Tecnologie Web

Aurora Store

Alternativa per la compravendita Apps.

Progetto Universitario, CdL Informatica, UniMoRe.
Anno Accademico 2023.

Elia Tolin

Matricola 136515.

260162@studenti.unimore.it

Sommario

Introduzione al progetto

Il progetto Aurora Store mira a creare una piattaforma di e-commerce dedicata alla vendita e all'acquisto di applicazioni per dispositivi Android e iOS. L'obiettivo principale è semplificare il processo di vendita rispetto ai competitor come AppStore e PlayStore, che presentano procedure burocratiche e complesse. Qualsiasi utente registrato può partecipare alla vendita e all'acquisto di applicazioni, e anche la navigazione in anonimo è possibile. Per migliorare l'esperienza d'acquisto, è presente un sistema di valutazione delle app acquistate, dove gli utenti possono valutare le applicazioni da essi acquistate.

Utenti

In AuroraStore ci sono due tipi di utenti, registrato e non registrato (chiamato anche anonimo).

Utente non registrato

Un utente non registrato può navigare liberamente nel portale nelle pagine a lui consentite, ciò dove tecnicamente non viene richiesto sapere l’utente che ha effettuato la richiesta e

nelle pagine in cui la navigazione è sicura per gli altri utenti.
Questo utente può fare le seguenti operazioni:

1. Navigazione nella pagina iniziale.
    a. In questa pagina si potrà navigare da anonimi ma non si avrà il beneficio dell'utilizzo ottimale del Sistema di Raccomandazione, descritto in seguito.
2. Ricerca utenti e apps.
3. Visualizzazione utenti e apps.

In caso di tentativo di pubblicazione di un'app o aggiunta al carrello di un'app verrà richiesto all'utente il Login o in generale in una pagina a lui non consentita.

Utente registrato

Un utente registrato può compiere tutte le azioni dell'utente non registrato è in più tutte le funzioni che la piattaforma propone. Questo utente può pubblicare e vendere un'app e comprare un'app sia per iOS che Android, oltre alle funzioni personali quali vendite, acquisti, etc. Questo utente ha il vantaggio inoltre di avere il Sistema di Raccomandazione (descritto successivamente), che si basa sul dispositivo dell'utente che viene richiesto al momento della registrazione di esso. L'utente ha la limitazione che non può comprare la sua stessa App.

Admin

Tramite le funzionalità proposte dal Web Framework, è possibile creare utenti "admin" che possono monitorare e gestire il portale in tutti i suoi oggetti: App, Utenti, Ordini, etc.

App

L'App è il prodotto rappresentante l'applicazione in vendita.
Viene descritta tramite proprietà specifiche, in maniera non esaustiva possiamo vedere:

1. Nome app
2. App Identifier (App ID)
3. Versione
4. Tipo dispositivo

Ci possono essere applicazioni con lo stesso App Identifier, in quanto la stessa applicazione può essere pubblicata sia per Android che iOS, la stessa App per piattaforme diverse ha lo stesso AppID. Le applicazioni vengono caricate dagli utenti registrati.

Ordini

Un utente registrato può effettuare degli acquisti tramite il portale aggiungendo un app o più nel carrello. Una volta aggiunta, tramite il bottone dedicato, può vedere il Carrello e procedere al Checkout. Se viene richiesto la visione del carrello, quand'è vuoto viene mostrato gli errori.
Durante la fase di checkout viene richiesto all'utente l'indirizzo di fatturazione (oppure può utilizzare un indirizzo di default se ha già acquistato altre volte) ed il metodo di pagamento. Una volta che l'utente effettua l'acquisto gli viene mostrato un messaggio che lo invita al pagamento, una volta che lo sviluppatore vede il pagamento potrebbe evadere l'ordine. Attualmente esiste solo lo stato evaso. Una volta inserito l'indirizzo e completato l'acquisto il sistema salva l'indirizzo per l'utente nel suo profilo. L'utente può vedere la cronologia degli acquisti.

Il sistema a scopo esemplificativo evade direttamente tutti gli ordini una volta ricevuti.

Categorie

Aurora Store è dotata di categorie di divisione delle app, ed vengono utilizzate per categorizzare al meglio. Inoltre è presente una sezione di filtro che permette di vedere le applicazioni pubblicate esclusivamente su quella categoria. Ingloba sia le applicazioni per iOS che per Android.

Vendita

Per la vendita di un'applicazione è necessario il suo caricamento tramite il form adatto richiamato dal bottone *"Vendi un'app"*. La piattaforma richiede le informazioni riguardo l'applicazione e la pubblicherà nella categoria e sezione corretta.
Inoltre l'utente ha una sezione dove vedere le vendite che ha effettuato riassuntivo di :

- Numero applicazioni vendute.

- Prezzo di ogni applicazione.
- Nome applicazione.
- Stato vendita.

Inoltre l'utente può procedere a vedere le sue informazioni sotto forma di grafico riassuntivo delle vendite per giorni.

Valutazioni

Un utente che ha acquistato un'app ha la possibilità di valutare essa, recandosi sull'app interessata e premendo il tasto *"Valuta"*.
Le valutazioni sono da 1 a 5. Il sistema calcolerà la media delle valutazioni per assegnare la valutazione finale. La valutazione finale è visibile a tutti gli utenti e verrà utilizzata anche dal *"Sistema di Raccomandazione"* per raccomandare le applicazioni.

Filtri

Nella sezione *"Tutte le apps"* è possibile visualizzare tutte le apps in vendita ed inoltre applicare dei filtri.
I filtri a disposizione sono i seguenti:

- Filtrare per apps Android
- Filtrare per apps iOS
- Filtrare per prezzo crescente
- Filtrare per prezzo decrescente

Sistema di raccomandazione

Il sistema di raccomandazione suggerisce in modo dinamico che prodotti visualizzare per l'utente che sta visualizzando la pagina iniziale (*homepage.html).*

Il testo delle sezioni viene mostrato dinamicamente in base al dispositivo dell'utente se è autenticato.

```
user = None
device_user = None
if request.user.is_authenticated:
    user = UserProfile.objects.filter(user=request.user).first()
if user:
    device_user = user.user_device
```

In modo tale che ad esempio se l'utente ha dispositivo iOS il testo per la sezione delle app iOS diventerà: *"Le migliori app per i tuoi dispositivi iOS"* mentre per la sezione Android vedrà *"Le migliori app per gli utenti Android"*. Mentre gli utenti anonimi avranno il testo della sezione neutro : *"Le migliori app per gli utenti iOS (*oppure *Android)".*

Il sistema raccomanda le applicazioni ordinandole per piattaforma (Android o iOS) e restituisce ordinate le prime 4 con le recensioni migliori. Mentre nella terza sezione, il sistema restituisce le app con il maggior numero di download indipendentemente dalla piattaforma.

```
recommendation_list = sorted(recommendation_list, key=lambda app_el: (
    app_el.sales_count), reverse=True)
ios_convenient_list = sorted(iosconvenient_list, key=lambda app_el: (
    app_el.raiting), reverse=True)
android_convenient_list = sorted(androidconvenient_list, key=lambda app_el: (
    app_el.raiting), reverse=True)
```

Aspetti tecnici

Introduzione Tecnica

Il progetto è stato sviluppato tramite l'utilizzo del Web Framework Django e altre tecnologie, tra cui HTML e CSS. Sono presenti piccoli script in Ajax e Javascript.

In fase di sviluppo, è stato utilizzato un semplice Dockerfile per agevolare lo sviluppo e per la sua comodità di portabilità.

Divisione progetto

Nel progetto è stata utilizzata una suddivisione logica delle varie “app” del progetto Django in modo tale che siano funzionali allo scopo macroscopicamente. Inoltre è stata creata una cartella contenente dei templates esterni, come descritto successivamente.

Aurorastore

Non è una divisione a livello logico ricercata ma bensì una convenzione della tecnologia Django, viene utilizzata come entry point e definizione delle caratteristiche del progetto. Tramite questa particolare app è stato configurato il progetto in tutte le sue parti.

Si cita in maniera non esaustiva le parti di maggiore interesse in cui si è intervenuti nel file *settings*:

- **TEMPLATES**: All’interno di questa proprietà sono stati configurati Templates e la loro posizione, questo permetterà al *Templates Engine* di trovarli ed utilizzarli.
    - Inoltre sono stati configurati due *Custom Context*, descritti nelle sezioni successive.
- **STATICFILES_DIRS**: All’interno di questa proprietà sono state definite le cartelle “static”.
- **MEDIA_ROOT**: Tramite questa proprietà è stata definita la cartella Media, utilizzata nei form per il salvataggio delle immagini.
- **LOGIN_REDIRECT_URL**: Tramite questa proprietà viene definita la pagina di redirect successiva al Login dell’utente.

Accounts

Ha la funzione di gestire la registrazione dell’utente e il suo indirizzo di fatturazione tramite le seguenti operazioni:

1. Modifica indirizzo fatturazione.
2. Eliminazione indirizzo fatturazione.
3. Visualizzazione sintetica di tutti gli indirizzi fatturazione.
4. Visualizzazione nel dettaglio di un indirizzo fatturazione.

Core

Contiene tutta la logica dell'applicazione e delle parti principali di essa, tra cui le seguenti macrofunzioni:

1. Profilo utente.
    a. Visualizzazione profilo personale dove sono contenute le proprie app.
    b. Visualizzazione profili altrui.
    c. Vendite effettuate.
        i. Visualizzazione in maniera grafica di essa.
    d. Acquisti effettuati.
2. App
    a. Valutazione.
    b. Eliminazione, Aggiunta, Modifica.
    c. Visualizzazione.
3. Gestione vendita
    a. Checkout
    b. Gestione ordini
4. Ricerca nella piattaforma
    a. App
    b. Utenti

Templates delle Apps

Ogni app ha una cartella chiamata Templates, in queste folder sono stati inseriti i *templates*, cioè file HTML contenenti Tag speciali che Django utilizza per generare dinamicamente il contenuto della pagina richiesta. Vengono utilizzati, a quanto detto in precedenza, anche per separare la logica delle applicazioni dalla presentazione del loro contenuto, questo rende più semplice la creazione e la manutenzione del progetto.

Questi template vengono richiamati dai metodi View (*views.py*).

Templates esterni alle Apps.

Esiste una cartella esterna alle apps chiamata Templates. Questo folder contiene dei template chiamati di “base”, in quanto vengono utilizzati senza necessariamente essere contestualizzati.

Ad esempio possiamo trovare:

- **base.html**: Questo componente è un “blocco” di HTML che implementa l’header che è in comune in tutte le pagine.
- **Folder registration:** Questa folder contiene tutte le pagine per il login/logout e gestione della password. Mentre come detto in precedenza, la parte di registrazione e di modifica degli indirizzi è legato all’account ed è presente nella corretta applicazione: Accounts.

Shared Context

Sono stati inseriti dei *shared context* per fare in modo che in qualsiasi template, compreso il *base.html,* fossero presenti le categorie e i tipi di dispositivi supportati.
La configurazione dei *shared context* è stata dichiarata nei Settings del progetto, sotto la dichiarazione dei Templates.

```
TEMPLATES = [
             . . .
            'OPTIONS': {
            'context_processors': [
                  . . .
                  'core.context.shared_context.shared_categories_context',
                  'core.context.shared_context.shared_devices_context'
      ], . . .
```

Essi sono dichiarati sotto la folder *context* dell’app core:

```
def shared_categories_context(request):
      return {'categories': AppCategories}
def shared_devices_context(request):
      return {'devices': Devices}
```

Tramite lo shared context, per esempio, il *base.html* può accedere alle categorie e può gestire in modo dinamico la generazione delle categorie nel menu a cascata, in modo tale che se viene aggiunta una categoria è subito presente nel menù senza andare ad aggiungerla manualmente nel template.

```
<div class="dropdown-menu" aria-labelledby="navbarDropdown">
   {% for category in categories %}
   <a class="dropdown-item" href="{% url 'categories' category %}">{{ category.label }}</a>
   <div class="dropdown-divider"></div>
   {% endfor %}
</div>
```

Folder component

E' stata creata nell'Apps core, nella sottocartella Templates una folder chiamata component, questa contiene dei componenti riutilizzabili durante la scrittura dei Template.

Ad esempio il componente che mostra l'applicazione inserita dagli utenti o comunque quando c'è richiesto di presentare un'app, essa è un componente riutilizzabile.
Questo porta i vantaggi di una maggiore semplicità di scrittura codice, manutenzione e portabilità. In questo modo viene semplificata anche la fase di sviluppo e modifica.

```
ore > templates > component > app_component.html > div.col-lg-3.col-md-6.mb-4 > div.card > div.card-body.text-center
<div class="col-lg-3 col-md-6 mb-4">
    <div class="card">
        <div class="view overlay">
            {% if app.image %}
            <img src="{{ app.image.url }}" class="card-img-top" width="100" height="200">
            {% endif %}
            <a href="{{ app.get_absolute_url }}">
                <div class="mask rgba-white-slight"></div>
            </a>
        </div>
        <div class="card-body text-center">
            <h3 class="h3">{{ app.get_category_display }}</h3>
            <h5 class="h5 h5">
                <strong>
                    <a href="{{ app.get_absolute_url }}" class="dark-grey-text">{{ app.name }}
                    </a>
                </strong>
            </h5>
            <h4 class="h4 font-weight-bold blue-text">
                {% if app.price == 0 %}
                Gratis
                {% else %}
                {{ app.price }}0 €
                {% endif %}
                {% if app.is_for_android %}
                <h6 style="color: red"> {{ app.get_device_label }} </h6>
                {% else %}
                <h6 style="color: green"> {{ app.get_device_label }} </h6>
                {% endif %}
                <strong>Valutazione : </strong>
                <strong style="color:orange;">{{ app.raiting }}</strong>
            </h4>
            <p class="text-muted">{{ app.data|timesince }} fa</p>
        </div>
    </div>
</div>
```

Models Enum

E' stato creato un particolare file chiamato *"models_enum.py"* che contiene due particolari oggetti che rappresentano le categorie e i device. Entrambi le classi vanno ad estendere "*models.TextChoices*" che permette di utilizzarlo comodamente come field in un form. La scelta inizialmente è caduta sugli Enum ma dopo alcune ricerche e prove si è capito che la scelta migliore è quella di estendere quella particolare classe di Django, per la sua grande comodità d'uso.

Questi oggetti inoltre permettono di dichiarare il campo in questo modo:

'name' = 'value', 'label'

Questo ha permesso di utilizzare delle *label* personalizzate per mostrare a video la possibile scelta. Inoltre ha dato il vantaggio di utilizzare l'oggetto praticamente ovunque senza portarsi dietro e copiare ovunque l'oggetto, quindi una miglioria di portabilità. Successivamente è stato utilizzato anche nella home per creare il menù a cascata come precedentemente riferito. Inoltre tramite i shared context sono iniettati in tutti i contesti.

```
'''Enums with categories apps'''


class AppCategories(models.TextChoices):
    cat_sport = 'sport', 'Sport'
    cat_game = 'game', 'Giochi'
    cat_utility = 'utility', 'Utility'
    cat_arcade = 'arcade', 'Arcade'
    cat_weather = 'weather', 'Meteo'
    cat_health = 'health', 'Medicina'
    cat_news = 'news', 'News'
    cat_other = 'other', 'Altro'


'''Enums with devices'''


class Devices(models.TextChoices):
    ios = 'ios', 'iOS'
    android = 'android', 'Android'
```

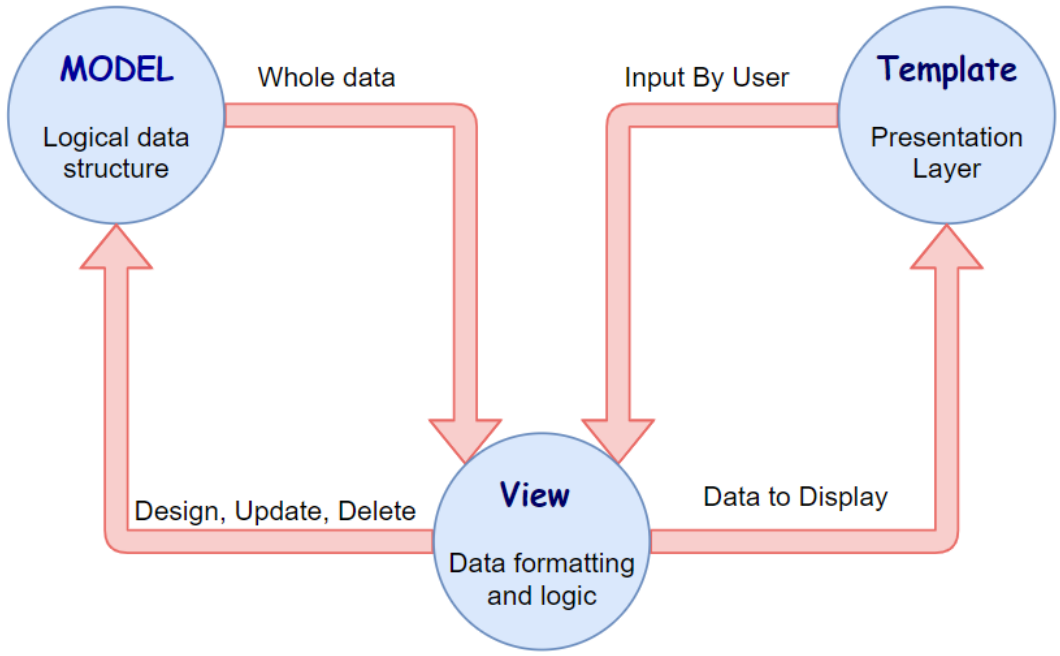
Messaggi ed Error Handling

I messaggi e gli errori vengono mostrati a video all'interno del *base.html* al di sotto dell'header. I messaggi sono di tre livelli, *"Info", "Warning"* ed *"Error".* Questi vengono mostrati a video con colori differenti (tramite classi *bootstrap*), in base alla loro criticità.

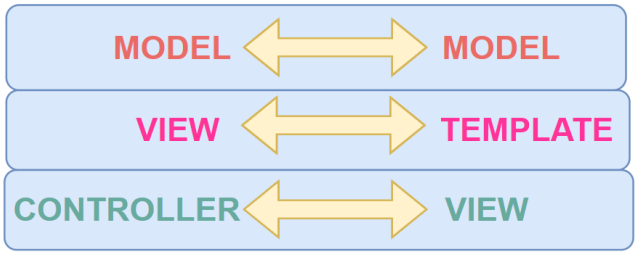
```
{% if messages %}
{% for message in messages %}
{% if message.level == DEFAULT_MESSAGE_LEVELS.ERROR %}
<div class="alert alert-danger alert-dismissible fade show" role="alert">
   {{ message }}
   <button type="button" class="close" data-dismiss="alert" aria-label="Close">
      <span aria-hidden="true">&times;</span>
   </button>
</div>
{% endif %}
{% if message.level == DEFAULT_MESSAGE_LEVELS.INFO %}
<div class="alert alert-success alert-dismissible fade show" role="alert">
   {{ message }}
   <button type="button" class="close" data-dismiss="alert" aria-label="Close">
      <span aria-hidden="true">&times;</span>
   </button>
</div>
{% endif %}
{% if message.level == DEFAULT_MESSAGE_LEVELS.WARNING %}
<div class="alert alert-warning alert-dismissible fade show" role="alert">
   {{ message }}
   <button type="button" class="close" data-dismiss="alert" aria-label="Close">
      <span aria-hidden="true">&times;</span>
   </button>
</div>
{% endif %}
{% endfor %}
{% endif %}
```

MTV e MVC

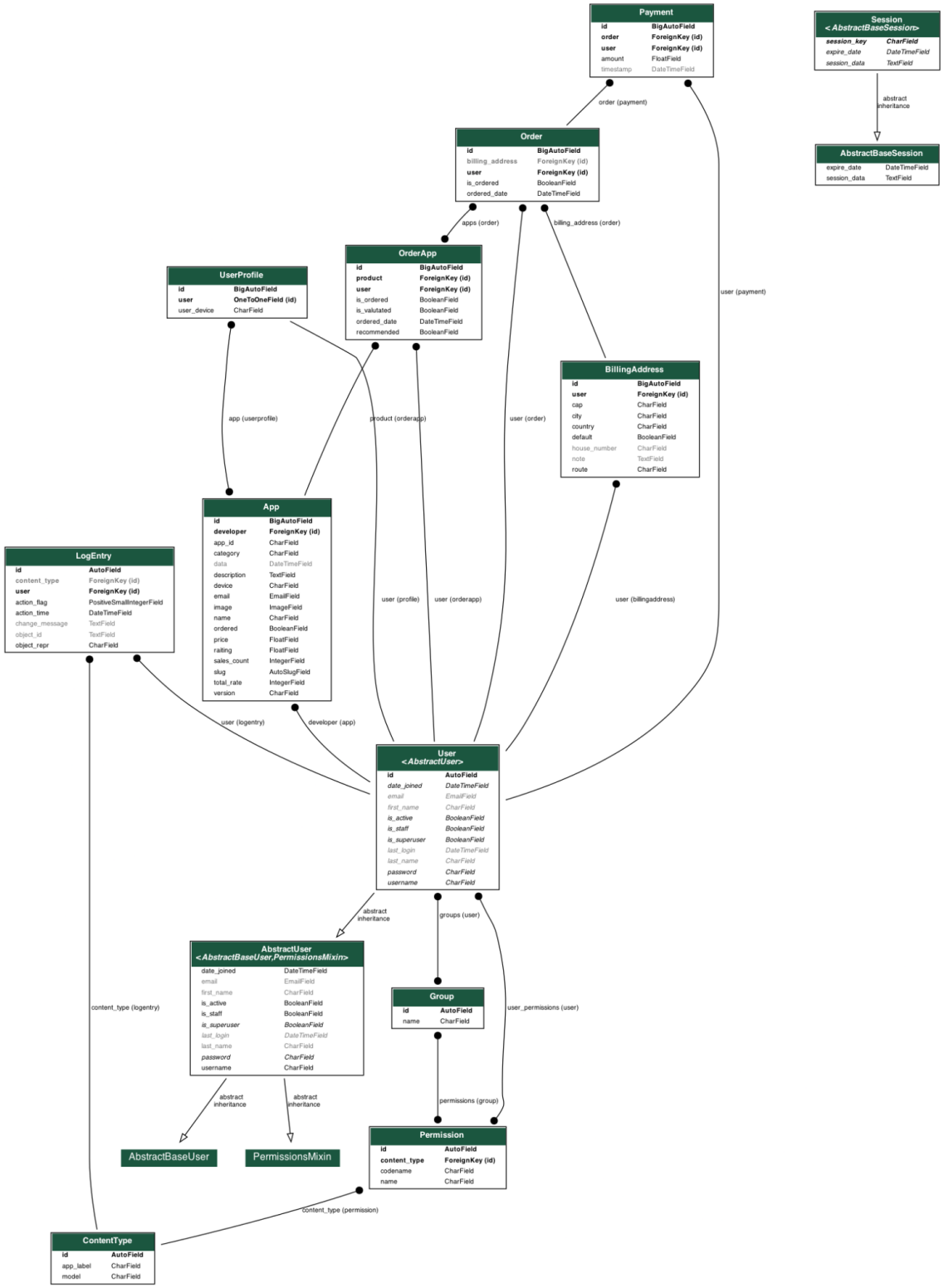
Il progetto utilizza il design pattern *MVC (Model - View - Controller)* ma tramite Django è possibile implementare una versione *MTV (Model - Template - View)*.
Questo perché come detto in precedenza il progetto utilizza dei template (HTML) che vengono mostrati dalle View, e queste entità utilizzano i Models per estrapolare i dati.



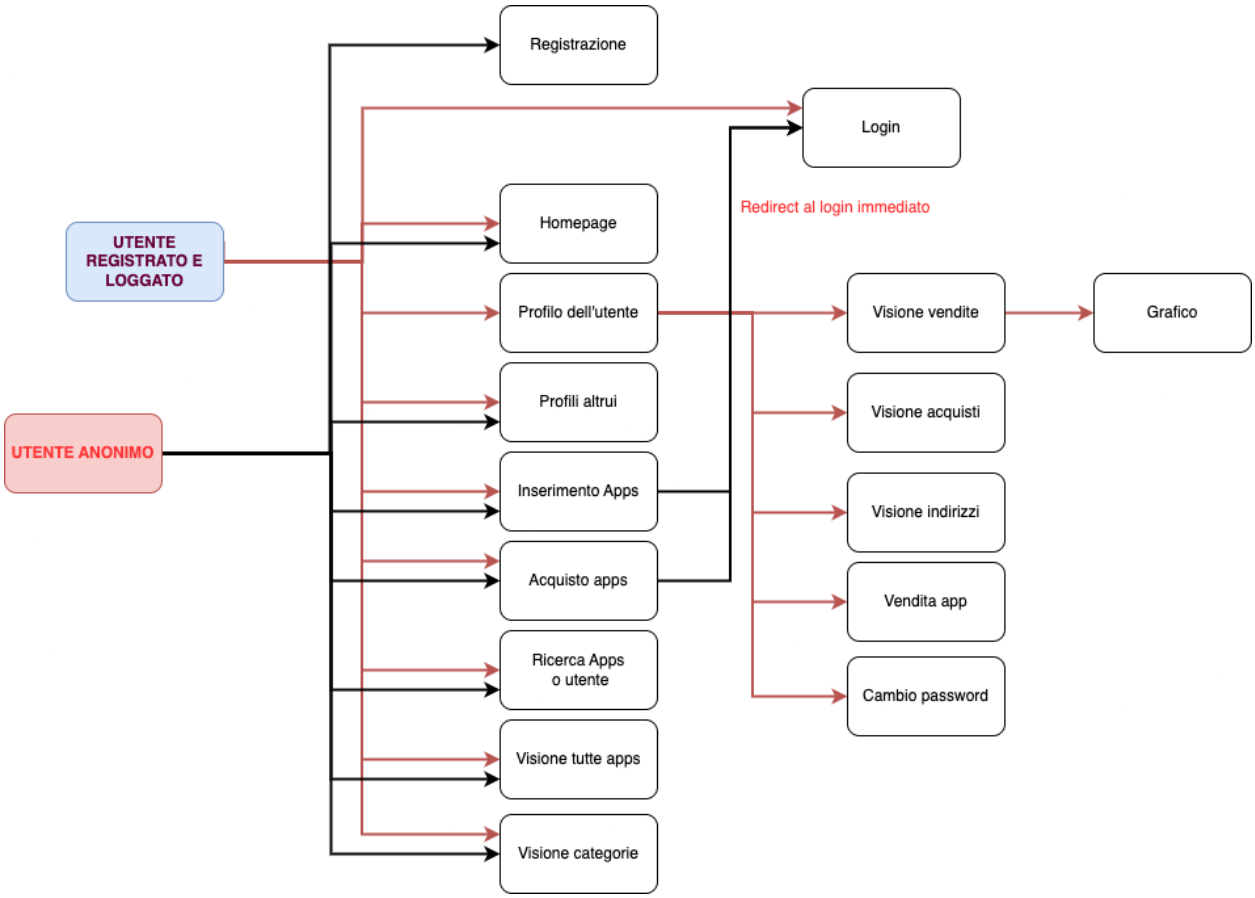
La corrispondenza tra MVC e MTV è la seguente:



UML delle classi



Schema funzionale



Come si può notare dallo schema l'utente non registrato può "vedere" il bottone per accedere ad esempio alla vendita delle app ma non può farlo senza previo login, quindi subisce immediatamente un redirect alla pagina dedicata.

Acquisto Apps: si parla della macrofunzione del processo di acquisto quindi compresa anche la funzione del Checkout dell'ordine.

Test

In totale sono presenti 8 test.
Uno per l'apps "Accounts" mentre gli altri sette sono dedicati alla parte di "Core".
Ogni file *tests.py* ha la propria funzione di setup dei test dove vengono creati e inizializzati gli oggetti richiesti per i test.

Test del login - test_login(self)

Nel test del login viene effettuato il tentativo di accesso corretto ed errato.
Aspettativa: Con credenziali errate il l'accesso viene negato, con credenziali corrette viene consentito.

Test creazione app - test_app_empty_field_create(self):

Nel test viene effettuata la creazione di un'app senza riempire i campi.
Aspettativa: Non viene effettuato redirect alla pagina successiva e tutti i campi devono presentare l'errore *"Questo campo è obbligatorio"*.

Test creazione app - test_app_create(self):

Nel test viene effettuata la creazione di un'app con tutti i campi.
Aspettativa: Viene effettuato redirect alla pagina successiva (*homepage*) correttamente.

Test eliminazione app - test_app_delete(self):

Nel test viene effettuata il tentativo di eliminazione di un'app da utente loggato (tramite post e get) e da utente anonimo.
Aspettativa: Con utente loggato, tramite chiamata GET l'applicazione non viene eliminata ma mentre tramite POST l'eliminazione avviene correttamente. Mentre da utente loggato ma non proprietario dell'app, l'eliminazione non avviene.

Test redirect login - test_login_required(self):

Nel test avviene il tentativo di creazione di un'app da utente non loggato.
Aspettativa: Avviene il redirect sulla pagina di login.

Test visione profilo - test_user_profile(self):

Nel test viene richiesto di visualizzare il proprio profilo (utente loggato) ed un'altro utente da utente loggato e da utente anonimo
Aspettativa: La visualizzazione del profilo avviene correttamente sempre.

Test cambio indirizzo - test_address_change(self):

Nel test viene richiesto il cambio indirizzo da utente loggato, tentato cambio senza riempire i campi, tentato cambio indirizzo con campi riempiti e indirizzo non proprio

Aspettativa: Il cambio tramite chiamata GET fallisce. Il cambio senza riempire i parametri, tutti i campi devono presentare l'errore *"Questo campo è obbligatorio"*. Il tentato cambio indirizzo con campi riempiti va a buon fine. Il tentato cambio indirizzo non proprio viene negato.

Test eliminazione indirizzo - test_address_delete(self):

Nel test viene richiesto l'eliminazione indirizzo da utente loggato ed indirizzo non proprio

Aspettativa: L'eliminazione tramite chiamata GET fallisce. La tentata eliminazione del proprio indirizzo tramite POST va a buon fine. La tentata eliminazione indirizzo non proprio viene negato.

Tecnologie utilizzate

Il progetto è stato sviluppato su Visual Studio Code.

Django

Come ormai citato più volte, è stato utilizzato il Web Framework Django, offre una vasta gamma di funzionalità e strumenti integrati per lo sviluppo di applicazioni web. È un framework altamente personalizzabile e scalabile che rende facile la creazione di applicazioni robuste e affidabili. Inoltre, Django fornisce una solida sicurezza integrata e un modello di sviluppo semplice che rende più efficiente lo sviluppo e la manutenzione del codice.

Inoltre sono state utilizzate i seguenti package:

- **django-autoslug**: E' stata utilizzata questa estensione per rendere univoche le applicazioni a partire dal App ID.
- **django-crispy-forms**: Per la creazione dei form in maniera semplice e affidabile, altamente integrato al Web Framework.

- **Pillow**: Per la gestione delle immagini.
- **django-extensions**: Per la creazione del UML delle classi.

Bootstrap

Bootstrap è stato utilizzato perché offre un insieme completo di componenti HTML, CSS e JavaScript pre-progettati che possono essere utilizzati per creare rapidamente un layout esteticamente appagante e responsive per le pagine web.

Javascript

Utilizzato per l'elaborazione delle pagine e delle informazioni al suo interno. Utilizzato anche per gestire le reazioni delle pagine in base a determinati eventi.

Ajax

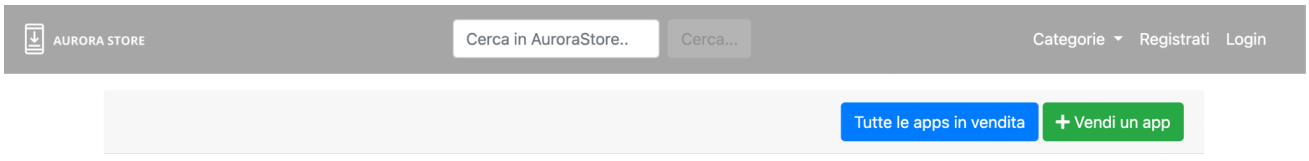
Utilizzati per elaborare i dati in maniera Asincrona, tramite una comunicazione da Browser e Server. Ad esempio nel progetto è stato utilizzato il grafico.

Formattazione PEP8 Python

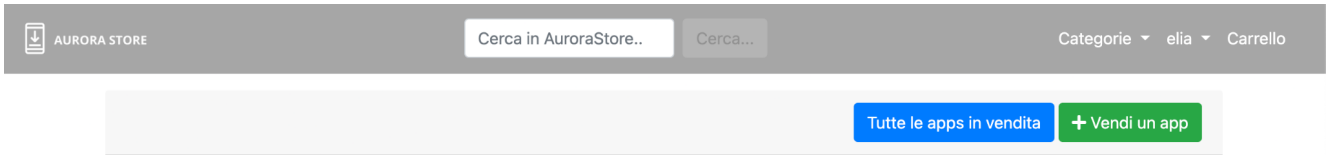
Nella realizzazione di questo progetto è stato utilizzato il formato di codifica PEP8 per tutto il codice scritto in Python. PEP8 è una guida per la formattazione del codice che mira a garantire un'uniformità nella scrittura del codice Python e a rendere il codice più leggibile e facile da mantenere.

Risultati UI

Header utente non autenticato



Header utente autenticato



Applicazione e la sua visualizzazione dettagliata

Tutte le apps in vendita con filtri

Conclusioni personali

Il progetto che ho sviluppato è stato molto appassionante in quanto le app sono la mia passione e anche uno dei miei due lavori. C'è la possibilità di ampliarlo ulteriormente tramite l'inserimento di recensioni degli utenti e la gestione delle procedure di rimborso e di reso. Il risultato mi soddisfa sia dal punto di vista estetico che funzionale.

Inoltre mi ha fatto molto piacere ampliare il mio Stack tecnologico tramite l'utilizzo di Django.

Inoltre la scrittura di questa relazione mi ha permesso di allenarmi per la mia futura Tesi.