Università degli Studi di Modena e Reggio Emilia

Dipartimento di Ingegneria "Enzo Ferrari"
Corso di Laurea in Ingegneria Informatica

PROGETTO E REALIZZAZIONE DI UN MARKETPLACE SU PIATTAFORMA PRESTASHOP

Relatore:
Prof.ssa Sonia Bergamaschi

Candidato:
Antonio Circiello

Anno Accademico 2013/2014

PAROLE CHIAVE

e-commerce
marketplace
PrestaShop
php
query

INDICE

PANORAMICA DEI CAPITOLI

Capitolo 1. Questo è un capitolo introduttivo in cui viene spiegato l'inquadramento e l'obiettivo del progetto TuuDoo. Viene quindi illustrato il concetto di e-commerce e ne vengono argomentate le varie caratteristiche.

Capitolo 2. In questo capitolo viene spiegato nel dettaglio lo strumento software PrestaShop, usato come base per TuuDoo. Vengono illustrate l'installazione, la struttura di base e alcuni principi del suo funzionamento. Si conclude con un panoramica del database e delle classi che interagiscono con esso.

Capitolo 3. Questo capitolo è dedicato a alla ricerca delle possibili soluzioni e al confronto tra varie alternative valide. Vengono analizzati due moduli marketplace sotto vari aspetti. Si passa poi a illustrare le varie fasi di testing e la fase, di carattere più economico, di acquisto del modulo scelto.

Capitolo 4. Viene affrontato più approfonditamente il sistema modulare di PrestaShop e vengono illustrate le varie fasi necessarie alla creazione di un modulo. Vengono analizzati nel dettaglio i vari file, funzioni e query che compongono il modulo Sellers Block, appositamente creato per il progetto.

Capitolo 5. Questo è un capitolo di carattere molto meno tecnico, in cui vengono spiegate le fasi finali del progetto e il testing sul campo.

INDICE DELLE FIGURE

INDICE DELLE TABELLE

INDICE DEI METODI

Capitolo 1
Introduzione

1.1 OBIETTIVO

Il lavoro di tesi è stato svolto a fronte di un tirocinio di 3 mesi presso l'azienda CarpiNet S.a.s. di Carpi.
CarpiNet è un'azienda specializzata in tecnologie internet, in grado di offrire un'ampia gamma di soluzioni per il business, dalla consulenza strategica e tecnologica alla progettazione di applicazioni, dai servizi di Application Service Provider alla realizzazione di interfacce Web, dall'accesso alla rete alla realizzazione di siti Internet [1]. Il data center di CarpiNet è allacciato ad una rete in fibra ottica in un locale isolato e condizionato sotto la completa supervisione del personale.

Figura 1: Logo CarpiNet S.a.s.

Scopo del progetto è realizzare un marketplace dove aziende o singoli utenti possano, previa contratto con i gestori, vendere i propri prodotti. Il nome deciso per il marketplace è TuuDoo.
Nella lingue italiana, con la parola marketplace si indica un sito internet di intermediazione per la compravendita di un bene o un servizio, in altre parole un marketplace è un mercato online in cui sono raggruppate le merci di diversi venditori o diversi siti web.

Il venditore deve poter accedere ad una sua area riservata da dove gestire tutte le sue attività, dall'inserimento e la modifica dei propri prodotti, alla gestione dei propri ordini e delle relative spedizioni.
Il sito deve gestire un salvadanaio unico, in cui confluiranno tutti i proventi delle vendite; sarà poi compito dei gestori recapitare ai venditori i guadagni che spettano loro, previa trattenuta di una commissione.

Come software di partenza si è scelto di usare PrestaShop 1.6, CMS open source per e-commerce.
La scelta è ricaduta su PrestaShop e non su altri CMS simili (es. Magento) poiché nell'ambito aziendale era già stato usato in passato per altri progetti e perché PrestaShop dispone di una community online molto attiva che si sarebbe rivelata utile in caso di problemi o incertezze relativi al progetto TuuDoo.
PrestaShop, che permette ad un solo venditore di commercializzare i propri prodotti, deve essere opportunamente esteso in modo da diventare una sorta di vetrina sulla quale più venditori possano gestire la propria attività di vendita, ognuno indipendentemente dagli altri.
Per effettuare questa modifica, la politica da adottare prevede una ricerca e utilizzo di software e materiale già preparato che possa essere utile allo scopo. Tale software verrà poi eventualmente modificato, adattato ed ampliato per soddisfare a pieno le specifiche di progetto.
Una volta messa in piedi la struttura di base, il sito verrà adattato alle scelte grafiche e verrà sottoposto a dei clienti per una prova sul campo.

1.2 E-COMMERCE

Il significato del termine e-commerce ha subito vari mutamenti con il tempo. Inizialmente indicava il supporto alle attività commerciali in forma elettronica, generalmente per inviare documenti commerciali quali fatture o ordini di acquisto in formato elettronico.
In seguito il termine è stato usato per indicare anche altri concetti. Con e-commerce ci si riferisce quindi all'insieme di transazioni per la commercializzazione di beni e servizi tra produttore (offerta) e consumatore (domanda) realizzata tramite Internet.
Si può intendere l'e-commerce anche come l'insieme delle applicazioni dedicate alle transazioni commerciali. [4]

1.2.1 Tipologia di prodotto

Alcune tipologie di prodotti o servizi possono essere più idonee di altre per un'attività di commercio elettronico.
Le imprese che hanno realizzato le migliori performance in questo campo, pur tuttavia restando delle entità completamente virtuali, vendono solitamente prodotti informatici come supporti di archiviazione, brani musicali, film, videogiochi, materiali didattici, software.
Un caso interessante è quello delle aziende specializzate nell'offerta di buoni sconto spendibili dal cliente presso altre aziende.
Tuttavia, anche la vendita di prodotti non digitali può permettere ai negozianti in rete di riscuotere un certo successo. I prodotti non digitali che più si prestano ad essere venduti in linea sono quelli che creano possibile imbarazzo nell'acquirente oppure che presentano caratteristiche standard e non necessitano di essere provati o valutati dal vivo (es. libri).
Un altro buon esempio di prodotti vendibili via Internet è quello dei pezzi di ricambio, sia per utenti finali che per attività industriali. Dal momento che i negozianti non accumulano pezzi di ricambio presso i punti vendita, spesso devono successivamente ordinarli; la competizione, in questo caso, non è tra commercio elettronico e commercio tradizionale ma tra il sistema di ordine al fornitore e il sistema di ordine al grossista. In questa nicchia, un fattore di successo è la possibilità di fornire al cliente informazioni precise, dettagliate e affidabili.
D'altro canto, i prodotti non facilmente adattabili all'e-commerce sono quelli che hanno una componente rilevante riguardo al loro odore, gusto o percezione tattile, quelli che necessitano di essere provati o osservati dal vivo per poter cogliere tutti gli aspetti caratteristici.

1.2.2 Tipologie di e-commerce

I mercati relativi all'e-commerce sono generalmente classificati in tre principali categorie che, a loro volta, possono includere sottocategorie specifiche:

- **Business to Business (B2B)**, transazioni commerciali elettroniche tra imprese. Più specificamente, indica le relazioni che un'impresa detiene con i propri fornitori per attività di approvvigionamento, pianificazione e monitoraggio della produzione, sussidio nelle attività di sviluppo del

prodotto oppure le relazioni che l'impresa detiene con clienti professionali, cioè altre imprese, collocati in punti diversi della filiera produttiva. Il volume di transazioni B2B è molto elevato rispetto a quello di transazioni B2C, una delle ragioni consiste nel fatto che le imprese hanno adottato tecnologie di commercio elettronico molto più di quanto abbiano fatto i consumatori. Sottocategorie del B2B possono essere il commercio Intra-Business, che coinvolge un'azienda con sedi distribuite o più aziende appartenenti allo stesso gruppo, il B2G (Business to Government), in cui la pubblica amministrazione acquista beni e servizi dal settore privato, e il G2B.

- **Business to Consumer (B2C)**, relazioni che un'impresa commerciale detiene con i suoi clienti per le attività di vendita e assistenza. Gli intermediari costituiscono attualmente la maggior parte delle aziende operanti nel B2C, sono aziende che facilitano le transazioni tra clienti e venditori ricavando una percentuale del valore della transazione stessa.
- **Consumer to Consumer (C2C)**, transazioni che avvengono tra singoli soggetti per via telematica attraverso appositi siti internet, che fungono da intermediari. Tali siti gestiscono l'ambiente nel quali gli utenti interagiscono, lasciando ampia autonomia nella regolazione delle modalità delle transazioni che vengono appunto decise dai singoli venditori e acquirenti. Le entrate di questi siti derivano maggiormente dalla percentuale che ricevono sulle transazioni degli utenti. Tipico esempio di tale forma di e-commerce sono le aste online. [3]

1.2.3 Fattori del successo

Per avere successo in un'attività di commercio elettronico sono necessari alcuni fattori chiave.
Secondo uno studio effettuato da parte di alcuni ricercatore dell'Università degli Studi di Brescia [2], i fattori maggiormente impattanti sulle decisioni di acquisto dei visitatori di un sito sono, in ordine di importanza decrescente:

- divertimento e soddisfazione personale nell'acquisto;
- sicurezza e fiducia ispirata;
- utilità percepita dal cliente;
- influenza dei social network;
- piena disponibilità di internet e della tecnologia;
- competenza dei navigatori;

- facilità d'uso del sito;
- esperienza nell'acquisto online.

Dalla ricerca è emerso che gli elementi maggiormente tecnologici, quali la facilità d'uso del sito, risultano essere meno interessanti, ovvero portano meno clienti, di elementi legati alle scelte di marketing tradizionale. I social network sembrano avere un'influenza blandamente positiva per quanto riguarda il successo di un sito di e-commerce, mentre possono essere estremamente dannosi qualora il giudizio da loro espresso riguardo al sito sia negativo.
L'e-commerce coinvolge sia aspetti funzionali che emozionali, devono quindi essere curati gli elementi che accrescono il potenziali di coinvolgimento, interazione e divertimento.
Inoltre la fiducia riposta nel sito sembra avere un impatto superiore rispetto agli elementi di usability, che tanto preoccupano i web designer.

1.2.4 Problematiche

Sotto alcuni aspetti, per un'attività di commercio elettronico possono sorgere delle difficoltà.
Senza delle buone ricerche di mercato, si potrebbe fraintendere il comportamento della clientela, non raggiungendo così i target di vendita prefissati.
Una mancata analisi dello scenario concorrenziale e l'incapacità di prevedere le reazioni nell'ambiente in cui opera l'impresa possono comportare difficoltà per lo sviluppo dell'attività. Cosa faranno i concorrenti? Scoppierà la guerra dei prezzi? Come reagiranno il governo e gli enti dei mercati coinvolti?
Altri problemi potrebbero sorgere sul piano organizzativo interno. I dipendenti, il sistema hardware, i software adottati e i flussi di informazione tra questi soggetti, possono tutti insieme padroneggiare la strategia adottata?
La realizzazione di un'impresa di *e-commerce* può richiedere un considerevole dispendio di tempo e denaro, e l'incapacità di comprendere la giusta sequenza dei processi imprenditoriali e la tempistica relativa a tali operazioni può portare a rilevanti aumenti dei costi, rispetto a quanto preventivato.

Tra le problematiche più delicate dell'e-commerce vi è la sicurezza nelle modalità di pagamento. Le modalità più diffuse sono il bonifico bancario, il contrassegno e

il pagamento con carta di credito, quest'ultima la più interessata dal problema della sicurezza.
Oggigiorno, i dati trasferiti tra venditore e cliente non vengono più trasmessi in chiaro, il che costituirebbe una grande falla per la sicurezza in quanto le informazioni sarebbero suscettibili ad intercettazioni e all'utilizzo da parte di terzi.
La maggior parte dei siti di e-commerce odierni utilizza livelli di crittografia elevati, ad esempio:

- **SSL/TLS** (Transport Layer Security), protocollo che abbinato all' HTTP permette di ottenere l' HTTPS. Questo protocollo garantisce l'invio delle informazioni personali sotto forma di pacchetti criptati e la loro integrità. Ad oggi, è il sistema più usato.
- **SET** (Secure Electronic Transaction), protocollo nato dalla collaborazione di Visa e MasterCard che prevede che entrambe le parti della transazione siano identificate con certezza prima che essa inizi. L'identificazione avviene tramite certificati digitali rilasciati alle parti dal proprio istituto bancario. Per l'utilizzo di questo protocollo sono necessari software aggiuntivi sul server del venditore.

Con la diffusione dell'e-commerce sono andate diffondendosi anche nuovi tipi di truffe volte principalmente a colpire l'acquirente, alcuni esempio posso essere la vendita di prodotti da parte di siti civetta (a seguito del pagamento la merce non viene inviata o ne viene solo simulata la spedizione) oppure la realizzazione di siti clonati con la finalità di rubare informazioni personali.
La normativa italiana prevede che tutti i siti di e-commerce riportino sulla home page la partita IVA e la denominazione dell'azienda. I siti più importanti hanno un certificato digitale che consente di verificarne l'autenticità.

1.2.5 Accoglienza

Lo shopping elettronico si è sviluppato molto lentamente, anche per categorie di prodotti appropriati per questa tipologia di commercio.
La lenta diffusione potrebbe essere giustificata da alcuni motivi:

- Dubbi in fatto di sicurezza. Molte persone non usano la carta di credito in internet per timore di frodi.
- Mancanza di gratificazione immediata. Quando il prodotto ordinato non arriva per giorni o settimane si perde l'attrattiva stessa dell'acquisto.

- Assenza dell’aspetto sociale dello shopping.
- Il problema dell’accesso al commercio in rete. Nelle nazioni in via di sviluppo la scarsa diffusione degli accessi ad internet riduce di molto il potenziale per l’e-commerce.

Capitolo 2
Piattaforma PrestaShop

2.1 PRESTASHOP

PrestaShop è un CMS open source. Un CMS (Content Management System) è uno strumento software, installato su un server web, il cui compito è facilitare la gestione dei contenuti di siti web, svincolando il webmaster da conoscenze tecniche specifiche di programmazione web.

Figura 2: Logo di PrestaShop

PrestaShop è nato nel 2007 e la sua peculiarità sta nel fatto che è interamente pensato per lo sviluppo e la gestione di siti di e-commerce.

I temi PrestaShop sfruttano il motore di template Smarty, il quale permette una netta separazione tra contenuti, grafica e programmazione. Grazie a questa caratteristica il web designer può occuparsi della grafica del sito lavorando sui file di template e CSS, mentre il web developer può lavorare sui file PHP. Infatti anche la documentazione ufficiale è divisa in due parti, una per gli sviluppatori e una per i designer.

PrestaShop è uno dei CMS per e-commerce più diffusi ed è utilizzato sia da neofiti che da professionisti.

Alcuni motivi del suo successo sono l'interfaccia semplice ed intuitiva ed il sistema modulare (per implementare una funzionalità basta installare il relativo modulo). Inoltre attorno a PrestaShop c'è una comunità molto attiva, che

contribuisce al constante miglioramento del software e allo sviluppo di moduli per implementare sempre nuove funzionalità.
L'applicazione è suddivisa in due parti:

- una sezione di amministrazione (back end), che serve ad organizzare e supervisionare la produzione dei contenuti;
- una sezione applicativa (front end), che l'utente web usa per fruire dei contenuti e delle applicazioni del sito.

Per maggiori dettagli si veda la documentazione presente al sito [5].

2.2 INSTALLAZIONE E CONFIGURAZIONE

Per essere installato, PrestaShop 1.6 necessita di:

- registrazione di un dominio web o di uno spazio hosting;
- web server: Apache 1.3, Apache 2.x, Nginx o Microsoft IIS;
- PHP 5.2 o superior installato e abilitato;
- MySQL 5.0 o superiore installato e abilitato;
- credenziali per l'accesso FTP e per l'accesso al database.

Una volta scaricato PrestaShop dal sito ufficiale e scompattata la cartella, bisogna caricare i file sullo spazio di hosting scelto mediante FTP.
FTP (File Transfer Protocol) è un protocollo per la trasmissione di dati basato su TCP.
Nel mio caso lo spazio di hosting è un server dell'azienda e il client FTP usato è FileZilla.
Prima di poter installare PrestaShop, è necessario che sul server MySQL ci sia un database pronto ad accogliere i dati di PrestaShop. In caso negativo, il database va creato.
Come strumento di amministrazione del database ho usato phpMyAdmin.
Per lanciare l'installazione basta cercare sul browser l'indirizzo al quale installare PrestaShop (in questo caso verrà automaticamente rilevata la presenza o meno di un PrestaShop installato e verrà fatta partire l'installazione) oppure l'indirizzo della cartella /install (la sottocartella d'installazione tra i file caricati sul server).

Il processo guidato di installazione è articolato in sei passaggi:

1. Introduzione al processo di installazione e scelta della lingua di default.

2. Accettazione delle condizioni delle due licenze Open Software License 3.0 e Academic Free License 3.0.
3. Controllo dei requisiti di sistema e dei parametri del server. Se tutti i requisiti sono rispettati questa pagina non viene mostrata, altrimenti viene visualizzata una lista degli errori riscontrati.
4. Impostazione delle informazioni riguardanti lo store quali il nome, il settore merceologico principale, lo Stato e delle informazioni dell'account di amministratore del sito.
5. Configurazione del sistema e del database. In questa pagina bisogna indicare dove si trova il server del database, i dettagli di accesso e il prefisso dei nomi delle tabelle che verranno create (ps_ di default). È inoltre possibile testare la connessione al database.
6. Pagina di riepilogo in cui vengono mostrate le credenziali dell'account di amministratore.

Per questioni di sicurezza è imperativo, una volta terminata l'installazione, eliminare la cartella /install dal server (tramite il client FTP). Inoltre è consigliabile rinominare la cartella /admin in quanto tramite questo indirizzo si accede all'area amministrativa del sito.

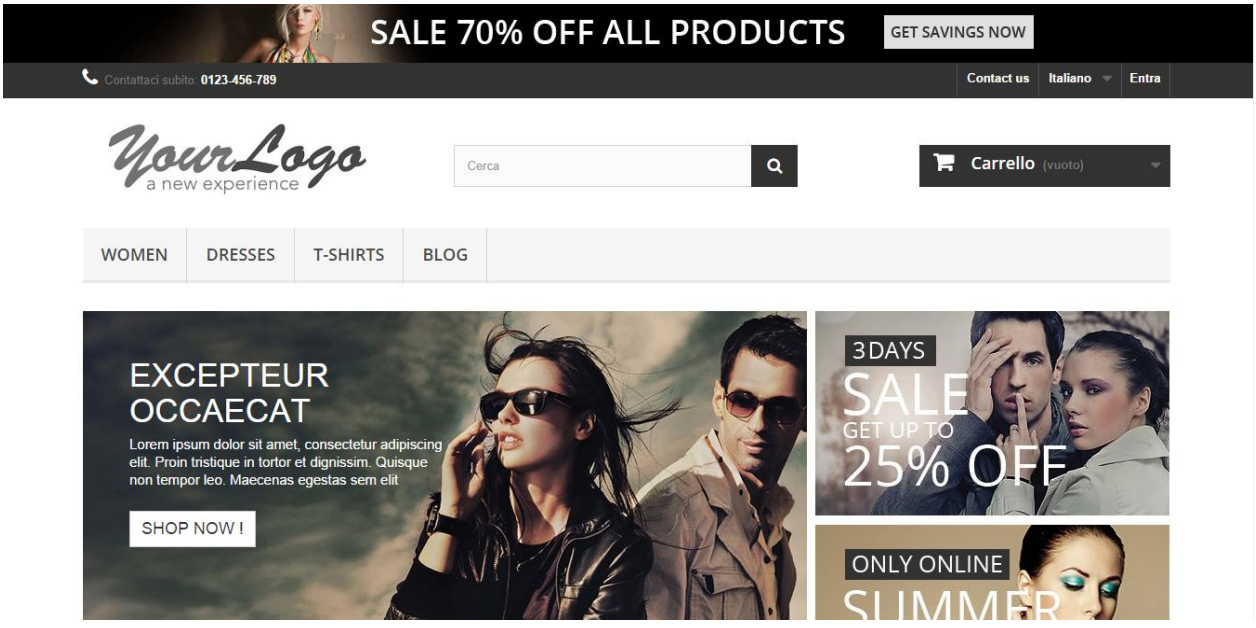

Figura 3: Home page di PrestaShop

PrestaShop può essere installato anche in locale su una macchina, per fare ciò bisogna prima installare l'ambiente adeguato, cioè il web server Apache, l'interprete PHP, il server MySQL per il database e lo strumento phpMyAdmin. Questo è conosciuto come AMP (Apache+MySQL+PHP).

Un'installazione in locale è utile per lo sviluppo e per effettuare dei test, in questo modo eventuali errori non intaccheranno la corretta funzionalità dello store. Allo stesso modo, può essere utile conservare una copia di backup dei file presenti sul server.

Per configurare al meglio PrestaShop e fare in modo che abbia prestazioni ottimali è possibile modificare alcune opzioni dal back-office.
Nella sezione SEO (Search Engine Optimization) del menù Preferences si può generare il file robots.txt tramite l'apposito pulsante.
Il protocollo di esclusione robot indica le regole utilizzate dai crawler per applicare restrizioni di analisi sulle pagine di un sito web, queste regole sono contenute nel file robots.txt.
Un crawler è un software che analizza i contenuti di una rete in modo metodico e automatizzato, in genere per conto di un motore di ricerca.
In altre parole generando il file robots.txt si fa in modo che le pagine della parte amministrativa di PrestaShop non vengano indicizzate dai motori di ricerca.

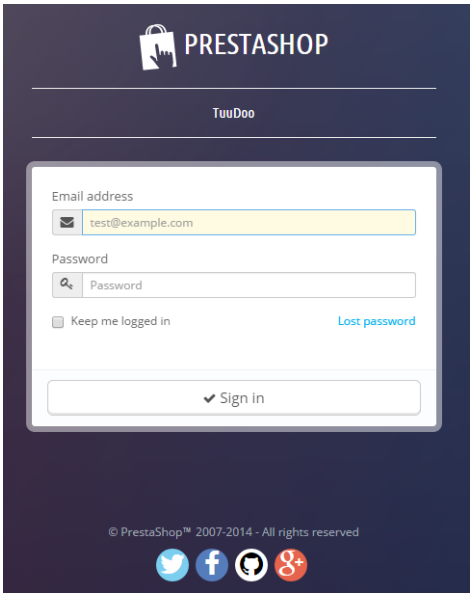

Figura 4: Schermata di login per il back-office

Una volta terminata la fase di sviluppo, per incrementare le prestazioni, è possibile attivare vari tipi di caching e altre ottimizzazioni riguardanti il codice HTML e JavaScript.
PrestaShop usa dei cookie crittografati per memorizzare le informazioni della sessione, sia per quanto riguarda gli utenti/clienti sia per gli amministratori.

2.3 ARCHITETTURA

PrestaShop è stato concepito in modo che i moduli sviluppati da terze parti possano essere facilmente montati, rendendolo così un software per e-commerce estremamente personalizzabile.
La personalizzazione di PrestaShop si basa su tre possibilità:

- temi;
- moduli;
- overriding.

Un modulo è un'estensione che permette al developer di aggiungere:

- funzionalità addizionali a PrestaShop;
- visualizzare elementi addizionali sul sito (selezione prodotti…);
- comunicazioni con altri servizi di e-commerce (piattaforme di pagamento, logistica…).
- …

L'overriding è di per sé un sistema. PrestaShop usa codice orientato agli oggetti. Uno dei vantaggi di ciò è la possibilità di rimpiazzare o estendere facilmente parti del codice di base con del proprio codice aggiuntivo senza aver bisogno di toccare il codice di base.
Il nuovo codice va così a sostituire o integrare il codice di base facendo così funzionare PrestaShop nel modo che si preferisce, nonostante i file di base rimangano invariati.

PrestaShop è basato su un'architettura multi-tier a tre livelli:

- **Object/Data.** L'accesso al database è controllato attraverso i file presenti nella cartella /classes.
- **Data Control.** I contenuti inseriti dall'utente sono controllati attraverso i file presenti nella cartella di root.
- **Design.** Tutti i file relativi ai temi sono nella cartella /themes.

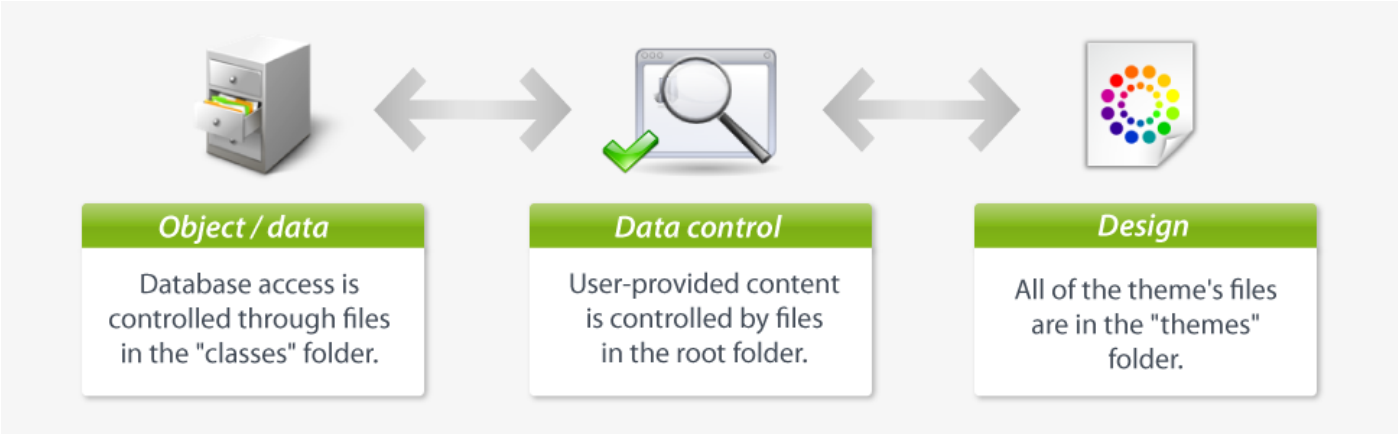

Figura 5: Architettura di PrestaShop

Un'architettura multi-tier è un'architettura software in cui le varie funzionalità del software sono logicamente separate, cioè suddivise su più strati o livelli software differenti in comunicazione tra loro. [4]

Il three-tier è un modello di architettura software e allo stesso tempo uno schema di progettazione software. Oltre ai vantaggi abituali di software modulare con interfacce ben definite, l'architettura three-tier è destinata a consentire a qualsiasi dei tre livelli di essere aggiornato o sostituito indipendentemente dal cambiamento di requisiti o tecnologia.
L'architettura a tre livelli di PrestaShop presenta lo stesso principio di un'architettura MVC (Model-View-Controller), solo in un modo più semplice e più accessibile.
Il model rappresenta il comportamento dell'applicazione (elaborazione dei dati, accesso al database…), il view è l'interfaccia con la quale l'utente interagisce, visualizza i dati forniti dal model e gestisce le azioni dell'utente, infine il controller gestisce la sincronizzazione degli eventi tra model e view e li aggiorna quando serve.
Un'architettura three-tier ha molti vantaggi:

- Il codice è più facile da leggere.
- Gli sviluppatori possono aggiungere o modificare il codice più velocemente.
- I graphic designers possono lavorare entro i confini della cartella /themes senza necessità di comprendere codice PHP.
- Gli sviluppatori possono lavorare su dati aggiuntivi e moduli.

Il team di sviluppo di PrestaShop ha scelto di non usare un framework PHP in modo da consentire una migliore leggibilità e quindi velocizzare l'editing. Questo comporta anche performance migliori in quanto il software contiene solo il codice necessario e non anche varie altre librerie generiche.

2.4 FILE STRUCTURE

Gli sviluppatori di PrestaShop hanno cercato di separare al meglio le varie parti del software.
I file sono divisi in cartelle e organizzati in questo modo:

- /admin(nome personalizzabile durante l'installazione): contiene tutti i file di PrestaShop relativi al back-office. Per accedere a questa cartella è necessaria un'identificazione per motivi di sicurezza.
- /cache: contiene cartelle temporanee generate e riusate per alleviare il carico sul server.

- /classes: contiene tutti i file relativi al modello a oggetti di PrestaShop. Ogni file rappresenta e contiene una classe PHP e i suoi metodi.
- /config: contiene i file di configurazione. Questi file non dovrebbero mai essere editati direttamente, in quanto gestibili direttamente dall'installer di PrestaShop e dal back office.
- /controllers: contiene tutti i file relativi ai controllers (quelli del modello di architettura MVC). Ogni file controlla una specifica parte di PrestaShop.
- /css: contiene tutti i file CSS non legati ad un tema, in genere quelli usati per il back-office.
- /docs: contiene alcune documentazioni, licenze ed esempi di file per l'importazione dati.
- /download: contiene i prodotti virtuali inseriti sul sito e acquistabili/scaricabili dall'utente. I nomi dei file sono crittografati in MD5.
- /img: contiene tutte le immagini di default di PrestaShop, icone e file immagini che non appartengono ad un tema. Qui si possono trovare le figure per i prodotti, per le categorie e per il back-office.
- /install: contiene i file relativi all'installer di PrestaShop. Questa cartella va eliminata dopo l'installazione.
- /js: contiene i file JavaScript non legati ad un tema, in genere quelli usati per il back-office. In questa cartella si trova il framework jQuery.
- /localization: contiene tutti i file relativi alla localizzazione geografica, cioè quei file che contengo informazioni relative a valute, tasse, lingue, Stati, unità di misura.
- /log: contiene i file di log generati da PrestaShop in vari momenti.
- /mails: contiene tutti i file HTML e i file di testo relativi alle email mandate da PrestaShop. C'è una sotto cartella per ogni lingua. PrestaShop ha uno strumento apposito per scrivere e modificare le proprie email.
- /modules: contiene tutti i moduli di PrestaShop, ognuno nella propria cartella.
- /overrides: questa è la cartella in cui posizionare i file che andranno a estendere o modificare le classi o i controller di default (sistema di overriding). Posizionandoli in questa cartella, non vengono modificati i file originali, in quanto situati in altre cartelle, che quindi saranno disponibili per eventuali futuri aggiornamenti di PrestaShop.
- /pdf: contiene i file di template (.tpl) relativi alla generazione di file PDF (fatture, bolle di consegna…). Cambiando questi file è possibile modificare l'aspetto dei PDF generati.
- /themes: contiene tutti i temi installati, ognuno nella propria cartella.

- /tools: contiene strumenti esterni che sono integrati in PrestaShop. Qui si trovano Smarty (motore di template), TCPDF (generatore di file PDF), Swift (mail sender), PEAR XML Parser (tool PHP)…
- /translations: contiene una sottocartella per ogni lingua. Per cambiare le traduzioni bisogna però usare lo strumento interno di PrestaShop e non modificare manualmente questi file.
- /upload: contiene i file caricati dai clienti per i prodotti personalizzabili (es. una foto da stampare su una maglietta).
- /webservice: contiene i file che permettono ad applicazioni di terze parti di accedere a PrestaShop tramite la sua API (Application Programming Interface).

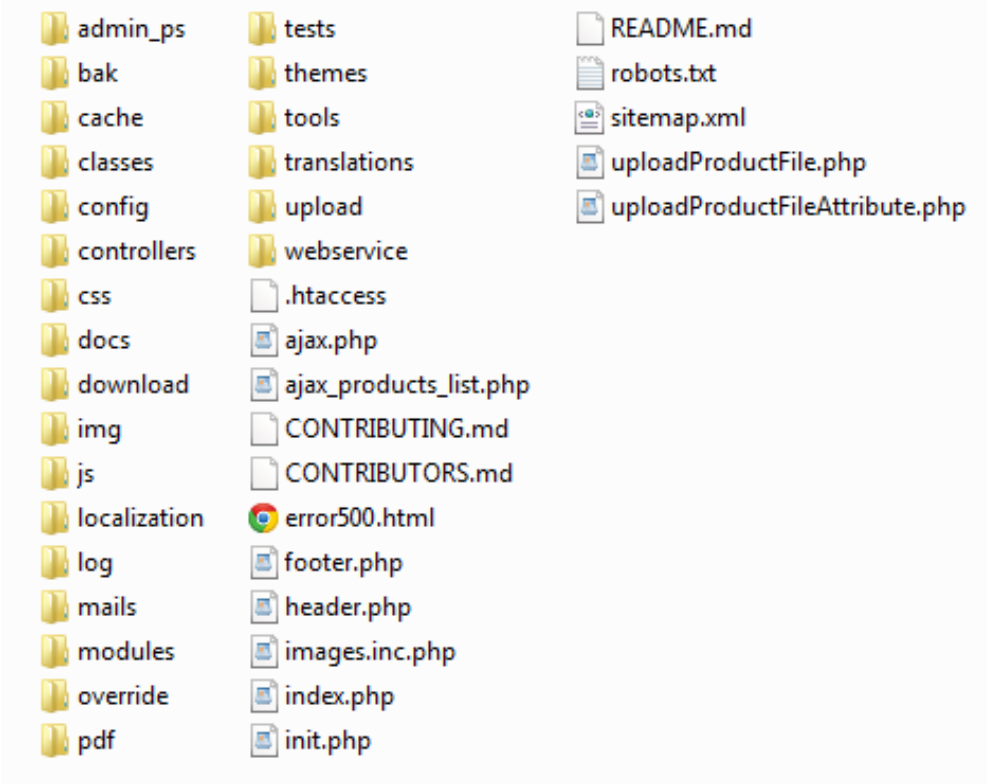

Figura 6: File structure di PrestaShop

2.5 FUNZIONAMENTO

2.5.1 Dispatcher

Il dispatcher gestisce le redirezioni URL.

Un URL (Uniform Resource Locator) è una sequenza di caratteri che identifica univocamente l'indirizzo di una risorsa in Internet.
Ad esempio, invece di usare più file PHP nella cartella di root come category.php o product.php, PrestaShop usa solo il file index.php.
Gli URL del front-office saranno quindi di questo tipo:

/index.php?id_category=3&controller=category,
/index.php?id_product=7&controller=product.

In più, il dispatcher supporta l'URL-rewriting, ovvero converte gli URL in Friendy URLs, cioè URL di più facile lettura per gli umani.
Quindi, un URL di PrestaShop, quando l'URL-rewriting non è attivo è così:

http://sitename.com/index.php?controller=category&id_category=3&id_lang=1;

mentre quando è attivo risulta essere simile a questo:

http://sitename.com/en/3-music.

Questo sistema ha svariati vantaggi:

- è facile aggiungere un controller;
- si possono personalizzare i friendly URLs, il che è molto positivo per la SEO;
- c'è un unico punto di accesso al software, ciò aumenta l'affidabilità di PrestaShop e facilita gli sviluppi futuri.

Il dispatcher fa uso delle classi Controller, FrontController e AdminController.

2.5.2 Controllers

In un'architettura MVC, un controller gestisce la sincronizzazione degli eventi tra model e view e li tiene aggiornati. Riceve tutti i comandi dell'utente e fa partire le relative azioni da eseguire.
Se un'azione necessita di modificare dei dati, il controller "chiede" al model di cambiare i dati e il model a sua volta notifica il cambiamento al view che in questo modo può aggiornarsi.
Tutti i controllers di PrestaShop estendono la classe Controller attraverso una delle sue sottoclassi:

- AdminController;
- ModuleAdminController;
- FrontController;
- ModuleFrontController.

Questi controller si possono trovare nella cartella /classes/controller.

L'ordine di esecuzione delle funzioni di un controller è:

1. __construct(): setta tutte le variabili;
2. init(): inizializza il controller;
3. setMedia() o setMobileMedia(): aggiunge le specifiche CSS e JavaScript alla pagina in modo che possano essere combinate, compresse e messe in cache;
4. postProcess(): gestisce ajaxProcess;
5. initHeader();
6. initContent(): inizializza il contenuto;
7. initFooter();
8. display() o displayAjax(): visualizza il contenuto.

Nome del controller	Descrizione
AddressController	Usato per modificare l'indirizzo di un utente.
AuthController	Usato per il login degli utenti.
CartController	Usato per gestire i carrelli dei client.
CategoryController	Usato per gestire le categorie.
CompareController	Usato per comparare I prodotti.
ContactController	Usato per mandare messaggi.
DiscountController	Usato per rilevare lo sconto riservato ad un cliente.
GuestTrackingController	Usato per gestire gli ordini di visitatori non registrati.
HistoryController	Usato per accedere alla cronologia degli ordini di un cliente.
IdentityController	Usato per gestire le informazioni personali dell'utente.
IndexController	Usato da index.php per visualizzare la home page.

Nome del controller	Descrizione
MyAccountController	Usato per gestire l'account di un utente.
NewProductsController	Usato per selezionare i nuovi prodotti inseriti.
OrderConfirmationController	Usato per la conferma dell'ordine.
OrderController	Usato per gestire il checkout.
PageNotFoundController	Usato per gestire la pagina "404 Not Found"
PasswordController	Usato per resettare una password persa.
PricesDropController	Usato per selezionare i prodotti in sconto.
ProductController	Usato per selezionare un prodotto.
SearchController	Usato per ottenere i risultati di una ricerca sul sito.

Tabella 1: Principali controllers di Prestashop

2.5.3 Smarty

PrestaShop usa il motore di template Smarty per generare le views.
Smarty consente di separare il codice PHP (business logic) dal codice HTML (presentation logic) e di generare contenuti web mediante il rimpiazzo di speciali Smarty tag all'interno del documento.
Uno Smarty tag è una direttiva, racchiusa da speciali caratteri (tipicamente parentesi graffe), interpretata dal motore di Smarty.
Queste direttive possono essere variabili, funzioni o anche istruzioni di controllo del flusso.
Smarty permette ai programmatori PHP di definire delle funzioni da includere nei tag stessi di Smarty.
Esso consente la programmazione di template attraverso una serie di caratteristiche built-in, tra cui:

- espressioni regolari;
- foreach, while;
- if, elseif, else;
- modificatori di variabile;
- funzioni create dall'utente;
- calcolo matematico all'interno del template
- …

Uno degli aspetti caratteristici di Smarty è la compilazione dei template. Smarty legge i file dei template e crea script PHP a partire da questi. Una volta creati, questi script vengono eseguiti da quel momento in poi, di conseguenza si evita una costosa analisi dei template ad ogni richesta, e ogni template può avvantaggiarsi pienamente di strumenti per velocizzare l'esecuzione.
Per ulteriori dettagli su Smarty si veda il sito [6].

2.5.4 Hooks

Gli hooks sono un modo per associare il codice a degli specifici eventi di PrestaShop.
Il più delle volte sono usati per inserire del contenuto in una pagina.
Gli hooks possono essere usati anche per effettuare specifiche azioni sotto determinate circostanze (es. mandare una mail al cliente).

Il sistema di denominazione degli hooks prevede un prefisso, esso indica il tipo di situazione per la quale l'hook è destinato:

- **action**, questi hooks sono attivati da specifici eventi di PrestaShop;
- **display**, questi hooks si risolvono in qualcosa di visualizzabile nel front-office o nel back-office.

Front-office: home page e pagine generiche del sito	
Nome dell'hook	**Descrizione**
displayHeader	Chiamato con i tag <head> dell'HTML. Posizione ideale per aggiungere file JavaScript o CSS.
displayTop	Chiamato nell'header della pagina.
displayLeftColumn	Chiamato quando viene caricata la colonna sinistra.
displayRightColumn	Chiamato quando viene caricata la colonna destra.
displayFooter	Chiamato nel footer della pagina.
displayHome	Chiamato al centro della home page.
actionSearch	Chiamato dopo una ricerca. Posizione ideale per analizzare la ricerca e agire sulle query o sui risultati.

Front-office: pagina del prodotto	
Nome dell'hook	**Descrizione**
displayProductButtons	Chiamato all'interno del blocco "Aggiungi al carrello" del prodotto.
actionProductOutOfStock	Chiamato all'interno del blocco "Aggiungi al carrello" del prodotto.
displayProductTab	Chiamato nella lista dei pulsanti relativi ad un prodotto, ad esempio "More info", "Manuale", "Accessori", etc.
displayProductTabContent	Chiamato quando uno dei pulsanti sopra citati viene cliccato.
Front-office: pagina del carrello	
Nome dell'hook	**Descrizione**
actionCartSave	Chiamato subito dopo la creazione o l'aggiornamento di un carrello.
displayCustomerAccountForm	Chiamato nel form di creazione di un account.
actionCustomerAccountAdd	Chiamato subito dopo la creazione di un account.
displayCustomerAccount	Chiamato sulla pagina di gestione dell'account. Ideale per aggiungere un link ad una sezione/funzionalità.
actionAuthentication	Chiamato subito dopo l'identificazione di un utente, solo se le credenziali sono valide.
Front-office: pagina per il pagamento	
Nome dell'hook	**Descrizione**
displayPayment	Chiamato durante il processamento di un'ordine, quando bisogna mostrare la lista delle possibili soluzioni di pagamento. Ideale per abilitare la scelta di un modulo di pagamento creato appositamente.
displayBeforePayment	Chiamato quando bisogna mostrare la lista delle possibili soluzioni di pagamento. Posizione ideale per redirezionare un'utente invece che mostrare tale lista.

Back-office: hooks generici	
Nome dell'hook	**Descrizione**
displayBackOfficeHeader	Chiamato con i tag <head> dell'HTML. Posizione ideale per aggiungere file JavaScript o CSS.
displayBackOfficeFooter	Chiamato nel footer della pagina.
displayBackOfficeHome	Chiamato al centro della homepage.
Back-office: ordini e dettagli dell'ordine	
Nome dell'hook	**Descrizione**
actionValidateOrder	Chiamato durante il processo di creazione di un nuovo ordine.
actionPaymentConfirmation	Chiamato quando lo stato di un ordine diventa "Pagamento Accettato".
actionOrderStatusUpdate	Chiamato quando viene cambiato lo stato di un ordine.
displayInvoice	Chiamato quando vengono visualizzati i dettagli di un ordine.
Back-office: prodotti	
Nome dell'hook	**Descrizione**
actionProductSave	Chiamato quando si salva un prodotti.
actionWatermark	Chiamato quando viene aggiunta un'immagine ad un prodotto.
Back-office: statistiche	
Nome dell'hook	**Descrizione**
displayAdminStatsGraphEngine	Chiamato quando viene visualizzato un grafico per le statistiche.
displayAdminStatsModules	Chiamato quando viene visualizzata la lista delle statistiche dei moduli.
Back-office: clienti	
Nome dell'hook	**Descrizione**
displayAdminCustomers	Chiamato quando vengono visualizzati i dettagli di un cliente.

Tabella 2: Principali hooks di Prestashop

2.6 DATABASE

2.6.1 Struttura

Con il termine data model si intende la descrizione di oggetti rappresentati da un sistema informatico insieme con le loro proprietà e relazioni, sono tipicamente oggetti "reali" come prodotti, corrieri, clienti e ordini. [4]
I data model sono spesso usati per aiutare la comunicazione tra gli uomini d'affari che definiscono i requisiti di un sistema informatico e il personale tecnico che definisce il progetto secondo questi requisiti.
Un data model è specificato in una notazione spesso in forma grafica e determina esplicitamente la struttura dei dati.
Un'istanza di data model può essere espressa in tre modi, in conformità con gli standard ANSI (American National Standard Institute):

- **Conceptual Data Model** – Identifica le relazioni di alto livello tra le entità. Tipicamente include solo i concetti principali e non ha sufficienti dettagli per costruire un effetivo database, il numero di oggetti deve essere piccolo. Primary e foreign keys non sono specificate.
- **Logical Data Model** – Descrive i dati con più dettagli possibile, senza però specificare come questi saranno fisicamente implementati nel database. Include tutte le entità, i loro attributi e le relazioni tra esse, primary e foreign keys sono specificate. A questo livello è prevista una normalizzazione.
- **Physical Data Model** – Rappresenta come il modello viene implementato nel database. Mostra tutte le tabelle e le loro relazioni, includendo i nomi delle colonne, il tipo di dato, i vincoli, primary e foreign keys. A seconda delle specifiche di progetto è possibile che sia necessaria una denormalizzazione. Il Physical Data Model è diverso per diversi RDBMS. [7/8]

	Conceptual	Logical	Physical
Nome delle entità	✓	✓	
Relazioni tra le entità	✓	✓	
Attributi		✓	
Primary Keys		✓	✓
Foreign Keys		✓	✓
Nomi delle tabelle			✓
Nomi delle colonne			✓
Tipi di dato delle colonne			✓

Tabella 3: Confronto tra data model

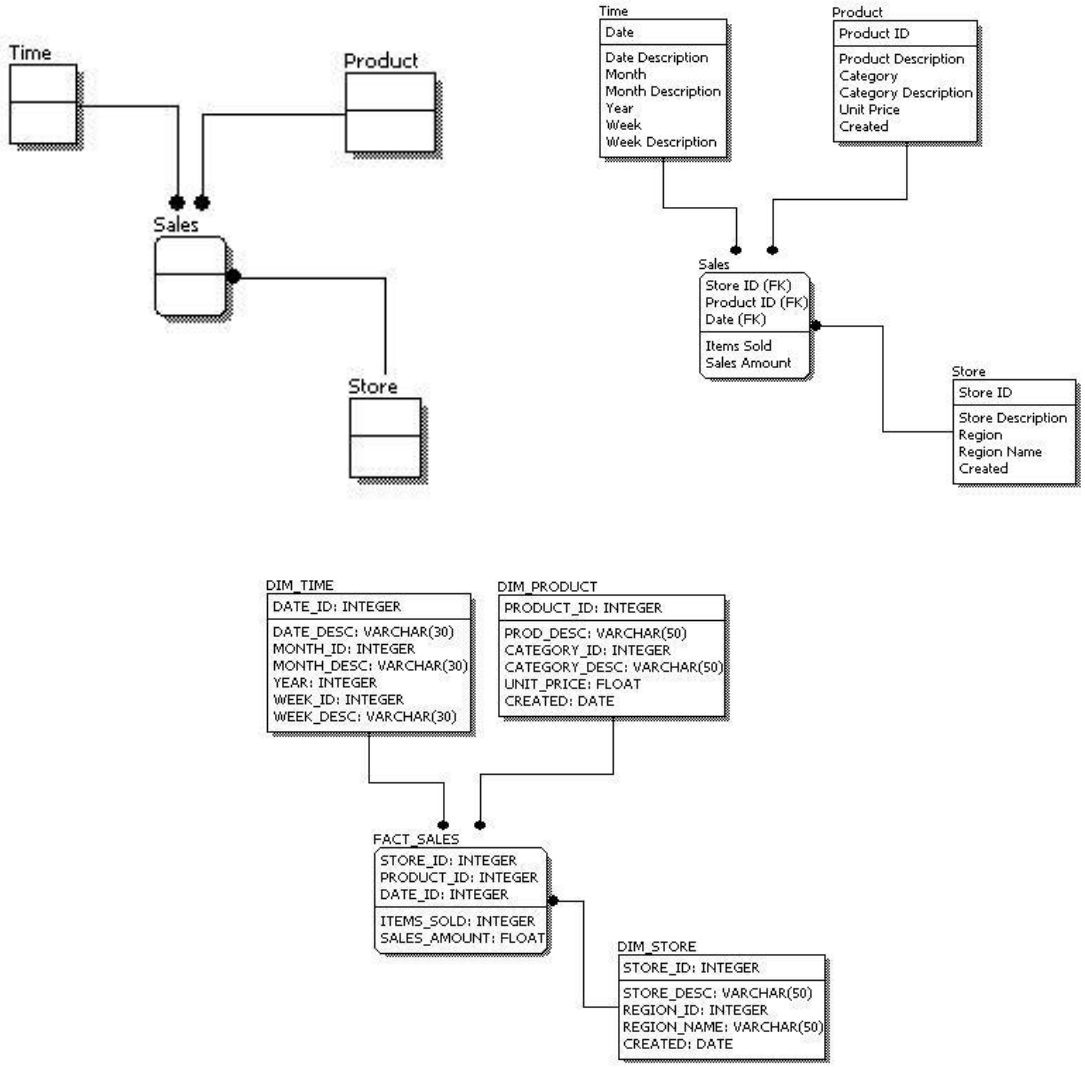

Figura 7: Esempi di Conceptual (sinistra), Logical (destra) e Conceptual Data Model (basso)

Di seguito è riportato il Physical Data Model del database di PrestaShop e il particolare di alcune parti di esempio.

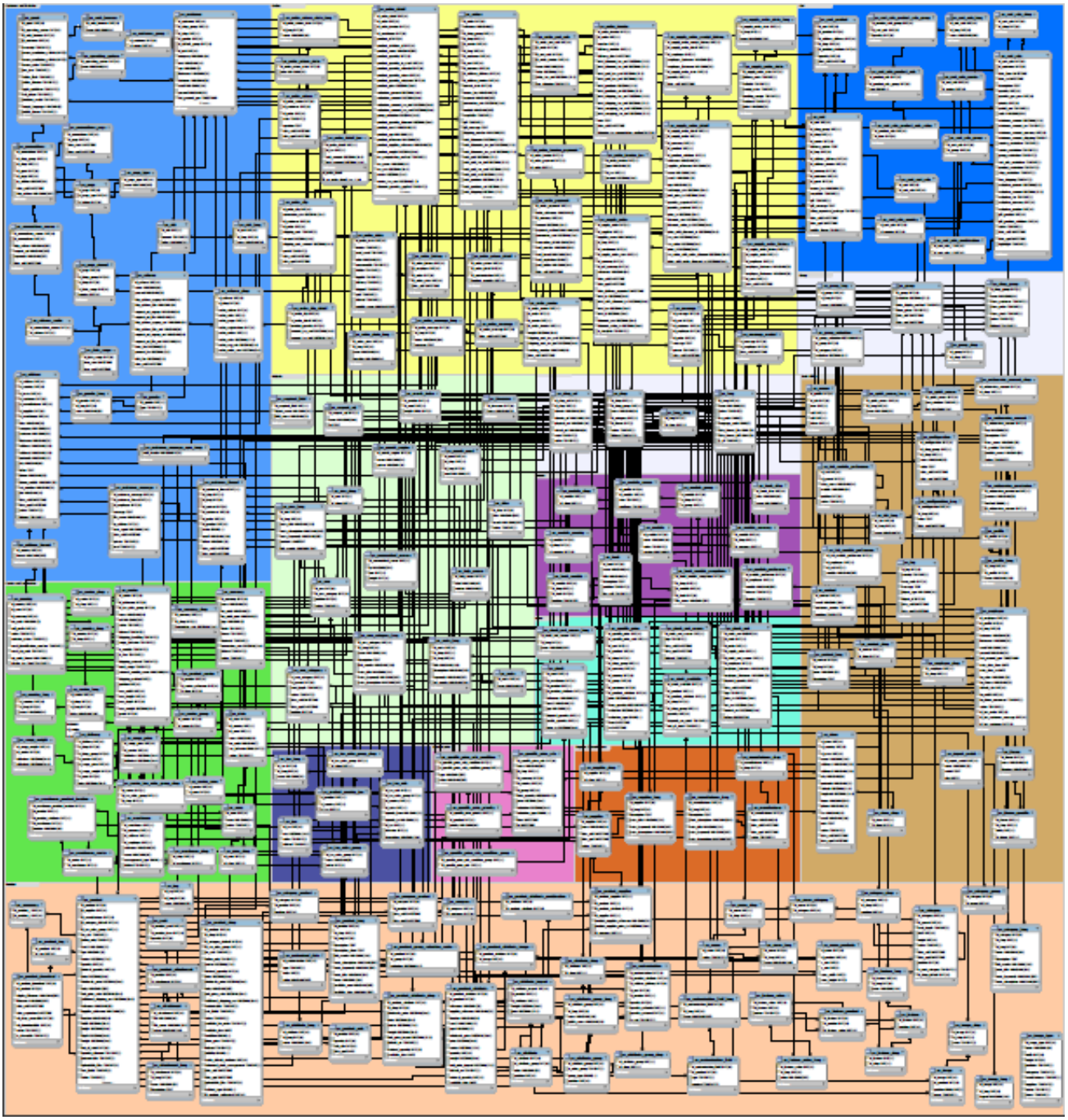

Figura 8: Schema del database di PrestaShop

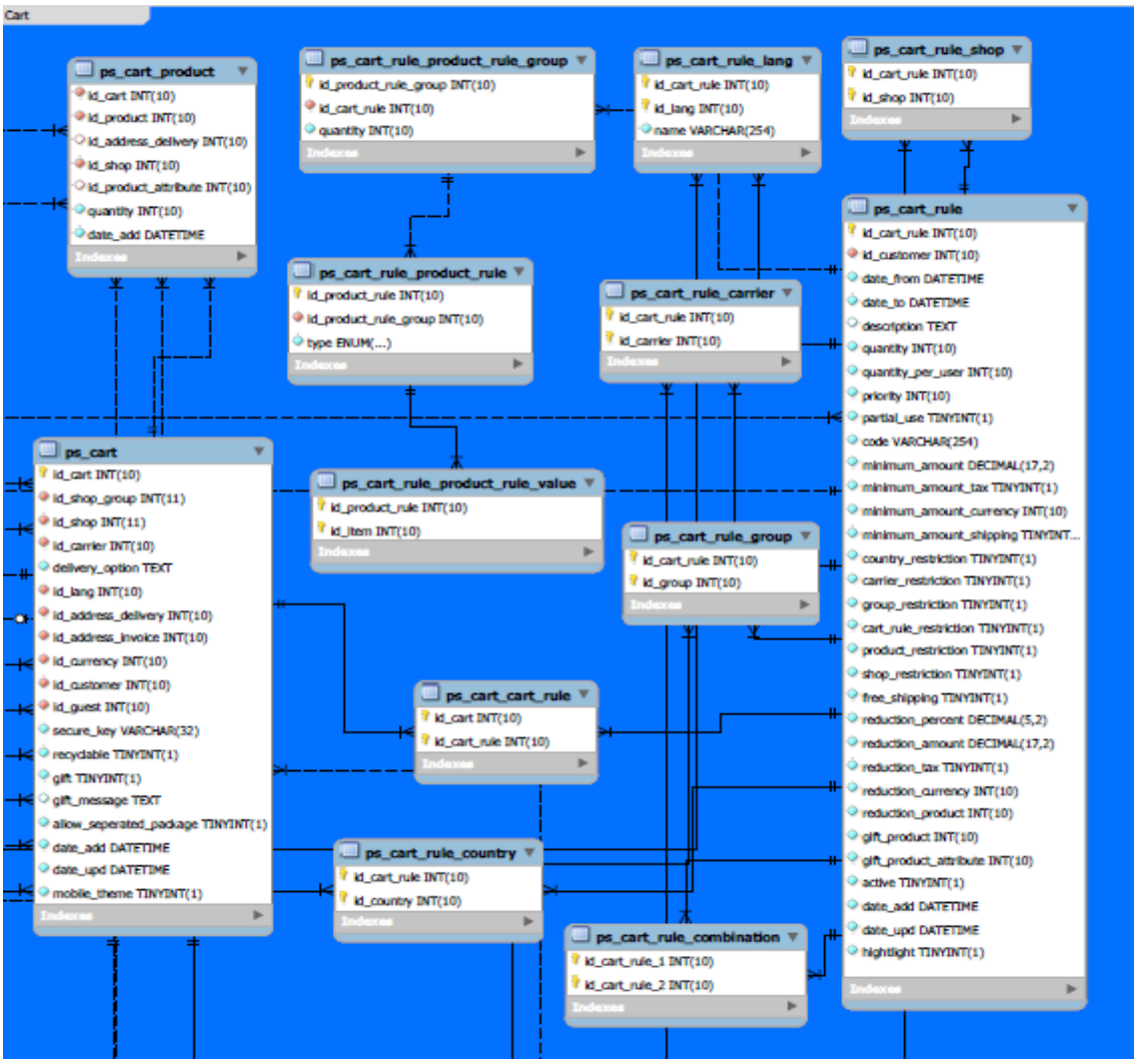

Figura 9: Particolare dello schema, tabelle relative al carrello

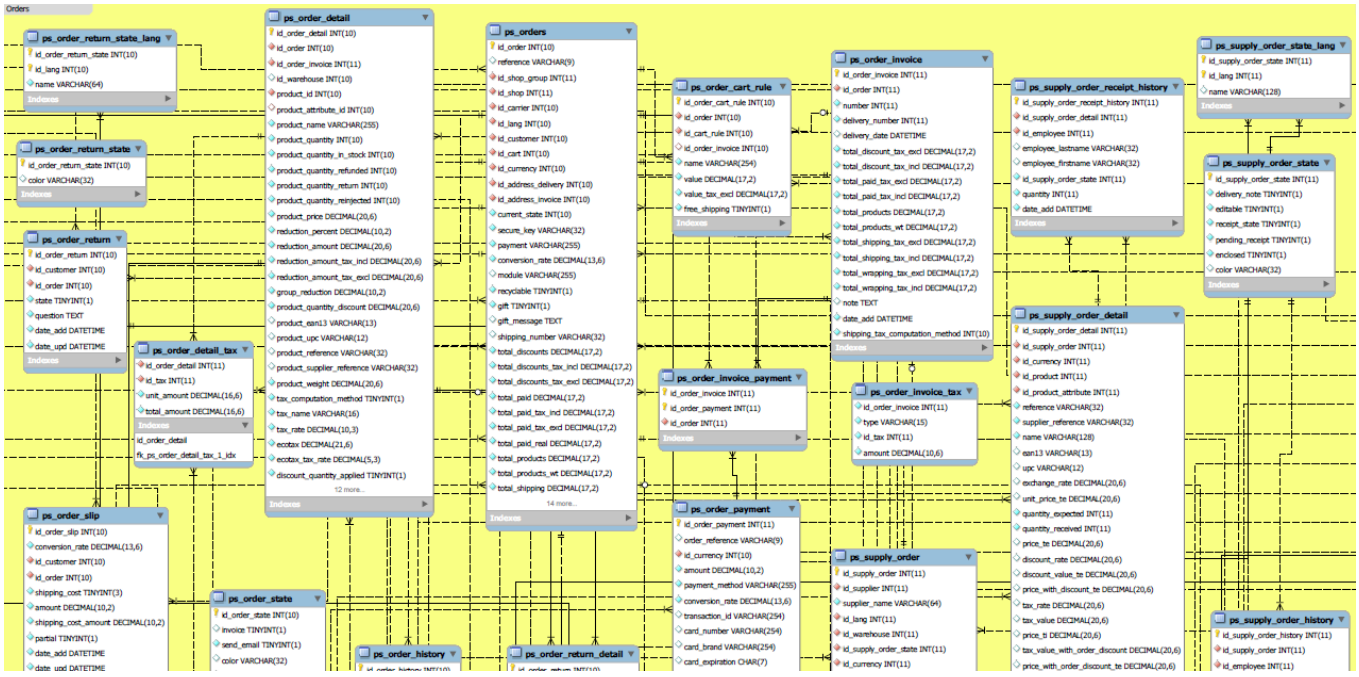

Figura 70: Particolare dello schema, tabelle relative agli ordini

Il nome delle tabelle del database inizia con il prefisso di PrestaShop (es. ps_product), impostabile durante l'installazione. Nelle query bisogna usare "_DB_PREFIX_" come prefisso.

Es. ... `FROM` `'. _DB_PREFIX_.'customer` ...

Il nome è sempre singolare ed è lo stesso dell'oggetto che riflette. Una tabella che contiene delle traduzioni ha lo stesso nome della tabella dell'oggetto con l'aggiunta del suffisso "_lang" (es. ps_product_lang).
Quando una tabella stabilisce una relazione tra due entità, il nome di entrambe le entità è menzionato nel nome della tabella (es. ps_category_product).

Nome della tabella	Descrizione	Primary Key
ps_access	Permessi dei profili di coloro che possono accedere al back-office.	id_profile, id_tab
ps_accessory	Accessori dei prodotti.	id_product_1, id_product_2
ps_address_format	Formato degli indirizzi per le diverse nazioni.	id_country
ps_attachment	Allegati dei prodotti.	id_attachment
ps_attachment_lang	Nomi e descrizioni degli allegati ai prodotti.	id_attachment, id_lang
ps_attribute	Attributi dei prodotti.	id_attribute
ps_attribute_lang	Nomi degli attributi dei prodotti.	id_attribute, id_lang
ps_carrier	Corrieri.	id_carrier
ps_carrier_lang	Descrizione dei corrieri.	id_carrier
ps_cart	Carrelli dei clienti.	id_cart
ps_cart_product	Prodotti nei carrelli.	id_cart, id_product, id_product_attribute
ps_category	Categorie.	id_category
ps_category_lang	Nomi e descrizioni delle categorie.	id_category, id_lang
ps_configuration	Impostazioni di configurazione.	id_configuration
ps_configuration_lang	Traduzioni delle impostazioni di configurazione.	id_configuration, id_lang

Nome della tabella	Descrizione	Primary Key
ps_connections	Indirizzo IP e pagine visitate dai visitatori.	id_connections
ps_connections_page	Tempi di inizio e fine per le pagine visitate.	id_connections, id_page, time_start
ps_country	Nazioni.	id_country
ps_country_lang	Nomi delle nazioni.	id_country, id_lang
ps_currency	Valute.	id_currency
ps_customer	Utenti registrati.	id_customer
ps_customer_group	Gruppi associati ad ogni utente registrato.	id_customer, id_group
ps_customer_message	Messaggi degli utenti tramite l'apposito form.	id_customer_message
ps_delivery	Prezzi dei corrieri.	id_delivery
ps_discount	Sconti.	id_discount
ps_discount_category	Restrizioni degli sconti per determinate categorie.	id_category, id_discount
ps_discount_lang	Descrizioni degli sconti.	id_discount, id_lang
ps_employee	Utenti che possono accedere al back-office.	id_employee
ps_feature	Caratteristiche dei prodotti.	id_feature
ps_feature_lang	Nomi delle caratteristiche dei prodotti.	id_feature, id_lang
ps_group	Gruppi e riduzioni per gli utenti.	id_group
ps_group_lang	Nomi dei gruppi di utenti.	id_group, id_lang
ps_group_reduction	Riduzioni per i gruppi di utenti.	id_group_reduction
ps_guest	Utenti non registrati.	id_guest
ps_help_access	Articoli di aiuto visti e relative versioni.	id_help_access
ps_hook	Nomi degli hooks e descrizioni.	id_hook

Nome della tabella	Descrizione	Primary Key
ps_hook_module	Posizione dei moduli in ogni hook.	id_module, id_hook
ps_image	Immagini dei prodotti.	id_image
ps_image_lang	Didascalie delle immagini dei prodotti.	id_image, id_lang
ps_image_type	Dimensioni delle immagini dei prodotti.	id_image_type
ps_lang	Lingue supportate.	id_lang
ps_log	Log degli errori PHP e warnings.	id_log
ps_meta	Nomi delle pagine.	id_meta
ps_module	Cartelle dei moduli.	id_module
ps_module_country	Restrizioni dei moduli in base alle nazioni.	id_module, id_country
ps_operating_system	Sistemi operativi riconosciuti.	id_operating_system
ps_orders	Ordini.	id_order
ps_order_detail	Prodotti e dettagli degli ordini.	id_order_detail
ps_order_discount	Sconti degli ordini.	id_order_discount
ps_order_history	Cronologia degli ordini.	id_order_history
ps_pack	Pacchetti di prodotti.	id_product_pack, id_product_item
ps_page	Pagine.	id_page
ps_pagenotfound	Informazioni sulla pagina “404 Not Found”.	id_pagenotfound
ps_payment_cc	Informazioni di carte di credito.	id_payment_cc
ps_product	Prodotti.	id_product
ps_product_attribute	Attributi dei prodotti.	id_product_attribute
ps_product_download	Prodotti scaricabili dal sito.	id_product_download
ps_product_lang	Nomi e descrizioni dei prodotti.	id_product, id_lang
ps_product_tag	Tag dei prodotti.	id_product, id_tag

Nome della tabella	Descrizione	Primary Key
ps_profile	Profili che possono accedere al back-office.	id_profile
ps_profile_lang	Nomi dei profili che possono accedere al back-office.	id_profile, id_lang
ps_search_engine	Motori di ricerca riconosciuti.	id_search_engine
ps_search_index	Indice delle parole chiave del motore di ricerca di PrestaShop.	id_product, id_word
ps_search_word	Parole chiave del motore di ricerca di PrestaShop.	id_word
ps_sekeyword	Parole chiave di motori di ricerca esterni.	id_sekeyword
ps_state	Regioni.	id_state
ps_tag	Tag dei prodotti.	id_tag
ps_tax	Tasse.	id_tax
ps_tax_lang	Nomi delle tasse.	id_tax, id_lang
ps_timezone	Fusi orari.	id_timezone
ps_web_browser	Browser riconosciuti.	id_web_browser
ps_zone	Zone dei corrieri.	id_zone

Tabella 4: Principali tabelle del database di PrestaShop

2.6.2 Reverse engineering sullo schema

Una cosa interessante potrebbe essere risalire al modello concettuale da cui è stato ricavato il Physical Data Model del database di PrestaShop, di cui sopra sono riportate alcune parti.
Ho estratto dallo schema alcune tabelle significative, che ho usato come esempio per la reverse engineering.

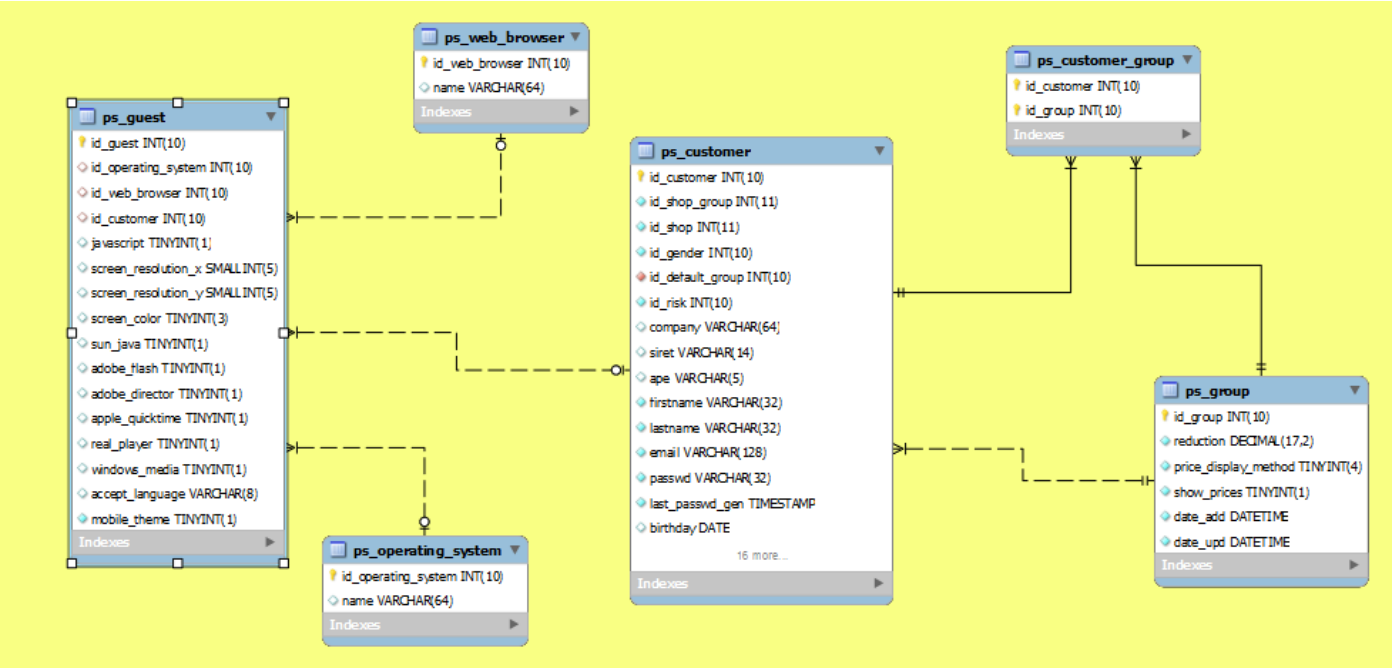

Figura 81: Parte di PSD usata per la reverse engineering

A partire da queste tabelle del Physical Data Model, ho ricostruito il corrispondente modello E/R.
Il modello entity-relationship è un modello per la progettazione concettuale dei dati ad un alto livello di astrazione, formalizzato dal prof. Peter Chen nel 1976 [9]. Viene spesso utilizzato in una prima fase della progettazione di una base di dati, in cui bisogna tradurre le informazioni ricavate dall'analisi di un determinato dominio in uno schema concettuale.
I principali costrutti del modello sono:

- **Entità**, rappresentano classi di oggetti con proprietà comuni ed esistenza autonoma. Nello schema sono rappresentate graficamente con dei rettangoli con i nomi delle entità all'interno.
- **Associazioni,** dette anche relazioni, rappresentano un legame tra due o più entità. Il grado dell'associazione è dato dal numero di entità legate. Sono rappresentate graficamente con dei rombi contenenti i nomi delle associazioni.
- **Attributi,** descrivono le entità e le associazioni. La scelta degli attributi riflette il livello di dettaglio con il quale vogliamo rappresentare le informazioni sulle entità o sulle associazioni. Sono rappresentati graficamente con degli ellissi al cui interno sono specificati i nomi degli attributi o anche semplicemente indicandone solo i nomi. [4]

Per ciascuna entità o associazione si definisce una chiave, cioè un'insieme minimale di attributi che identifica univocamente un'istanza di entità o associazione.
Le cardinalità delle associazioni vengono specificate per ciascuna entità che partecipa ad un'associazione e dicono quante volte un'occorrenza di una di queste entità può essere legata ad occorrenze delle altre entità coinvolte (minimo e massimo delle occorrenze).
Di seguito è riportato il modello E/R che ho ricostruito.

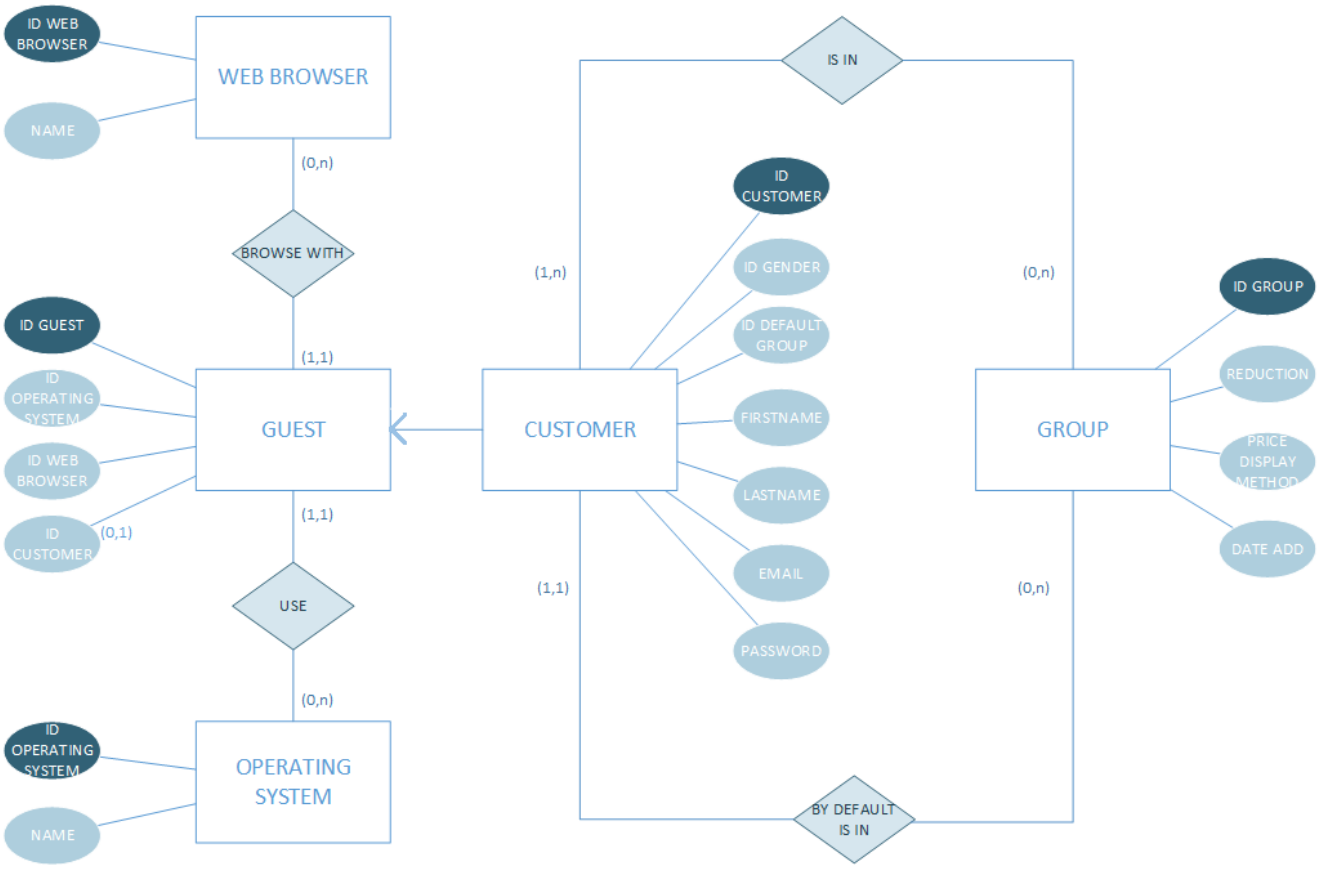

Figura 9: Modello E/R ricostruito dal PDM

Come si può vedere dalle immagini, le tabelle del database sono diventate delle entità, tranne la tabella ps_customer_group che è stata tradotta in un'associazione molti-a-molti. Nello schema E/R ho riportato alcuni attributi principali, le chiavi primarie sono state evidenziate.
Inoltre non sono più presenti dettagli implementativi, come i nomi esatti di tabelle e colonne, i tipi di dato e l'occupazione di memoria.
L'entità Guest rappresenta una generalizzazione dell'entità Customer, la generalizzazione è di tipo parziale sovrapposta.

L'attributo Id Customer dell'entità Guest può quindi non essere specificato, come si osserva dalla relativa cardinalità.
Un'ulteriore differenza tra i due schemi è che le cardinalità delle relazioni sono invertite.

2.6.3 Classe DBQuery

La classe DBQuery è un query builder che aiuta nella creazione di query SQL.

Es.
```
$sql = new DbQuery();
$sql->select('*');
$sql->from('cms', 'c');
$sql->innerJoin('cms_lang', 'l', 'c.id_cms = l.id_cms AND l.id_lang
= '.(int)$id_lang);
$sql->where('c.active = 1');
$sql->orderBy('position');
return Db::getInstance()->executeS($sql);
```

Alcuni metodi di questa classe sono:

Metodo e parametri	Descrizione
__toString()	Genera la query.
build()	Genera la query (ritorna una stringa).
from(string $table, mixed $alias = null)	Imposta le tabelle per una clausola FROM.
groupBy(string $fields)	Aggiunge una condizione GROUP BY.
having(string $restriction)	Aggiunge una condizione con clausola HAVING, ogni condizione viene separata da un AND.
innerJoin(string $table, string $alias = null, string $on = null)	Aggiunge una clausola INNER JOIN.
join(string $join)	Aggiunge una clausola JOIN.
leftJoin(string $table, string $alias = null, string $on = null)	Aggiunge una clausola LEFT JOIN.
limit(string $limit, mixed $offset = 0)	Limita i risultati di una query.
naturalJoin(string $table, string $alias =	Aggiunge una clausola NATURAL JOIN.

Metodo e parametri	Descrizione
null)	
orderBy(string $fields)	Ordina i risultati di una query, ORDER BY.
select(string $fields)	Aggiunge dei campi ad una SELECT.
where(string $restriction)	Aggiunge una condizione con clausola WHERE, ogni condizione viene separata da un AND.

Tabella 5: Metodi della classe DBQuery

2.6.4 Classe ObjectModel

Questo è l'oggetto principale del modello ad oggetti di PrestaShop. Può essere soggetto ad overriding, con molta precauzione.
È una classe di tipo Active Record. Le tabelle e gli attributi del database di PrestaShop sono incapsulati in questa classe, quindi essa è legata ad un record nel database. Ogni volta che viene istanziato un oggetto, un nuovo record viene aggiunto al database. Ogni oggetto prende i dati dal database e, quando viene aggiornato, si aggiorna anche il record del database al quale è legato. La classe implementa l'accesso ad ogni attributo.
Per definire il model si usa la variabile $definition.

```
Es. public static $definition = array(
        'table' => 'cms',
        'primary' => 'id_cms',
        'multilang' => true,
        'fields' => array(
           'id_cms_category' => array('type' => self::TYPE_INT,
'validate' => 'isUnsignedInt'),
           'position' => array('type' => self::TYPE_INT),
           'active' => array('type' => self::TYPE_BOOL),

           // Language fields
           […]
           ),
       );
```

Un'overriding sui metodi di questa classe influenza il comportamento di tutte le altre classi e metodi.

Alcuni metodi di questa classe sono:

Metodo e parametri	Descrizione
__construct($id = NULL, $id_lang = NULL)	Costruisce l'oggetto.
add($autodate = true, $nullValues = false)	Salva l'oggetto corrente nel database (aggiunge o aggiorna).
associateTo(integer|array $id_shops)	Associa un elemento al suo contesto.
delete()	Cancella l'oggetto corrente dal database database.
deleteImage(mixed $force_delete = false)	Cancella le immagini associate con l'oggetto.
deleteSelection($selection)	Cancella vari oggetti dal database.
save($nullValues = false, $autodate = true)	Salva l'oggetto corrente nel database (aggiunge o aggiorna).
toggleStatus()	Cambia lo stato di un oggetto nel database.
update($nullValues = false)	Aggiorna l'oggetto corrente nel database.
validateFields($die = true, $errorReturn = false)	Controlla la validità di un campo prima di iniziare l'interazione con il database.

Tabella 6: Metodi della classe ObjectModel

2.6.5 Accesso al database

Il più delle volte, creare un modulo o estendere PrestaShop significa usare o inserire dati nel database. Conoscere il funzionamento della classe DB è essenziale per il developer.
La classe DB è in realtà formata da due classi:

- la classe astratta Db, che si può trovare in /classes/db/Db.php;
- una sottoclasse che estende Db, tre astrazioni sono supportate come sottoclassi, MySQL, MySQLi e PDO.

DB è uno pseudo-singleton, dal momento che può essere istanziato manualmente, poichè il suo costruttore è public.

Il singleton è un design pattern creazionale che ha lo scopo di garantire che di una determinata classe venga creata una e una sola istanza, e di fornire un punto di accesso globale a tale istanza.

Il metodo insert() è stato creato per generare automaticamente l'inserimento nel database. Bisognerebbe preferirlo all'INSERT con una query, a meno che la query non sia complessa (con query innestate o funzioni SQL).
Eseguendo il seguente codice

```
$target = Tools::getValue('id');
$name = Tools::getValue('name');
Db::getInstance()->insert('target_table', array(
    'id_target' => (int)$target,
    'name'      => pSQL($name),
));
```

si genera la seguente query SQL

```
INSERT INTO `prefix_target_table` (`id_target`, `name`) VALUES (10,
'myName').
```

Bisogna essere sicuri che I dati siano sempre controllati e protetti quando si inseriscono nel database, per questo nel precedente codice sono stati usati un casting esplicito e la funzione pSQL(), la quale protegge da attacchi SQL injection.
Il metodo update() lavora come insert(), ma per l'aggiornamento dei dati (query UPDATE), stessa cosa per delete().
Il metodo delete() viene anche usato dal sistema di cache di PrestaShop per eliminare dalla cache le query interessate.
Il metodo execute() esegue la query SQL passatagli come parametro, esso ritorna un valore booleano e non una risorsa database che può essere usata. Questo metodo cancella anche la cache usata quindi dovrebbe essere usato per query "di scrittura" (INSERT, UPDATE…).
Simile funzionamento ha il metodo query(), con le seguenti differenze :

- non gestisce la cache;
- ritorna una risorsa database che può essere usata da altri metodi della classe DB.

Questo metodo è sua volta usato, come metodo di basso livello, da tutti gli altri metodi che usano query SQL.
Il metodo executeS() esegue la query passatagli e rende i dati risultati disponibili attraverso un array multidimensionale. Viene usato per query "di lettura" (SELECT, SHOW…) e il risultato viene messo in cache.

Con getRow() viene visualizzata solo la prima riga del risultato della query da eseguire, il metodo aggiunge automaticamente una clausola LIMIT alla query (che quindi in origine ne doveva essere priva); fa uso di cache.
Simile è il metodo getValue(), che recupera il primo valore della prima riga del risultato della query. getValue() non protegge il codice da eventuali attacchi, i dati devono essere messi in sicurezza separatamente.

Capitolo 3
Benchmark dei moduli multi-vendor

3.1 RICERCA E CONFRONTO MODULI

Per questo progetto si è scelto di procedere partendo da eventuali moduli già presenti sul mercato piuttosto che partire da zero.
Questo approccio è stato scelto con l'intento di risparmiare risorse temporali, finanziarie e umane cercando di ottenere comunque un risultato soddisfacente.
Il modulo da ricercare avrebbe dovuto modificare il normale funzionamento di PrestaShop, permettendo a una determinata categoria di utenti registrati di poter inserire i propri prodotti e di poterne gestire l'intero processo di vendita-spedizione.
In questo modo il singolo shop si trasforma in una vetrina sulla quale vari venditori possono agire in modo concorrenziale.
Mi è stata fornita un'indicazione riguardo ad un modulo per PrestaShop, Agile Multiple Seller, su cui l'azienda aveva già messo gli occhi, seppur non avendone approfondito la conoscenza.

La mia ricerca è iniziata osservando altri siti operanti con modalità simili a quelle ricercate dalla mia azienda e osservendone la struttura per quanto riguardava la parte di interesse (cioè la gestione dei vari venditori).

Figura 10: Logo Agile

Ho proseguito con una ricerca generica in rete di eventuali soluzioni applicabili al CMS scelto per la realizzazione del sito.

La ricerca mi ha portato ad identificare due possibili alternative, in quanto molto ricorrenti come soluzioni per un marketplace. La prima alternativa era il modulo che già mi era stato indicato, Agile Multiple Seller, prodotto e venduto da Agile; la seconda alternativa era il modulo PrestaShop Advanced Marketplace, prodotto e venduto da Webkul.

Durante la ricerca mi sono imbattuto anche in altri moduli che però soddisfavano solo in parte le caratteristiche richieste e che quindi sono stati posti in secondo piano in quanto sarebbero risultati più dei compromessi che delle soluzioni.

Figura 14: Logo WebKul

Gli sviluppatori di entrambi i moduli hanno presentato e descritto il loro prodotto sul forum di PrestaShop, inoltre altre informazioni sono reperibili sui relativi siti. Sono partito da queste informazioni per poi analizzare e comparare i due moduli, con la finalità di individuare quello che più si sarebbe adattato alle esigenze aziendali.

Ho analizzato i moduli sotto vari aspetti: in modo generico, dal punto di vista dei venditori, dal punto di vista dell'admin del multistore e infine riguardo ai pagamenti.

3.1.1 Confronto generico

	Agile Multiple Seller
Prezzo	144 $
Compatibilità versione Prestashop 1.6	Sì
Qualità supporto	Reviews molto positive (1 mese gratis)
Installazione	Standard
Licenza	Dominio singolo (più test servers), codice parzialmente offuscato (parti in PHP), 5 upgrade gratis in un anno
Manuale	No
	Prestashop Advanced Marketplace
Prezzo	199 $
Compatibilità versione Prestashop 1.6	Sì
Qualità supporto	Reviews molto positive (3 mesi gratis)
Installazione	Standard

Licenza	Dominio singolo, codice non offuscato(possibili variazioni sul supporto)
Manuale	Sì

Tabella 7: Benchmark moduli, parte generale

La compatibilità con la versione del CMS usato per il sito è rispettata da tutti e due i moduli, e in tutti e due i casi non sono richieste, per l'installazione del modulo, azioni aggiuntive alla procedura guidata implementata in PrestaShop. Le opinioni degli utenti della community di PrestaShop riguardo al supporto e all'assistenza in caso di problemi con i moduli sono risultate essere decisamente positive (e io stesso ho avuto modo di provarlo in fase di testing nel caso del modulo Agile).

3.1.2 Confronto gestione venditore

	Agile Multiple Seller
Modalità per diventare seller	Iscrizione front-office | Registrazione da parte dell'admin
Lato gestione seller	Back-office, front-office
Gestione prodotti	Sì
Gestione ordini	Sì
Categorie private	Sì
Usare categorie pubbliche	Sì
Gestione spedizioni e corrieri	Modulo aggiuntivo (81 $)
Friendly URL	Sì
HTML editor per info seller	Sì
Importare dati tramite file CSV	Sì
Comunicazioni seller-customer	Modulo aggiuntivo (45 $)

	Prestashop Advanced Marketplace
Modalità per diventare seller	Richiesta all'admin (integrata nel modulo)
Lato gestione seller	Front-office modificato
Gestione prodotti	Sì
Gestione ordini	Sì
Categorie private	Sì
Usare categorie pubbliche	Sì
Gestione spedizioni e corrieri	Modulo aggiuntivo (149 $)
Friendly URL	Sì
HTML editor per info seller	No

Importare dati tramite file CSV	No
Comunicazioni seller-customer	Non pervenuto

Tabella 8: Benchmark moduli, parte venditore

Nel caso di Agile Multiple Seller, un utente può diventare venditore attraverso una richiesta all'admin del sito (e relativa approvazione) oppure, se l'opzione è abilitata, semplicemente tramite l'accettazione dei relativi termini e condizioni. Il venditore può gestire la propria sezione sia da front-end che da back-end.
Per quanto riguarda Advanced PrestaShop Marketplace invece l'utente può diventare venditore solo tramite richiesta all'admin e i venditori non possono accedere da back-end.
Per quanto riguarda la gestione delle spedizioni interamente da parte del venditore, in tutti e due i casi è necessario un modulo aggiuntivo.
Il modulo Agile Multiple Seller, a differenza dell'altro, permette inoltre al venditore di importare eventuali dati che aveva in un database precedente.

3.1.3 Confronto gestione amministratore

	Agile Multiple Seller
Controllo sul seller	Totale
Gestione prodotti seller	Sì
Gestione categorie seller	Sì
Gestione categorie pubbliche	Sì
Gestione ordini seller	Sì
Gestione corrieri e spedizioni pubbliche	Sì
Lato gestione admin	Back-office
Esclusione di alcuni moduli dal seller	Sì
Gestione permessi seller	Sì
	Prestashop Advanced Marketplace
Controllo sul seller	Totale
Gestione prodotti seller	Sì
Gestione categorie seller	Sì
Gestione categorie pubbliche	Sì
Gestione ordini seller	Sì
Gestione corrieri e spedizioni pubbliche	Sì
Lato gestione admin	Back-office
Esclusione di alcuni moduli dal seller	No
Gestione permessi seller	No

Tabella 9: Benchmark moduli, parte amministratore

L’admin ha in ogni caso il pieno controllo su tutto ciò che avviene nello store, può creare, modificare ed eliminare prodotti, categorie, ordini (questi non sono eliminabili per questioni legali), corrieri; sia elementi pubblici dello shop che privati dei venditori.
L’amministratore gestisce tutto dal back-office.
Con il modulo Agile Multiple Seller l’admin può anche are in modo che alcuni determinati moduli possano essere interdetti all’utilizzo dei venditori. Inoltre può controllare totalmente i permessi del profilo venditore, personalizzando temporaneamente o permanentemente le azioni eseguibili e le schermate visibili dai venditori.

3.1.4 Confronto modalità di pagamento

	Agile Multiple Seller
Modalità di raccolta soldi	Raccolti dall'admin | Raccolti dal seller (moduli aggiuntivi) | Divisi tra seller e admin (modulo aggiuntivo)
Commissioni	Modulo aggiuntivo (74.80 $)
Carrello	Prodotti seller multipli |Prodotti seller singolo
	Prestashop Advanced Marketplace
Modalità di raccolta soldi	Raccolti dall'admin | Divisi tra admin e seller (modulo aggiuntivo 149 $)
Commissioni	Globali | Per seller, calcolo automatico
Carrello	Prodotti seller multipli |Prodotti seller singolo

Tabella 10: Benchmark moduli, parte pagamenti

Sul lato della raccolta dei pagamenti il modulo Agile permette una scelta tra tre modalità, permettendo di decidere il soggetto a cui vanno, interamente o parzialemente, i soldi.
Comunque, da specifiche di progetto, i ricavi sono raccolti dal gestore del multistore, quindi passano in secondo piano le differenze tra i due moduli sulla gestione delle commissioni.

I due moduli sono risultati comparabili riguardo svariate caratteristiche.
Entrambi soddisfano gli obiettivi iniziali della ricerca, inoltre entrambi sono estendibili con ulteriori moduli accessori (sviluppati dagli stessi sviluppatori del modulo principale) all’occorrenza.

Il modulo Agile Multiple Seller ha però alcune funzioni aggiuntive rispetto al modulo Advanced PrestaShop Marketplace ed inoltre era già stato individuato in precedenza dall'azienda.
Questi motivi hanno fatto sì che una scelta provvisoria ricadesse appunto su Agile Multiple Seller.
Per maggiori informazioni sui moduli confrontati vedere i rispettivi siti [10] e [11].

3.2 TEST DEMO MODULO AGILE, ACQUISTO, TEST MODULI AGILE

Una volta individuato il modulo adatto, ho richiesto di poterlo provare sui server dell'azienda produttrice per confermare effettivamente che tutte le specifiche richieste erano soddisfatte e che il modulo funzionasse correttamente.
L'azienda Agile ha a disposizione vari server sui quali girano vari siti che possono essere messi a disposizione di eventuali compratori per una prima prova dei moduli da loro prodotti.
Sul sito messo a mia disposizione ho allestito un piccolo sistema di prova per cercare di testare il modulo in tutte le sue funzionalità.
Il test prevedeva:

- 1 admin, 2 venditori, 2 clienti registrati;
- 1 categoria pubblica, 2 categorie private (una per venditore);
- 2 prodotti per venditore (uno nella categoria pubblica, l'altro nella privata), 1 prodotto pubblico;
- Un venditore inserisce i propri dati e prodotti, l'admin inserisce dati e prodotti dell'altro;
- Un cliente compra 3 prodotti (uno dalla categoria privata di un seller, uno dell'altro seller dalla categoria pubblica e il prodotto pubblico);
- L'altro cliente serviva solo per vedere cosa cambiava associandolo o meno ad un particolare venditore.

Ho appurato il corretto funzionamento del sistema in tutte le sue parti e sotto il punto di vista di ogni agente. Particolare attenzione è stata prestata alla visuale del venditore sulle varie sezioni da lui gestite e sull'indipendenza del modulo principale dai moduli accessori.

Appurata così la completa idoneità del modulo Agile Multiple Seller, si è deciso di rendere definitiva la scelta e di procedere analizzando le soluzioni di acquisto dello stesso.

L’azienda Agile prevede degli sconti sull’acquisto al superamento di determinate soglie di prezzo.
Mi è stato chiesto di analizzare i moduli accessori e i vari costi per valutarne l’effettiva utilità e poter arrivare così ad alcune alternative di acquisto per massimizzare il risparmio e l’utilità dei moduli acquistati.

Modulo	Prezzo
Agile Multiple Seller	144.00 $
Agile Seller Shipping	81.00 $
Agile Shipping Estimation	30.00 $
Agile Seller Commission	74.80 $
Agile Seller Messenger	45.00 $
Agile Seller Ratings	55.25 $
Agile Seller List Options	68.00 $

Tabella 11: Prezzi moduli Agile

Totale acquisti	Sconto
250 $	15%
350 $	20%
550 $	25%
850 $	30%
1000 $	35%

Tabella 12: Sconti disponibili

Alternativa	Prezzo scontato
Multi Seller + Seller Shipping	225 $
Multi Seller + Seller Shipping + Shipping Estimation	216.75 $
Multi Seller + Seller Shipping + Seller Messenger	229.5 $

Tabella 13: Alternative di acquisto

La scelta finale è stata quella di acquistare Agile Multiple Seller insieme con i moduli Agile Seller Shipping e Agile Seller Messenger.
Il modulo Agile Seller Messenger non è di vitale importanza per la realizzazione del progetto ma è stato selezionato per poter superare la prima soglia di prezzo ed accedere così al primo sconto.
Non è stata scelta l’alternativa con il modulo Agile Shipping Estimation in quanto, nonostante il minor prezzo della soluzione, la compatibilià con PrestaShop di quel modulo era garantita solo fino ad una versione di quattro anni più vecchia rispetto a quella usata per il progetto.

Una volta acquistati i moduli, sono stati caricati sul server, installati e configurati. È seguita una seconda fase di testing sui server della mia azienda, durante la quale ho provato tutti e tre i moduli acquistati.
Ho usato un sistema di prova similare a quello testato in precedenza:

- 1 admin, 2 venditori, 1 cliente registrato;
- 1 categoria pubblica, 2 categorie private (una per venditore);
- 2 prodotti per venditore (uno nella categoria pubblica, l'altro nella privata);
- 2 corrieri (uno per venditore);
- Un venditore inserisce i propri dati e prodotti, l'admin inserisce dati e prodotti dell'altro;
- Un cliente compra 2 prodotti (uno dalla categoria privata di un seller, uno dell'altro seller dalla categoria pubblica);
- Test del funzionamento della messaggistica tra venditore e cliente.

Durante i test è emerso un problema (non di grande importanza per le specifiche del progetto) non rilevato sul server demo. Accedendo dal back-office, un venditore poteva inserire i propri prodotti oltre che nelle categorie pubbliche e nelle sue categorie private, anche in categorie private di altri venditori.
Dopo vari contatti con l'assistenza Agile il problema è stato arginato facendo in modo che l'accesso al back-office da parte dei venditori potesse essere attivato e disattivato a piacere da parte dell'admin.

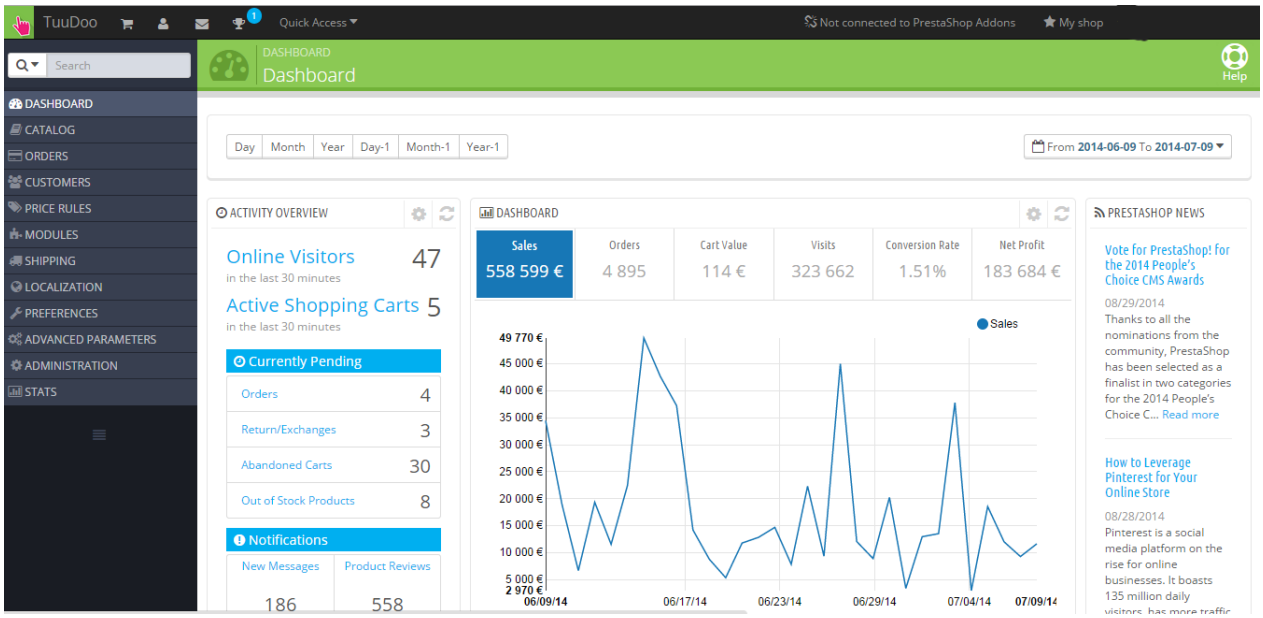

Figura 15: Schermata del back-office

Capitolo 4
Sviluppo modulo Sellers Block

4.1 STRUTTURA MODULI PRESTASHOP

L'estensibilità di PrestaShop ruota intorno ai moduli, che sono piccoli programmi che fanno uso di funzionalità di PrestaShop e le cambiano o ne aggiungono altre al fine di rendere PrestaShop di più facile utilizzo o più personalizzato.
Un modulo consiste di un file PHP principale, insieme ad altri file PHP se servono, e di tutte le immagini e file di template (.tpl) necessari per visualizzare l'informazione.
Qualsiasi modulo, una volta installato su uno shop online, può interagire con uno o più hooks. Gli hooks permettono di agganciare il codice alla view corrente al momento del parsing (es. quando si visualizza il carrello). Nello specifico, un hook è una scorciatoia a vari metodi disponibili dall'oggetto Module assegnati allo stesso hook.
I moduli possono far visualizzare una grande varietà di contenuti (blocchi, testo…), svolgere molti compiti, interfacciarsi con altri strumenti e molto altro ancora.
I moduli possono essere resi configurabili, più configurabile è un modulo, più facile è il suo utilizzo, e soddisferà i bisogni di un range maggiore di utenti.
Uno dei principali vantaggi del sistema modulare è quello di poter aggiungere funzionalità a PrestaShop senza necessità di modificare i file di base, e quindi di

rendere più facili gli upgrade in quanto non c'è bisogno di trasferire anche i cambiamenti.
Sia i moduli di default di PrestaShop che i moduli di terze parti si trovano nella cartella /modules. Alcuni moduli possono anche fare parte di un tema e in questo caso vanno cercati nella cartella del tema in questione.

Nome del file/cartella	Descrizione	Dettagli
name_of_the_module.php	File principale.	Il file PHP principale dovrebbe avere lo stesso nome della cartella del modulo. Per esempio, per il modulo Sellers Block: Nome cartella: /modules/sellersblock Nome file principale: /modules/ sellersblock / sellersblock.php
config.xml	File di configurazione della cache.	Questo file viene generato automaticamente da PrestaShop dopo la prima installazione del modulo.
logo.png	Icona che rappresenta il modulo nel back-office.	File PNG di dimensioni in pixels 32x32.
/views	Contiene i file per le view.	
/views/templates	Contiene i file di template del modulo (.tpl).	Se il modulo deve funzionare con versioni PrestaShop 1.4 o precedenti i template vanno messi nella cartella /templates, nella cartella principale del modulo.
/views/templates/admin	Sottocartella per i template usati dai controller per l'amministrazione.	
/views/templates/front	Sottocartella per i template usati dai controller per il front-office.	
/views/templates/hook	Sottocartella per i template usati dagli hooks.	

Nome del file/cartella	Descrizione	Dettagli
/views/templates/css	Sottocartella per i file CSS.	Se il modulo deve funzionare con versioni PrestaShop 1.4 o precedenti i file CSS vanno messi nella cartella /css, nella cartella principale del modulo.
/views/templates/js	Sottocartella per i file JavaScript.	Se il modulo deve funzionare con versioni PrestaShop 1.4 o precedenti i file JavaScript vanno messi nella cartella /js, nella cartella principale del modulo.
/views/templates/img	Sottocartella che contiene le immagini.	Se il modulo deve funzionare con versioni PrestaShop 1.4 o precedenti le immagini vanno messe nella cartella /img, nella cartella principale del modulo.
/controllers	Questa cartella contiene i controller.	Le sottocartelle di /controllers sono le stesse di /view/templates.
/override	Cartella per il codice che estende o modifica le classi di base.	Questo è molto utile se si presenta la necessità di modificare la classi di default di PrestaShop.
/translations	Cartella per i file di traduzione.	fr.php, en.php, es.php, etc.
/themes/[theme_name]/modules	Cartella per sovrascrivere i .tpl, se necessario.	
/upgrade	Cartella per i file di upgrade.	

Tabella 14: File structure di un modulo PrestaShop

Del precedente elenco, solo i primi tre file sono necessari per un modulo base: il file principale, il file di configurazione cache (generato automaticamente) e l'icona.
Gli altri file possono essere usati, nel caso servano, ma un modulo può funzionare ugualmente anche senza di essi. [5]

4.2 SVILUPPO MODULO SELLERS BLOCK

Una volta preparato il sito di TuuDoo con PrestaShop e installati i moduli Agile acquistati, la struttura del marketplace era già pronta e il funzionamento era quello desiderato.
Però il modulo Agile Multiple Seller non riflette il cambiamento da store a multistore anche dalla parte del front end, ovvero il cliente finale che vuole acquistare dei prodotti non è in grado di capire che questi prodotti sono inseriti e gestiti da diversi venditori fintanto che non apre la scheda di uno di essi, sulla quale in basso sono visualizzati i dati del venditore.
Quindi non è possibile operare una scelta sulla base del venditore preferito, a meno che non si vogliano aprire una ad una le pagine di tutti i prodotti del sito per vedere quali gli appartengano.

Per risolvere questo problema si è deciso di creare un modulo apposito che visualizzasse una lista di tutti i venditori del sito e che, scegliendone uno in particolare, ne elencasse i prodotti.
Questo modulo, chiamato Sellers Block è una sorta di modulo aggiuntivo al modulo Agile Multiple Seller in quanto fa uso di tabelle e dati del database che sono presenti solo grazie all'installazione di quest'ultimo.

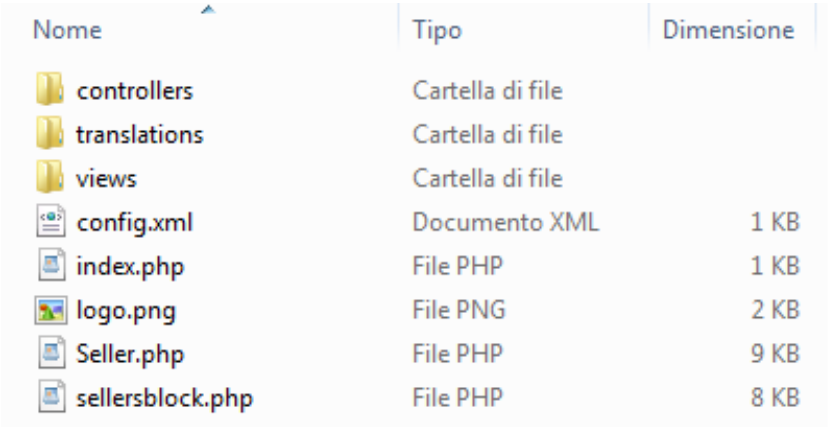

Figura 116: Cartella principale del modulo Sellers Block

4.2.1 sellersblock.php

Per prima cosa, bisogna creare una cartella con il nome del modulo in minuscolo e senza spazi, nel mio caso "sellersblock". In questa cartella va inserito il file PHP principale, chiamato con lo stesso nome della cartella, sellersblock.php.

Andrò ora a descrivere e spiegare le parti principali che formano questo file.

```
if (!defined('_PS_VERSION_'))
    exit;
```

Il file deve iniziare con questa verifica, che controlla l'esistenza della costante _PS_VERSION_. Questa costante è sempre presente se PrestaShop è installato e ne indica la versione. Se la verifica dà esito negativo il modulo non viene caricato. Questa istruzione serve a prevenire a visitatori malintenzionati di caricare direttamente il file.
Il file contiene la classe principale del modulo, chiamata come la cartella però in CamelCase, e deve estendere la classe Module, così da ereditarne metodi e attributi.

```
public function __construct()
{
    $this->name = 'sellersblock';
    $this->tab = 'front_office_features';
    $this->version = '1.0';
    $this->author = 'Antonio Circiello';
    $this->need_instance = 0;
    $this->ps_versions_compliancy = array('min' => '1.5', 'max' =>
_PS_VERSION_);
    $this->bootstrap = true;
    $this->dependencies = array('agilemultipleseller');

    parent::__construct();

    $this->displayName = $this->l('Sellers Block');
    $this->description = $this->l('Displays a block listing product
sellers.');

    $this->confirmUninstall = $this->l('Are you sure you want to
uninstall?');
}
```

Metodo 1: __contruct()

Un costruttore è una funzione di una classe che viene automaticamente chiamata quando si crea una nuova istanza della classe. Nel caso di PrestaShop, il costruttore è il primo metodo chiamato quando viene caricato il modulo.
In questa funzione vengono assegnati alcuni attributi all'istanza: nome (che funge da identificatore interno), tab (la parte di PrestaShop interessata dal modulo), versione…

Segue la chiamata al costruttore della classe padre, che deve essere fatta dopo la creazione della variabile $this->name e prima dell'uso della funzione $this->l() (metodo di PrestaShop per le traduzioni).

```
public function install()
{
   if (Shop::isFeatureActive())
      Shop::setContext(Shop::CONTEXT_ALL);

   Configuration::updateValue('SELLER_DISPLAY_TEXT', true);
   Configuration::updateValue('SELLER_DISPLAY_TEXT_NB', 5);
   Configuration::updateValue('SELLER_DISPLAY_FORM', false);

   $success = (parent::install() && $this->registerHook('header') &&
      $this->registerHook('actionObjectSellerDeleteAfter') &&
      $this->registerHook('actionObjectSellerAddAfter') &&
      $this->registerHook('actionObjectSellerUpdateAfter'));

   if ($success)
   {
      $theme = new Theme(Context::getContext()->shop->id_theme);
      if ((!$theme->default_left_column ||
         !$this->registerHook('leftColumn'))
         && (!$theme->default_right_column ||
         !$this->registerHook('rightColumn')))
      {
         $this->_errors[] = $this->l('This module need to be hooked in a
column and your theme does not implement one');
         parent::uninstall();
         return false;
      }
   }
   return $success;
}
```

Metodo 2: install()

```
public function uninstall()
{
   if (!parent::uninstall())
      return false;

   $result = Configuration::deleteByName('SELLER_DISPLAY_TEXT');
   $result &= Configuration::deleteByName('SELLER_DISPLAY_TEXT_NB');
   $result &= Configuration::deleteByName('SELLER_DISPLAY_FORM');

   return $result;
}
```

Metodo 3: uninstall()

Alcuni moduli necessitano di eseguire delle azioni durante la loro installazione, come controllare delle impostazioni di PrestaShop oppure registrare nel database le proprie impostazioni. Nello stesso modo, le impostazioni cambiate durante

l'installazione devono essere riportate alla normalità con la disintallazione del modulo.
I metodi install() e unistall() rendono possibile controllare ciò che avviene appunto quando l'amministratore dello shop installa o disintalla il modulo.
In questo caso, nel metodo per l'installazione viene controllato che il modulo possa essere agganciato agli hook necessari, cioè quelli delle colonne laterali del sito, l'header e altri appositamente creati nel file stesso per gestire la cache. Se il controllo dà esito negativo non avviene l'installazione del modulo.
Vengono inoltre inserite nel database delle informazioni per poter configurare il modulo, informazioni che devono essere opportunamente eliminate nel metodo per la disintallazione.
Durante l'installazione del modulo, viene aggiunta una riga alla tabella ps_module del database.

I blocchi di codice illustrati finora fanno uso dell'oggetto Configuration. Questo è un oggetto specifico di PrestaShop, ideato per aiutare gli sviluppatori nel gestire le impostazioni dei moduli, immagazzinando le impostazioni nel database di PrestaShop senza bisogno di utilizzare query SQL. I dati gestiti da Configuration si trovano nella tabella ps_configuration.
I metodi principali di questo oggetto sono:

- Configuration::get(), recupera un valore specifico dal database;
- Configuration::getMultiple(), recupera più valori dal database e ritorna un array PHP;
- Configuration::updateValue(), aggiorna una variabile esistente nel database con un nuovo valore o, se la variabile non esiste, la crea con quel valore;
- Configuration::deleteByName(), cancella la variabile dal database.

```
   public function getContent()
   {
      $output = '';
      if (Tools::isSubmit('submit'.$this->name))
      {
         $text_list = (int)(Tools::getValue('SELLER_DISPLAY_TEXT'));
         $text_nb = (int)(Tools::getValue('SELLER_DISPLAY_TEXT_NB'));
         $form_list = (int)(Tools::getValue('SELLER_DISPLAY_FORM'));
         if ($text_list && !Validate::isUnsignedInt($text_nb))
            $errors[] = $this->l('There is an invalid number of
elements.');
         elseif (!$text_list && !$form_list)
            $errors[] = $this->l('Please activate at least one system
list.');
```

```
            else
            {
                Configuration::updateValue('SELLER_DISPLAY_TEXT', $text_list);
                Configuration::updateValue('SELLER_DISPLAY_TEXT_NB', $text_nb);
                Configuration::updateValue('SELLER_DISPLAY_FORM', $form_list);
                $this->_clearCache('sellersblock.tpl');
            }
            if (isset($errors) && count($errors))
                $output .= $this->displayError(implode('<br />', $errors));
            else
                $output .= $this->displayConfirmation($this->l('Settings
updated.'));
        }
        return $output.$this->renderForm();
    }
```

Metodo 4: getContent()

Un modulo può anche avere una pagina di configurazione, visibile dal back-office, in modo che l'utente possa cambiare qualche impostazione. Il link a questa pagina appare aggiungendo il metodo getContent() alla classe principale. Esso è un metodo standard di PrestaShop, la cui sola esistenza manda un messaggio al back-office dichiarando che c'è una pagina di configurazione.
Ciò che è contenuto nel metodo (uno tra i primi ad essere chiamato quando viene caricata la pagina di configurazione) crea effettivamente la pagina.
La variabile $output contiene il codice HTML finale che crea la pagina. In caso di errori durante i controlli eseguiti dal codice la pagina inizierà quindi con un relativo messaggio di errore.
Se i controlli vanno a buon fine significa che è possibile inserire i valori nel database, tramite l'oggetto Configuration.
Alla fine viene usato il metodo renderForm(), da creare appositamente, per aggiungere a $output il contenuto della pagina.

```
    public function renderForm()
    {
        $fields_form = array(
            'form' => array(
                'legend' => array(
                    'title' => $this->l('Settings'),
                    'icon' => 'icon-cogs'),
                'input' => array(
                    array(
                        'type' => 'switch',
                        'label' => $this->l('Use a plain-text list'),
                        'name' => 'SELLER_DISPLAY_TEXT',
                        'desc' => $this->l('Display sellers in a plain-text
list.'),
                        'values' => array( […] ),
                    array(
```

```
                    'type' => 'text',
                    'label' => $this->l('Number of elements to display'),
                    'name' => 'SELLER_DISPLAY_TEXT_NB',
                    'class' => 'fixed-width-xs'),
                array(
                    'type' => 'switch',
                    'label' => $this->l('Use a drop-down list'),
                    'name' => 'SELLER_DISPLAY_FORM',
                    'desc' => $this->l('Display sellers in a drop-down
list.'),
                    'values' => array( […] )
                    )
                ),
                'submit' => array(
                    'title' => $this->l('Save'),)
            ),
        );

      $helper = new HelperForm();
      $helper->show_toolbar = false;
      $helper->table =  $this->table;
      $lang = new Language((int)Configuration::get('PS_LANG_DEFAULT'));
      $helper->default_form_language = $lang->id;
      $helper->allow_employee_form_lang =
Configuration::get('PS_BO_ALLOW_EMPLOYEE_FORM_LANG') ?
Configuration::get('PS_BO_ALLOW_EMPLOYEE_FORM_LANG') : 0;
      $helper->identifier = $this->identifier;
      $helper->submit_action = 'submit'.$this->name;
      $helper->currentIndex = $this->context->link-
>getAdminLink('AdminModules', false).'&configure='.$this-
>name.'&tab_module='.$this->tab.'&module_name='.$this->name;
      $helper->token = Tools::getAdminTokenLite('AdminModules');
      $helper->tpl_vars = array(
         'fields_value' => $this->getConfigFieldsValues(),
         'languages' => $this->context->controller->getLanguages(),
         'id_language' => $this->context->language->id);

      return $helper->generateForm(array($fields_form));
   }
```

Metodo 5: renderForm()

In preparazione alla generazione del form, bisogna costruire un array con vari titoli, textfield e pulsanti.
La variabile $fields_form contiene un array multidimensionale, ogni array contiene la descrizione dettagliata del tag che descrive. A partire da questa variabile, PrestaShop genera il form HTML.
Le classi Helper di PrestaShop permettono di generare degli elementi HTML standard per il back-office o per le pagine di configurazione dei moduli, esse sono:

- **HelperForm**, usato per generare form per oggetti di tipo ObjectModel;

- **HelperOptions**, usato per generare form di configurazione i cui valori vengono messi nella tabella ps_configuration, es. la pagina Preferences del back-office;
- **HelperList**, usato per generare una tabella di elementi, che possono o meno essere oggetti ObjectModel;
- **HelperView**, usato per generare pagine view;
- **HelperHelpAccess**, usato per generare il link alla guida nella toolbar.

La funzione renderForm() usa molti attributi di HelperForm, devono essere settati prima di generare il form stesso a partire da $fields_form.

```
   public function hookLeftColumn($params)
   {
      if (!$this->isCached('sellersblock.tpl', $this->getCacheId()))
      {
         $sellers = Seller::getSellers();
         foreach ($sellers as &$seller)
         {
            $seller['image'] = $this->context->language->iso_code.'-
default';
            if (file_exists(_PS_IMG_DIR_.'as/'.$seller['id_sellerinfo'].'-
'.ImageType::getFormatedName('medium').'.jpg'))
               $seller['image'] = $seller['id_sellerinfo'];
         }

         $this->smarty->assign(array(
            'sellers' => $sellers,
            'text_list' => Configuration::get('SELLER_DISPLAY_TEXT'),
            'text_list_nb' => Configuration::get('SELLER_DISPLAY_TEXT_NB'),
            'form_list' => Configuration::get('SELLER_DISPLAY_FORM'),
            'display_link_seller' =>
               $this->context->link->getModuleLink('sellersblock',
'seller'),
             ));
      }
      return $this->display(__FILE__, 'sellersblock.tpl', $this-
>getCacheId());
   }
```

Metodo 6: hookLeftColumn()

Per fare in modo di visualizzare qualcosa sul front-office bisogna implementare gli hooks necessari, il che è stato fatto nella funzione install().
Aggiungere del codice agli hooks necessita di alcuni metodi, uno per ogni hook:

- **hookLeftColumn(),** aggancia il codice alla colonna destra del sito, nel mio caso usa la funzione getSellers() della classe Seller (che illustrerò in seguito) per avere una lista dei venditori presenti nel database e visualizzarla così nella colonna grazie al file di template sellersblock.tpl;

- **hookRightColumn(),** stessa cosa di hookLeftColumn() ma per la colonna destra;
- **hookHeader(),** aggiunge un link ad un'eventuale file CSS per la visualizzazione del modulo.

Le colonne, così come varie altre parti della struttura di una pagina, sono previste e implementate di default da PrestaShop ed è possibile usarle per agganciarci dei moduli.
Nel metodo hookLeftColumn() viene usato il Context per cambiare una variabile Smarty. La funzione Smarty assign() rende possibile settare i nomi delle variabili del template con i valori inseriti nella tabella ps_configuration del database.
L'hook header non è parte dell'header che si può vedere, ma consente di inserire del codice nel tag <head> del file HTML generato. Ciò serve per aggiungere nel tag <head> della pagina del front-office il corretto tag <link> al file CSS o JavaScript del modulo.
Un modulo può essere trapiantato in vari hooks dal back-office, anche se ciò è inutile se il modulo non implementa i metodi necessari per quegli hooks.
Il codice completo di sellersblock.php si trova nell'appendice.

4.2.2 config.xml

Durante l'installazione del modulo, PrestaShop crea il file config.xml nella cartella del modulo, dove memorizza le informazioni di configurazione.
Questo file rende possibile ottimizzare il caricamento della lista dei moduli nel back-office.
Per il codice di questo file si rimanda all'appendice.
Alcuni dettagli:

- **is_configurable** indica se il modulo ha una pagina di configurazione o no;
- **need_instance** indica se il modulo deve essere istanziato quando viene visualizzato nella lista dei moduli, questo è utile nel caso il modulo debba controllare la configurazione di PrestaShop e visualizzare dei warning di conseguenza;
- **limited_countries** è usato per indicare gli Stati ai quali il modulo è riservato, e quindi limitato solo ad essi.

4.2.3 sellersblock.tpl

Ottenuto l'accesso alla colonna, bisogna visualizzare qualcosa sul front-office.
Le parti visibili di un modulo sono definite in file .tpl posizionati in specifiche cartelle:

- /views/templates/front/, per elementi di front-office;
- /views/templates/admin/, per elementi di back-office;
- /views/templates/hook/, per elementi agganciati a PrestaShop e quindi visualizzabili sia sul front che sul back-office.

Dal momento che il file sellersblock.tpl è richiamato da un hook, va inserito nella cartella del modulo in /views/templates/hook/.
Inoltre ho creato un altro template molto simile e con lo stesso nome ma in modo che si adattasse al tema scelto per TuuDoo quindi questo altro file va posizionato internamente alla cartella del tema (sempre però in un'apposita sottocartella per il modulo); infatti PrestaShop cerca il file prima nella giusta cartella del tema e solo in seguito, se la ricerca ha esito negativo, in quella del modulo generale.
Per visionare il codice vedere l'appendice.

È praticamente normale codice HTML, fatta eccezione per alcune chiamate Smarty:

- { l s='…' mod='…' } serve per registrare la stringa in un apposito pannello di PrestaShop che gestisce le traduzioni;
- {if} {else} {/if} sono istruzioni condizionali di Smarty;
- {$display_link_seller} è una variabile Smarty.

Questo file permette di visualizzare sulla colonna un blocco contenente una lista di tutti i venditori presenti nel marketplace. L'elenco può essere configurato in modo che possa essere visualizzato in un menù a tendina oppure per aumentare/ridurre il numero di venditori mostrati. A seconda di dove si clicca con il mouse si viene riportati ad un pagina del front-office che mostra un'elenco più dettagliato dei venditori oppure ad una pagina specifica per un venditore ed i suoi prodotti.

Figura 17: Parte visibile del modulo Sellers Block nella colonna della pagina

4.2.4 seller.php

Questo file altro non è che un controller front-end, inserito nella cartella /controllers/front/ interna alla cartella del modulo. Un controller di questo tipo deve estendere la classe ModuleFrontController.

I metodi init() e initContent() sono entrambi metodi per inizializzare il contenuto della pagina e richiamano i loro corrispondenti nella classe padre.
Il metodo init() effettua un controllo sui valori passati tramite URL e gestisce i relativi messaggi di errore.
Nel metodo initContent() viene operata una scelta a seconda di ciò che l'utente ha selezionato (lista completa dei venditori o lista dei prodotti di un venditore singolo, ovvero se nell'URL è presente o meno un identificativo, di cui è già stata controllata la validità, per il venditore) e in ognuno dei due casi viene settato e visualizzato il template corretto.
La funzione setTemplate() serve per inglobare il template appositamente creato in una pagina completa, con header, footer e barre laterali.
Per visionare il codice si rimanda all'appendice.

4.2.5 seller.tpl, seller-list.tpl e sellersblock.css

seller-list.tpl è il file di template che visualizza la lista dettagliata dei venditori, ognuno con il relativo logo, numero di prodotti inseriti e link alla pagina dei prodotti.
seller.tpl è il file di template che visualizza tutti prodotti inseriti da un venditore e una descrizione dello stesso, se presente.

Entrambi i file includono altri template del tema, in modo che l'impaginazione, la visualizzazione degli errori e quella dei prodotti siano identiche a quelle delle altre pagine. Per uniformare completamente con il tema i dati mostrati, ho realizzato un apposito file CSSche estendesse le regole del tema ai template creati.
Il codice di questi file si può trovare nell'appendice.
Tutti questi file, dal momento che sono stati ideati per essere graficamente conformi al tema scelto per TuuDoo, vanno posizionati internamente alla cartella del tema.

4.2.6 Seller.php

Ho creato questo file con l'unico scopo di raggruppare tutte le funzioni che vanno a leggere dal database e che spesso sono state usate nei file descritti precedentemente.

```
public static function getSellers($get_nb_products = false, $id_lang = 0,
$p = false, $n = false, $all_group = false, $group_by = false)
    {
        if (!$id_lang)
            $id_lang = (int)Configuration::get('PS_LANG_DEFAULT');
        if (!Group::isFeatureActive())
            $all_group = true;

        $sellers = Db::getInstance(_PS_USE_SQL_SLAVE_)->executeS('
            SELECT si.*, sil.*
            FROM `'. DB_PREFIX .'sellerinfo` si
            '.Shop::addSqlAssociation('sellerinfo', 'si').'
            INNER JOIN `'. DB_PREFIX .'sellerinfo_lang` sil ON
(si.`id_sellerinfo` = sil.`id_sellerinfo` AND sil.`id_lang` =
'.(int)$id_lang.')
            '.($group_by ? ' GROUP BY si.`id_sellerinfo`' : '' ).'
            ORDER BY sil.`company` ASC
            '.($p ? ' LIMIT '.(((int)$p - 1) * (int)$n).','.(int)$n : ''));
        if ($sellers === false)
            return false;

        if ($get_nb_products)
        {
            $sql_groups = '';
            if (!$all_group)
            {
                $groups = FrontController::getCurrentCustomerGroups();
                $sql_groups = (count($groups) ? 'IN ('.implode(',',
$groups).')' : '= 1');
            }
```

```
            foreach ($sellers as $key => $seller)
            {
                $sellers[$key]['nb_products'] =
Db::getInstance(_PS_USE_SQL_SLAVE_)->getValue('
                SELECT COUNT(DISTINCT p.`id_product`)
                FROM `'. DB_PREFIX .'product_owner` p
                '.Shop::addSqlAssociation('product', 'p').'
                INNER JOIN `'. DB_PREFIX .'sellerinfo` s ON (p.`id_owner` =
s.`id_seller`)
                WHERE s.`id_sellerinfo` = '.(int)$seller['id_sellerinfo'].'
                AND product_shop.`visibility` NOT IN ("none")
                '.($all_groups ? '' : ' AND p.`id_product` IN (
                    SELECT cp.`id_product`
                    FROM `'. DB_PREFIX .'category_group` cg
                    LEFT JOIN `'. DB_PREFIX .'category_product` cp ON
(cp.`id_category` = cg.`id_category`)
                    WHERE cg.`id_group` '.$sql_groups.'
                )'));
            }
        }

        $total_sellers = count($sellers);
        $rewrite_settings = (int)Configuration::get('PS_REWRITING_SETTINGS');
        for ($i = 0; $i < $total_sellers; $i++)
            $sellers[$i]['link_rewrite'] = ($rewrite_settings ?
Tools::link_rewrite($sellers[$i]['name']) : 0);
        return $sellers;
    }
```

Metodo 7: getSellers()

Questa funzione ha il compito di eseguire una query sul database in modo da recuperare tutti i venditori (tabella ps_sellerinfo) e le loro informazioni nella lingua corrente dello shop (tabella ps_sellerinfo_lang).
Entrambe queste tabelle sono state create dal modulo Agile Multiple Marketplace per il suo funzionamento. Per combinare i valori delle due tabelle ho usato un'inner join dal momento che ogni valore di ps_sellerinfo ha almeno un riscontro tra i valori di ps_sellerinfo_lang.
Un inner join crea una nuova tabella combinando i valori delle due tabelle di partenza basandosi su una certa regola di confronto. [12]
La query compara ogni riga della tabella ps_sellerinfo con ciascuna riga della tabella ps_sellerinfo_lang cercando di soddisfare la regola di confronto definita. Quando la regola di join viene soddisfatta, i valori di tutte le colonne delle due tabelle vengono combinati in un'unica riga nella costruzione della tabella risultante.
La query prevede inoltre un'ordinamento alfabetico dei venditori.

Se richiesto, tramite un valore booleano passato come parametro, la funzione continua con un'altra query che, per ogni venditore ottenuto dalla query

precedente, ne conta il numero di prodotti inseriti (tenendo conto delle visibilità degli stessi relativamente al cliente che sta visitando il sito). Questa query opera un inner join sulle tabelle ps_product_owner e ps_sellerinfo.
La funzione ritorna l'array multidimensionale risultante dalla prima query.

```
   static protected $cacheCompany = array();
   public static function getCompanyById($id_seller, $id_lang = 0)
   {
      if (!isset(self::$cacheCompany[$id_seller]))
         self::$cacheCompany[$id_seller] =
Db::getInstance(_PS_USE_SQL_SLAVE_)->getValue('
            SELECT `company`
            FROM `'._DB_PREFIX_.'sellerinfo_lang`
            WHERE `id_sellerinfo` = '.(int)$id_seller.'
            AND `id_lang` = '.(int)$id_lang.'
            ');

      return self::$cacheCompany[$id_seller];
   }

   public static function getIdByCompany($company, $id_lang = 0)
   {
      $result = Db::getInstance()->getRow('
         SELECT `id_sellerinfo`
         FROM `'._DB_PREFIX_.'sellerinfo_lang`
         WHERE `company` = \''.pSQL($company).'\'
         AND `id_lang` = '.(int)$id_lang.'
         ');

      if (isset($result['id_sellerinfo']))
         return (int)$result['id_sellerinfo'];

      return false;
   }

   public static function sellerExists($id_seller)
   {
      $row = Db::getInstance()->getRow('
         SELECT `id_sellerinfo`
         FROM '._DB_PREFIX_.'sellerinfo s
         WHERE s.`id_sellerinfo` = '.(int)$id_seller
         );

      return isset($row['id_sellerinfo']);
   }

   static protected $cacheDescription = array();
   public static function getDescriptionById($id_seller, $id_lang = 0)
   {
      if (!isset(self::$cacheDescription[$id_seller]))
         self::$cacheDescription[$id_seller] =
Db::getInstance(_PS_USE_SQL_SLAVE_)->getValue('
            SELECT `description`
            FROM `'._DB_PREFIX_.'sellerinfo_lang`
```

```
            WHERE `id_sellerinfo` = '.(int)$id_seller.'
            AND `id_lang` = '.(int)$id_lang.'
            ');

        return self::$cacheDescription[$id_seller];
    }
```

Metodo 8: getCompanyById(), getIdByCompany(), sellerExists() e getDescriptionById()

Queste funzioni eseguono query molto semplici e fanno uso dei metodi getRow() e getValue(), illustrati precedentemente, per recuperare singoli valori dal database.
La funzione getCompanyById() recupera il nome del venditore dal database a partire dall'identificativo numerico passato come parametro, getIdByCompany() si comporta in modo opposto.
La funzione getDescriptionById() recupera invece la descrizione del venditore in modo analogo a getCompanyById(). Queste tre funzioni agiscono sulla tabella ps_sellerinfo_lang, in quanto il nome e la descrizione variano a seconda della lingua corrente.
sellerExists() verifica la presenza o meno di un venditore all'interno del database di TuuDoo a partire dal suo identificatore. Questa funzione agisce sulla tabella ps_sellerinfo, in quanto l'id identifica il venditore in maniera assoluta in tutto il database, indipendentemente dalla lingua in cui il cliente sta visualizzando il sito.

```
   public static function getProducts($id_seller, $id_lang, $p, $n,
$order_by = null, $order_way = null, $get_total = false, $active_category =
true, Context $context = null)
    {
        […]

        $groups = FrontController::getCurrentCustomerGroups();
        $sql_groups = count($groups) ? 'IN ('.implode(',', $groups).')' : '=
1';

        if ($get_total)
        {
            $sql = 'SELECT p.`id product`
                    FROM `'. DB_PREFIX_.'product_owner` p
                    '.Shop::addSqlAssociation('product', 'p').'
                    INNER JOIN `'. DB_PREFIX_.'sellerinfo` s ON (p.`id_owner`
= s.`id seller`)
                    WHERE s.id_sellerinfo = '.(int)$id_seller.'
                    '.($front ? ' AND product_shop.`visibility` IN ("both",
"catalog")' : '').'
                    AND p.`id product` IN (
                        SELECT cp.`id product`
                        FROM `'. DB_PREFIX_.'category_group` cg
                        LEFT JOIN `'. DB_PREFIX_.'category_product` cp ON
(cp.`id_category` = cg.`id_category`)'.
```

```
                    ($active_category ? ' INNER JOIN
`'. DB_PREFIX .'category` ca ON cp.`id_category` = ca.`id_category` AND
ca.`active` = 1' : '').'
                    WHERE cg.`id_group` '.$sql_groups.'
                    )';

        $result = Db::getInstance(_PS_USE_SQL_SLAVE_)->executeS($sql);
           return (int)count($result);
    }

  [...]

    $sql = 'SELECT p.*, product_shop.*, stock.out_of_stock,
IFNULL(stock.quantity, 0) as quantity,
MAX(product_attribute_shop.`id_product_attribute`) id_product_attribute,
pl.`description`, pl.`description_short`, pl.`link_rewrite`,
pl.`meta_description`, pl.`meta_keywords`, pl.`meta_title`, pl.`name`,
pl.`available_now`, pl.`available_later`, MAX(image_shop.`id_image`)
id_image, il.`legend`, s.`company` AS seller_company,
DATEDIFF( product_shop.`date_add`,
          DATE_SUB(
             NOW(),
             INTERVAL
'.(Validate::isUnsignedInt(Configuration::get('PS_NB_DAYS_NEW_PRODUCT')) ?
Configuration::get('PS_NB_DAYS_NEW_PRODUCT') : 20).' DAY
             )
          ) > 0 AS new
             FROM `'. DB_PREFIX .'product` p
             '.Shop::addSqlAssociation('product', 'p').'
             LEFT JOIN `'. DB_PREFIX .'product_attribute` pa ON
(p.`id_product` = pa.`id_product`)
             '.Shop::addSqlAssociation('product_attribute', 'pa', false,
'product_attribute_shop.`default_on` = 1').'
             LEFT JOIN `'. DB_PREFIX .'product_lang` pl ON (p.`id_product`
= pl.`id_product` AND pl.`id_lang` =
'.(int)$id_lang.Shop::addSqlRestrictionOnLang('pl').')
             LEFT JOIN `'. DB_PREFIX .'image` i ON (i.`id_product` =
p.`id_product`)
             '.Shop::addSqlAssociation('image', 'i', false,
'image_shop.cover=1').'
             LEFT JOIN `'. DB_PREFIX .'image_lang` il ON (i.`id_image` =
il.`id_image` AND il.`id_lang` = '.(int)$id_lang.')
             LEFT JOIN (`'. DB_PREFIX .'product_owner` po
                INNER JOIN `'. DB_PREFIX .'sellerinfo` si ON
(po.`id_owner` = si.`id_seller`)
                INNER JOIN `'. DB_PREFIX .'sellerinfo_lang` s ON
(si.`id_sellerinfo` = s.`id_sellerinfo` AND s.`id_lang` =
'.(int)$id_lang.')) ON (po.`id_product` = p.`id_product`)
             '.Product::sqlStock('p', 0).'
             WHERE si.`id_sellerinfo` = '.(int)$id_seller.'
             '.($front ? ' AND product_shop.`visibility` IN ("both",
"catalog")' : '').'
             AND p.`id_product` IN (
                SELECT cp.`id_product`
                FROM `'. DB_PREFIX .'category_group` cg
                LEFT JOIN `'. DB_PREFIX .'category_product` cp ON
(cp.`id_category` = cg.`id_category`)'.
```

```
                ($active_category ? ' INNER JOIN `'. DB_PREFIX .'category`
ca ON cp.`id_category` = ca.`id_category` AND ca.`active` = 1' : '').'
                WHERE cg.`id_group` '.$sql_groups.'
                )
            GROUP BY product_shop.id_product
            ORDER BY '.$alias.'`'.bqSQL($order_by).'`
'.pSQL($order_way).'
            LIMIT '.(((int)$p - 1) * (int)$n).','.(int)$n;

        $result = Db::getInstance(_PS_USE_SQL_SLAVE_)->executeS($sql);

        if (!$result)
            return false;
        if ($order_by == 'price')
            Tools::orderbyPrice($result, $order_way);

        return Product::getProductsProperties($id_lang, $result);
    }
```

Metodo 9: getProducts()

Questa funzione ha il compito di prelevare dal database tutti i dati relativi a prodotti appartenenti ad un singolo venditore (il cui identificatore è passato come parametro).
La prima query della funzione serve a contare i prodotti inseriti dal venditore cercato, essa è identica alla query che, nella funzione getSellers(), conta i prodotti di ogni venditore.

La seconda query è quella che preleva effettivamente tutte le informazioni, dei prodotti e del venditore che li tratta.
Per poter avere tutte le informazioni del prodotto bisogna agire sulle tabelle:

- ps_product, contenente informazioni sui prodotti;
- ps_product_attribute, contenente informazioni sugli attributi che un prodotto può avere;
- ps_product_lang, contenente nomi e descrizioni dei prodotti nelle varie lingue supportate;
- ps_image, contenente le immagini dei prodotti;
- ps_image_lang, contenente le didascalie delle immagini dei prodotti nelle varie lingue supportate.

Per poter, invece, collegare un prodotto al suo venditore, le tabelle in gioco sono:

- ps_product_owner, che collega l'id di un prodotto con l'id del relativo venditore;
- ps_sellerinfo, contenente informazioni sui venditori;
- ps_sellerinfo_lang, contenente nomi e descrizioni dei venditori nelle varie lingue supportate.

Queste ultime tre tabelle sono tabelle create dal modulo Agile Multiple Seller. Specifico che l'id_owner, attributo della tabella ps_product_owner, è un numero intero che identifica un account che può accedere al back-office del sito (quindi venditori e amministratori del sito) mentre l'id_sellerinfo, attributo e chiave primaria della tabella ps_sellerinfo, è un numero intero che identifica un account di un venditore.
Gli identificatori vengono assegnati in base all'ordine cronologico della creazione degli account quindi sicuramente un venditore non avrà mai id_owner (collegato a id_seller in ps_sellerinfo) e id_sellerinfo uguali, in quanto almeno un account da amministratore è stato creato prima.
Dunque è indispensabile passare per la tabella ps_sellerinfo, un collegamento diretto tra ps_product_owner e ps_sellerinfo_lang non è possibile.

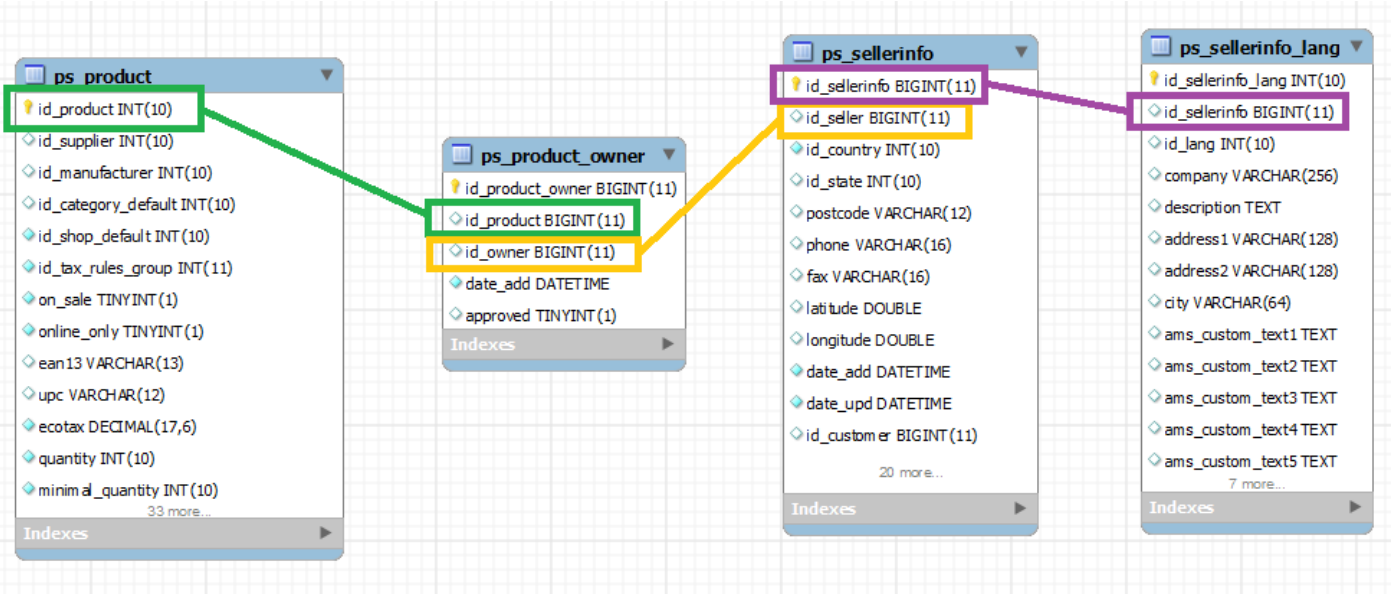

Figura 18: Percorso delle Foreign Keys che lega un prodotto al suo venditore

Durante la sua installazione, il modulo Agile Multiple Seller ha creato le tabelle necessarie nel database senza modificare le tabelle di default di PrestaShop. In questo modo la tabella ps_product contiene solo l'elenco dei prodotti senza che essi siano legati ad alcun venditore, funzione che è destinata alla tabella ps_product_owner; e neppure quest'ultima tabella è significativa per avere informazioni sull'effettivo venditore.
Il fatto di non modificare le tabelle di default, un po' come il sistema di overriding, è utile in caso di futuri aggiornamenti di PrestaShop e del suo database ed esprime a pieno il concetto di sistema modulare, che è uno dei punti di forza di PrestaShop.

Contrariamente a ciò, nello sviluppo specifico del modulo Sellers Block, questa impostazione del database ha portato ad un'utilizzo di tecniche di join più frequentemente del previsto.
Un'attributo id_sellerinfo inserito direttamente nella tabella dei prodotti avrebbe alleggerito di tre join la query, che invece prevede un utilizzo di otto join.

Tornando all'analisi della query, a partire dalla tabella ps_product vengono usati cinque left join (in quanto non necessariamente ogni prodotto è dotato di attributi o immagini), l'ultimo dei quali con la tabella risultante da due operazioni di inner join tra le tre tabelle sopracitate che collegano il prodotto al venditore.
Un outer join non richiede che ci sia corrispondenza esatta tra le righe di due tabelle. Il risultato di una query left outer join, o semplicemente left join, contiene sempre tutti i record della tabella di sinistra mentre vengono estratti dalla tabella di destra solamente le righe che trovano corrispondenza nella regola di confronto del join. [12]

La query prosegue con un controllo sui permessi che ha il cliente in questione, selezionando cioè solo i prodotti che hanno visibilità per lui, e con un ordinamento in base ad un'alias, che altro non è che un criterio scelto dall'utente (ordine alfabetico, in base al prezzo, in base alla giacenza…).

```
   public static function getSellerLink($id_seller, $id_lang = null,
$id_shop = null, $ssl = null)
   {

      static $force_ssl = null;
      if ($ssl === null)
      {
         if ($force_ssl === null)
            $force_ssl = (Configuration::get('PS_SSL_ENABLED') &&
Configuration::get('PS_SSL_ENABLED_EVERYWHERE'));
         $ssl = $force_ssl;
      }

      if (Configuration::get('PS_MULTISHOP_FEATURE_ACTIVE') && $id_shop !==
null)
         $shop = new Shop($id_shop);
      else
         $shop = Context::getContext()->shop;

      $base = (($ssl && $this->ssl_enable) ? 'https://'.$shop->domain_ssl :
'http://'.$shop->domain);

      if (!$id_lang)
         $id_lang = Context::getContext()->language->id;
```

```
    return
$base.'/index.php?id_seller='.(int)$id_seller.'&id_lang='.(int)$id_lang.'&f
c=module&module=sellersblock&controller=seller&id_lang='.(int)$id_lang;
    }
```

Metodo 10: getSellerLink()

Questa funzione ha lo scopo di costruire l'URL per la pagina che visualizza i prodotti di un venditore. Ogni volta che il cliente clicca su uno dei vari tasti che porta a quella pagina, viene chiamata questa funzione, a cui viene passato come parametro l'id del venditore. La funzione inserisce l'identificatore nell'URL che quindi quando verrà analizzato indirizzerà il controller seller.php verso il caricamento del giusto template.
Per il codice completo di Seller.php visionare l'appendice.

Dopo aver preparato tutti i file necessari, il modulo è completo. L'ho quindi caricato sul server e installato su PrestaShop. Ho inoltre tradotto il modulo sia in italiano che in inglese, e comunque è già predisposto per altre future traduzioni in altre lingue.

Capitolo 5
Conclusioni

5.1 FASI FINALI E COMPLETAMENTO DEL PROGETTO

Una volta installati i moduli Agile Multiple Seller, Agile Seller Shipping, Agile Seller Messenger (acquistati) e Sellers Block (sviluppato), il sito è strutturalmente operativo.
La grafica e i prodotti ora presenti sono però quelli di esempio di PrestaShop. Il prossimo passo è adattare la grafica del sito alle specifiche per il progetto TuuDoo e inserire dei prodotti di prova più significativi per la realtà in cui TuuDoo andrà ad operare.
Il graphic designer dell'azienda si è occupato della parte grafica, creando le immagini (logo, simbolo del carrello…), modificando i file CSS del tema di PrestaShop e adattando le fotografie, appositamente scattate per il progetto TuuDoo, ai vari spazi e blocchi della home page.
Io ho provveduto a inserire dei venditori e dei prodotti di prova a partire da dati reali di possibili clienti che si erano mostrati interessati al progetto.

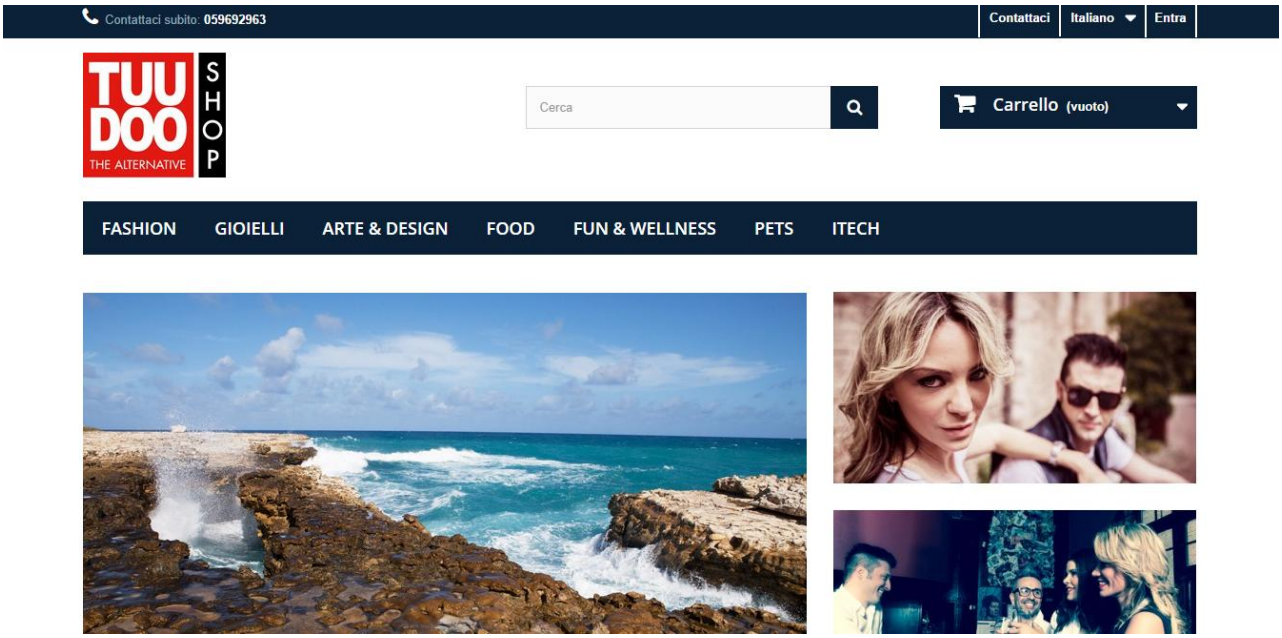

Figura 19: Homepage di TuuDoo

Con una grafica quasi definitiva e dei dati verosimili, il prossimo step sarebbe stato quello di testare il funzionamento effettivo del sito, quindi effettuando un ordine e il relativo pagamento con soldi reali.
PrestaShop ha già installati di default alcuni moduli per i pagamenti, mentre altri sono reperibili gratuitamente sullo store ufficiale. Il modulo per poter accettare pagamenti tramite PayPal e con carte di credito o prepagate è stato appunto preso dallo store ufficiale di PrestaShop.
Come già spiegato nell'obiettivo, tutti i soldi ricavati devono confluire in un conto unico, e solo in seguito essere divisi tra i venditori.
Il funzionamento del modulo Agile Multiple Seller non va ad intralciare il funzionamento dei moduli di default o ufficiale di PrestaShop per il pagamento, almeno relativamente alle specifiche di progetto riguardanti le modalità di pagamento.
Se i soldi, tutti o in parte, avessero dovuto raggiungere il venditore del prodotto all'atto del checkout di un cliente finale (modalità previste dal modulo Agile), sarebbero stati necessari dei moduli integrativi, sempre sviluppati da Agile.
I moduli per il pagamento sono dunque stati configurati con i dati adeguati e il test ha mostrato risultati positivi.

Il sito TuuDoo è quindi pronto per essere provato da alcuni venditori interessati. Ho creato due manuali per illustrale la gestione di un account venditore (uno per la gestione front end e uno per la gestione back end), spiegando in poche parole,

con l'ausilio di alcuni screenshot, come inserire prodotti e corriere e gestire ordini.
Ho quindi inserito nel database i dati dei clienti interessati a provare il sito come venditori e ho poi mandato loro una mail di invito su TuuDoo, allegando i manuali.
Il sito è quindi ora in fase di testing sul campo, con venditori veri, in modo che in futuro possa essere adattato meglio alle esigenze di coloro che lo dovranno usare e in modo da risolvere eventuali problemi non ancora riscontrati che potrebbero sorgere. Il sito è comunque online e funzionante e il progetto TuuDoo si è concluso con i risultati sperati.

Il sistema modulare di PrestaShop può favorire sviluppi futuri in quanto a funzionalità implementate dal sito e riguardo la sua struttura. Probabilmente la grafica subirà alcuni cambiamenti per rendere il sito più elegante e pulito.

5.2 BIBLIOGRAFIA E SITOGRAFIA

[1] carpinet.it

[2] "The Propensity of E-commerce Usage: the Influencing Variables", Bonera M. (2011), Management Research Review, Vol. 34, Iss.7.

[3] treccani.it

[4] wikipedia.org

[5] doc.prestashop.com

[6] smarty.net

[7] Steve Hoberman, Donna Burbank, & Chris Bradley (2009). Data Modeling for the Business. Technics Publications, LLC

[8] 1keydata.com/datawarehousing/data-modeling-levels.html

[9] Peter Pin-Shan Chen (1976). The Entity-Relationship Model: Toward a Unified View of Data.

[10] addons-modules.com

[11] webkul.com

[12] Beneventano, Bergamaschi, Guerra, Vincini, “Progetto di Basi di Dati Relazionali: lezioni ed esercizi”, Pitagora Editrice Bologna