nostri clienti. I nostri clienti sono identificati da un codice alfanumerico univoco[2] (Ad esempio "CL1", "CL2" ecc.) che gli viene assegnato al momento dell'inserimento[3] nel sistema. Le informazioni più importanti associate ad ogni cliente sono il nome, il cognome, il codice fiscale[4] nel caso siano privati o la partita IVA[5] nel caso siano aziende, uno o più indirizzi, uno o più numeri di telefono, uno o più indirizzi e-mail e una descrizione facoltativa[6]. I clienti devono essere in grado di comprare i nostri prodotti.
Ogni nostro prodotto[7] è identificato da un codice alfanumerico (Ad esempio "PR1", "PR2" ecc.), ma su di esso è anche stampato un codice a barre[8]. Le informazioni più importanti associate ai prodotti sono il nome, la descrizione, il prezzo unitario[9] e la quantità che abbiamo in ogni magazzino. I prodotti si dividono in varie categorie e in vari brand[10]. Per una migliore gestione dei magazzini e delle spedizioni vogliamo anche avere a disposizione alcuni parametri quali il peso e la dimensione del prodotto.

Un'altra figura importante del sistema sono i nostri agenti[11]. I nostri agenti si occupano di trovare nuovi clienti, a cui vendere i nostri prodotti. Ogni agente è identificato da un codice alfanumerico (Ad esempio "AG1", "AG2" ecc.). Le informazioni rilevanti associate ai nostri agenti sono il nome e il cognome, il codice fiscale e/o la partita IVA, uno o più indirizzi, uno o più numeri di telefono, uno o più indirizzi e-mail e una descrizione facoltativa.

Il sistema deve essere in grado di gestire sia gli ordini che facciamo ai nostri clienti[12], sia gli ordini che facciamo ai nostri fornitori[13].

Gli ordini che facciamo ai nostri clienti sono identificati da un codice numerico e da una data. Quando un ordine viene inserito si trova nello stato[14] "in gestione"; successivamente potrà passare ad altri stati come "evaso", "spedito", "ricevuto". Altre informazioni rilevanti dell'ordine sono il cliente associato[15], l'utente che ha eseguito l'ordine[16], il prezzo totale[17] dell'ordine (Con e senza IVA[18]) e una serie di dettagli d'ordine caratterizzati da prodotto, prezzo unitario e quantità.

Gli ordini che facciamo ai nostri fornitori sono molto simili. Essi sono identificati da un numero progressivo e da una data. Sono inoltre caratterizzati da un fornitore associato[19], da l'utente che ha eseguito l'ordine, da uno stato, da un prezzo totale (Con e senza IVA) e da una serie di dettagli d'ordine.

Gli ordini vanno a modificare le quantità di prodotti presenti nei nostri magazzini.

Il sistema deve essere in grado di gestire più magazzini. Ogni magazzino[20] è identificato da un codice alfanumerico (Ad esempio "WH1", "WH2" ecc.) ed è caratterizzato da un nome, da una descrizione, da uno o più indirizzi, da uno o più numeri di telefono e da uno o più indirizzi e-mail. Deve inoltre essere possibile assegnare ad ogni magazzino un ruolo[21] e/o un una categoria.

Un'altra figura rilevante del sistema sono i nostri fornitori. Essi sono identificati da un codice alfanumerico (Ad esempio "SU1", "SU2" ecc.) e sono caratterizzati da un nome/ragione sociale[22], da un codice fiscale e/o da una partita iva e da una descrizione testuale facoltativa.

Coloro che possono accedere al sistema sono gli utenti. Gli utenti sono identificati da una e-mail e per accedere devono inserire una password alfanumerica[23]. L'unica informazione rilevante per un utente è il nome, in modo che questo possa essere visualizzato in seguito al login[24].

È inoltre richiesta una funzionalità di logging[25] dei movimenti del magazzino. Gli ordini in entrata[26] causano un carico[27] di prodotti nel magazzino, mentre gli ordini in uscita[28] causano uno scarico[29]. Per motivi statistici desideriamo poter accedere a queste informazioni. Alcuni dati rilevanti possono essere la data, il tipo di operazione eseguita (Carico, scarico), la causa, la quantità spostata, il prodotto spostato, l'utente che ha eseguito l'operazione e il magazzino.
Associati agli ordini possono o meno esserci dei documenti. Questi documenti possono essere di diverse tipologie (Fattura[30], DDT[31] ecc.) e sono identificati da un numero progressivo[32] e da una data. Altre informazioni rilevanti possono essere lo stato e una nota facoltativa.

2.5 – Disambiguazione dei termini

Si procederà ora con una rapida analisi del testo sopra riportato, in modo da eliminare eventuali ambiguità terminologiche, che possono compromettere l'esatta comprensione del testo. Questa analisi descriverà ogni termine "tecnico" del dominio applicativo in questione, in modo da facilitare la comprensione anche ai "non addetti ai lavori".

In seguito sarà riscritta l'intervista in una forma più sintetica e organizzata per poter poi procedere alla vera e propria fase di progettazione concettuale, nella quale verranno rappresentati i primi diagrammi E/R scheletro, che fungeranno da base al processo delle raffinazioni successive **[ER02]**. Una volta raffinati tutti i diagrammi E/R, questi verranno integrati per rappresentare lo schema concettuale completo.

Si procede ora con l'identificazione dei termini ambigui o troppo generici, in modo da fornirne una descrizione più dettagliata e accurata. (Tabella 1)

Tabella 1 – Termini ambigui o troppo generici

N. Riferimento	Termine	Significato
1	Sistema	Può essere inteso come "sistema software".
2	Codice alfanumerico univoco	Insieme di numeri e lettere che formano un codice. Questo codice può identificare univocamente nel sistema l'entità al quale si riferisce.
3	Inserimento	Possibili sinonimi possono essere "caricamento" o "introduzione".
4	Codice fiscale	Codice alfanumerico di lunghezza fissa di 16 caratteri che serve ad identificare univocamente ai fini fiscali i cittadini italiani.
5	Partita IVA	Sequenza di cifre che identifica univocamente un soggetto che esercita un'attività rilevante ai fini dell'imposizione fiscale indiretta (IVA).
6	Descrizione facoltativa	Il salvataggio deve essere possibile anche se viene lasciato vuoto il campo della descrizione.
7	Nostro prodotto	Viene inteso come prodotto "da noi venduto" e non come "da noi prodotto".
8	Codice a barre	Insieme di elementi grafici a contrasto elevato disposti in modo da poter essere letti da un sensore a scansione e

		decodificati per restituire l'informazione contenuta.
9	Prezzo unitario	Prezzo del singolo prodotto. Ci possono essere ambiguità nel caso i prodotti siano venduti a casse o in altre forme.
10	Brand	Marca del prodotto.
11	Agenti	Figura atta a promuovere i contratti commerciali di vendita tra l'azienda committente e i clienti potenziali.
12	Ordini che facciamo ai nostri clienti	Perifrasi che equivale a "Ordini in uscita" o a "Ordini di vendita"
13	Ordini che facciamo ai nostri fornitori	Perifrasi che equivale a "Ordini in entrata" o a "Ordini di acquisto"
14	Stato	Stato di avanzamento dell'ordine.
15	Cliente associato	Cliente al quale viene emesso l'ordine.
16	L'utente che ha eseguito l'ordine	L'utente che ha compilato ed emesso effettivamente l'ordine.
17	Prezzo totale	Somma totale dei prodotti fra prezzo unitario e quantità di ogni dettaglio d'ordine.
18	Con o senza iva	Importo lordo o importo netto.
19	Fornitore associato	Fornitore associato all'ordine in entrata.

20	Magazzino	Luogo dove vengono effettivamente immagazzinati i prodotti
21	Ruolo	Utilizzo specifico di quel magazzino (Principale, secondario, di appoggio ecc.).
22	Ragione Sociale	Nome di una società di persone.
23	Password alfanumerica	Parola d'ordine segreta composta da numeri e lettere necessaria per il processo di autenticazione al sistema.
24	Login	Processo di autenticazione
25	Logging	Tracciamento.
26	Ordini in entrata	Ordini ai fornitori.
27	Carico	Operazione che aumenta le quantità di uno o più prodotti in magazzino.
28	Ordini in uscita	Ordini ai clienti.
29	Scarico	Operazione che diminuisce le quantità di uno o più prodotti in magazzino.
30	Fattura	Documento fiscale obbligatorio emesso da un soggetto fiscale per comprovare la cessazione di beni o prestazione di servizi e il diritto a riscuoterne il prezzo.
31	DDT	Documento di trasporto emesso per giustificare il trasferimento di un materiale da cedente a cessionario.

32	Numero progressivo	Codice numerico che viene incrementato di un'unità all'emissione del documento.

2.6 – Analisi dei concetti fondamentali

Utilizzando un testo ristrutturato e ordinato verranno ora individuati i concetti chiave dai quali si partirà per la creazione dei primi schemi scheletro, i quali verranno raffinati e integrati nelle successive fasi di progettazione fino ad ottenere lo schema definitivo.

Anagrafica dei clienti

I clienti sono identificati da un codice alfanumerico, che non sia necessariamente il codice fiscale o la partita IVA (Ad esempio "CL1", "CL2" ecc.). Ogni cliente è caratterizzato da un nome e da un cognome, da un codice fiscale e/o partita IVA e da una descrizione e/o note facoltative. Deve inoltre essere possibile assegnare ad ogni cliente uno o più indirizzi e uno o più contatti che possono essere di diverse tipologie (Telefono, Cellulare, FAX, Mail ecc.). È richiesta inoltre una categorizzazione dei clienti.

Anagrafica degli agenti

Gli agenti sono identificati da un codice alfanumerico, che non sia necessariamente il codice fiscale o la partita IVA (Ad esempio "AG1", "AG2" ecc.). Ogni agente è caratterizzato da un nome e da un cognome, da un codice fiscale e/o partita IVA e da una descrizione e/o note facoltative. Deve inoltre essere possibile assegnare ad ogni agente uno o più indirizzi e uno o più contatti che possono essere di diverse tipologie (Telefono, Cellulare, FAX, Mail ecc.). È richiesta inoltre una categorizzazione degli agenti.

Anagrafica dei prodotti

I prodotti sono identificati da un codice alfanumerico, che non sia necessariamente il codice a barre stampato su di esso (Ad esempio "PR1", "PR2" ecc.). Ogni prodotto è

caratterizzato da un codice a barre, da un nome, da una descrizione e/o note facoltative e da un prezzo unitario. I prodotti presenti in magazzino sono caratterizzati da una quantità e da altre caratteristiche rilevanti per il loro immagazzinamento (Come ad esempio peso, altezza, larghezza e profondità). È richiesta inoltre una categorizzazione dei prodotti.

Anagrafica dei magazzini

I magazzini sono identificati da un codice alfanumerico (Ad esempio WH1, WH2 ecc.). Ogni magazzino è caratterizzato da un nome e da una descrizione e/o note facoltative. Deve inoltre essere possibile assegnare ad ogni magazzino uno o più indirizzi e uno o più contatti che possono essere di diverse tipologie (Telefono, Cellulare, FAX, Mail ecc.). È richiesta inoltre una categorizzazione dei magazzini e una gestione dei ruoli (Magazzino principale o secondario, adibito all'e-commerce ecc.)

Anagrafica dei fornitori

I fornitori sono identificati da un codice alfanumerico, che non sia necessariamente il codice fiscale o la partita IVA (Ad esempio "SU1", "SU2" ecc.). Ogni fornitore è caratterizzato da un nome (Ragione sociale), da un codice fiscale e/o partita IVA e da una descrizione e/o note facoltative. Deve inoltre essere possibile assegnare ad ogni fornitore uno o più indirizzi e uno o più contatti che possono essere di diverse tipologie (Telefono, Cellulare, FAX, Mail ecc.). È richiesta inoltre una categorizzazione dei fornitori.

Utenti

Gli utenti sono identificati da una e-mail e per accedere al sistema devono inserire una password alfanumerica. L'unica informazione rilevante per un utente è il nome, in modo che questo possa essere visualizzato in seguito al login.

Ordini in uscita

Gli ordini in uscita sono identificati da un codice numerico e da una data. Ogni ordine in uscita è caratterizzato da uno stato, da una descrizione e/o note facoltative e da un prezzo totale (Tassato o non tassato). Ad ogni ordine in uscita è assegnato un cliente, un agente, un utente che ha inserito l'ordine nel sistema e una serie di dettagli d'ordine (Prodotti e relative quantità).

Ordini in entrata

Gli ordini in entrata sono identificati da un codice numerico e da una data. Ogni ordine in entrata è caratterizzato da uno stato, da una descrizione e/o note facoltative e da un prezzo totale (Tassato o non tassato). Ad ogni ordine in entrata è assegnato un fornitore, un utente che ha inserito l'ordine nel sistema e una serie di dettagli d'ordine (Prodotti e relative quantità).

Tracciamento movimenti magazzino

Gli ordini in entrata e in uscita causano una modifica nelle quantità dei prodotti in magazzino. È richiesto il tracciamento di questi movimenti per motivi statistici. Ogni tracciamento è caratterizzato da una data, da un tipo (Carico o scarico), da una causa e/o da note facoltative e dalla quantità che è stata mossa (Positiva o negativa). Ogni tracciamento è legato al prodotto, al magazzino e all'utente che ha eseguito il movimento.

Salvataggio documenti

Il sistema deve essere in grado di registrare i documenti associati agli ordini in entrata e in uscita. Questi documenti sono identificati da un numero e da una data. Ogni documento è caratterizzato da un tipo (Fattura, DDT ecc.), da uno stato e da note facoltative.

Grazie al lavoro di analisi che è stato svolto fino a questo punto, è ora possibile individuare le entità su cui formare lo schema scheletro.

Osservando la versione schematizzata del testo, appare chiaro che gli elementi chiave dello schema sono le varie anagrafiche (Clienti, Prodotti, Agenti, Magazzini, Fornitori) e le varie entità che supportano le funzionalità di inserimento di ordini e tutto ciò che ne consegue (Carico e scarico da magazzini, generazione documenti ecc.).

Uno dei punti che rimangono più oscuri in questa prima fase di analisi è il meccanismo di categorizzazione dei clienti, degli agenti, dei prodotti, dei magazzini e dei fornitori. È richiesta pertanto una seconda breve intervista col committente per mettere in chiaro quale tipo di categorizzazione sia richiesta.

2.7 – Seconda Intervista col committente

In questo paragrafo viene trascritta la seconda intervista col committente, richiesta dall'azienda *Librasoft snc*, per eliminare le ambiguità presenti nella prima intervista, per quanto riguardava la categorizzazione dei clienti, degli agenti, dei prodotti, dei magazzini e dei fornitori. Esistono infatti numerose tipologie di categorizzazione e raggruppamento in informatica (Gerarchie, tag/label ecc.) e occorre fin dalle prime fasi di progettazione capire che uso intende farne il committente, in modo da utilizzare il pattern più adeguato.

I sistemi di categorizzazione da noi richiesti servono per porre ordine all'interno del sistema. Sicuramente il sistema di categorizzazione più interessante è quello dei prodotti. Dovendo il sistema interfacciarsi con il nostro e-commerce, dev'essere possibile riprodurre lo stesso tipo di categorizzazione dei prodotti. I prodotti devono essere divisi in una gerarchia di categorie: esistono una o più categorie "radici", all'interno delle quali si trovano le sottocategorie. Ogni sottocategoria appartiene a una e una sola categoria.
Deve comunque essere possibile associare uno o più tag di ricerca a ogni prodotto indipendentemente dalla categoria a cui appartiene.
Per quanto riguarda i clienti la situazione è simile. Desideriamo un sistema di categorizzazione dei clienti per poter eseguire determinate operazioni su un gruppo preciso di essi. Alcune di queste operazioni possono essere l'invio delle mail, l'applicazione di sconti speciali ecc.
Lo stesso vale per agenti e fornitori.
Tutto ciò non vale per quanto riguarda la categorizzazione dei magazzini. Quella dei magazzini più che una categorizzazione è un'assegnazione del ruolo.

Questa seconda intervista col committente ha colmato qualche dubbio, ma ne ha sollevati molti altri. Si rimanda quindi alle prossime fasi di progettazione un'analisi più approfondita su quello che sembra essere uno dei punti più cruciali e più sensibili del sistema. Probabilmente la soluzione più adeguata sarà quella di adottare un approccio "ibrido" fra categorizzazione gerarchica ad albero e categorizzazione a tag.

3 – Progettazione Concettuale

Occorre innanzitutto definire una strategia di progettazione **[ER02]**. Nel nostro caso la scelta ricade fra:

- Progettazione Top-Down
- Progettazione Bottom-Up

Nel primo paragrafo verrà scelta la strategia di progettazione più adeguata e sarà spiegato il perché di tale scelta. In seguito si procederà alla vera e propria progettazione concettuale.

3.1 – Scelta della strategia di progettazione

I concetti cardine del dominio applicativo sono pochi e le relazioni che intercorrono fra di essi sono semplici. Al contrario, all'interno di uno stesso concetto, ci sono molte relazioni, alcune delle quali ancora da mettere bene a fuoco, e scelte progettuali che possono portare ad anomalie e a cali di prestazioni.
Per questi motivi è vantaggioso optare per un strategia di progettazione top-down.

"La progettazione top-down è uno stile di progettazione in cui si inizia specificando ***parti complesse*** *e* ***suddividendole successivamente in parti più piccole*** *(divide et impera). Questo è l'esatto opposto della progettazione bottom-up.*
Il nome top-down significa ***dall'alto verso il basso****: in "alto" viene posto il problema e in "basso" i* ***sotto problemi che lo compongono****. Il nome ricorda anche una raffigurazione a piramide: l'obiettivo finale è la* ***cima della piramide****, e i sotto problemi che lo compongono* ***formano la base****.*
Il top down ***parte dall'obiettivo*** *e da esso fa scaturire la strategia direttamente adatta a determinare l'obiettivo stesso, quindi* ***valorizza il perché e da esso fa dipendere il come****; nell'ambito della strategia mirata a determinare direttamente l'obiettivo individua le risorse necessarie, precisa quelle disponibili e identifica quelle mancanti,* ***propone successivamente ogni risorsa mancante come sub-obiettivo*** *ovvero come sotto-problema in cui ciascun sub-obiettivo richiede una sub-strategia ad esso correlata."* **[WIKI]**

Alcuni dei vantaggi di questa scelta di strategia di progettazione sono:

- Permette di semplificare la soluzione del problema, suddividendolo in problemi più semplici;
- Permette di suddividere il lavoro tra persone diverse, che si possono occupare ciascuna di un sotto problema;
- Permette di rendere più comprensibile il lavoro e quindi più facile la manutenzione successiva.

Come previsto dalla strategia di progettazione top-down, saranno ora individuati i concetti principali e ne sarà rappresentato lo schema scheletro. Successivamente si procederà ad un rapido raffinamento di ogni concetto.

3.2 – Individuazione dei nodi principali

Sarà ora sviluppato lo schema Entity/Relationship, assumendo come base di partenza le interviste trascritte nel capitolo precedente; si procederà, tramite raffinamenti successivi, all'arricchimento delle aree in cui il sistema informativo sarà suddiviso, fino a giungere alla rappresentazione completa del dominio preso in esame.

Prima di procedere allo sviluppo dello schema E.R. scheletro, ricordiamo i principali concetti estratti nel capitolo precedente:

- Clienti
- Agenti
- Prodotti
- Magazzini
- Fornitori
- Utenti
- Ordini in uscita
- Ordini in entrata
- Movimenti magazzino
- Documenti in uscita

- Documenti in entrata
- Categorizzazione per:
    - Prodotti
    - Clienti
    - Agenti
    - Fornitori
    - Magazzini

Come possiamo notare da questo riepilogo, alcuni dei concetti presenti nel sistema altro non sono che **Persone**. Clienti, Agenti, Fornitori e Utenti sono quattro sotto-classi della super-classe "Persona" (Figura 4) e formano quindi una gerarchia di generalizzazione **[ER02]**.
Altre possibili gerarchie sono quella fra Ordini in entrata e Ordini in uscita, come sotto-classi della super-classe "Ordine" (Figura 5) e quella fra Documenti in entrata e Documenti in uscita, come sotto-classi della super-classe "Documento" (Figura 6).

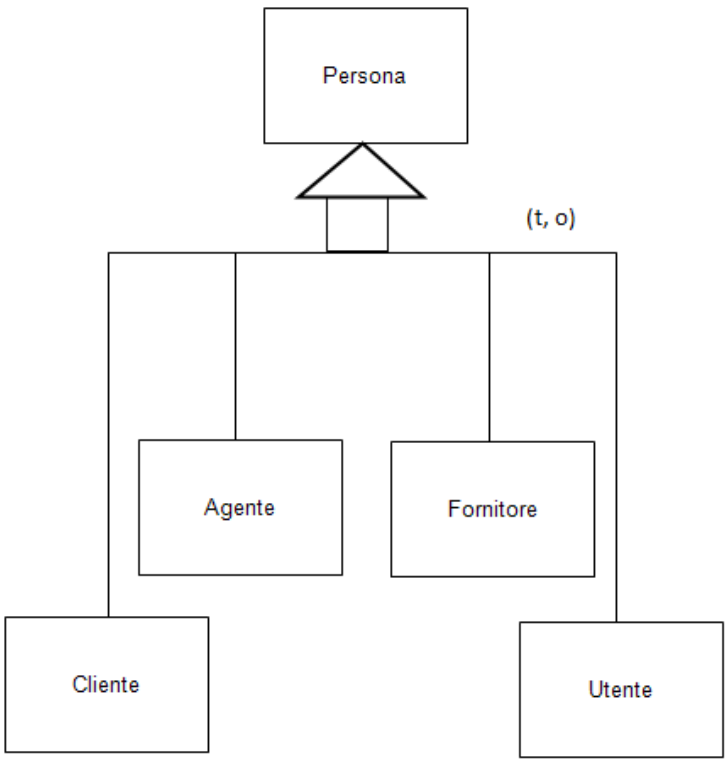


Figura 4 – Gerarchia Persona

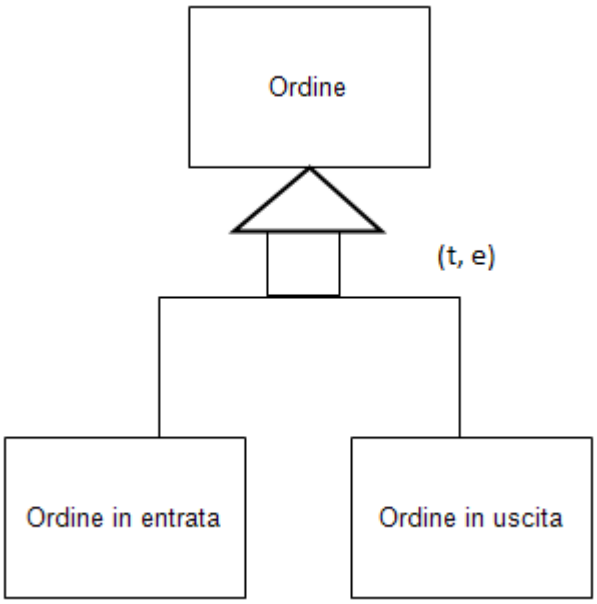


Figura 5 – Gerarchia Ordine

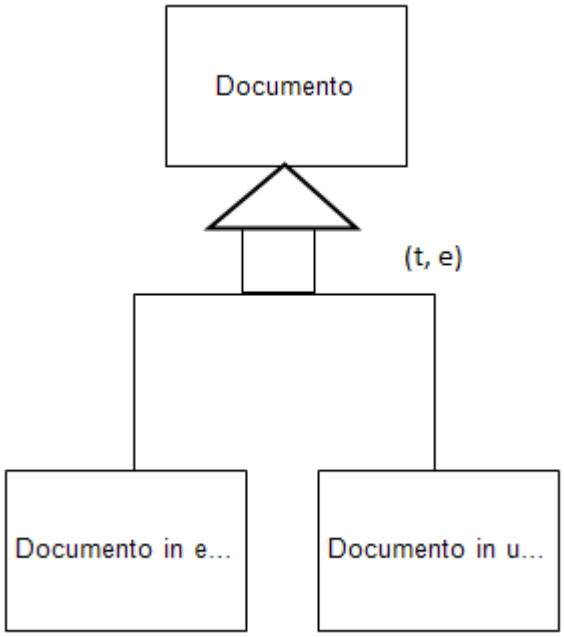


Figura 6 – Gerarchia Documento

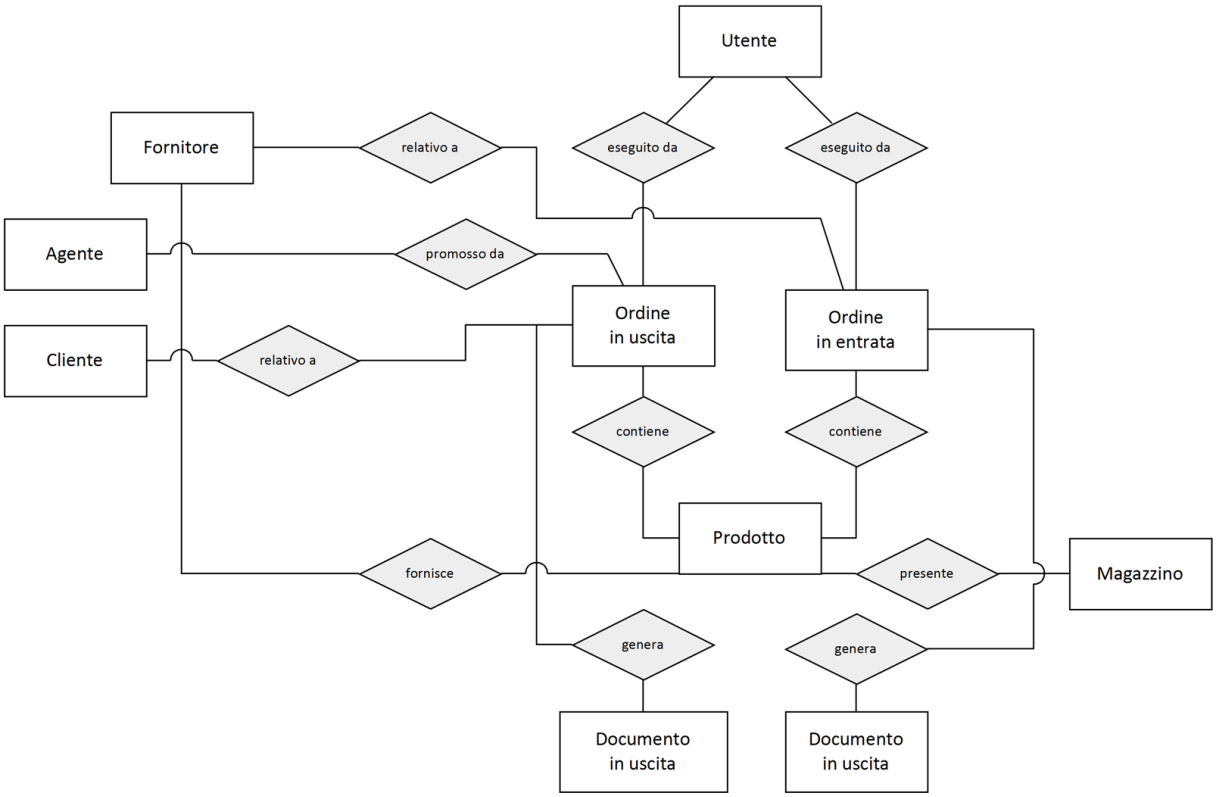
Per rendere più chiaro lo schema scheletro, più semplice la progettazione concettuale e date le richieste del committente, risulta vantaggioso trattare separatamente le entità di questa gerarchia, in modo da non generare confusione nelle successive fasi di progettazione. Queste entità hanno infatti molte caratteristiche comuni, ma hanno funzioni e ruoli completamente diversi.

Da questo punto in poi perciò le quattro entità sopra citate verranno trattate separatamente, in modo da modellare le richieste del committente in maniera più adeguata.

Un discorso analogo vale per la gerarchia degli ordini e dei documenti.

3.3 – Schema scheletro

Grazie a una attenta analisi dei requisiti, che ha portato all'individuazione di 10 entità, è possibile procedere alla creazione dello schema scheletro che rappresenta le entità individuate e le relazioni che intercorrono tra esse.



3.4 – Raffinamento e sviluppo delle entità

Si procede ora, come previsto dalla strategia di progettazione Top-Down, ad un raffinamento delle singole entità. Le singole entità saranno modellate in modo tale che rispecchino, per quanto possibile, le richieste espresse dal committente durante le interviste. Dopo aver raffinato tutte le entità, si sposterà l'attenzione sulle relazioni che intercorrono fra di esse. Infine, una volta raffinate entità e relazioni, si integreranno i risultati di questa fase di progettazione per ottenere lo schema concettuale finale, che servirà da input per la successiva fase della progettazione: la progettazione logica.

3.4.1 – Raffinamento delle entità “Cliente”, “Agente” e “Fornitore”

Come già preannunciato nel capitolo precedente, le entità in questione altro non sono che persone. Dal testo dell’intervista è emerso che, per tutte le entità di tipo “Persona”, è richiesto un salvataggio delle informazioni anagrafiche. Si procede quindi all’introduzione degli attributi necessari per il salvataggio di queste informazioni.

Di seguito una possibile rappresentazione grafica delle entità (Figura 7):

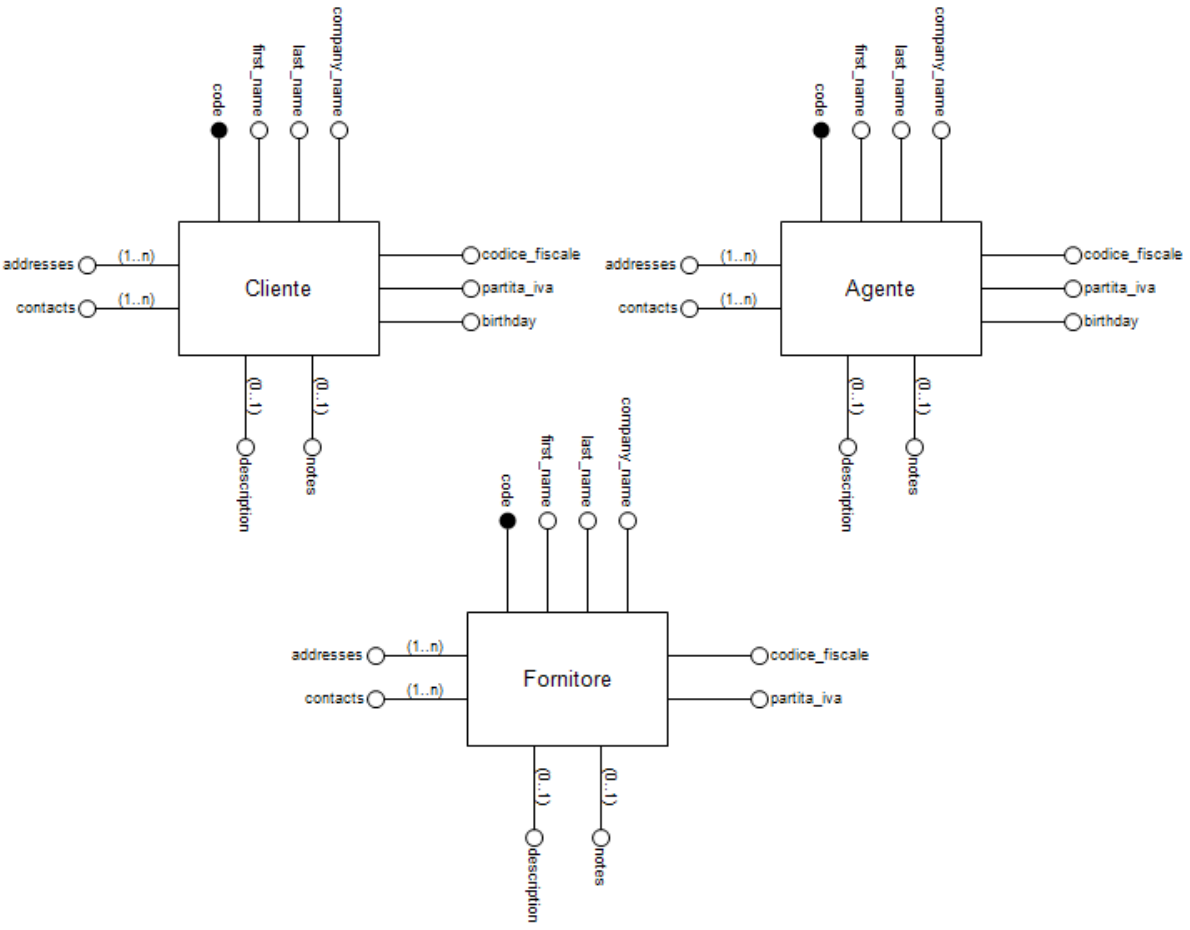


Figura 7 – Entità Cliente, Agente e Fornitore

Da notare che gli attributi **addresses** e **contacts** sono attributi composti con cardinalità (1..n) e che gli attributi **description** e **notes** hanno cardinalita (0..1).
Per una descrizione completa di tutti gli attributi si rimanda il lettore all’Appendice A. (Tavole 1, 2, 3)

Le entità possono considerarsi raffinate.

3.4.2 – Raffinamento dell'entità "Utente"

L'entità utente è stata esclusa dalla gerarchia "Persona" in quanto il committente non desidera un salvataggio dei dati anagrafici per i suoi utenti. Dal testo dell'intervista è infatti emerso che l'unica informazione anagrafica rilevante ai fini funzionali è il nome dell'utente, mentre le informazioni più rilevanti sono quelle che ne consentono l'autenticazione.
Si procede quindi all'introduzione degli attributi necessari per il salvataggio di queste informazioni.

Di seguito una possibile rappresentazione grafica dell'entità (Figura 8):

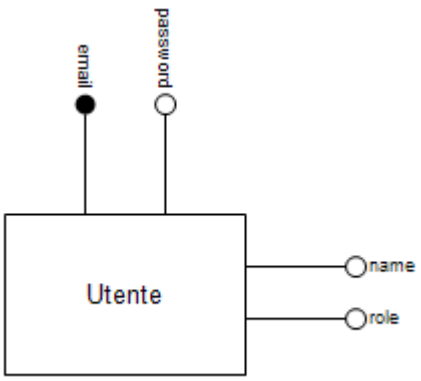


Figura 8 – Entità Utente

Per una descrizione completa di tutti gli attributi si rimanda il lettore all'Appendice A. (Tavola 4)
L'entità **Utente** può considerarsi raffinata

3.4.3 – Raffinamento dell'entità "Prodotto"

Le informazioni rilevanti per l'entità Prodotto sono quelle che ne consentono l'identificazione (Codice, Codice a barre) e che ne rappresentano le caratteristiche (Nome, descrizione, prezzo, dimensioni e peso). Si procede quindi all'introduzione degli attributi necessari per il salvataggio di queste informazioni.

Di seguito una possibile rappresentazione grafica dell'entità (Figura 9):

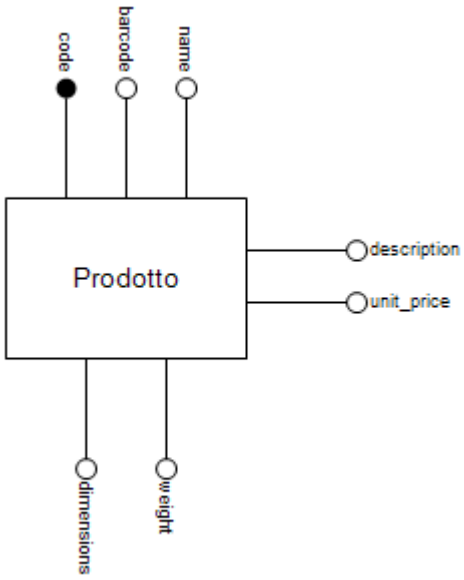


Figura 9 – Entità Prodotto

Da notare che l'attributo **dimensions** è un attributo composto. Esso rappresenta le misure reali del prodotto ed è un'informazione utile per motivi di immagazzinamento e spedizione.
Per una descrizione completa di tutti gli attributi si rimanda il lettore all'Appendice A. (Tavola 5)
L'entità **Prodotto** può considerarsi raffinata.
Ci sono tuttavia un paio di aspetti richiesti che non sono ancora stati modellati:

1. Gestione della quantità dei prodotti in magazzino.
2. Gestione della categorizzazione gerarchica dei prodotti.

Si rimanda la modellazione di questi due aspetti alle successive fasi di progettazione.

3.4.4 – Raffinamento dell'entità "Magazzino"

È richiesta una gestione multi-magazzino ed è quindi necessaria l'introduzione di una nuova entità che consenta il salvataggio dei dati anagrafici dei magazzini.
Sono stati inoltre aggiunti due attributi per la gestione del ruolo del magazzino nel sistema.

Di seguito una possibile rappresentazione grafica dell'entità (Figura 10):

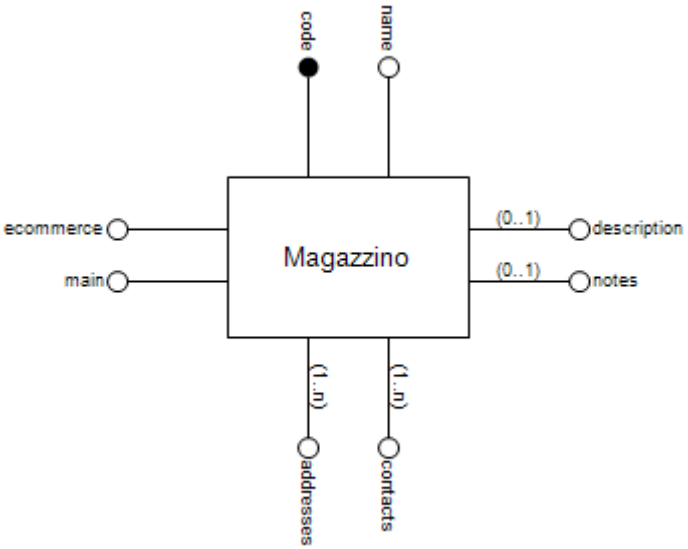


Figura 10 - Entità Magazzino

Da notare che gli attributi **addresses** e **contacts** sono attributi composti con cardinalità (1..n) e che gli attributi **description** e **notes** hanno cardinalita (0..1).
Per una descrizione completa di tutti gli attributi si rimanda il lettore all'Appendice A. (Tavole 2, 3, 6)
L'entità **Magazzino** può considerarsi raffinata

3.4.5 – Raffinamento delle entità "Ordine in uscita" e "Ordine in entrata"

Oltre che alla gestione degli Ordini di vendita, è richiesta la gestione degli ordini di acquisto. Vengono quindi sviluppate le entità relative agli ordini, entrambe caratterizzate da numero e data d'ordine, stato e descrizione facoltativa.

Di seguito una possibile rappresentazione grafica delle entità (Figura 11):

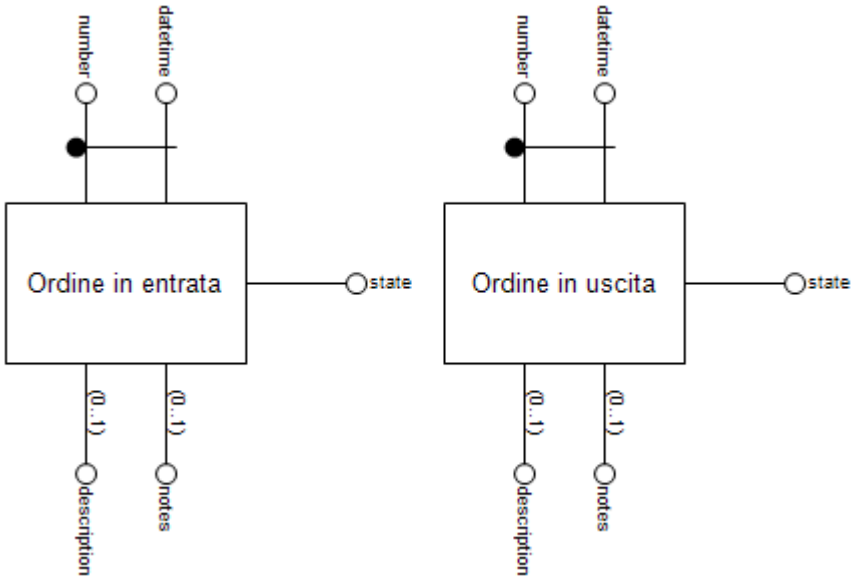


Figura 11 - Entità Ordine in entrata e Ordine in Uscita

Le entità possono considerarsi raffinate.

Per una descrizione completa di tutti gli attributi si rimanda il lettore all'Appendice A. (Tavola 7)

C'è tuttavia un aspetto che non è ancora stato modellato: la quantità di ogni prodotto ordinato. Si rimanda la modellazione di questo aspetto alle successive fasi di progettazione.

3.4.6 – Raffinamento delle entità "Documento in uscita" e "Documento in entrata"

Il sistema dev'essere in grado di gestire i documenti relativi agli ordini in entrata e in uscita. Questi documenti sono identificati da un numero e da una data e caratterizzati da altri attributi quali il tipo e lo stato.

Di seguito una possibile rappresentazione grafica delle entità (Figura 12):

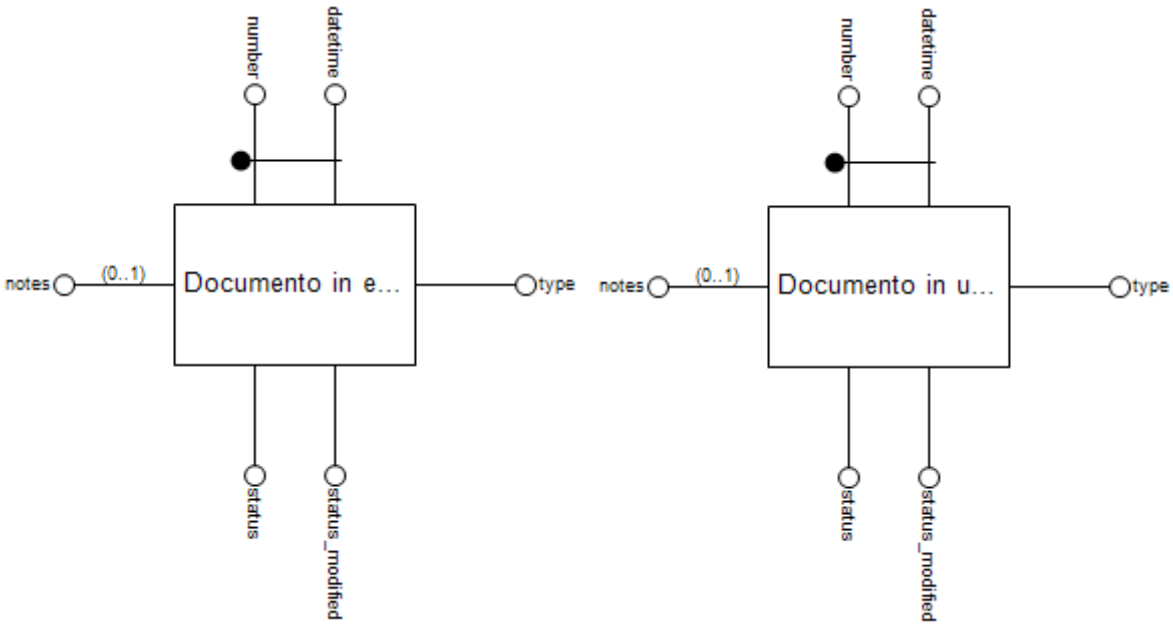


Figura 12 - Entità Documento in entrata e Documento in uscita

Per una descrizione completa di tutti gli attributi si rimanda il lettore all'Appendice A. (Tavola 8)

Le entità possono considerarsi raffinate.

3.4.7 – Sviluppo dell'entità "Etichetta"

Dai precedenti raffinamenti non è stato modellato un aspetto espressamente richiesto dal committente: il sistema di categorizzazione di prodotti, clienti, fornitori, agenti e magazzini.

Occorre quindi introdurre una nuova entità "Etichetta".

L'etichetta sarà identificata dal nome e caratterizzata da un tipo e da una descrizione.

Di seguito una possibile rappresentazione grafica dell'entità (Figura 13):

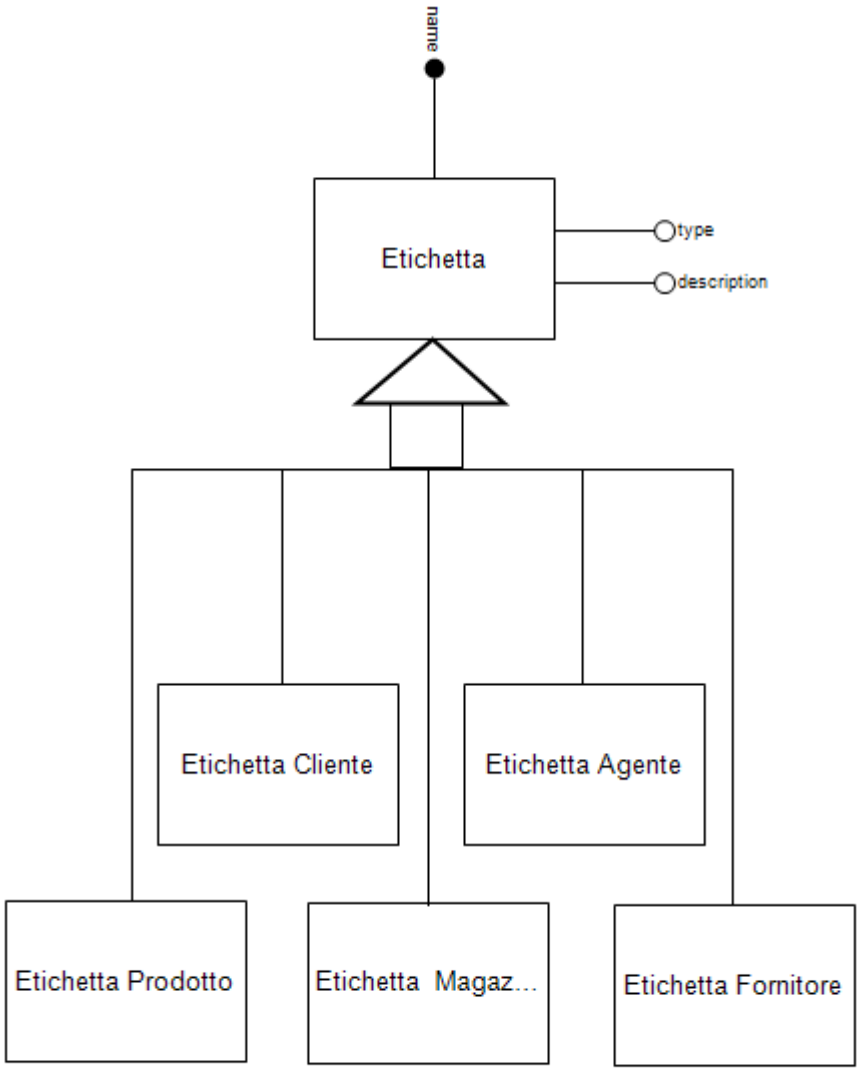


Figura 13 - Entità Etichetta

Questo sistema di etichettamento non prevede tuttavia un'organizzazione gerarchica delle etichette. Bisognerà quindi creare una relazione unaria **[ER02]** sull'entità, per la gestione delle sotto-etichette. Questo secondo aspetto sarà trattato durante il raffinamento delle relazioni.

3.5 – Raffinamento delle relazioni

Come già preannunciato nel paragrafo precedente, si procede ora ad un raffinamento delle relazioni più significative. Le relazioni saranno raffinate in modo tale che il sistema sia in grado di gestire le richieste espresse dal committente durante le interviste. Una volta raffinate tutte le

relazioni, si integreranno i risultati di questi ultimi due paragrafi per ottenere lo schema concettuale finale.

3.5.1 – Raffinamento della relazione "Prodotto – Magazzino"

Come si può notare dallo schema scheletro, la relazione che intercorre fra l'entità **Prodotto** e l'entità **Magazzino** è una relazione molti a molti. Questa relazione necessita di un raffinamento, in quanto, durante il raffinamento delle entità, non è stato modellato un aspetto essenziale per il corretto funzionamento del sistema: la gestione delle quantità dei prodotti in magazzino. Essendo la relazione in questione una relazione molti a molti, la quantità del prodotto in magazzino è un attributo proprio della relazione.
Rappresentando graficamente quanto detto finora si ha (Figura 14):

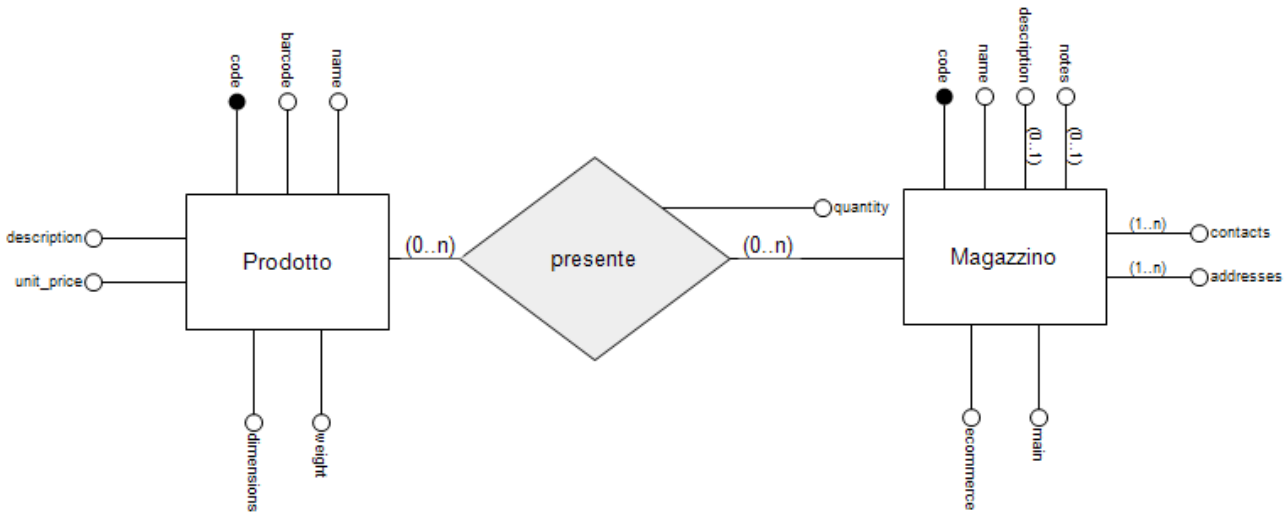


Figura 14 - Relazione Prodotto - Magazzino

3.5.2 – Raffinamento della relazione "Prodotto – Ordine"

Come si può notare dallo schema scheletro, la relazione che intercorre fra l'entità **Prodotto** e l'entità **Ordine** (Sia in uscita che in entrata) è una relazione molti a molti. Questa relazione necessita di un raffinamento, in quanto, durante il raffinamento delle entità, non è stato modellato un aspetto essenziale per il corretto funzionamento del sistema: la gestione delle quantità dei prodotti nell'ordine. Essendo la relazione in questione una relazione molti a molti, la quantità del prodotto nell'ordine è un attributo proprio della relazione.

Rappresentando graficamente quanto detto finora si ha (Figura 15):

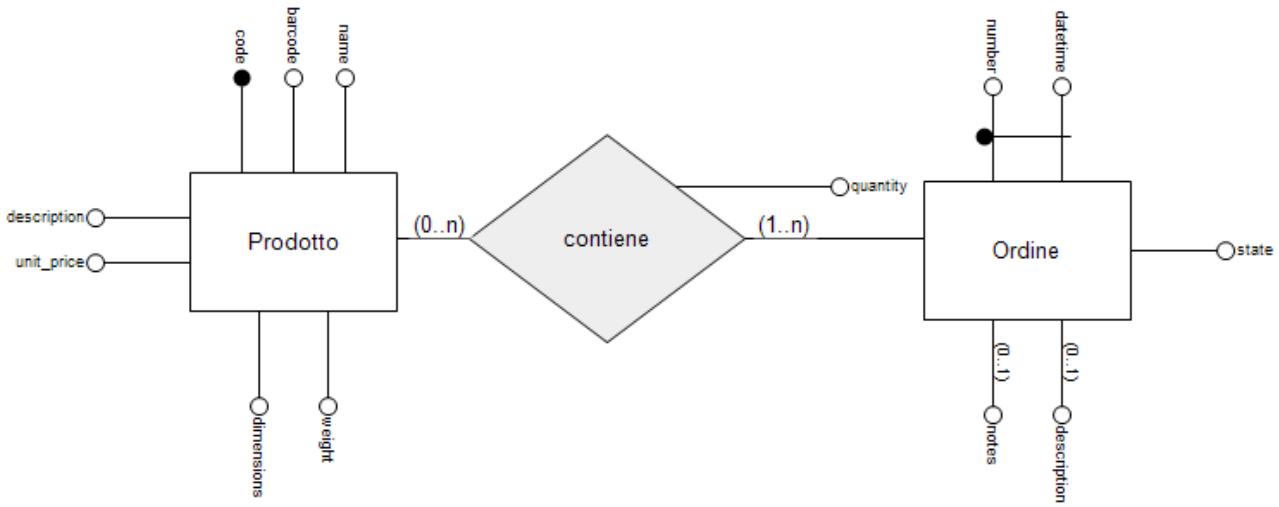


Figura 15 - Relazione Prodotto – Ordine

3.5.3 – Introduzione e raffinamento della relazione "Etichetta – Etichetta"

Come già detto nei precedenti paragrafi è necessario introdurre una relazione unaria sull'entità **Etichetta** per modellare l'organizzazione gerarchica delle etichette. Occorre però fare attenzione alle cardinalità di questa relazione: bisogna fare in modo che un'etichetta possa avere più figli, ma che non sia figlia di più padri.
Una possibile rappresentazione grafica è la seguente (Figura 16):

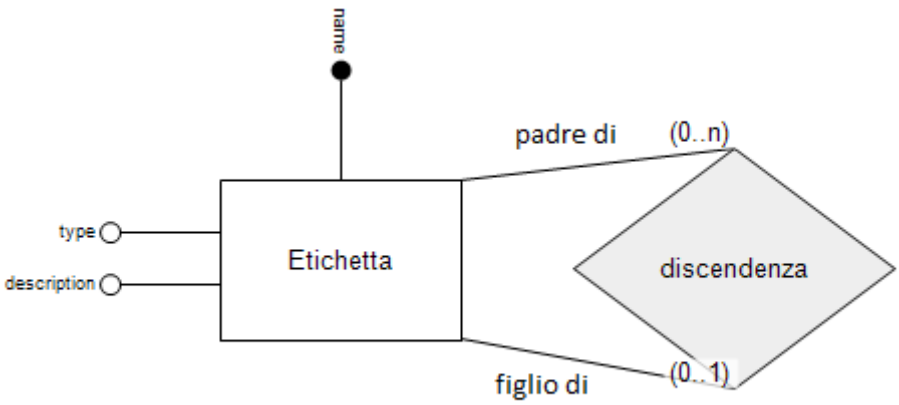
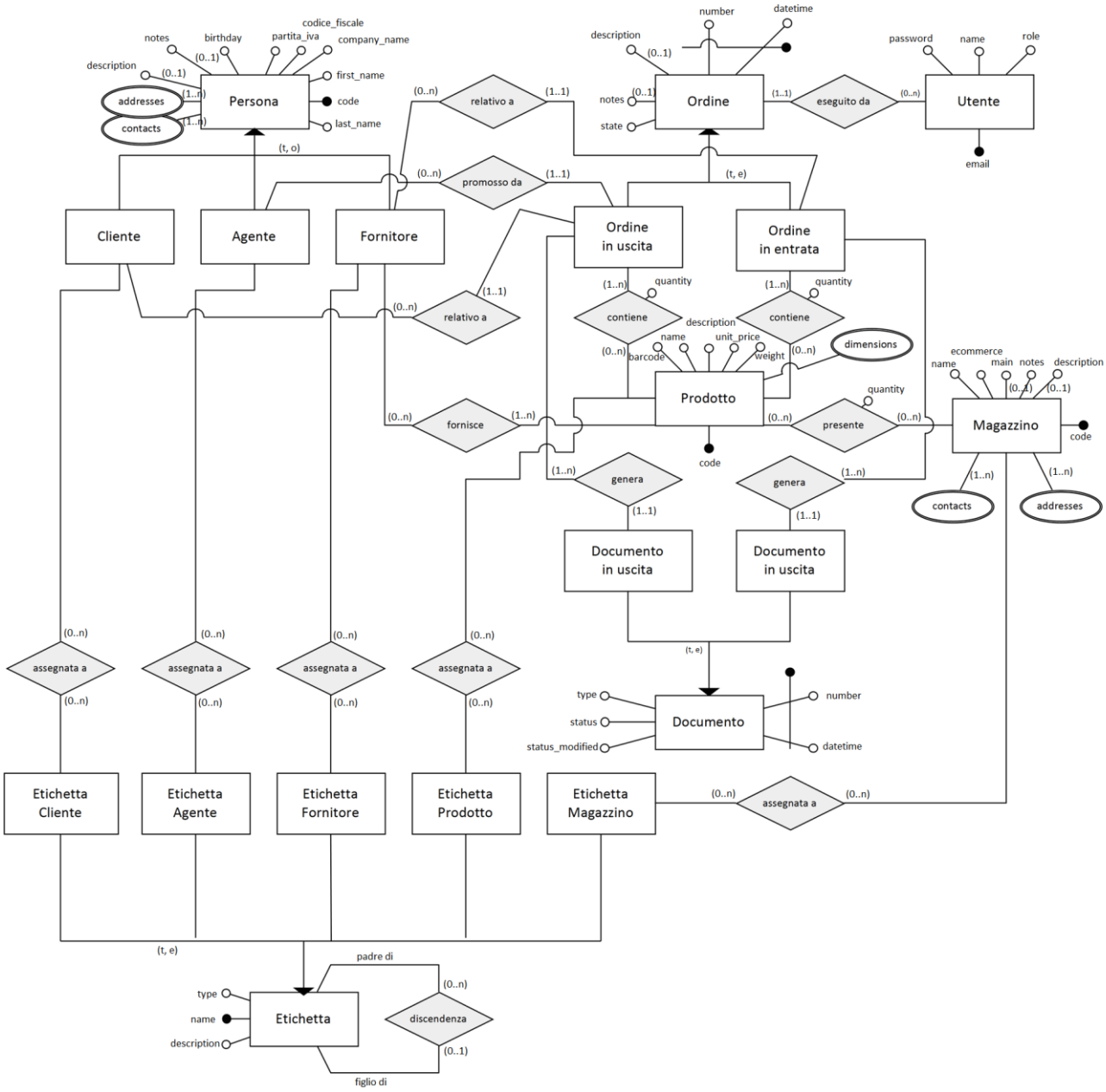


Figura 16 - Relazione unaria su Etichetta

In questo modo un'etichetta può avere 0..n figli e 0..1 padri (0 nel caso sia un'etichetta radice).

3.6 – Schema concettuale finale

Di seguito la rappresentazione grafica dello schema concettuale finale. Questo schema è stato ottenuto grazie all'approccio top-down scelto a inizio capitolo; lo schema scheletro è stato analizzato e raffinato in ogni sua parte, in modo che modellasse tutti gli aspetti individuati durante la fase di analisi dei requisiti.



Questo schema concettuale fungerà da input per la progettazione logica che si affronterà nel prossimo capitolo. Si può considerare conclusa la progettazione concettuale.

4 – Progettazione Logica

Per completare la progettazione del sistema, si procederà ora con la progettazione logica. Oltre che a eseguire le tipiche operazioni di ristrutturazione dello schema concettuale, al fine di eliminare costrutti non rappresentabili nel modello relazionale **[ER02]**, bisognerà tener conto delle esigenze della piattaforma e del framework utilizzato per l'implementazione del progetto.
Si procede ora con la ristrutturazione dello schema concettuale.

4.1 – Ristrutturazione dello schema concettuale

Le modifiche che è necessario apportare allo schema concettuale sono le seguenti:

- Eliminazione delle gerarchie di generalizzazione.
- Modifica delle associazioni collegate alle entità che fanno parte della gerarchia.
- Eliminazione degli attributi composti.

Nei prossimi paragrafi si analizzeranno uno ad uno i punti precedentemente citati.
Successivamente, durante la fase di realizzazione del progetto logico, tutte le associazioni con cardinalità fissa saranno eliminate e sostituite da chiavi importate.
Per le relazioni molti a molti sarà invece necessario eseguire una reificazione dell'associazione.

4.1.1 – Eliminazione delle gerarchie di generalizzazione e modifiche alle associazioni a esse collegate

Nel passaggio da schema concettuale a schema logico, le gerarchie di generalizzazione si possono risolvere in tre modi: collasso verso l'alto, collasso verso il basso, sostituzione con associazioni. **[ER02]**
Nello schema concettuale finale possiamo individuare 4 gerarchie di generalizzazione:

1. Persona – Cliente, Agente, Fornitore
2. Ordine – Ordine in uscita, Ordine in entrata
3. Documento – Documento in uscita, Documento in entrata

4. Etichetta – Etichetta Cliente, Etichetta Agente, Etichetta Fornitore, Etichetta Prodotto, Etichetta Magazzino

Saranno ora analizzate una ad una tutte le gerarchie di generalizzazione per determinare quale strategia è meglio adottare.

Persona – Cliente, Agente, Fornitore

La gerarchia in questione è di tipo totale (In quanto tutte le persone del sistema sono di una di queste tre sotto entità) e sovrapposta. È conveniente, in questo caso, eseguire un collasso verso il basso in modo da marcare la distinzione fra queste tre entità. Inoltre, così facendo, non è necessario apportare modifiche alle relazioni, in quanto esse erano già collegate alle sotto-entità.
Per eliminare l'entità persona sarà quindi necessario replicarne tutti gli attributi nelle sotto-entità.

Ordine – Ordine in uscita, Ordine in entrata

La gerarchia in questione è di tipo totale (In quanto gli ordini possono essere solo d'acquisto o di vendita) ed esclusiva. È conveniente, in questo caso, eseguire un collasso verso il basso in modo da marcare la distinzione fra queste due entità. È necessario, tuttavia, duplicare la relazione "eseguito da" con utente per ristabilire tutte le relazioni presenti prima del collasso.
Per eliminare l'entità ordine sarà quindi necessario replicarne tutti gli attributi nelle sotto-entità.

Documento – Documento in uscita, Documento in entrata

La gerarchia in questione è di tipo totale (In quanto i documenti corrispondono o a un acquisto o a una vendita) ed esclusiva. È conveniente, in questo caso, eseguire un collasso verso il basso in modo da marcare la distinzione tra queste due entità e creare un parallelismo con gli ordini.

Etichetta – Etichetta Cliente, Etichetta Agente, Etichetta Fornitore, Etichetta Prodotto, Etichetta Magazzino

La gerarchia in questione è di tipo totale ed esclusivo. Al contrario delle gerarchie precedentemente gestite, è conveniente in questo caso collassare verso l'alto, in quanto collassando verso il basso saremmo costretti a replicare la relazione "discendenza" su tutte e cinque le sotto-entità. Il collasso verso l'alto avviene introducendo un attributo che esprime con che entità si relaziona l'etichetta. Le cinque relazioni "assegnata a" convergeranno tutte su etichetta e avranno cardinalità molti a molti.

4.1.2 – Eliminazione degli attributi composti

Nel passaggio da schema concettuale a schema logico, gli attributi composti si possono risolvere in due modi: trasformazione in più attributi semplici, trasformazione in entità. **[ER02]** Nello schema concettuale finale possiamo individuare 3 attributi composti:

- Addresses
- Contacts
- Dimensions

Saranno ora analizzati, uno a uno, tutti gli attributi composti per determinare quale strategia è meglio adottare.

Addresses e contacts

Essendo attributi composti comuni a più entità, risulta vantaggioso trasformarli in vere e proprie entità, per poi associarle alle entità che ne fanno uso. Si introducono quindi due nuove entità **Indirizzi** e **Contatti**.

Di seguito una possibile rappresentazione grafica delle due nuove entità introdotte (Figura 17):

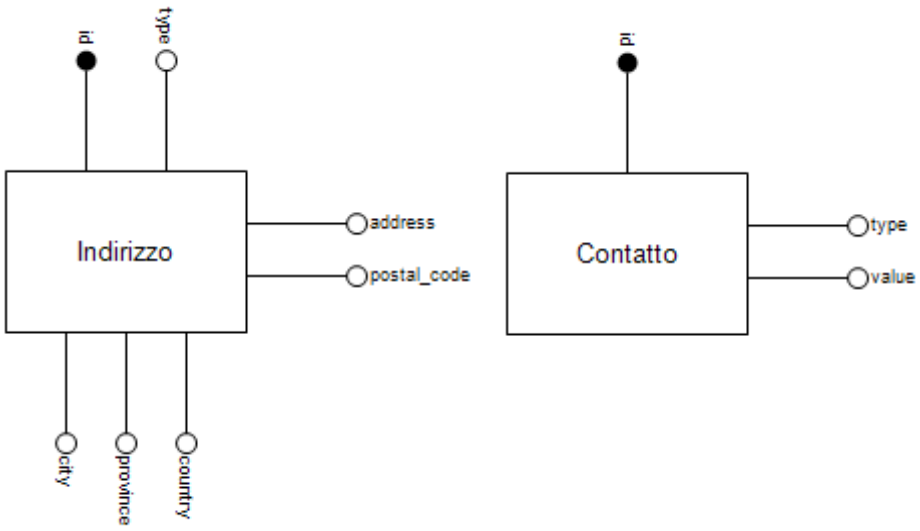


Figura 17 - Entità "Indirizzo" e "Contatto"

Dovendo queste due nuove entità relazionarsi con Clienti, Agenti, Fornitori e Magazzini sarà utilizzato un pattern simile a quello utilizzato per l'entità Etichetta; è prevista quindi l'introduzione di un attributo che esprime con che entità si relazionano le entità in questione.

Dimensions

In questo caso l'attributo composto non è comune a più entità. È preferibile perciò procedere a una divisione in attributi semplici, che saranno aggiunti all'entità **Prodotto.**
Questa soluzione comporta un maggior utilizzo di spazio, ma un più veloce accesso alle informazioni sui prodotti, i quali rappresentano uno dei punti centrali di molte operazioni.

4.2 – Modifiche richieste dal framework

Per poter utilizzare le funzionalità dell'ORM (Object-Relational Mapping) offerto dal framework utilizzato per l'implementazione del progetto, sono necessarie alcune modifiche dello schema.
Il framework CakePHP applica difatti la filosofia *"Conventions over Configuration"*: rispettando le convenzioni su nomi di tabelle e attributi, si ha un significativo risparmio di tempo per quel che riguarda la configurazione del framework. **[CAKE]**

Di seguito una lista delle modifiche necessarie:

- La chiave primaria di ogni relazione si deve chiamare **id** e dev'essere un attributo numerico. Il framework, grazie a questo attributo, gestisce in automatico tutti gli aspetti relativi all'unicità della chiave primaria e al controllo dell'integrità referenziale. **[ER02]**
- Il nome di ogni relazione dev'essere in inglese e al plurale. Questo consente un corretto utilizzo dello strumento di scrittura delle query, offerto dal framework, chiamato "Query Builder". Seguendo questa convenzione di nomenclatura, ci si potrà riferire alla relazione usando il plurale e alla singola tupla usando il singolare
- Aggiungendo i campi **created, modified** e **deleted** ad ogni relazione si ha in automatico il supporto del timestamp behaviour e del softdelete behaviour, offerti dal framework. Questi moduli software chiamati behaviour rappresentano un "Comportamento". Al verificarsi di un determinato evento, come ad esempio un inserimento, una modifica o una cancellazione, questi moduli vengono attivati ed eseguono determinate operazioni sui dati.

4.3 – Traduzione delle entità e delle associazioni

Partendo dallo schema concettuale ristrutturato nei due paragrafi precedenti, si procede alla traduzione delle entità e delle associazioni. Per la traduzione delle associazioni con cardinalità fissa sono state introdotte una o più foreign key nella relazione di partenza (o di destinazione). Per le associazioni molti a molti è invece necessaria una reificazione **[ER01]**, con conseguente introduzione di nuove relazioni.
Per una descrizione completa dello schema relazionale finale si rimanda il lettore all'Appendice B. (Tavola 12)

Si può considerare conclusa la progettazione logica.

5 – Tecnologie Utilizzate

In questo capitolo verranno descritte nel dettaglio le tecnologie e gli strumenti utilizzati per la realizzazione del sistema informativo presentato nei capitoli precedenti. Queste tecnologie fanno parte delle cosiddette "tecnologie web 2.0", in quanto permettono un elevato livello di interazione con l'utente, che diventa il protagonista di questo scenario.

5.1 – Cos'è un framework?

Nei precedenti capitoli si è parlato di "esigenze del framework" senza effettivamente spiegare cos'è un framework e quali vantaggi porta rispetto alla programmazione classica.
Un framework (Che letteralmente significa "Struttura di lavoro") è una struttura di supporto alla progettazione e alla realizzazione di sistemi software.

Esso è solitamente composto da:

- Una serie di librerie che supportano uno o più linguaggi di programmazione.
- Un ambiente di sviluppo (IDE).
- Un debugger o altri strumenti di testing per aiutare il programmatore.

Lo scopo del framework è quello di offrire una base di partenza, facendo risparmiare tempo allo sviluppatore che altrimenti sarebbe costretto a scrivere da zero codice già esistente (L'espressione tipicamente associata è "reinventare la ruota").
Un framework può anche essere visto come una cornice di supporto, dentro alla quale il programmatore "dipinge" i contenuti veri e propri della sua applicazione.

Di seguito alcuni vantaggi nell'utilizzo di un framework rispetto alla programmazione classica:

- **Sviluppo più rapido**: Non ci si deve preoccupare di lavorare sullo strumento che si sta utilizzando, dedicando quindi maggior tempo al progetto. La maggior parte dei problemi che si incontrano durante lo sviluppo di un progetto sono, molto probabilmente, già stati

affrontati da qualcun altro in precedenza e ci sarà sicuramente uno strumento del framework che risolve il problema.

- **Riusabilità e condivisibilità**: Utilizzando gli strumenti offerti dal framework si produce codice riusabile in altri progetti e condivisibile con la community.
- **Possibilità di usare moduli esterni**: Su servizi come Github **[GIT]** o Bitbucket **[BIT]** sono reperibili centinaia di moduli già scritti e testati per espandere le funzionalità di base del framework.
- **Possibilità di ricercare personale già formato**: Nel caso sia necessario ampliare il team di sviluppo, è sufficiente scegliere, fra i candidati, quelli che hanno già esperienza col framework utilizzato, in modo che questi siano produttivi in un tempo molto minore.
- **Formazione personale**: Viceversa, imparando a sviluppare con framework diffusi, si hanno meno difficoltà nella ricerca di un posto di lavoro.

5.2 – Il framework CakePHP

CakePHP **[CAKE01]** è un framework open-source per lo sviluppo di applicazioni web, basate sul linguaggio di scripting server-side PHP **[PHP01] [PHP02]**, che mette a disposizione tutti gli strumenti di cui ha bisogno il programmatore per implementare le funzionalità progettate. Esso è compatibile con numerosi DBMS (MySQL **[SQL01],** PostgreSQL, Microsoft SQL Server, SQLite ecc.) e con numerosi HTTP Server.

Per iniziare un nuovo progetto utilizzando il framework, si scarica un'applicazione scheletro tramite **Composer**, un diffuso tool per la gestione delle dipendenze dei moduli PHP.

Eseguendo da riga di comando l'istruzione:

```
composer create-project --prefer-dist -s dev cakephp/app
[app_name]
```

Viene scaricata nella cartella corrente l'applicazione scheletro di CakePHP.

5.2.1 – Il pattern architetturale di CakePHP

CakePHP adotta come pattern architetturale il molto diffuso e consolidato MVC (Model – View – Controller | Modello – Vista – Controllo in italiano).
Esso è basato sulla separazione dei compiti fra i componenti software, i quali interpretano tre ruoli principali:

- Il **Model** fornisce i metodi per accedere ai dati dell'applicazione.
- Le **View** visualizzano i dati contenuti nel model e si occupano dell'interazione col controller.
- Il **Controller** riceve i comandi dell'utente (in genere attraverso la view) e li attua, modificando lo stato degli altri due componenti.

Questo schema consente una semplice separazione fra logica applicativa (a carico di controller e model) e interfaccia utente (a carico delle view). Le interazioni fra questi tre componenti software dipendono dalle tecnologie utilizzate (Linguaggio di programmazioni, librerie ecc.).

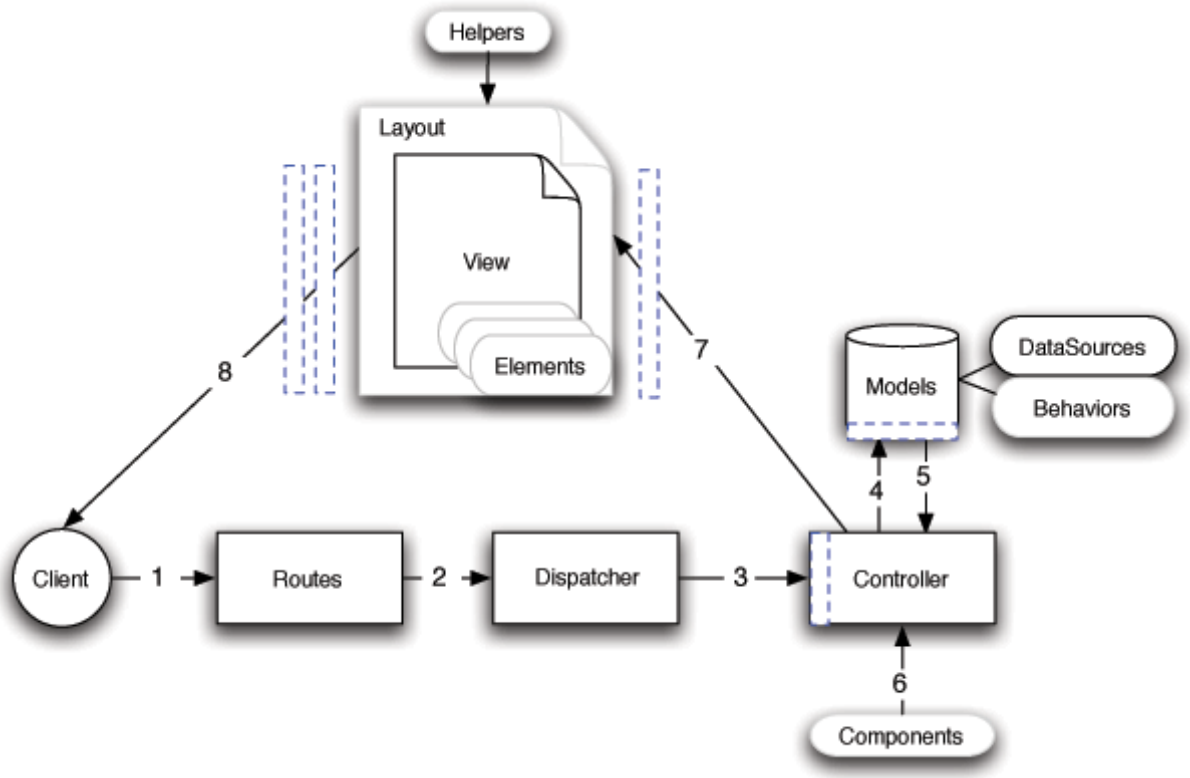


Figura 18 - Tipica richiesta in CakePHP

Una tipico ciclo di richiesta in CakePHP (Figura 18) inizia con la richiesta, da parte di un utente, di una pagina o di una risorsa dell'applicazione. La richiesta attraversa quindi i seguenti passaggi:

1. La richiesta viene instradata.
2. Una volta instradata, il dispatcher selezionerà il controller in grado di gestire la richiesta.
3. Viene chiamata la corrispondente azione del controller che interagirà con i Model.
4. Il controller delega alla view la creazione e la generazione dei risultati ottenuti dal model.

Questa divisione dei compiti fra model, view e controller aiuta il programmatore a capire dove andare ad agire per implementare le funzionalità progettate e permette di mantenere l'applicazione leggera ed efficiente.

5.2.2 – Struttura del progetto

In seguito al download, tramite Composer, dell'applicazione scheletro di CakePHP, si può analizzare la struttura della directory del progetto.
Questa directory contiene le seguenti sotto-directory (Tabella 2)

Tabella 2 - Struttura della directory

Cartella	Contenuto
bin	Contiene gli eseguibili del framework
config	Contiene i pochi file di configurazione utilizzati dal framework. (Connessione al database, sicurezza, caching ecc.)
logs	Contiene i file di log
plugins	Contiene i plugin aggiuntivi
src	Contiene i file sorgenti dell'applicazione. Verrà in seguito approfondita la struttura di questa cartella.
tests	Contiene i test case da eseguire sull'applicazione.

vendor	Contiene i file dai quali CakePHP dipende.
webroot	Rappresenta la radice pubblica dell'applicazione. All'interno di questa cartella vengono comunemente collocati i file che dovranno essere pubblicamente raggiungibili.

La cartella più interessante è sicuramente la cartella "src", dentro alla quale si svolge infatti la maggior parte dello sviluppo.
La cartella "src" è formata dalle seguenti sotto-cartelle (Tabella 3):

Tabella 3 - Struttura della cartella "src"

Cartella	Contenuto
Console	Contiene i comandi da eseguire tramite console per automatizzare alcuni task.
Controller	Contiene i Controller dell'applicazione.
Locale	Contiene le stringhe per l'internazionalizzazione dell'applicazione
Model	Contiene i model necessari per interfacciare l'applicazione con il DB.
View	Contengono le view e i template necessari a mostrare all'utente i risultati ottenuti dal model.
Template	

5.2.3 – Object-Relational Mapping

L'Object-Relational Mapping (ORM) è una tecnica di programmazione che favorisce l'integrazione di sistemi software, aderenti al paradigma della programmazione orientata agli oggetti, con sistemi RDBMS.
Un prodotto ORM fornisce, mediante un'interfaccia orientata agli oggetti, tutti i servizi inerenti alla persistenza dei dati, astraendo nel contempo le caratteristiche implementative dello specifico RDBMS utilizzato. **[WIKI]**

I vantaggi nell'uso di un tale sistema sono innumerevoli. Alcuni di questi sono:

- Superamento dell'incompatibilità tra programmazione orientata agli oggetti e modello relazionale.
- Elevata portabilità rispetto alla tecnologia DBMS utilizzata.
- Riduzione della quantità di codice sorgente da redigere.
- Meccanismi di caching dei dati, che portano alla riduzione dei tempi di caricamento.
- Gestione della concorrenza nell'accesso ai dati.

Come già detto in precedenza, il framework CakePHP implementa questo paradigma di programmazione in maniera molto intuitiva. Esso predilige infatti l'adozione di convenzioni, rispetto a lunghi processi di configurazione; una volta apprese le convenzioni sulla nomenclatura di classi, file e tabelle, vi è un consistente risparmio di tempo per la configurazione del progetto.
Come già detto nel precedente capitolo, le convenzioni di nomenclatura di CakePHP richiedono che i nomi delle tabelle del database siano in inglese e al plurale.
Questo consente la generazione automatica del codice scheletro di Model, View e Controller grazie ad un processo chiamato **Baking. [CAKE02]**

Prendendo, ad esempio, la relazione *Clients*, si può generare tutto il codice necessario ad eseguire operazioni CRUD sulla tabella in questo modo:

- Si apre il prompt dei comandi
- Si naviga fino alla cartella del progetto
- Si eseguono i seguenti comandi

```
cake bake model clients
cake bake view clients
cake bake controller clients
```

Queste tre istruzioni generano i seguenti file:

```
Progetto/src/Model/Entity/Client.php
Progetto/src/Model/Table/ClientsTable.php
```

```
Progetto/src/Template/Clients/add.ctp
Progetto/src/Template/Clients/edit.ctp
Progetto/src/Template/Clients/index.ctp
Progetto/src/Template/Clients/view.ctp
Progetto/src/Controller/ClientsController.php
```

Ripetendo le stesse operazioni per le altre tabelle del database si ottiene una applicazione scheletro che implementa le principali funzionalità CRUD sul database.

5.2.4 – Model, View e Controller in CakePHP

Models

I model sono le classi che formano il "business layer" dell'applicazione. Essi sono responsabili di gestire tutto ciò che riguarda i dati dell'applicazione, la loro validazione e le loro interazioni con gli altri dati.

A livello implementativo, i model sono classi PHP che estendono la classe base AppModel, padre di tutte le classi model dell'applicazione. Esse solitamente rappresentano una tabella o una singola tupla, ma possono anche essere usati per manipolare dati esterni come file.

Views

Le view sono classi responsabili di generare specifici output in seguito a una richiesta da parte dell'utente. Solitamente gli output generati sono sotto forma di HTML **[HTML]**, XML o JSON

[TECWEB], ma possono anche essere generati veri e propri file come i PDF.

A livello implementativo, le view sono file scritti in HTML e PHP e hanno una estensione .ctp (Cake TemPlate). Essi vengono divisi in quattro categorie:

- **Views**: File che visualizzano i dati passati dalla corrispondente azione del controller. Rappresentano il nucleo della risposta dell'applicazione.
- **Elements**: Piccole e riutilizzabili view. Vengono di solito inserite all'interno delle View.

- **Layouts**: File che contengono il codice presentazionale dell'applicazione. Le view sono solitamente create all'interno di un layout.
- **Helpers**: Classi che incapsulano una logica applicativa che necessita di essere riutilizzata in più punti dell'applicazione. Un classico esempio è il formhelper, il quale assiste il programmatore nella creazione delle form.

Controllers

I controller sono classi responsabili della gestione della logica dell'applicazione, facendo da intermediari fra i model e le view. Una volta che il dispatcher ha passato la richiesta al controller, questo la interpreta ed esegue l'azione corrispondente.

A livello implementativo, i controller sono classi PHP che estendono la classe base AppController, padre di tutte le classi controller dell'applicazione. All'interno della classe troviamo una o più funzioni pubbliche chiamate actions. Queste actions si occupano di elaborare la richiesta e di restituire una risposta. Alcuni classici esempi di actions sono view(), edit(), add() e index().

5.3 – Il framework Bootstrap

Bootstrap **[CSS02]** è il più famoso framework HTML, CSS **[CSS01]** e Javascript **[JS01]** per lo sviluppo di interfacce web responsive e mobile-first. Inizialmente nato dal team di sviluppatori del noto social network Twitter, è oggi indipendente e vanta di più di 30.000 fork su GitHub **[GIT]**.

Nella sostanza si tratta di un pacchetto di file HTML, CSS e Javascript che, una volta inseriti nella directory del progetto, consentono di creare un'interfaccia grafica moderna e pulita in maniera molto semplice.

Bootstrap definisce infatti un gran numero di classi CSS che, sapientemente assegnate agli elementi dell'interfaccia, danno un aspetto gradevole e ormai standardizzato. Inoltre, dalla versione 3.0, bootstrap ha adottato la filosofia "mobile-first" in modo da favorire la creazione di siti web responsive.

Alcuni dei vantaggi che derivano dall'utilizzo di questo framework sono:

- **Look moderno e standardizzato**: Standardizzando l'interfaccia utente ne beneficia la user experience (UX).
- **Supporto Cross-Browser**: Il framework definisce infatti regole CSS che funzionino in maniera omogenea su tutti i browser attualmente più utilizzati.
- **Responsive design**: Dalla versione 2 del framework è stato introdotto il supporto al responsive design e, nella versione 3, questa è diventata l'impostazione di default. Sfruttando il supporto al responsive design offerto dal framework, è possibile creare interfacce utente "adattive" in maniera semplice e intuitiva.
- **Aumento della produttività**: Non dovendo preoccuparsi di tutte le problematiche dovute alla non definitiva standardizzazione di CSS3 si ha un significativo risparmio di tempo e aumento della produttività.

5.4 – Il framework jQuery

Così come si è scelto di adottare CakePHP per la programmazione server-side e Bootstrap per l'interfaccia utente, si è ritenuto conveniente adottare un framework anche per la programmazione client-side.
Le moderne applicazione web utilizzano il linguaggio di scripting client-side Javascript **[JS01]** per regolare l'interazione con l'utente e realizzare effetti dinamici tipici delle applicazioni desktop. Tuttavia, poiché ogni browser implementa un diverso motore Javascript, si può andare incontro a risultati inattesi e disomogenei, che implicano molte ore di debugging per il programmatore.

jQuery **[JQUER01]** è un framework Javascript che si propone di risolvere questi problemi e di fornire meccanismi che semplificano le operazioni più comuni come la manipolazione degli stili CSS e degli elementi HTML o l'esecuzione di chiamate AJAX **[JS01]**.
Il fulcro del framework, attorno al quale ruotano tutti i meccanismi, è l'oggetto/funzione $, alias della parola chiave jQuery:

```
$("#esempio"); //Un oggetto jQuery
jQuery("#esempio") //Lo stesso oggetto richiamato in modo diverso
```

Alcuni dei vantaggi che derivano dall'utilizzo di questo framework sono:

- **Leggero ed efficiente:** La versione minificata del framework occupa solamente 32kB ed offre innumerevoli funzionalità di supporto alla programmazione Client-Side.
- **Compatibile coi selettori CSS:** Per recuperare gli elementi sui quali si vuole intervenire può essere usata la stessa sintassi dei selettori CSS **[CSS01].**
- **Supporto Cross-Browser:** Rende omogeneo il comportamento dei motori Javascript dei vari browser.

5.5 – Strumenti di sviluppo

Verranno ora presentati gli strumenti di sviluppo utilizzati per l'implementazione e il testing del progetto.

5.5.1 – La piattaforma Wamp

Il framework CakePHP richiede alcuni strumenti di sviluppo prima di poter procedere all'implementazione dell'applicativo. Questi sono:

- HTTP Server. Viene suggerito Apache con il modulo "mod_rewrite" abilitato.
- Motore PHP 5.4.16 o superiore.
- Opzionalmente un DBMS. Il framework supporta numerosi DBMS fra i quali:
    - MySQL (5.1.10 o superiore) **[SQL01]**
    - PostgreSQL
    - Microsoft SQL Server (2008 o superiore)
    - SQLite 3

Fortunatamente è disponibile un software gratuito che integra tutti gli strumenti di sviluppo richiesti dal framework: **Wamp.**
Wamp è un ambiente di sviluppo web per Windows, che prende il nome dalle iniziali dei componenti software dai quali è composto:

- W = Windows. Il sistema operativo sul quale si appoggia l'ambiente di sviluppo.
- A = Apache. Il web server.
- M = MySQL. Il DBMS open-source sviluppato da Oracle.
- P = PHP. Il motore di scripting server-side.

L'ambiente fornisce inoltre un utilissimo strumento per la gestione dei Database: phpMyAdmin. Questo tool (Figura 19) permette di creare un database da zero, creare le tabelle ed eseguire operazioni di ottimizzazione sulle stesse. Presenta un feedback sulla creazione delle tabelle per evitare eventuali errori. Sono previste delle funzionalità per l'inserimento dei dati (popolazione del database), per le query, per il backup dei dati, ecc.

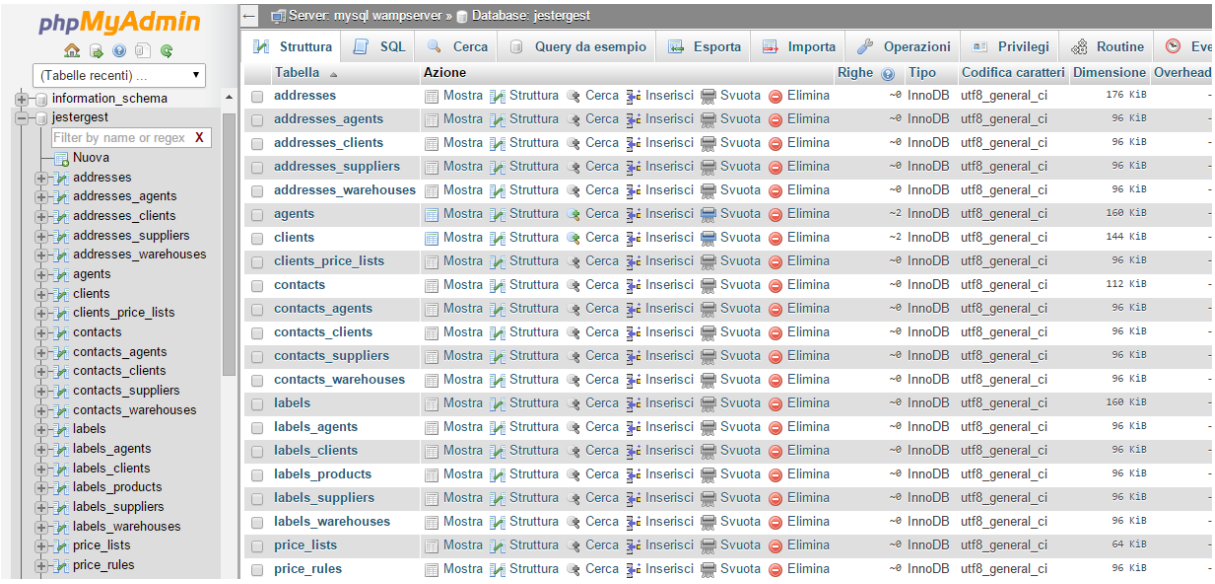


Figura 19 - phpMyAdmin

5.5.2 – Ambiente di sviluppo Netbeans

Per procedere alla vera e propria implementazione dell'applicativo ci si è avvalsi di un ambiente di sviluppo integrato (IDE) chiamato NetBeans. **[NETBE]**

NetBeans è un ambiente di sviluppo, ossia uno strumento destinato ai programmatori per scrivere, compilare ed eseguire il debug ed il deploy di programmi. È scritto in Java ma può supportare qualsiasi linguaggio di programmazione. Esiste anche un gran numero di moduli utili per estendere le sue funzionalità. Esso è gratuito, senza alcuna restrizione riguardante il suo uso.

Grazie alla sua compatibilità con PHP 5, è possibile avvalersi di avanzati strumenti di stesura del codice (Autocompletamento, Indentazione automatica ecc.) e di debugging (XDebug).

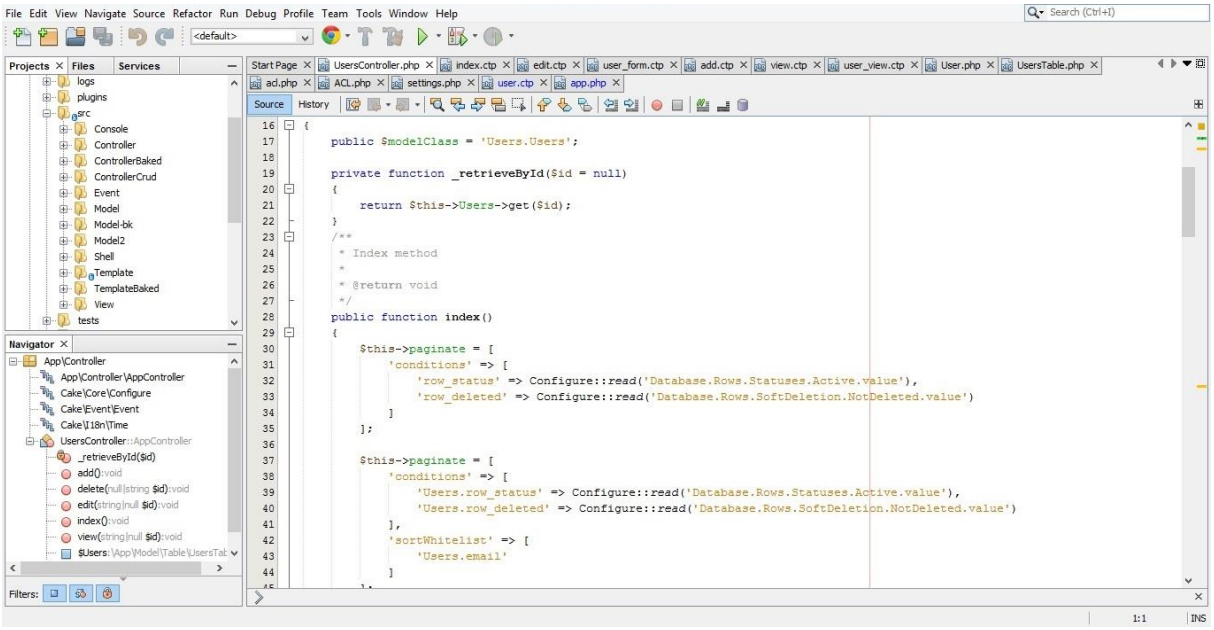


Figura 20 - Ambiente di sviluppo Netbeans

Dall'immagine (Figura 20) si può notare che la schermata è organizzata in più riquadri. Il primo riquadro in alto a sinistra consente di navigare fra le directory del progetto, mentre quello in basso a sinistra analizza gli elementi contenuti dentro al file che si sta visualizzando. La finestra principale è il vero e proprio editor di codice sorgente, il quale fornisce strumenti di highlighting, indentazione, autocompletamento e correzione del codice. È inoltre presente una scheda "History" per visualizzare lo storico delle modifiche apportate al codice sorgente.

Un'altra funzionalità molto importante offerta dall'ambiente è quella delle gestione di repository in remoto.
GitHub **[GIT]** e BitBucket **[BIT]** sono due servizi di hosting web-based per la gestione di repository in remoto e che offrono un sistema del controllo della revisione.

Collegando a NetBeans l'account personale di uno di questi due servizi, è possibile, dal menù "Team" (Figura 21), gestire il progetto in remoto, tramite i classici meccanismi di commit, push e pull offerti dai servizi sopra citati.

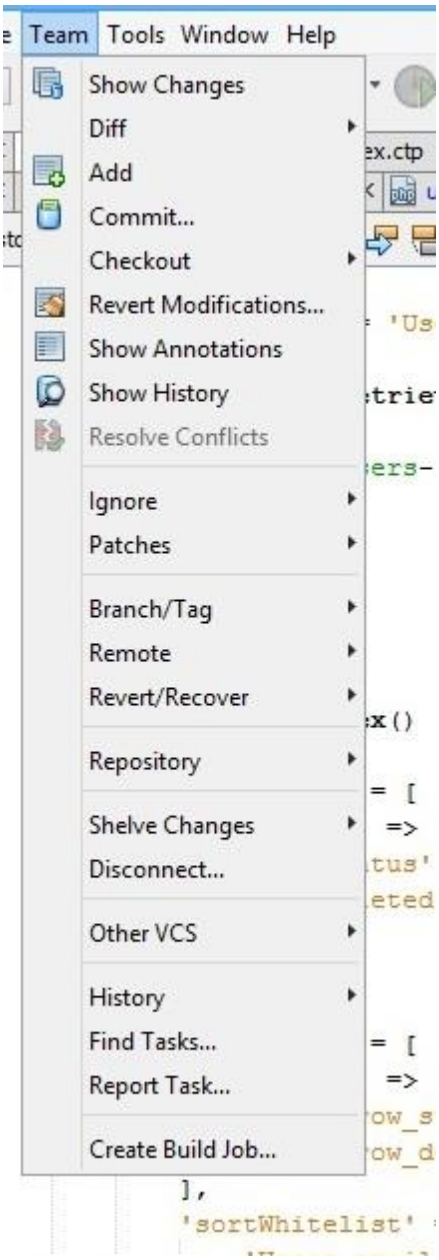


Figura 21 - Menu "Team"

5.5.3 – Tool di compilazione dei file LESS

Per questo progetto si è scelto di non usare i classici fogli di stile CSS, ma bensì la loro variante compilata LESS. **[CSS03]**

LESS è un linguaggio dinamico per la programmazione di stili che può essere compilato nei classici CSS o essere eseguito dal client o dal server a runtime. Offre numerosi meccanismi tipici dei linguaggi di programmazione come variabili, annidamento, operatori, funzioni e mixins. È open source.
Per non gravare sulle prestazione del client o del server facendo compilare i file LESS a runtime, si è utilizzato un tool per la compilazione dei file LESS: Prepros.

Prepros è un preprocessore CSS utilizzato per compilare i file scritti in LESS ed eseguire la minificazione dei file CSS e Javascript. Il suo utilizzo è molto semplice: aggiungendo il progetto tramite la pressione del bottone "Add new project" (Figura 22, riquadro rosso) e selezionando l'impostazione di autocompilazione (Figura 22, riquadro giallo), il software rileverà in automatico il salvataggio delle modifiche e si occuperà di compilare i file LESS in normali file CSS.

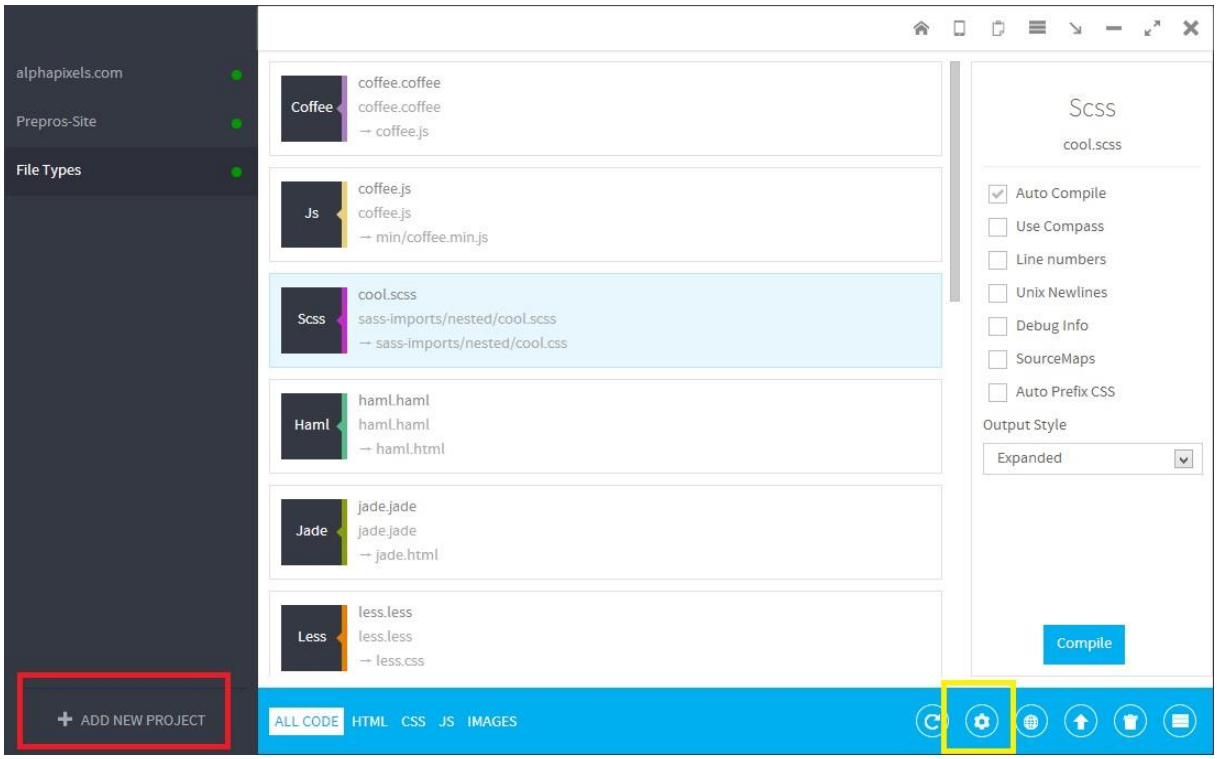


Figura 22 - Prepros

6 – Sistema realizzato

Sarà di seguito presentato il sistema finale, realizzato con l'utilizzo delle tecnologie introdotte nel capitolo precedente. Ci si soffermerà maggiormente sulla descrizione dell'interfaccia utente (UI), punto di forza delle applicazioni web 2.0.

6.1 – Layout delle pagine

Il layout delle pagine web è stato realizzato in modo che sia consistente in tutte le pagine.
Per semplificare questo processo, si è utilizzata la gestione dei layout di CakePHP, introdotti nel capitolo precedente. Creando un file "default.ctp" nella cartella "src/Template/Layout", questo farà da cornice a tutte le pagine della nostra applicazione e sarà sufficiente delegare al controller il riempimento del contenuto.

La struttura adottata per l'applicativo è sostanzialmente composta da tre sezioni: un Side Menù, una Top Bar e un Content (Figura 23).

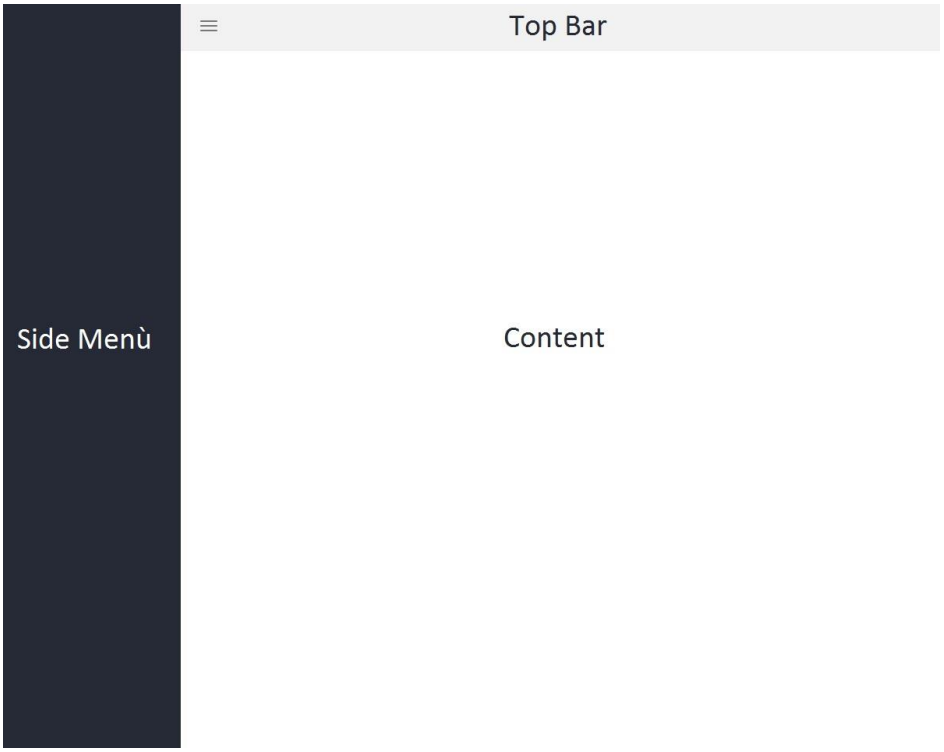


Figura 23 - Layout Pagine

Si procederà ora all'analisi delle tre sezioni.

6.1.1 – Top Bar

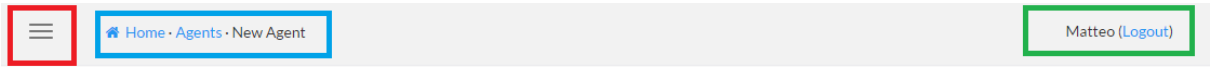


Figura 24 - Top Bar

La Top Bar (Figura 24) è a sua volta composta da tre sezioni:

1. Un bottone menù (Riquadro rosso). Questo bottone ha il compito di dilatare o contrarre il Side Menù, utilizzando un effetto di transizione CSS **[CSS01]**. Lo scopo della contrazione del Side Menù è quello di fornire più spazio alla sezione Content per visualizzare il contenuto.
2. Una breadcrumb (Riquadro azzurro). Lo scopo della breadcrumb è quello di aiutare l'utente nella navigazione delle pagine, fornendogli la possibilità di navigare alle pagine precedenti tramite l'attivazione dei link contenuti in essa.
3. Il nome dell'utente loggato (Riquadro verde). In questo modo l'utente è in grado di capire se la procedura di autenticazione è andata a buon fine e, nel caso volesse chiudere la sessione di lavoro, può effettuare la disconnessione (Logout).

6.1.2 – Side Menù

Il Side Menù (Figura 25) è composto dal logo e nome dell'azienda e da una serie di pulsanti che permettono la navigazione in tutte le pagine del sistema. Come già detto nel paragrafo precedente, in seguito alla pressione del bottone menù, il Side Menù passa alla sua forma collassata (Figura 26).

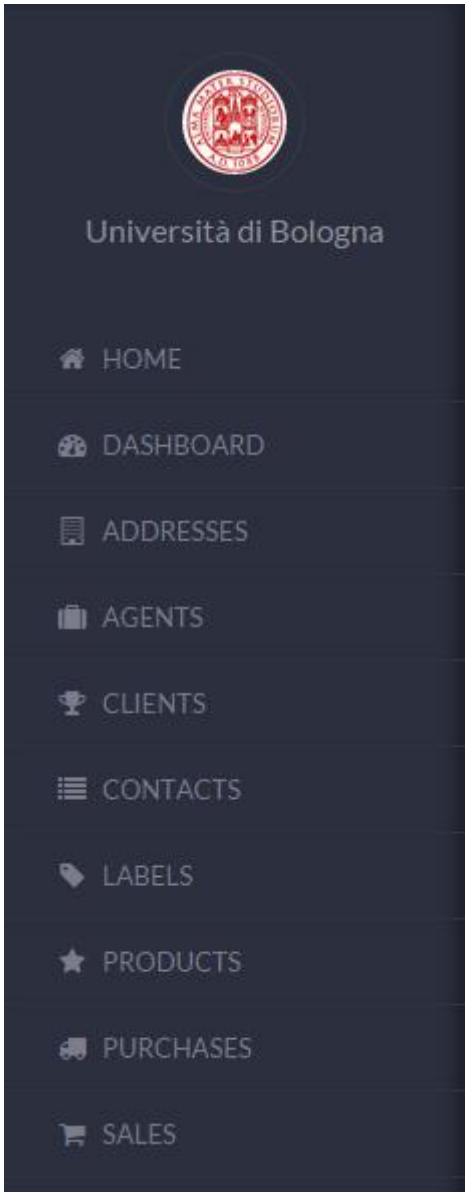


Figura 25 - Side Menù dilatato

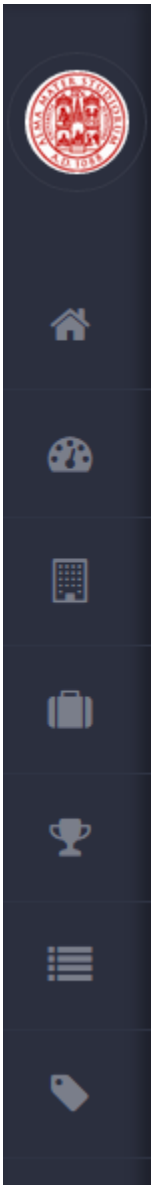


Figura 26 - Side Menù collassato

Al contrario della sua versione dilatata, il Side Menù collassato non presenta il nome dell'azienda e i pulsanti sono composti dalle sole icone.

È presente, in entrambi i casi, un effetto di selezione in seguito agli eventi di Hover e Click (Figura 27).

Figura 27 - Selezione di una voce del menù

6.1.3 – Content

Come già detto in precedenza, il content viene caricato dinamicamente dal controller, in seguito all'azione richiesta dall'utente. Verranno in seguito analizzati i content delle varie sezioni del sistema.

6.2 – Pagine di Login e di recupero della password

La pagina di autenticazione (Figura 28) è composta da una form dove vengono richieste l'email e la password dell'utente. È inoltre presente un checkbox che consente la permanenza della sessione (Remember me) e un link per la gestione delle password dimenticate (Figura 29).

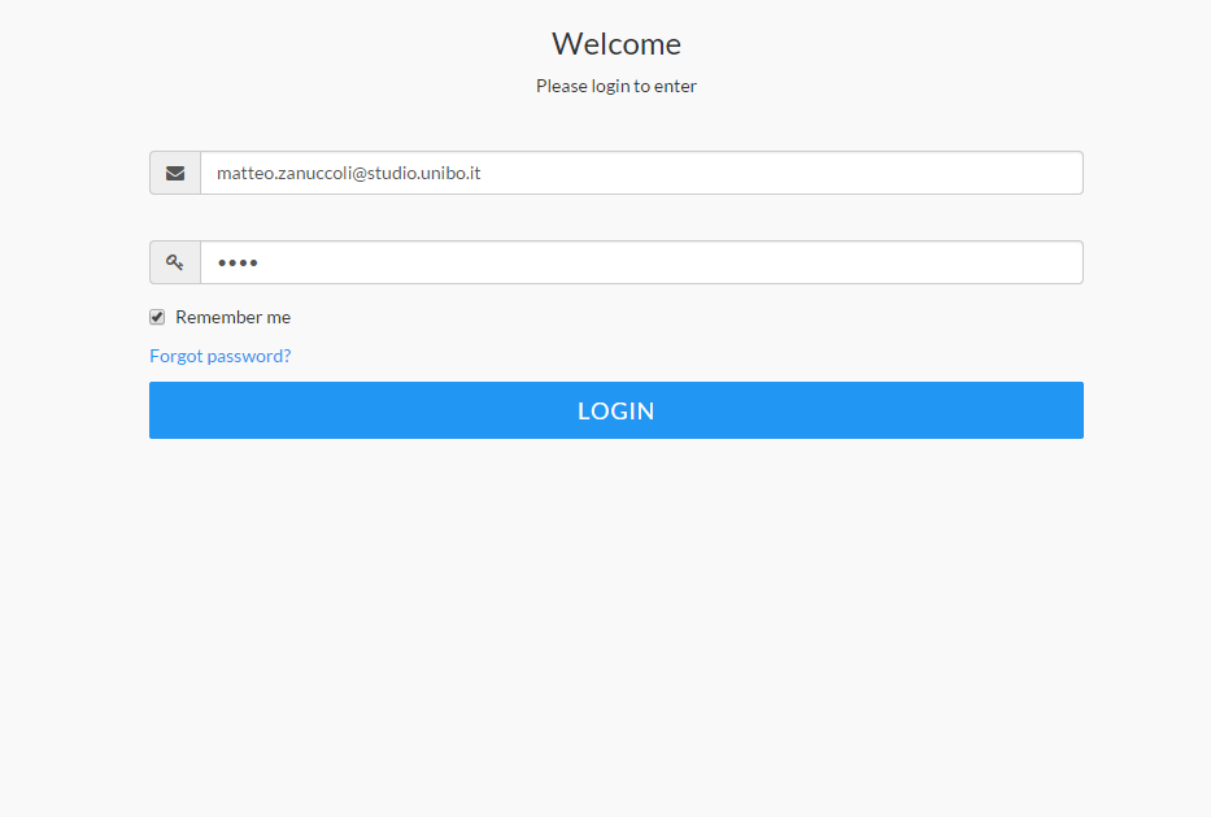


Figura 28 - Pagina di Login

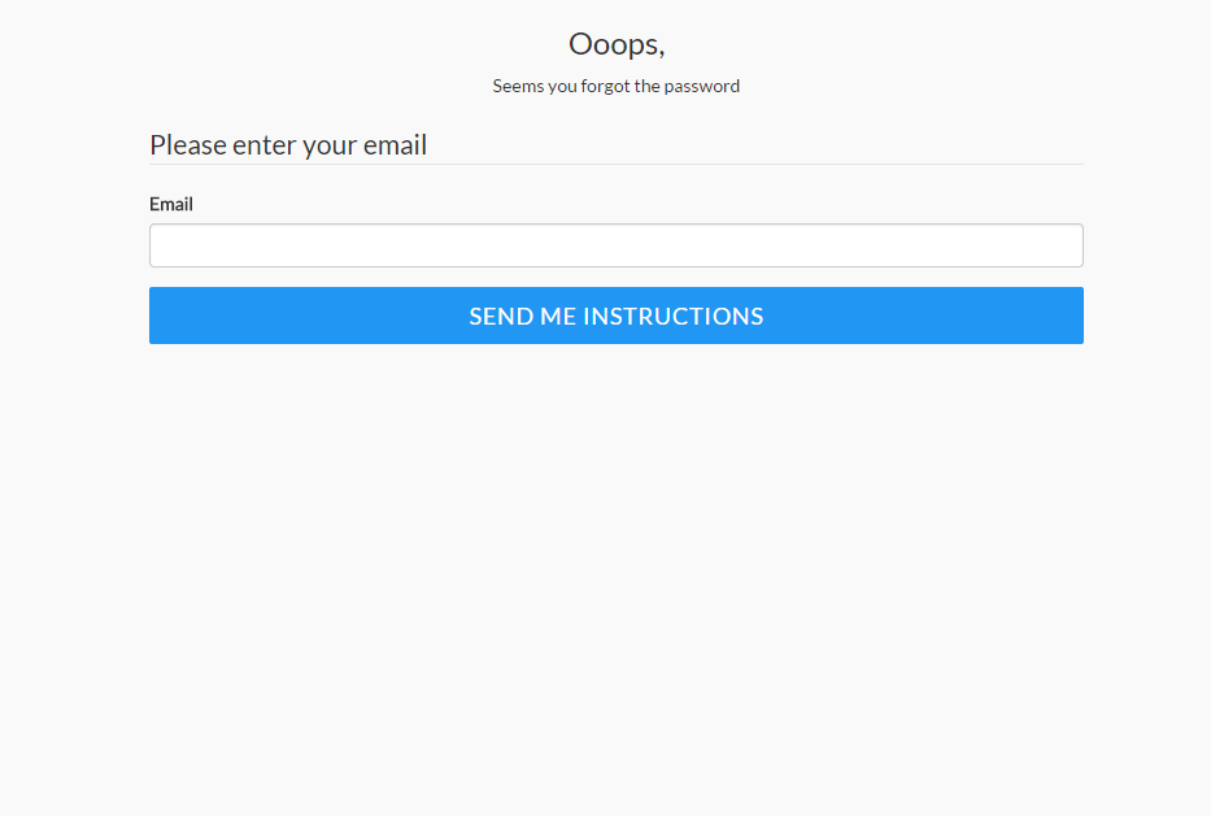


Figura 29 - Pagina di Recupero Password

Allo stato attuale di sviluppo del progetto, la creazione di nuovi account è permessa esclusivamente all'amministratore del sistema che, una volta autenticato, potrà accedere alla sezione "Users" e creare nuovi account.

6.3 – Pagine Home e Dashboard

Le pagine home e dashboard forniscono informazioni generali sullo stato attuale del sistema. Nel caso specifico di un software gestionale, esse mostrano, in genere, informazioni statistiche su vendite, prezzi, affluenza di clienti ecc. Quali dati dovranno mostrare le pagine in questione è un punto tutt'ora in fase di definizione. Di seguito un mock-up della pagina Dashboard (Figura 30)

Figura 30 - Mock-up della pagina Dashboard

6.4 – Tipologie di pagine

Per creare una soluzione omogenea sotto tutti gli aspetti, si è scelto di dividere le pagine in quattro macro-tipologie:

- Pagine index
- Pagine add
- Pagine view
- Pagine edit

Queste quattro tipologie di pagine corrispondono alle quattro azioni index(), add(), view() ed edit() presenti in tutti i controller.

Prendendo, ad esempio, la richiesta “Visualizza la lista dei clienti” CakePHP la gestirà seguendo i seguenti passi:

- È stata chiamata un’azione del controller “Clients”.
- Dentro al controller “Clients” viene individuata l’azione index().
- L’azione index() recupera i dati dal model e delega la visualizzazione dei dati recuperati alla view index.ctp.

Utilizzando questo pattern “Controller => Action” per tutti i controller, si ha un considerevole risparmio nella progettazione delle pagine web e una sensazione di uniformità che aiuta l’utente nella sua esperienza d’uso.

Di seguito una dettagliata descrizione di ognuna delle quattro tipologie di pagina.

6.4.1 – Pagine index

Le pagine index (Figura 31) visualizzano la lista delle tuple presenti nella tabella corrispondente alla sezione scelta.

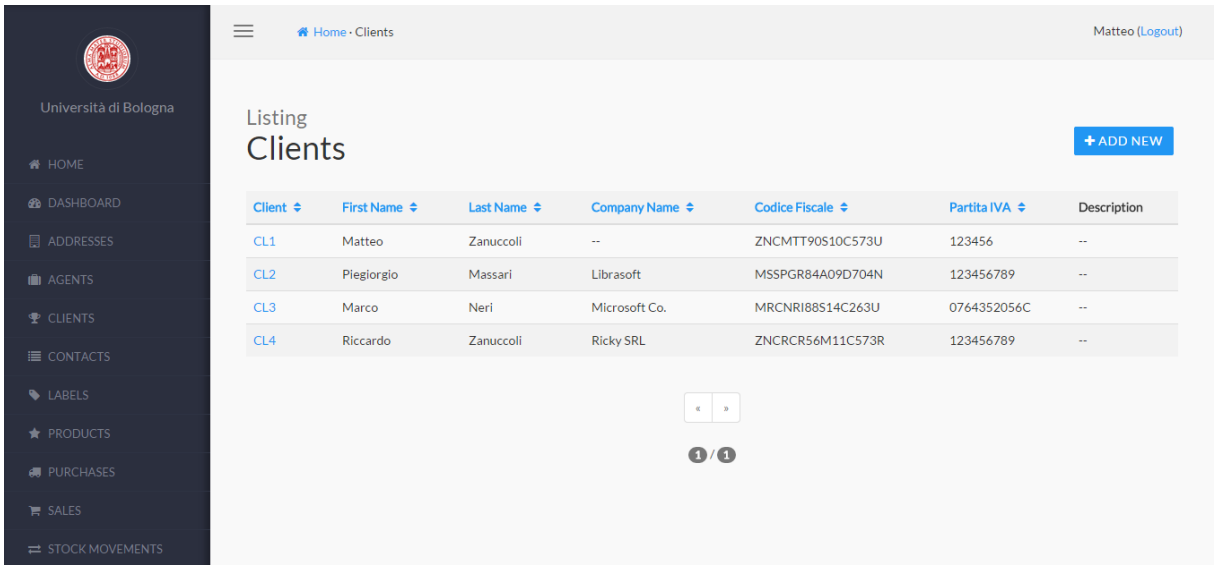
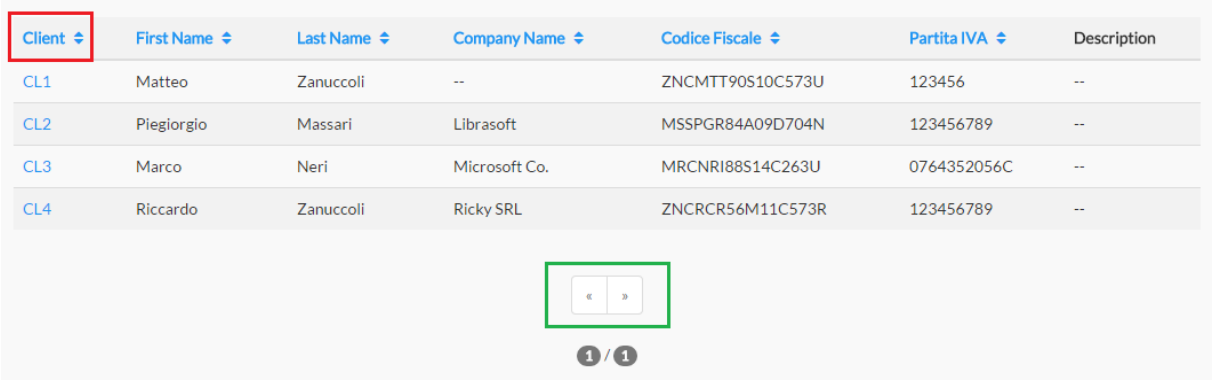


Figura 31 - Pagina Index

All’interno della pagina si possono individuare vari elementi:

- Nome dell’azione e del controller (Figura 32).
- Tabella che visualizza i dati reperiti dal model (Figura 33). Questi dati sono ordinabili tramite un click sull’header della colonna (Riquadro rosso) sulla quale si vuole eseguire l’ordinamento e sono paginati (Riquadro verde) grazie al PaginatorHelper offerto dal framework.
- Bottone “Add new” che chiama l’azione add() del controller corrispondente.

Figura 32 - Nome dell'azione e del controller



Client	First Name	Last Name	Company Name	Codice Fiscale	Partita IVA	Description
CL1	Matteo	Zanuccoli	--	ZNCMTT90S10C573U	123456	--
CL2	Piegiorgio	Massari	Librasoft	MSSPGR84A09D704N	123456789	--
CL3	Marco	Neri	Microsoft Co.	MRCNRI88S14C263U	0764352056C	--
CL4	Riccardo	Zanuccoli	Ricky SRL	ZNCRCR56M11C573R	123456789	--

Figura 33 - Tabella di visualizzaizone dei dati

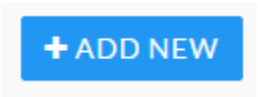


Figura 34 - Bottone "Add new"

6.4.2 – Pagine add

Le pagine add (Figura 35) consentono di aggiungere una nuova tupla nella tabella corrispondente alla sezione scelta.

All'interno della pagina si possono individuare vari elementi:

- Nome dell'azione e del controller (Figura 32).
- Form che permette l'inserimento dei campi richiesti. (Figura 36)
- Bottoni "Submit" e "Cancel" per inoltrare la form o cancellare l'azione di aggiunta. (Figura 37)

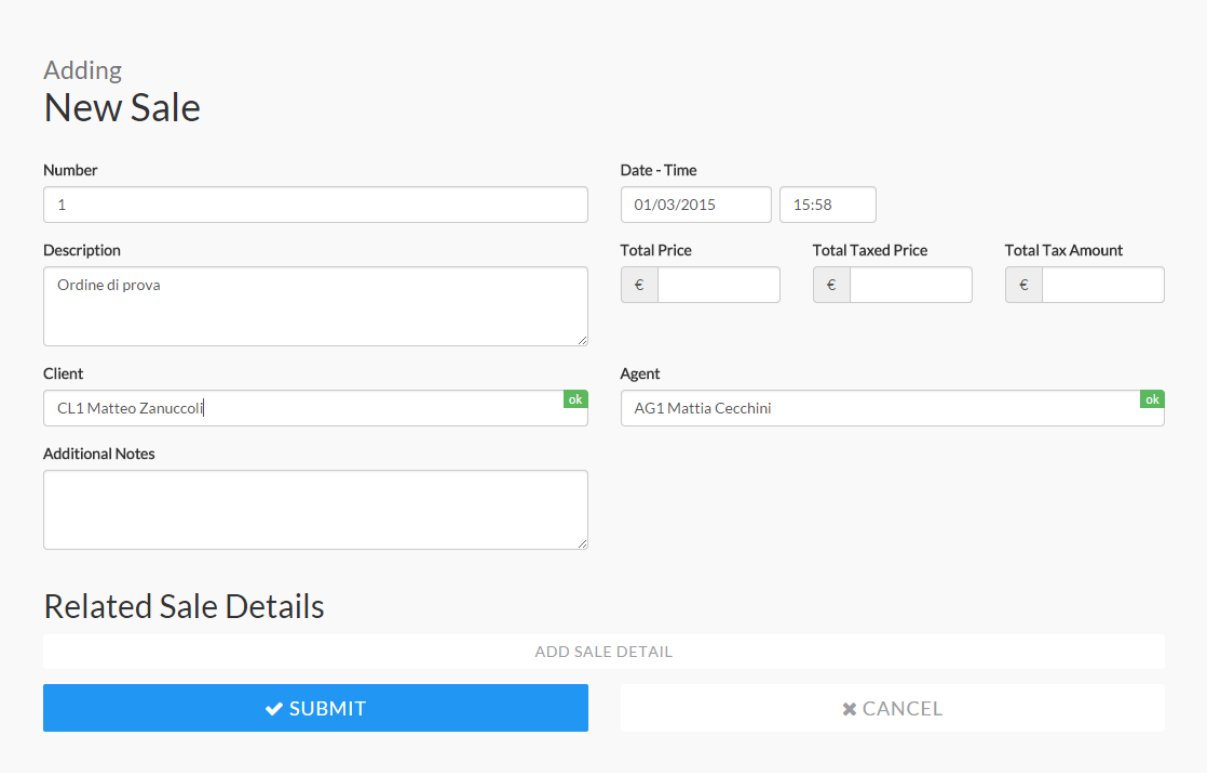


Figura 35 - Pagina Add

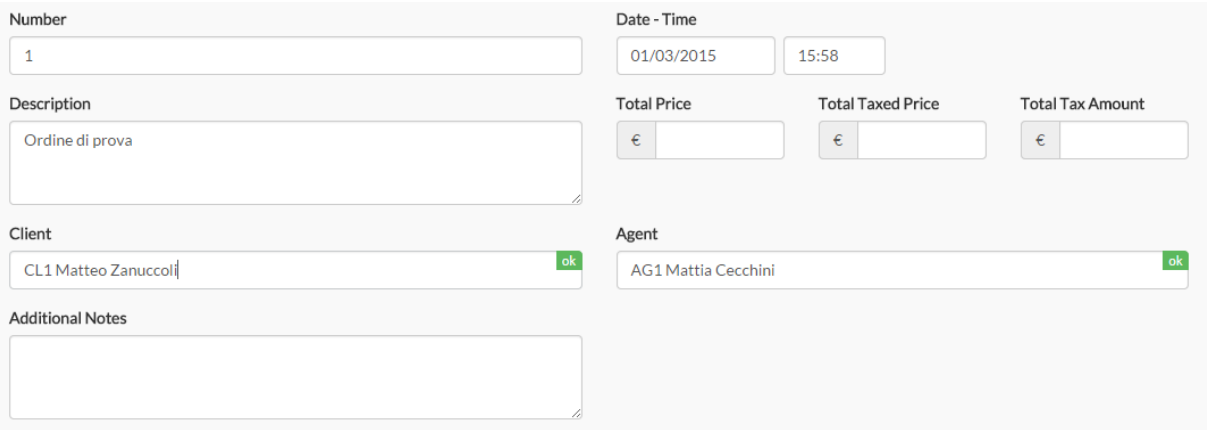


Figura 36 - Form d'inserimento dei campi richiesti

Figura 37 - Bottoni "Submit" e "Cancel"

Nel caso la tabella sia in relazione con altre tabelle (Come ad esempio “Sales” con “Sale_Details”) è presente un bottone speciale (Figura 38). La pressione di questo bottone fa comparire, grazie all’utilizzo di CSS e Javascript, una nuova form (Figura 38), la quale consente di inserire una tupla nella tabella associata.

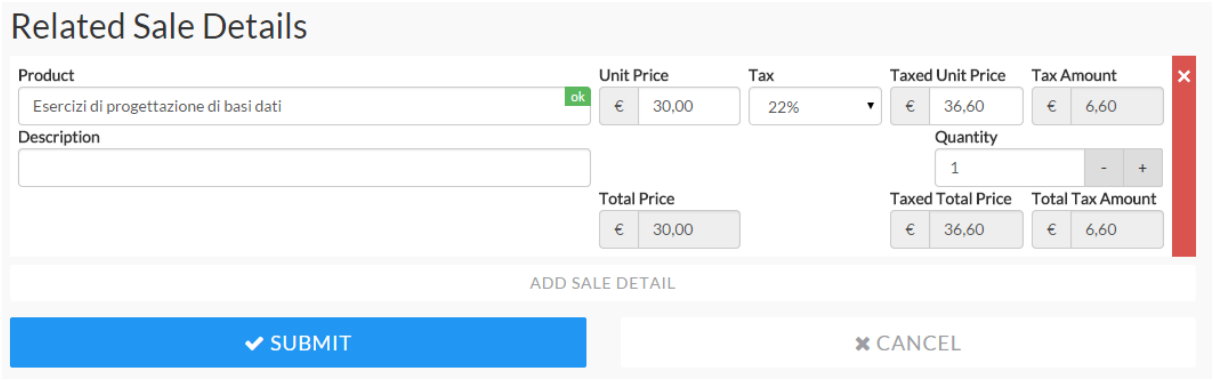


Figura 38 - Inserimento tupla in tabella associata

6.4.3 – Pagine view

Le pagine view (Figura 39) consentono di visualizzare tutti i dettagli relativi alla tupla selezionata, oltre a quelli già presenti nella relativa pagina index.

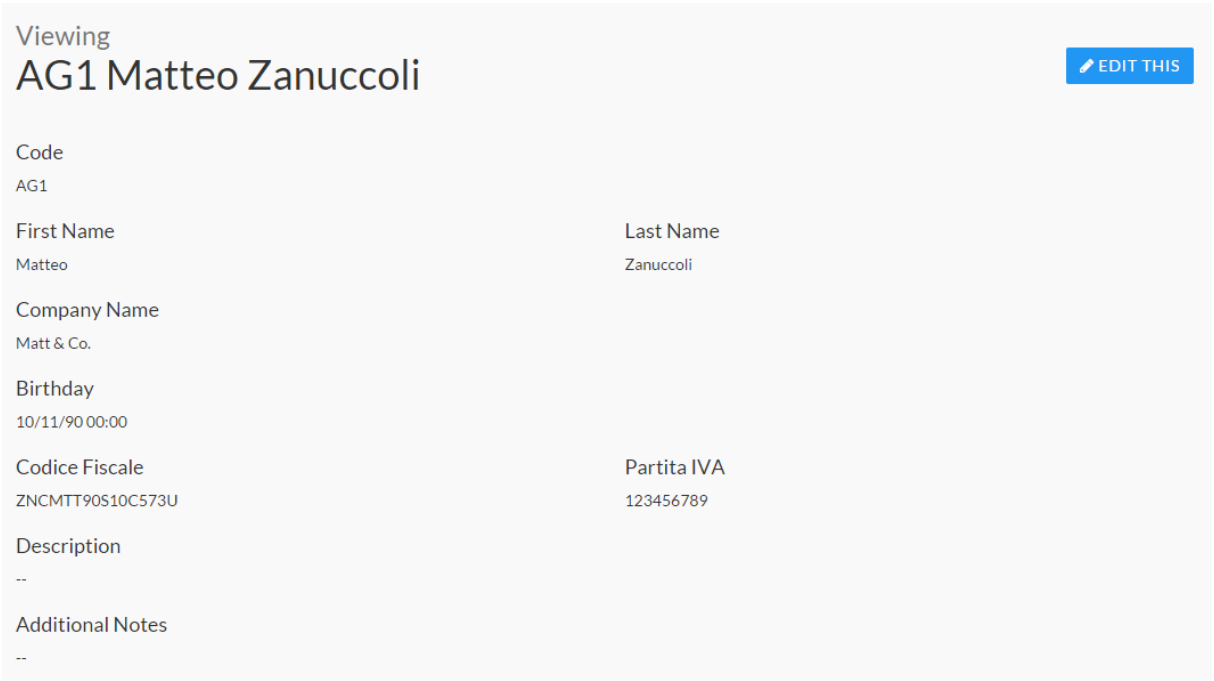


Figura 39 - Pagina View

All'interno della pagina si possono individuare vari elementi:

- Nome dell'azione e identificativo della tupla che si sta visualizzando. (Figura 40)
- Bootstrap Grid **[CSS02]** contenente tutti i dettagli della tupla. (Figura 41)

- Bottone “Edit this” che chiama l’azione edit() del controller corrispondente passandogli il campo id della tupla da modificare.

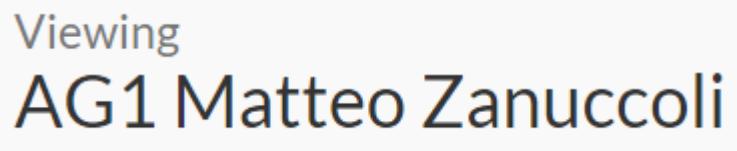


Figura 40 - Nome azione e identificativo tupla

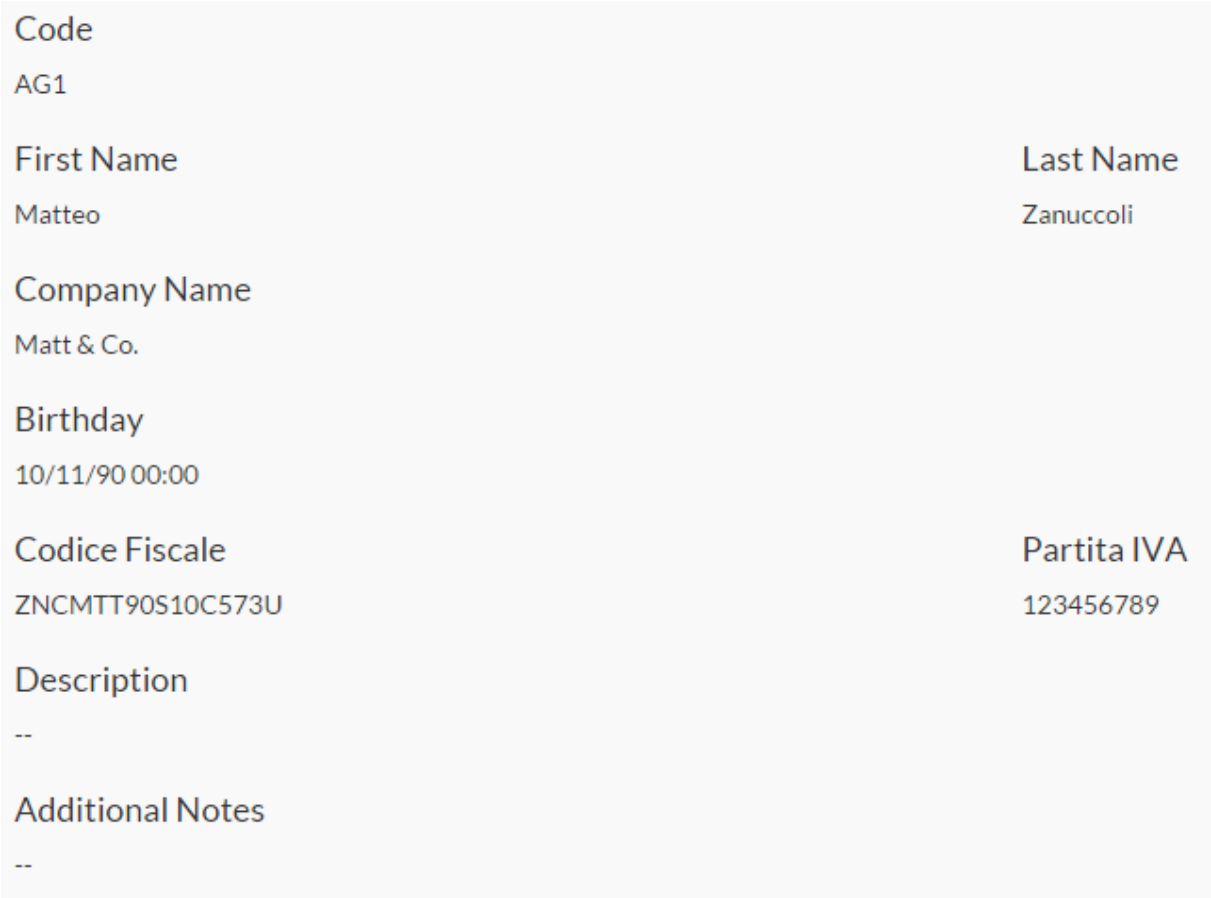


Figura 41 - Bootstrap Grid

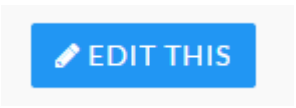


Figura 42 - Bottone "Edit this"

6.4.4 – Pagine edit

Le pagine edit consentono di modificare le tuple precedentemente inserite nella tabella corrispondente alla sezione scelta. Queste sono in tutto e per tutto uguali alle pagine add, tranne che per l'autoriempimento dei campi già precedentemente compilati.

All'interno della pagina si possono individuare vari elementi:

- Nome dell'azione e identificativo della tupla che si sta modificando. (Figura 40)
- Form che permette l'inserimento dei campi richiesti. (Figura 36)
- Bottoni "Submit" e "Cancel" per inoltrare la form o cancellare l'azione di modifica. (Figura 37)
- Sezione "Related" per aggiungere e/o modificare relazioni con altre tabelle, come visto in precedenza per le pagine add (Figura 43).
- Bottone "Delete this" per eliminare la tupla (Soft Delete).

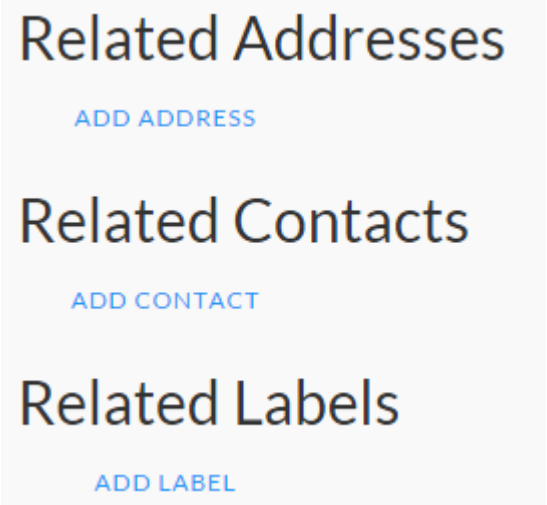


Figura 43 - Sezione related

Figura 44 - Bottone "Delete this"

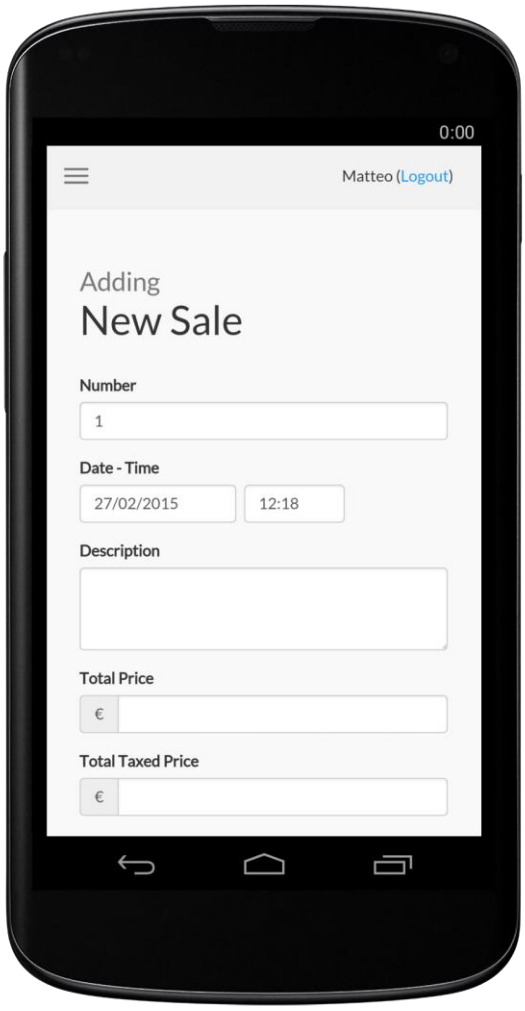
7 – Conclusioni e sviluppi futuri

L'oggetto di questo lavoro di tesi è stata la progettazione e la realizzazione di un software gestionale web-based su piattaforma cloud.
Il sistema informativo è nato come evoluzione di un progetto precedentemente sviluppato in collaborazione con l'azienda *Librasoft snc*.
L'esigenza di evolvere tale sistema è stata causata del cambio delle richieste del committente e ha portato ad una totale ristrutturazione della base di dati. Al fine di modellare adeguatamente le nuove esigenze, si sono seguite le classiche fasi del ciclo di vita di un software.
Si sono innanzitutto condotte più interviste col committente, al fine di individuare le caratteristiche principali del dominio applicativo da modellare. Successivamente, queste interviste sono state analizzate, sotto ogni aspetto, per eliminare ogni possibile incomprensione sulla terminologia utilizzata dal committente.
È poi iniziata la fase di progettazione. Partendo dalla progettazione concettuale si è modellato ogni aspetto del sistema; successivamente si è costruito lo schema logico a partire dallo schema concettuale rifinito. A questo punto, a partire dallo schema logico, è stato possibile definire lo schema relazionale della basi di dati.
Infine, è stato realizzato un modulo software che permettesse di eseguire tutte le varie operazioni, richieste dal committente durante la fase di specifica dei requisiti. Per soddisfare alcune richieste dell'utilizzatore finale, si è scelto di sviluppare l'applicativo avvalendosi delle più moderne tecnologie web.

7.1 – Sviluppi futuri

Sebbene molte delle funzionalità richieste siano già state implementate, questo non frena gli sviluppi futuri del progetto. Uno dei principali limiti individuati durante l'introduzione al progetto è quello di non poter funzionare in assenza di connessione. Uno dei possibili sviluppi futuri è, infatti, quello di implementare il funzionamento offline, utilizzando una nuova tecnologia web chiamata HTML5 Appcache. Questa tecnologia consente di scaricare parte degli elementi web sul proprio dispositivo, in modo da poter proseguire il lavoro in locale ed aggiornare i dati quando si torna online.

Un altro sviluppo futuro, in questo momento in fase di implementazione, è il supporto ai dispositivi mobili. Grazie ad opportune media-query, il sistema può adattarsi (Responsive) allo schermo di qualsiasi dispositivo, in modo da aumentarne la produttività.



Appendici

Appendice A

Tavola 1 - Attributi dell'entità Persona

Attributo	Descrizione
code	Codice alfanumerico utilizzato per identificare la persona. Utilizzando un codice interno invece che, ad esempio, il codice fiscale, si hanno numerosi vantaggi. Ad esempio è possibile inserire un cliente che al momento è sprovvisto di codice fiscale e modificarlo in un secondo momento.
first_name	Nome della persona.
last_name	Cognome della persona.
company_name	Nome della compagnia della persona.
codice_fiscale	Codice fiscale della persona.
partita_iva	Partita iva della persona
birthday	Data di compleanno della persona.
description	Descrizione della persona. Deve essere facoltativa.
notes	Note sulla persona. Devono essere facoltative.
addresses	Indirizzi della persona. Possono essere uno o più.
contacts	Contatti della persona. Possono essere uno o più e di diverse tipologie (Telefono, Cellulare, FAX, Mail ecc.)

Tavola 2 - Attributi dell'attributo multiplo Address

Attributo	Descrizione
type	Tipo dell'indirizzo (Di fatturazione, di spedizione ecc.)
address	Indirizzo comprensivo di numero civico.
postal_code	Cap (Codice di avviamento postale).
city	Città.
province	Provincia.
country	Nazione.

Tavola 3 - Attributi dell'attributo multiplo Contact

Attributo	Descrizione
type	Tipo del contatto (Telefono fisso, cellulare, fax, e-mail ecc.).
value	Valore del contatto.

Tavola 4 - Attributi dell'entità Utente

Attributo	Descrizione
email	Identificatore dell'utente. Utilizzato per la procedura di login.
password	Password dell'utente. Utilizzata per la procedura di login.
name	Nome dell'utente. Utilizzato per dare il benvenuto all'utente in seguito al login.
role	Ruolo dell'utente. È stato richiesto dal committente successivamente alle prime interviste.

Tavola 5 - Attributi dell'entità Prodotto

Attributo	Descrizione
code	Codice alfanumerico utilizzato per identificare il prodotto. Utilizzando un codice

	interno invece che, ad esempio, il codice a barre, si hanno numerosi vantaggi. Ad esempio è possibile inserire un prodotto che al momento è sprovvisto di codice a barre e modificarlo in un secondo momento.
barcode	Codice a barre del prodotto.
name	Nome del prodotto.
description	Descrizione del prodotto.
unit_price	Prezzo unitario del prodotto.
dimensions	Dimensioni del prodotto.
weight	Peso del prodotto

Tavola 6 - Attributi dell'entità Magazzino

Attributo	Descrizione
code	Codice alfanumerico utilizzato per identificare il magazzino.
name	Nome del magazzino.
description	Descrizione del magazzino. Deve essere facoltativa.
notes	Note sul magazzino. Devono essere facoltative.
ecommerce	Flag che stabilisce se il magazzino è adibito all'ecommerce. Al momento non ci sono limitazioni sul numero di magazzini adibiti all'ecommerce.
main	Flag che stabilisce se il magazzino è il magazzino principale. Al momento non ci sono limitazioni sul numero di magazzini principali.
addresses	Indirizzi del magazzino. Possono essere uno o più.

contacts	Contatti del magazzino. Possono essere uno o più e di diverse tipologie (Telefono, Cellulare, FAX, Mail ecc.)

Tavola 7 - Attributi dell'entità Ordine

Attributo	Descrizione
number	Codice numerico utilizzato insieme alla data per identificare l'ordine.
datetime	Data e ora utilizzata insieme al codice numerico per identificare l'ordine.
state	Stato dell'ordine.
description	Descrizione dell'ordine. Deve essere facoltativa.
notes	Note sull'ordine. Devono essere facoltative.

Tavola 8 - Attributi dell'entità Documento

Attributo	Descrizione
number	Codice numerico utilizzato insieme alla data per identificare il documento.
datetime	Data e ora utilizzata insieme al codice numerico per identificare il documento.
type	Tipo del documento (Fattura, DDT ecc.).
status	Stato del documento.
status_modified	Data e ora dell'ultima modifica dello stato del documento.
notes	Note sul documento. Devono essere facoltative.

Tavola 9 - Attributi dell'entità Etichetta

Attributo	Descrizione
name	Nome dell'etichetta. In questa fase di progettazione sarà usato il nome come

	identificatore interno in quanto non è ancora chiaro se possono esserci più categorie con lo stesso nome.
type	Tipo di etichettamento.
description	Descrizione dell'etichetta.

Tavola 10 - Descrizione Entità dello Schema Concettuale Finale

Nome Entità	Descrizione
Persona	Entità radice della gerarchia di persone. È identificata da un codice ed è caratterizzata da un nome, da un cognome, da un nome della compagnia, da un codice fiscale, da una partita iva, da un compleanno, da note e descrizioni facoltative e da uno o più indirizzi e contatti.
Cliente	Sotto entità dell'entità persona. È quella persona che viene assegnata a un ordine in uscita.
Agente	Sotto entità dell'entità persona. È quella persona che fa da promotore a un ordine in uscita.
Fornitore	Sotto entità dell'entità persona. È quella persona che viene assegnata a un ordine in entrata.
Ordine	Entità radice della gerarchia degli ordini. È identificata da un numero e da una data ed è caratterizzato da uno stato e da note e descrizione facoltativa.
Ordine in entrata	Sotto entità dell'entità ordine. È un ordine di acquisto.
Ordine in uscita	Sotto entità dell'entità ordine. È un ordine di vendita.

Utente	Entità che rappresenta gli utilizzatori del sistema. È identificata da una email ed è caratterizzato da un nome, da una password e da un ruolo.
Prodotto	Entità che rappresenta i prodotti. È identificata da un codice e sono caratterizzati da un codice a barre, da un nome, da una descrizione, da un prezzo unitario da un peso e dalle dimensioni.
Magazzino	Entità che rappresenta i magazzini. È identificata da un codice ed è caratterizzato da un nome, da una descrizione e note facoltative, da contatti e indirizzi e da flag che stabiliscono se è un magazzino principale e adibito all'ecommerce.
Documento	Entità che rappresenta i documenti. È identificata da un numero e da una data ed è caratterizzata da un tipo, da uno stato e da una data e ora di ultima modifica dello stato.
Documento in uscita	Sotto entità dell'entità documento. È un documento legato a un ordine in uscita.
Documento in entrata	Sotto entità dell'entità documento. È un documento legato a un ordine in entrata.
Etichetta	Entità che rappresenta le etichette. È identificata da un nome ed è caratterizzata da un tipo e da una descrizione.
Etichetta Cliente	Sotto entità dell'entità etichetta. È una etichetta legata a un cliente.
Etichetta Agente	Sotto entità dell'entità etichetta. È una etichetta legata a un agente.
Etichetta Fornitore	Sotto entità dell'entità etichetta. È una etichetta legata a un fornitore.

Etichetta Prodotto	Sotto entità dell'entità etichetta. È una etichetta legata a un prodotto.
Etichetta Magazzino	Sotto entità dell'entità etichetta. È una etichetta legata a un magazzino.

Tavola 11 - Descrizione Associazioni dello Schema Concettuale Finale

Nome associazione	Entità coinvolte	Descrizione
Eseguito da	Utente, Ordine	Stabilisce quale utente ha creato quel determinato ordine.
Relativo a (1)	Cliente, Ordine	Stabilisce a che cliente è associato l'ordine.
Relativo a (2)	Fornitore, Ordine	Stabilisce a che fornitore è associato l'ordine.
Promosso da	Agente, Ordine	Stabilisce quale agente ha promosso l'ordine.
Contiene (1)	Ordine in uscita, Prodotto	Stabilisce quali prodotti sono contenuti nell'ordine e in quale quantità.
Contiene (2)	Ordine in entrata, Prodotto	Stabilisce quali prodotti sono contenuti nell'ordine e in quale quantità.
Fornisce	Fornitore, Prodotto	Stabilisce quali prodotti sono forniti da quel determinato fornitore.
Presente	Prodotto, Magazzino	Stabilisce quali prodotti sono presenti in quel determinato magazzino e in quale quantità.
Genera (1)	Ordine in uscita, Documento in uscita	Stabilisce da quale ordine in uscita è stato generato un determinato documento in uscita.

Genera (2)	Ordine in entrata, Documento in uscita	Stabilisce da quale ordine in entrata è stato generato un determinato documento in entrata.
Discendenza	Etichetta, Etichetta	Relazione unaria creata per stabilire una relazione di parentela fra etichette.
Assegnata a (1)	Etichetta Cliente, Cliente	Stabilisce a quale cliente è assegnata quella determinata etichetta.
Assegnata a (2)	Etichetta Agente, Agente	Stabilisce a quale agente è assegnata quella determinata etichetta.
Assegnata a (3)	Etichetta Fornitore, Fornitore	Stabilisce a quale fornitore è assegnata quella determinata etichetta.
Assegnata a (4)	Etichetta Prodotto, Prodotto	Stabilisce a quale prodotto è assegnata quella determinata etichetta.
Assegnata a (5)	Etichetta Magazzino, Magazzino	Stabilisce a quale magazzino è assegnata quella determinata etichetta.

Appendice B

Tavola 12 - Schema Relazionale Finale

clients (id, code, first_name, last_name, company_name, codice_fiscale, partita_iva, birthday, description, notes, created, modified, deleted);
agents (id, code, first_name, last_name, company_name, codice_fiscale, partita_iva, birthday, description, notes, created, modified, deleted);
suppliers (id, code, first_name, last_name, company_name, codice_fiscale, partita_iva, birthday, description, notes, created, modified, deleted);
users (id, email, password, role, name, created, modified, deleted);
sales (id, user_id:users(id), client_id:clients(id), agent_id:agents(id) number, datetime, description, notes, status, total_price, total_taxed_price, total_tax_amount, created, modified, deleted);
purchases (id, user_id:users(id), supplier_id:suppliers(id), number, datetime, description, notes, status, total_price, total_taxed_price, total_tax_amount, created, modified, deleted);
products (id, code, barcode, name, description, purchase_unit_price, purchase_tax_id, purchase_tax_pct, purchase_unit_taxed_price, purchase_unit_tax_amount, sale_unit_price, sale_tax_id, sale_tax_pct, sale_unit_taxed_price, sale_unit_tax_amount, stock_minimum_quantity, shipping_unit_box_number, shipping_unit_width, shipping_unit_height, shipping_unit_depth, shipping_unit_weight, ecommerce_flag, created, modified, deleted);
warehouses (id, code, name, description, notes, ecommerce_flag, main_flag, created, modified, deleted);

sale_documents (id, sale_id:sales(id), number, datetime, type, status, notes, created, modified, deleted);
purchase_documents (id, purchase_id:purchases(id), number, datetime, type, status, notes, created, modified, deleted);
labels (id, parent_id:labels(id), type, name, description, row_scope, lft, rght, created, modified, deleted);
addresses (id, type, address, postal_code, city, province, country, row_scope, created, modified, deleted);
contacts (id, type, value, row_scope, created, modified, deleted);
sale_details (id, sale_id:sales(id), product_id:products(id), quantity, unit_price, total_price, tax_pct, tax_unit_value, tax_unit_price, tax_total_price, tax_total_value, created, modified, deleted);
purchase_details (id, purchase_id:sales(id), product_id:products(id), quantity, unit_price, total_price, tax_pct, tax_unit_value, tax_unit_price, tax_total_price, tax_total_value, created, modified, deleted);
stocks (id, product_id:products(id), warehouse_id:warehouses(id), user_id:users(id), current_quantity, previous_quantity, delta_quantity, created, modified, deleted);
products_suppliers (id, product_id:products(id), supplier_id:suppliers(id), created, modified, deleted);
labels_agents (id, label_id:labels(id), agent_id:agents(id), created, modified, deleted);
labels_clients (id, label_id:labels(id), client_id:clients(id), created, modified, deleted);
labels_products (id, label_id:labels(id), product_id:products(id), created, modified, deleted);
labels_suppliers (id, label_id:labels(id), supplier_id:suppliers(id), created, modified, deleted);

```
labels_warehouses (id, label_id:labels(id),
warehouse_id:warehouses(id), created, modified, deleted);
```

Bibliografia e Webgrafia

Libri

[ER01]

D. Maio, S. Rizzi, A. Franco. Esercizi di Progettazione di Basi di Dati. Esculapio, 2005.

[ER02]

P. Ciaccia, D. Maio. Lezioni di Basi di Dati. Esculapio, 2002.

[IS]

M. Golfarelli, D. Maio, S. Rizzi. Ingegneria dei Sistemi Informativi: Lezioni ed Esercizi di Modellazione dei Requisiti. Esculapio, 2000.

[TECWEB]

E.Castro, B.Hyslop. HTML5 e CSS3 Per il world wide web. Tecniche nuove, 2012.

[JS01]

E.Elliott. Programmare applicazioni JavaScript. Tecniche nuove, 2014.

[JQUER01]

E.Castledine, C.Sharkie. JQuery, guida completa. Apogeo, 2012.

[PHP01] [SQL01]

K.Yank. Sviluppare applicazioni con PHP e MySQL. Apogeo, 2012.

[CAKE01]

Chan K, Omokore J., Millar R.K., Pratical CakePHP Projects. Apress, 2009.

Web

[HTML]

http://www.w3.org/TR/html5/

[CSS01]

http://www.w3.org/Style/CSS/

[CSS02]

http://getbootstrap.com/css/

[CSS03]

http://lesscss.org/

[PHP02]

http://php.net/docs.php

[CAKE02]

http://book.cakephp.org/3.0/en/index.html

[JS02]

http://www.w3.org/standards/webdesign/script.html

[JQUER02]

http://api.jquery.com/

[NETBE]

https://netbeans.org

[PEWINT]

http://www.pewinternet.org/three-technology-revolutions/

[MATER]

http://www.google.com/design/spec/material-design/introduction.html

[FLAT]

http://fltdsgn.com/

[GIOBBY]

https://app.giobby.com/GiobbyBiz/Login.xhtml

[GIT]

https://github.com/

[BIT]

https://bitbucket.org/

[WIKI]

http://www.wikipedia.org