POLITECNICO DI TORINO

DIPARTIMENTO DI AUTOMATICA E INFORMATICA

Corso di Laurea in Ingegneria Informatica

Monografia di Laurea

LFS-Facebook

integrazione di un'applicazione e-commerce sulla piattaforma Facebook

Filippo Projetto

Dicembre 2013

Indice

Capitolo 1

Introduzione

1.1 L'azienda

Dal 2004 Redomino cura lo sviluppo, la commercializzazione e la manutenzione di piattaforme digitali open source per il business e dal 2007 fornisce servizi nei settori del digital marketing, del social media marketing e più di recente del green marketing e della co-creazione. La divisione green marketing di Redomino aiuta i clienti a comunicare al meglio i propri valori e le proprie eccellenze nel settore socio-ambientale con l'obiettivo di sviluppare innovative forme di business.

1.2 Obiettivi

Il tirocinio si è incentrato sull'apprendimento del linguaggio Python, del framework Django e delle API Facebook. E' stato realizzato, attraverso il supporto formativo del tutor aziendale e figure specializzate all'interno dell'azienda, un'applicazione web in grado di integrare come Facebook App un e-commerce. L'obiettivo non è stato quello di creare un sistema completo, ma dei moduli riusabili che prototipino il comportamento richiesto. Componenti fondamentali del tirocinio sono stati anche l'interazione con le comunita' open source che seguono i suddetti progetti, e la documentazione di quanto svolto.

1.3 Tecnologie al contorno

Le tecnologie utilizzate durante lo sviluppo del progetto sono quelle ritenute molto spesso necessarie per la maggior parte dei progetti di sviluppo software. Tra queste:

- software di controllo versione, in particolare quello fornito da https://github.com, che è stato utile per le seguenti azioni:
  - creazione di un repository: <git init>;
  - aggiunta di file al repository: <git add "nome file">, <git commit -m "messaggio che descrive la modifica">, <git push>;

  - mostra i cambiamenti tra file versionati e quelli in locale in fase di sviluppo: <git diff "nome file">;
  - ripristino ad una versione precedente di uno o più file del progetto, o: <git checkout "nome file">.

- un build system basato su Python per creare, assemblare e sviluppare applicazioni da piu parti, alcune delle quali potrebbe non essere scritte in Python. Creare un file di configurazione Buildout permette di riprodurre lo stesso software dopo lo sviluppo dello stesso senza perdite di tempo.

  "While not directly aiming to solve world peace, it perhaps will play a role in the future, as people will be less angry about application deployment and will have more time for making love and music. –Noah Gift, co-author of 'Python For Unix and Linux' from O'Reilly."

- virtualenv, un tool che permette di creare versioni isolate dell'ambiente Python, utile per svariati motivi, tra cui:

  - ogni virtualenv ha i permessi dell'utente che lo crea, quindi per installare pacchetti addizionali non occorrono i permessi di root;
  - lasciare l'installazione del sistema il più snella possibile;
  - aggiornare una o più librerie di Python, senza che venga compromesso il funzionamento di altri programmi.

- E' stato necessario inoltre ottenere un certificato SSL, in quanto le politiche Facebook riguardanti le applicazioni prevedono come requisito una connessione sicura https. Openssl toolkit è stato utilizzato per generare una RSA Private Key e un CSR (Certificate Signing Request). Il CSR può essere utilizzato in due modi:

  - può essere inviato ad una Certificate Authority, come Thawte or Verisign che verificherà l'identità del richiedente ed emetterà un signed certificate;
  - la seconda opzione è creare un CSR self-signed.

  Per i nostri scopi non è stato sufficiente ottenere un certificato self-signed.

Capitolo 2

LFS-Facebook

2.1 Progetti di partenza

La parte iniziale del tirocinio è stata impiegata principalmente nello studio di Python,il framework django e di due progetti open source: LFS e django-facebook.

LFS[1] è un negozio online che si basa su Python, django(Web framework per Python) e jQuery. A questo è stato aggiunto poi un tema responsivo (anch'esso basato su un altro progetto open-source, bootstrap), ed il tutto ha rappresentato la base sulla quale aggiungere delle funzionalità legate principalmente alla sua integrazione come applicazione fruibile su Facebook.

django-facebook[2], invece, permette ad applicazioni django di registrare gli utenti tramite backend Facebook, convertendo i dati ricevuti dall'operazione di log in con Facebook, in un modello consono alle modalità di storing attraverso django.

Inoltre, si è fatto anche un overview di Bootstrap[3], un framework per lo sviluppo web client-side, che ho trovato molto interessante in quanto ha come obiettivo principale quello di supportare la navigazione da mobile; in particolare, Bootstrap adatta in modo automatico le pagine web alle dimensioni dello schermo.

[1] http://www.getlfs.com

[2] http://pypi.python.org/pypi/django-facebook

[3] http://getbootstrap.com/

2.2 Struttura del pacchetto LFS-Facebook

LFS-Facebook è un pacchetto che aggiunge funzionalità social ad un e-commerce. In quanto applicazione scritta sfruttando il framework django, si struttura come riportato in figura 2.1. La struttura base è stata generata attraverso il comando:

```
python manage.py startapp myapp
```

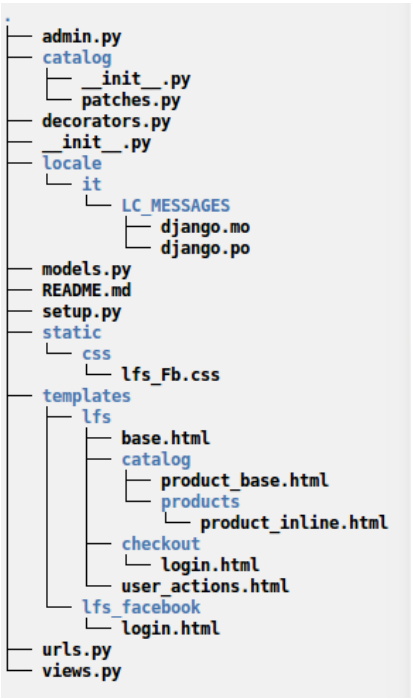


Figura 2.1. package structure

Descrizione dei principali contenuti del pacchetto:

- **views.py** definisce quelle funzioni che elaborano informazioni per poi chiamare il template html (avendo la possibilità di passare dei dati sottoforma di context) che verrà visualizzato;
- **urls.py** dato un URL, specifica quale funzione chiamare (che quindi risiederà in views.py), è quindi un semplice file che realizza un mapping;
- **models.py** definisce contenuto e comportamento dei dati (vedi 2.6);
- **locale** contiene informazione per l'internazionalizzazione (vedi 2.8);
- **templates** contiene il codice html delle pagine web, utilizzando anche il linguaggio di templating proprio di django;
- **decorators.py** definisce un nuovo decoratore (vedi 2.5);
- **catalog/patches.py** personalizzazione di funzioni pre-esistenti in LFS (vedi 2.5).

2.3 Gestione backend di autenticazione

Un'esigenza che un progetto di un'applicazione potrebbe avere è quella di dover/voler pescare da un'altra sorgente di usernames e passwords.
Questo è proprio il nostro caso, vogliamo usare come backend di autenticazione quello di Facebook. Per iniziare si è testata un'applicazione django come LFS con LDAP.

LDAP è un protocollo di livello applicativo per accedere e gestire servizi di directory su una rete IP. Uno degli usi più comuni di LDAP è quello di fornire un "single sign-on" dove per la coppia d'informazioni utente:password è condivisa tra diversi servizi. Nel nostro caso è stato utilizzato openLDAP, un'implementazione opensource del suddetto protocollo.

Per aggiungere questa funzionalità ad un'applicazione django è necessario modificare la variabile AUTHENTICATION BACKENDS del file setting.py, in particolare django mantiene una lista di "authentication backends" che egli verifica per ogni autenticazione.
Quando qualcuno cerca di autenticarsi, django prova l'autenticazione lungo tutta la lista di backend per cui è impostato. Attenzione che l'ordine è importante in quanto django si fermerà al primo match positivo.

Per quanto riguardo openLDAP sono necessarie anche tutta un'altra serie di impostazioni da specificare per l'accesso al server, e la ricerca; qui un esempio di configurazione minimale:

```
AUTH_LDAP_SERVER_URI = "ldap://ldap.example.com"

AUTH_LDAP_BIND_DN = "cn=django-agent,dc=example,dc=com"
AUTH_LDAP_BIND_PASSWORD = "phlebotinum"
AUTH_LDAP_USER_SEARCH = LDAPSearch("ou=users,dc=example,dc=com",
    ldap.SCOPE_SUBTREE, "(uid=%(user)s)")
```

vengono così configurate due operazioni fondamentali: Bind, che esegue l'autenticazione, e Search che esegue una ricerca. Nella struttura ad albero, ad ogni livello esiste un Relative distinguished name (RDN) che lo identifica (ad esempio ou=people). L'unione di tutti i RDN, presi in successione dal nodo foglia fino alla radice, costituisce il distinguished name (DN), una stringa che rappresenta univocamente una entry nella directory. Nell'esempio di configurazione viene fornito il DN di un utente che ha i permessi per fare determinate azioni, e diamo il sottoalbero dove ricercare gli utenti come primo parametro della LDAPSearch.

Mentre per quanto riguarda il backend di autenticazione di Facebook è sufficente aggiungere al file di setting:

```
AUTHENTICATION_BACKENDS = (
        'django_facebook.auth_backends.FacebookBackend',
        'django_auth_ldap.backend.LDAPBackend',
        'django.contrib.auth.backends.ModelBackend'
)
```

Da notare che va inserito in testa alla lista.

2.4 Integrazione dell'applicazione sulla piattaforma Facebook

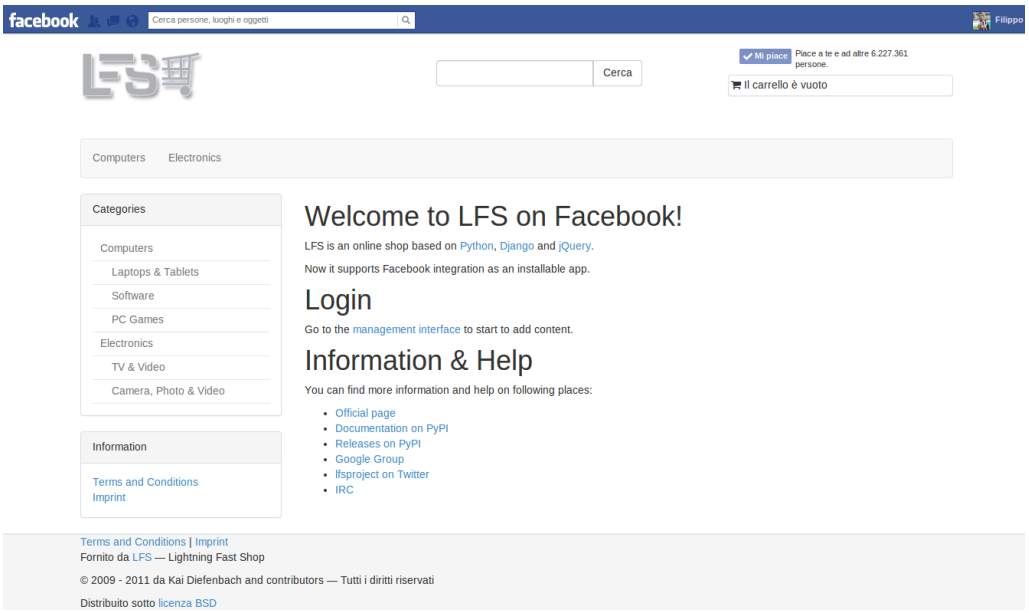


Figura 2.2. home page on Facebook

Per inserire l'applicazione tra le App di Facebook è necessario procurarsi innanzitutto le credenziali.

Qui i passi utili per ottenerle:

- raggiungere la Facebook Developers page[4] e autenticarsi con username e password del proprio account facebook;
- cliccare su Create New App;
- se non si vede quest'opzione, cliccare su Register as Developer;
- inserire un nome per l'applicazione nel campo App Name;
- accettare le normative della piattaforma Facebook;
- inserire il proprio dominio nel campo App Domains; poi, portarsi alla sezione Select how your app integrates with Facebook e selezionare Website with Facebook Login. Questa sezione mostrerà un campo Site URL dove digitare nuovamente il dominio;
- cliccare su Save Changes.

[4] https://developers.facebook.com/

Accedendo alla pagina principale dell'applicazione sono disponibili le due credenziali utili per il proseguo: ID applicazione e App Secret. Adesso vanno inserite nel file di setting del progetto:

```
FACEBOOK_APP_ID = 'YOUR APP ID'
FACEBOOK_APP_SECRET = 'YOUR APP SECRET NUMBER'
```

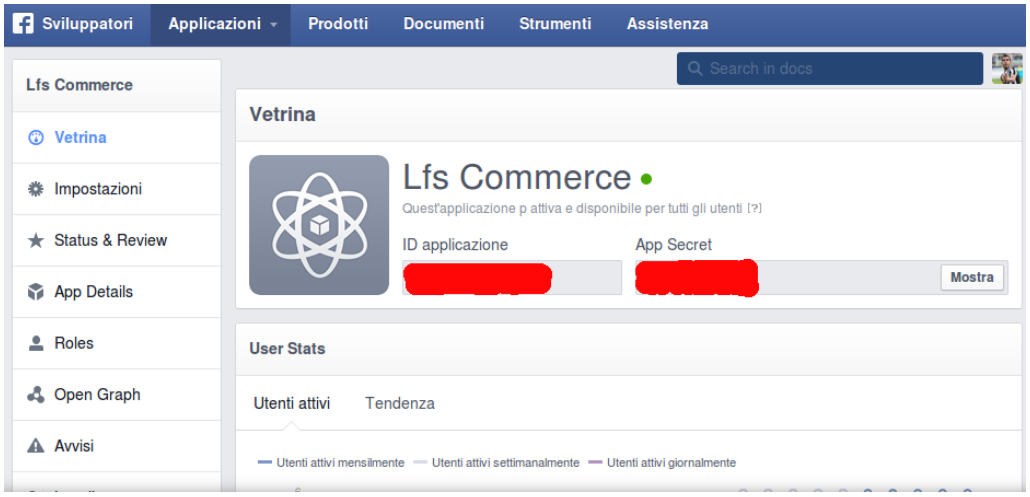


Figura 2.3. basic page for your app

Da notare che è richiesta una connessione https.

2.4.1 Facebook plugin

Facebook fornisce strumenti, servizi ed SDKs per integrare una social experience nelle apps. All'interno delle pagine html di LFS-Facebook si è fatto uso di social plugin il cui codice è fornito agli sviluppatori registrati sulla piattaforma tramite delle pagine nella documentazione che permettono di settare parametri per customizzare il formato dei vari plugin per poi generare il codice di cui fare copy and paste.

Inoltre si sono acquisite delle informazioni del profilo Facebook attraverso delle istruzioni FQL. Facebook Query Language, o FQL, permette di interfacciarci, in modo simile ad SQL, per interrogare i dati ottenuti dalle Graph API.

Le query hanno questa struttura:

```
SELECT [fields] FROM [table] WHERE [conditions].
```

Diversamente da SQL la clausola FROM può contenere solo una tabella. Ad ogni query ci si può riferire all'utente loggato attraverso me().

2.5 Protezione delle viste con autenticazione richiesta

L'applicazione si presenta senza un link ad una pagina di login, in quanto l'utente sarà automaticamente rediretto alla procedura di autenticazione quando e se necessario.

Per ottenere questo comportamento si è scelto di creare un decoratore che protegga le viste desiderate. Le viste da proteggere non sono imposte dal sistema, ma è tutto pienamente configurabile attravrso il file di setting. Infatti, è stato creato un dizionario di variabili booleane, a ciascuna delle quali corrisponde una pagina che è possibile proteggere settando come 'True' la variabile in questione.

```
VIEW_WITH_LOGIN_REQUIRED = {
    'add-to-cart': True,
    'shop': False,
    'category': False,
    'product': False,
    'checkout': False,
}
```

Se una vista è protetta da decoratore, sarà dunque scandito l'elenco di variabili: se la variabile corrispondente è uguale a True, sarà chiamato un altro decoratore, nativo del framework django, @login_required, che verificherà lo stato dell'utente, che se loggato procederà verso la pagina richiesta, altrimenti sarà rediretto alla pagina di login.

```
def permissions_required(view_name):

    def decorator(func):
        if settings.VIEW_WITH_LOGIN_REQUIRED[view_name]:
            return login_required(func)
        else:
            return func

    return decorator
```

Comunque si può scegliere di non proteggere alcuna vista e permettere la libera navigazione all'utente che agirà come utente anonimo, potendo aggiungere prodotti al carrello, andare alla cassa, decidere di completare l'acquisto e quindi effettuare il pagamento.

patches Affinchè siano chiamate la versioni delle nostre funzioni, rappresentanti le viste che devono sovrascrivere quelle già esistenti nel package di LFS, abbiamo usato il meccanismo delle patches. Per far questo abbiamo:

- creato un file nel package della nostra applicazione, nel quale importare la funzione di cui si vuole fare la patch;
- implementato la nostra funzione la quale deve avere gli stessi parametri di input di quella da sovrascrivere;
- e infine sostituito la vecchia funzione con la nostra: <nome funzione da sovrascrivere> = <nome nostra funzione>

2.5.1 Pagina di login

La pagina di login è stata modificata facendo un override di quella esistente attraverso il linguaggio di template di django, che permette, tra le altre cose, di ereditare da una pagina html e sovrascrivere i blocchi che si vuole.

```
{% extends "lfs/customer/login.html" %} -> inheritance
{% block wrapper %} -> blocco da sovrascrivere
...
{% endblock %}
```

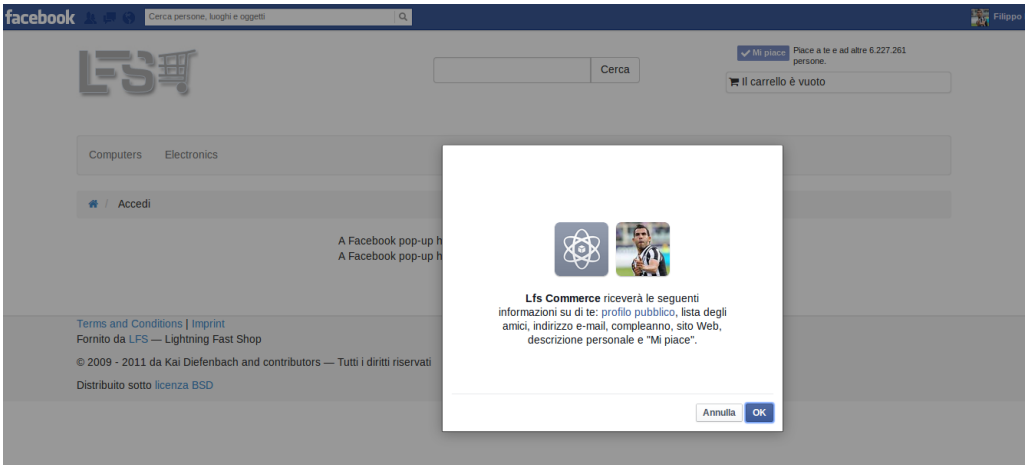


Figura 2.4. richiesta permessi da parte dell'applicazione

Si è scelto di presentare una pagina di login che non richieda nessuna interazione all'utente, infatti l'autenticazione si avvierà in automatico. Per far questo si è utilizzato del codice javascript di django-facebook che effettua il login facebook attraverso le facebook API; il tutto viene scatenato dall'istruzione F.connect(), ma essendo F una variabile che viene caricata asincronamente, è stato necessario inserire la F.connect all'interno della funzione setInterval (utility di javascript) che proverà a chiamare la F.connect ad intervalli di tempo prefissati finchè non avrà successo, ovvero fin quando F non sarà definita. Il meccanismo di chiamata periodica ad uno script verrà fermato attraverso il clearInterval().

```
var myVar = setInterval(function(){
    if(F !== undefined){
        F.connect(document.getElementById('facebook_form'),
        ['email', 'user_about_me', 'user_birthday',
        'user_website', 'user_likes']);
        clearInterval(myVar);
    }
    }, 500);
```

template override L'override di un template pre-esistente viene effettuato creando un file html con lo stesso identico nome e dovrà essere riprodotta l'alberatura 2.1. Per

esempio se voglio sovrascrivere lfs/templates/lfs/base.html, dovrò creare un file base.html in myapp/templates/lfs/base.html.

Dopodichè è necessario anche intervenire nel file di setting ed assicurarsi che fra le INSTALLED_ APPS, myapp appaia prima della applicazione di cui vogliamo sovrascrivere il template.

2.5.2 Pagina di checkout

La pagina di checkout è l'ultima pagina della user experience, prima di procedere all'inserimenrto dati per effettuare il pagamento e completare l'acquisto.

Anche di questa pagina è stato fatto un override per visualizzare ancora la possibilità di registrarsi prima dell'acquisto. Se si effettua il login sarà possibile gestire lo stato degli ordini e salvare i dati per prossime spedizioni. Il pulsante del login è un normale elemento html(non un faceboook plugin) il quale stile è stato reso simile a quello Facebook attraverso css.

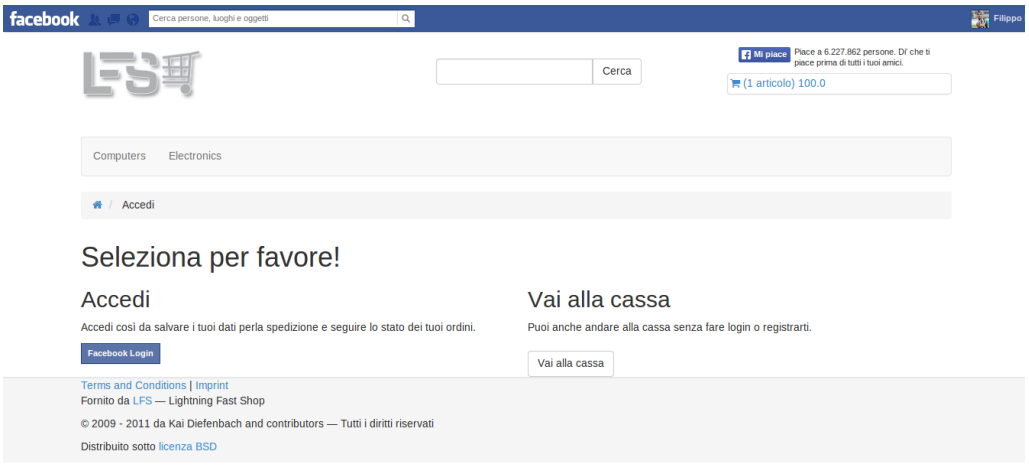


Figura 2.5. checkout page

2.6 Personalizzazione del modello utente

Il modello, con django, è l'unica fonte di informazione a riguardo dei dati che vogliamo memorizzare. Esso contiene i campi e il comportamento per la memorizzazione.

Quello che vogliamo fare è modificare il modello che definisce la gestione delle informazioni riferite all'utente. Per far questo, si è creato un file models.py, contenente una nuova classe che rappresenta il modello. Questa eredita da FacebookProfileModel la quale è la classe del pacchetto django-facebook che gestiva le informazioni ottenute dal backend di autenticazione di Facebook.

```
class LfsFbUser(FacebookProfileModel):
    user = models.OneToOneField(User)
```

```
    fb_like_flag = models.BooleanField(default=False)

        ...

    class Meta():
            db_table = 'fb_data'
```

Assicuriamoci a questo punto che l'applicazione dove risiede il models.py dov'è definita la classe LfsFbUser, sia tra le INSTALLED_APPS del file di setting, e poi bisogna lanciare il comando:

```
python manage.py syncdb
```

che crea le tabelle del database(non ancora create) per le apps specificate.

Per settare django affinchè usi il modello appena creato, come modello per l'autenticazione, bisogna modificare il file di setting, aggiungendo al fondo del file oppure modificando la voce esistente, in modo da avere:

```
AUTH_PROFILE_MODULE = 'LFS-Facebook.LfsFbUser'
```

2.7 Prodotti riservati ai fan

Come feature aggiuntiva si è pensato di dare la possibilità a chi mette in piedi l'e-commerce, di poter riservare l'acquisto di un prodotto soltanto ai Facebook fan, ovvero a chi ha messo il mi piace alla pagina Facebook del negozio.

Per far ciò bisogna configurare il sistema specificando l'identificativo della pagina facebook nel solito file di setting, come segue:

```
FACEBOOK_PAGE = 'YOUR PAGE ID'
```

Per distinguere un prodotto riservato ai fan, si utilizza una caratteristica di LFS che sono le PROPRIETA. queste si possono gestire da web, loggandosi attraverso un utente amminstratore ed entrando nella sezione gestione del sito. Cosa importante è il titolo della proprietà, in quanto questo attributo verrà poi utilizzato nel codice.

```
fb_reserved = "False"
    for p in product.get_properties():
        if p.title == settings.FACEBOOK_FAN_RESERVED_PROPERTY:
            fb_reserved = "True"
```

Il titolo da adottare è settabile anch'esso nel setting.py:

```
FACEBOOK_FAN_RESERVED_PROPERTY = "YOUR PROPERTY TITLE"
```

Trovate le modalità per marcare un prodotto come riservato, è importante vedere come controllare se l'utente soddisfa i requisiti per effettuare l'acquisto.

Per far questo si è ricorso ad una FQL Query che seleziona un user ID solo se presenta tra la lista di pagine ottenute con la query, la pagina Facebook dell'applicazione.

```
var fql_query = "SELECT uid
                 FROM page_fan
```

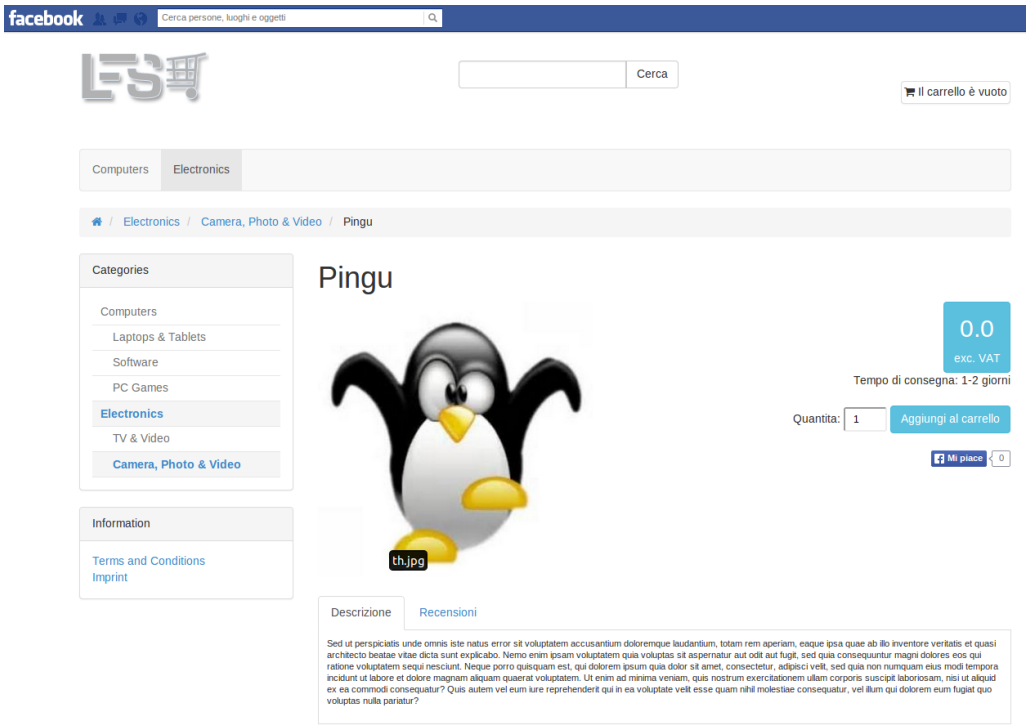
```
                WHERE page_id = "+fbPage+
                " and uid = "+user_id;
```

nota: Per effettuare questa query è necessario aver richiesto i permessi in fase di login.

Attraverso una Facebook API, eseguiamo la query, e ci muoviamo di conseguenza; in particolare, attraverso jQuery, se l'utente è abilitato all'acquisto, si abilita il pulsante di aggiunta al carrello e si mostra un determinato messaggio, altrimenti si disibalita il pulsante e si mostra un messaggio complementare.

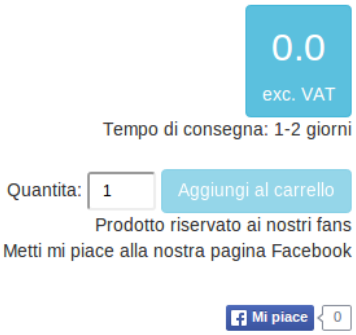
```
$(document).ready(FB.api({
    method: 'fql.query',
    query: fql_query
},
function(response){
    if(fbReserved == "True"){
        if (response.length == 1 && response[0].uid == user_id){
            allowPurchase(true);
        } else {
            allowPurchase(false);
        }
    }else{
        $("#add-to-cart").prop('disabled', false);

        $("#fb-no-like").hide();
        $("#fb-already-like").hide();
    }
}));
```



(a) no reserved product

(b) Facebook fan



(c) no Facebook fan

Figura 2.6. product view

2.8 Internazionalizzazione

Lo sviluppo di un'applicazione web richiede, sempre più frequentemente, di potere essere utilizzata da una moltitudine di utenti di lingue e culture diverse.

La soluzione a questa esigenza è di consentire a questi utenti di poter scegliere la lingua con cui desiderano navigare le pagine dell'applicazione stessa.

L'internazionalizzazione è il processo mediante il quale vengono eliminati dal codice sorgente i preconcetti culturali quali il formato della data, la formattazione dei numeri, la visualizzazione di immagini raffiguranti elementi tipici di un determinato paese (autobus, cassette della posta, ecc.).

La Localizzazione è il processo di adattamento di un'applicazione per permetterle di poter cambiare dinamicamente le risorse presenti nelle pagine: principalmente stringhe ma anche file di testo, immagini, file audio e ogni altro tipo di contenuto inseribile in una pagina web. Quindi codice del tipo:

lblMessaggio.Text = "Ciao Mondo!"

deve essere riscritto in modo che la stringa "Ciao Mondo!" non sia statica, ma possa variare in modalità runtime secondo la lingua scelta dall'utente.

django supporta pienamente la traduzione del testo, formattazione delle date, orari, numeri e time zones.

Questo viene fatto essenzialmente attraverso due passi:

- si specificano quali parti della propria app dovrebbero essere tradotte o formattate per culture e/o lingue specifiche;
- django usa questi ganci per localizzare la Web app per un utente specifico secondo le proprie impostazioni.

La parola 'Internationalization' è spesso abbreviata con 'i18n'. Questa abbreviazione è largamente usata, deriva dal fatto che ci sono 18 lettere tra la 'i' e la 'n'.

Sebbene il meccanismo di traduzione può essere avviato da molti tipi di file, per esempio nel codice python, javascript, etc... per il nostro lavoro è stato sufficiente supportare l'internazionalizzazione nell'html. Più in dettaglio, in testa alle pagine desiderate, in accordo con il linguaggio django, è stato inserito il tag

```
{% include 'i18n' %}
```

ed ogni elemento di cui si disporrà di una traduzione va inserito tra il tag

```
{% trans '<item>' %}
```

Successivamente è stato necessario scrivere la traduzione. Per far questo si utilizza il comando

```
django-admin.py makemessages -l <id_language>
```

il quale crea un file con l'elenco delle voci da tradurre e la giusta alberatura nel quale inserirlo. Ovviamente adesso va qui inserita la traduzione frase per frase. Successivamente lanciando il comando

```
<django-admin.py compilemessages>
```

verrà creato il file di estensione .mo utilizzato dal sistema per le traduzioni 2.1.

2.9 Installazione del pacchetto

Per rendere semplice l'installazione si è creato un file .cfg per sfruttare la tecnologia buildout. Questo file ricalca quello fornito con la distribuzione di LFS ed in particolare aggiunge i due pacchetti esterni utilizzati:

```
[sources]
lfs_facebook = git https://github.com/filippo91/lfs_facebook.git
lfs_responsivetheme =
        git https://github.com/redomino/lfs_responsivetheme.git
```

e nuove impostazioni per come configurare django, inserendo tra le librerie django-facebook:

```
[django]
recipe = djangorecipe
eggs =
    django-lfs
    gunicorn
    django_facebook
project = lfs_project
settings=settings
extra-paths =
    \${buildout:directory}/parts
    \${buildout:directory}/lfs_project
    \${buildout:directory}/src
    \${buildout:directory}/src/lfs_facebook
    \${buildout:directory}/src/lfs_responsivetheme
```

Per garantire che sia installata la stessa versione del software, basta inserire nel file .cfg:

```
extends =
    versions.cfg
```

che specifica un file in cui è elencata la versione da installare per ogni pacchetto.

Capitolo 3

Conclusioni

3.1 Possibili sviluppi

Il lavoro fatto è stato pacchetizzato, ed è scaricabile da github[1]. Seguendo le istruzioni inserite nel file di readme, si ricrea in pochi passi l'ambiente di sviluppo, scaricando cartelle e file del progetto con relative dipendenze in modo automatico attraverso il file di buildout.

Ovviamente, l'applicazione ha un ampio margine di miglioramento soprattutto in termini di caratteristiche aggiuntive da offrire, e quindi da sviluppare. Tra queste:

- specificare come indirizzo di posta elettronica quello Facebook, per ricevere novità, avvisi, o qualsiasi altra interazione;
- sfruttare il meccanismo delle notifiche di Facebook, ad esempio per aggiornamenti sulla nuova disponibilità di un prodotto;
- poter condividere una lista di oggetti desiderati attraverso un post in bacheca;
- sconti e/o altri vantaggi nel giorno del compleanno.

3.2 Esperienza in azienda

Valuto l'esperienza in azienda molto importante per la mia carriera in quanto mi ha messo a contatto con le reali esigenze nel mondo dello sviluppo delle applicazioni web e dell'informatica a livello extra-accademico. Il valore tecnico e la preparazione delle persone sono state un grande stimolo a dar sempre il massimo dell'impegno per apprendere il più possibile.

Sebbene inizialmente abbia avuto difficoltà nel capire la tecnologia, anche perchè questo è stato il primo approccio a linguaggi server-side in generale, python nello specifico, sono

[1] https://github.com/filippo91/lfs_facebook_buildout

riuscito nel capire i concetti base grazie alle capacità di problem solving acquisite attraverso questi anni di preparazione scientifica (liceo e percorso di laurea).

Interessante è stato esplorare il mondo open-source, leggere codice scritto da professionisti ed utilizzare questo codice per i propri progetti. Infatti, a differenza da quello che avviene spesso in ambito accademico, dove di frequente gli homework prevedono di partire da zero è realizzare tutto il codice ex novo, ho avuto l'opportunità di confrontarmi con un altro approccio allo sviluppo, ovvero quello di aggiungere funzionalità ad un progetto esistente, magari modificandone anche il comportamento nativo, ma comunque in modo da creare un plug-in, e quindi scrivendo moduli che si possano essere installati successivamente, senza modificare direttamente il codice pre-esistente.

Inoltre ritengo molto utile l'esperienza in quanto mi ha permesso di valutare l'area dello sviluppo web e capire se approfondire l'argomento, magari con dei corsi di un percorso di laurea specialistica.

Infine, ho avuto modo di apprezzare la filosfia open-source la quale comporta passare da una cultura aziendale/personale protezionistica e difensiva, ad un approccio aperto, in cui tutti danno quello che sanno fare meglio e tutti si arricchiscono imparando dal meglio.

"Se tu hai una mela, e io ho una mela, e ce le scambiamo, allora tu ed io abbiamo sempre una mela per uno. Ma se tu hai un'idea, ed io ho un'idea, e ce le scambiamo, allora abbiamo entrambi due idee." G.B. Shaw