

FootX

Progetto di Tani Federico

Indice

1. Descrizione del Progetto
2. Diagrammi
3. Tecnologie utilizzate
4. Test
5. Alcuni Screenshot

1. Descrizione del progetto

L'idea del progetto in questione fa riferimento ad un qualsiasi sito di compravendita. In questo caso si tratta di compravendita di un tipo di scarpe dette 'Sneakers'. La compravendita di questi prodotti è nata perchè la domanda è maggiore dell'offerta. Questo ha portato la nascita di un vero e proprio mercato secondario per gli appassionati.

Il progetto è stato diviso nelle seguenti App:

App Django:

- accounts (viene gestita la registrazione dell'utente)
- core (gestisce il cuore del progetto, cioè la parte relativa alla compravendita delle Sneakers)

Tipologie di utenti presenti:

-**Guest** : non è autenticato e quindi non ha possibilità di vedere e navigare in tutte le pagine del sito. Può visualizzare la HomePage ma non la parte relativa ai prodotti consigliati(non essendo un utente registrato non può comprare nessun prodotto e quindi il sito non ha prodotti da consigliarli).

Può utilizzare la ricerca per trovare un prodotto oppure trovare un utente e eventualmente vedere i prodotti che vende. Può visualizzare tutti i prodotti in vendita anche nel dettaglio e navigare per categoria.

-**Utente registrato** : si è registrato cliccando sul tasto 'Registrazione' e ha inserito tutte le informazioni necessarie(nome utente , password e indirizzo mail). Avvenuta la registrazione può navigare nel sito(fare tutto ciò che può fare un guest), mettere in vendita dei propri prodotti , comprare i prodotti di altri utenti.

L'admin del sito inoltre potrà eliminare gli annunci che non sono idonei(per esempio se si tratta di merce contraffatta o non rispettano le condizioni del sito), oppure eliminare direttamente un utente.

Gestione degli ordini :

Un utente registrato come detto precedentemente può acquistare dei prodotti. Se un utente non ha ancora nessun prodotto nel carrello, non ha un ordine attivo. Aggiungendo un prodotto nel carrello verrà creato un ordine. Nella pagina relativa al carrello può eventualmente eliminare un prodotto che ha aggiunto e decidere di procedere andando al checkout, altrimenti può continuare lo shopping e aggiungere altri prodotti al carrello. Inoltre in questa pagina vedrà il totale che andrà infine a pagare e che è la somma del prezzo dei prodotti presenti nel carrello.

Se l'utente decide di procedere col checkout verrà indirizzato in una pagina dove dovrà inserire i dati dell'indirizzo di spedizione (Stato, Città, CAP, Via e numero civico,interno ed eventuali note). Il campo interno ed il campo eventuali note non

sono obbligatori. Dopo aver inserito l'indirizzo, può decidere di salvarlo come default. Gli indirizzi salvati possono essere visualizzati, modificati (per renderli di default o meno) ed eliminati dall'utente. Inoltre un utente può avere un solo indirizzo di default. Al posto di compilare ogni volta il form dell'indirizzo, può decidere di utilizzare quello di default (sempre se è presente). Nella stessa pagina deve selezionare il metodo di pagamento (pagamento alla consegna). Dopo di che può finalizzare l'ordine oppure tornare al carrello se per caso ha avuto dei ripensamenti. Inoltre ogni utente avrà la sua cronologia degli ordini che ha effettuato, potrà vedere la data di emissione, il totale che ha speso e i prodotti di ogni ordine.

Come vendere un prodotto:

Un utente registrato può decidere di vendere dei prodotti. Per mettere l'annuncio del prodotto che vuole vendere deve cliccare il bottone 'Aggiungi Sneaker' che è presente nella homepage. Dovrà quindi mettere il nome del prodotto, il prezzo a cui lo vuole vendere, la taglia europea della scarpa, un'immagine, la condizione (se è un articolo nuovo oppure è stato usato) e una descrizione del prodotto (essa può essere utilizzata per descrivere la vestibilità, cioè se per esempio la scarpa in questione veste una taglia più stretta del normale, oppure il lavaggio). Dopo aver messo in vendita il prodotto, esso sarà visibile all'interno del sito e nel profilo dell'utente. L'utente può quindi visualizzare i suoi annunci dal suo profilo, decidere di cambiare il nome, la taglia, il prezzo, oppure di eliminare l'annuncio dal sito. Quando venderà dei prodotti, essi saranno visibili nella sezione 'le mie vendite'.

Recommendation System:

Il sistema di raccomandazione che ho deciso di implementare si basa sugli ordini che effettua un utente. È stata necessaria la creazione di un modello 'Item consigliato'

```

class ItemConsigliato(models.Model):
    num_item = models.PositiveIntegerField(default=0)
    condizioneN = models.PositiveIntegerField(default=0)
    condizioneU = models.PositiveIntegerField(default=0)
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    prezzo = models.FloatField(validators=[MinValueValidator(0.0)], default=0)
    sum_prezzo = models.FloatField(default=0)

    def __str__(self):
        return self.user.username

    class Meta:
        verbose_name_plural = 'ItemConsigliati'

```

Il modello in questione salva i vari dati di ogni ordine, tiene conto di quanti item ha ordinato un utente, del numero di item con condizione uguale a 'nuova' e del numero di item con condizione uguale a 'usata' e tiene conto della media dei prezzi dei vari item acquistati. Poi viene creata una lista di raccomandazioni, e all'interno di essa vengono salvati gli item che soddisfano delle particolari richieste, ovvero è una combinazione tra la condizione e tra il prezzo dell'item. Prima di tutto in base alla media del prezzo salvata nel modello 'ItemConsigliato' viene controllato se l'item ha un prezzo che è vicino a quello della media, cioè può essere al massimo la media + 100 oppure non può essere minore alla media -100. Poi viene controllata la condizione, se un utente ha ordinato più di due item con la stessa condizione e se gli item ordinati con questa condizione sono maggiori rispetto a quelli ordinati che hanno un'altra condizione. Se vengono soddisfatte queste richieste l'item viene aggiunto alla lista di raccomandazione. Gli items consigliati all'utente vengono mostrati da quello col prezzo minore.

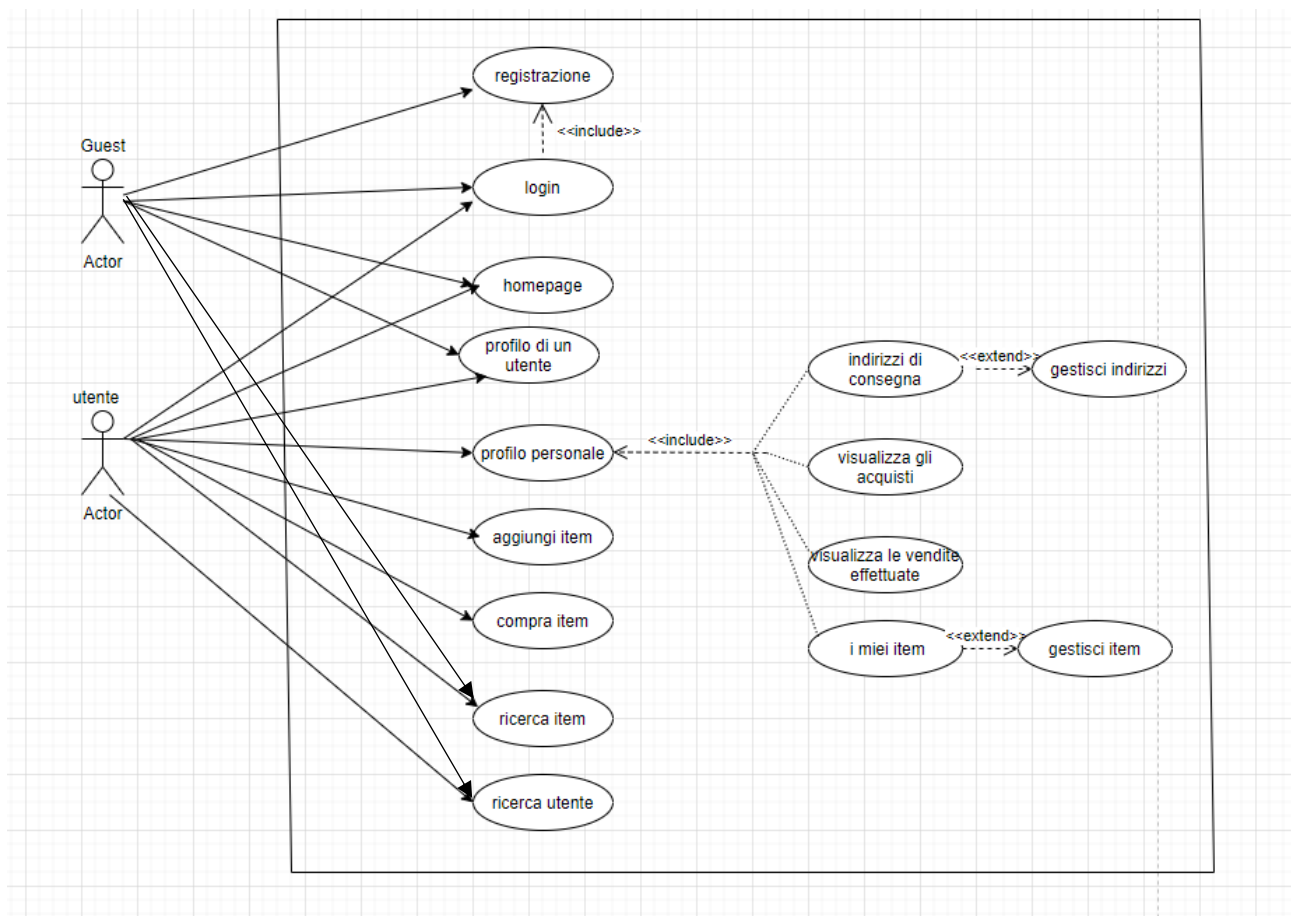
Homepage:

Nella homepage non viene mostrata la lista di tutti gli item ma vengono mostrate delle sezioni diverse contenenti gli item. In una sezione vengono mostrati gli item più convenienti con condizione uguale a 'nuova', in un'altra gli item più convenienti con condizione uguale a 'usata'. Con più convenienti si intendono quelli che hanno un prezzo minore o uguale a 300 euro. L'ultima sezione è quella

contenente gli item consigliati che fa riferimento al Recommendation System. Quest'ultima può essere vista solamente dagli utenti loggati.

2. Diagrammi

2.1 Diagramma dei casi d'uso

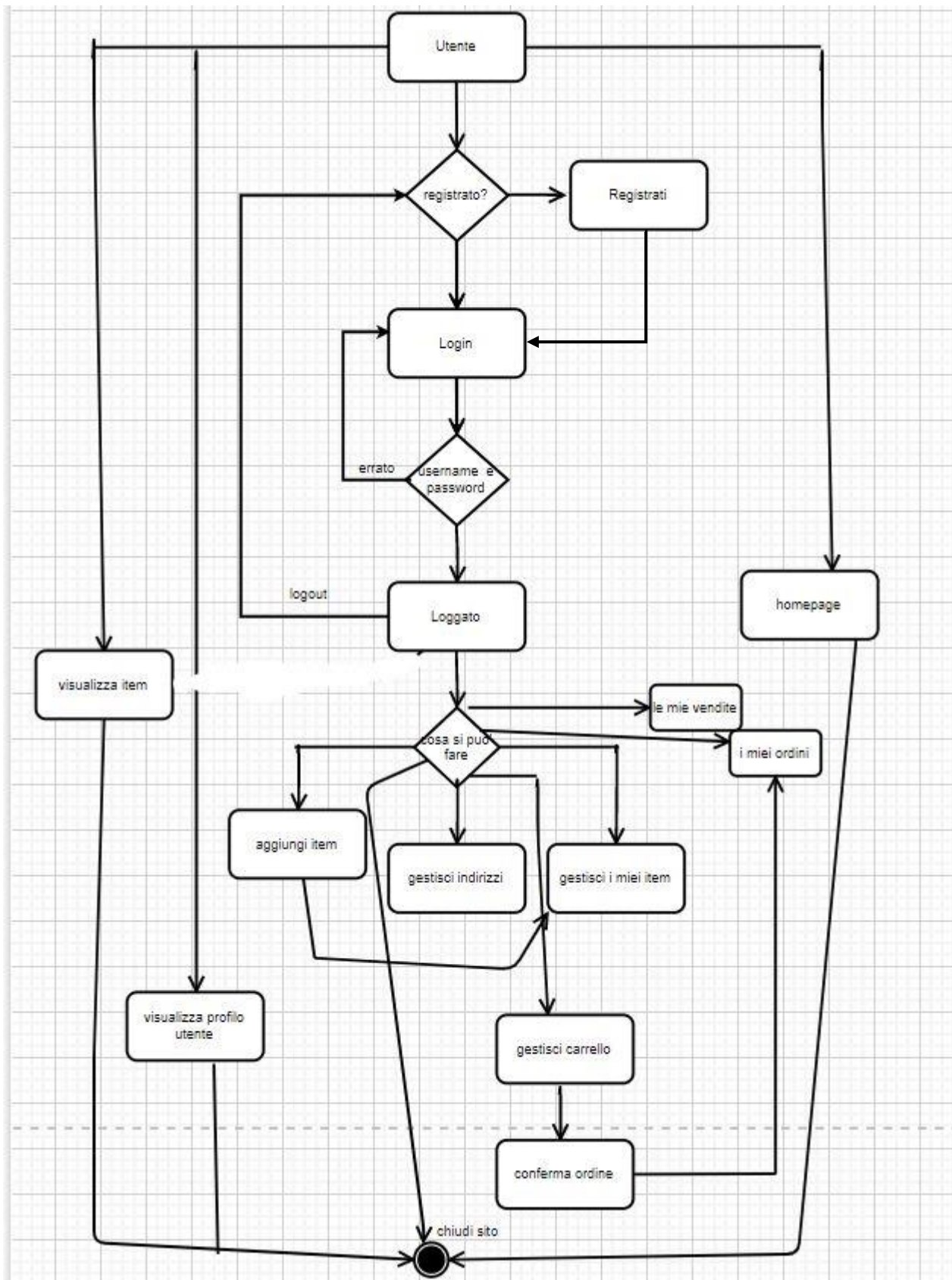


Scenario di un caso d'uso:

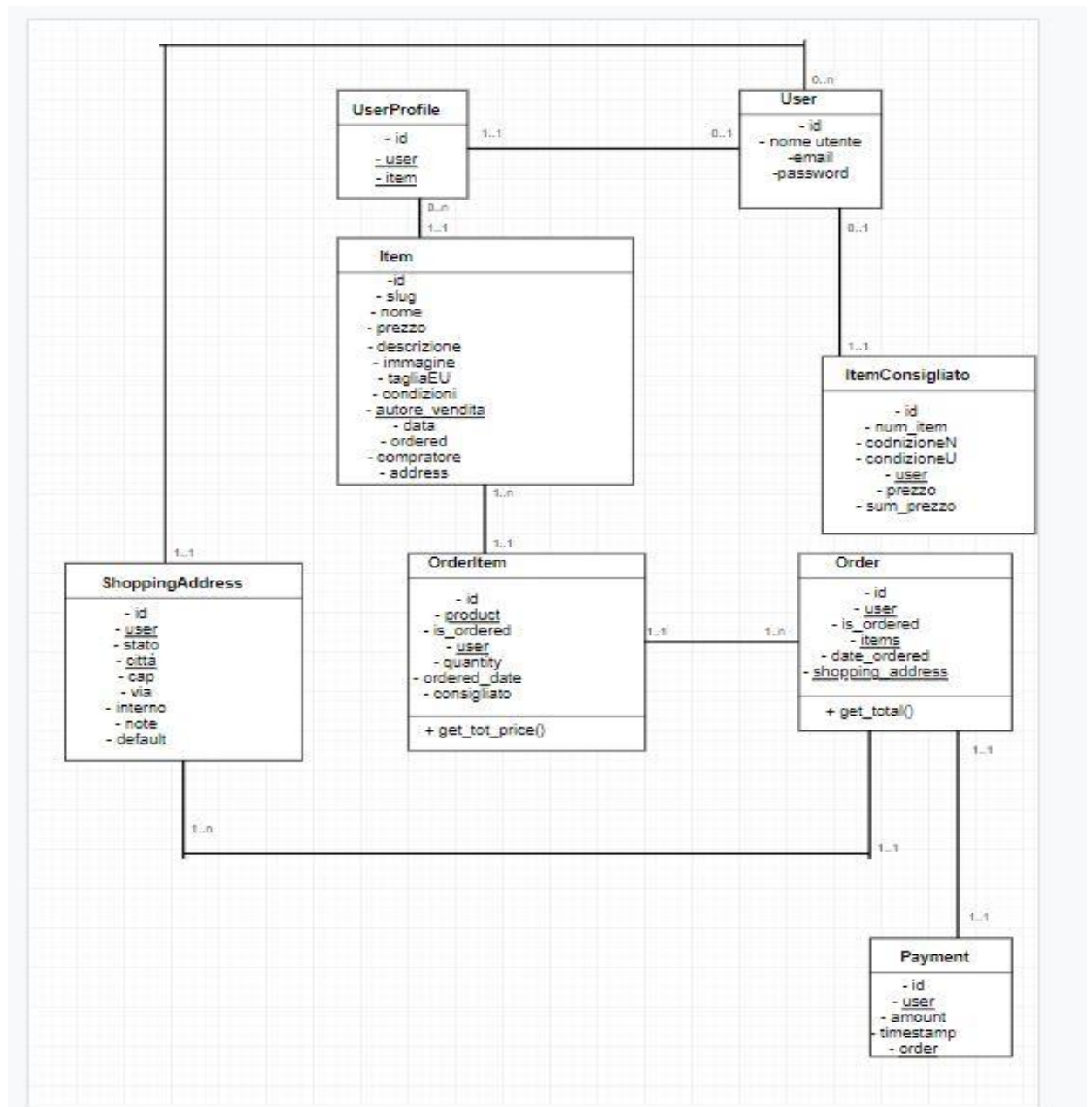
Giuseppe D'Alessandro, il 20 novembre decide di entrare nel sito footX per vedere se qualcuno vende un paio di scarpe che sta cercando da tanto tempo. Si logga correttamente mettendo il suo username e la sua password. Nella home vede se è presente quello che cerca, ma senza trovare quello che vuole, scrive il nome delle prodotto nella barra di ricerca. Gli appare l'annuncio di un utente da cui aveva già acquistato precedentemente, cosa che lo rassicura e decide di aggiungere al carrello il prodotto. A questo punto dopo aver inserito un indirizzo effettua l'ordine con successo. Per curiosità poi controlla gli altri ordini che ha effettuato in passato e

controlla quali item ha acquistato dallo stesso utente da cui ha comprato quest'ultimo prodotto.

2.2 Diagramma di attività



2.3 Diagramma delle classi



3.Tecnologie usate

Django:

Questo progetto è basato su Django, un framework MVC (Model View Controller) per lo sviluppo Web

La parte Model di Django è implementata direttamente come collezione di classi Python che andranno a rappresentare le tabelle del database.

La parte View tratta di funzioni Python che gestiscono il flusso dell'applicazione: grazie a essa possiamo definire comportamenti che le pagine avranno in funzione all'interazione con l'utente.

La parte Controller è realizzata tramite i file urls.py, che permettono di mappare le URL sulle opportune risorse richieste dall'utente.

Django di default presenta un'interfaccia di amministrazione che permette di gestire comodamente il database dell'applicazione.

Uno dei punti di forza di Django è la riusabilità dei componenti: in particolare permette di dividere il progetto in varie app in modo da facilitare la riusabilità e la leggibilità del codice.

Javascript:

Javascript è un linguaggio di programmazione, interpretato, orientato agli oggetti. E' conosciuto come linguaggio di scripting client-side per pagine web, e può essere utilizzato per dare un design e stabilire il comportamento delle pagine web quando viene scatenato un particolare evento da parte dell'utente.

Ajax:

È una tecnica di sviluppo software per la realizzazione di applicazioni web interattive. Lo sviluppo di applicazioni HTML con AJAX si basa su uno scambio asincrono di dati in background fra web browser e server, che consente l'aggiornamento dinamico di una pagina web senza esplicito ricaricamento da parte dell'utente. Lo scambio di dati è asincrono perché essi sono richiesti al server e caricati in background senza interferire con il comportamento della pagina esistente

Bootstrap:

Bootstrap è un framework rappresenta una delle soluzioni più utilizzate

per la progettazione di template per il web, soprattutto per quanto riguarda lo sviluppo responsive.

4.Test

Test nell'app accounts:

Ho testato la registrazione di un utente controllando che tutti i campi vengano inseriti correttamente, controllando anche se le due password inserite corrispondano, altrimenti si genera un errore.

```
def test_registrazione(self):
    # Test POST invalid data
    response = self.client.post('/accounts/registrazione/', {})
    self.assertFormError(response, 'form', 'username', 'Questo campo è obbligatorio.')
    self.assertFormError(response, 'form', 'email', 'Questo campo è obbligatorio.')
    self.assertFormError(response, 'form', 'password1', 'Questo campo è obbligatorio.')
    self.assertTemplateUsed(response, 'accounts/registrazione.html')
    self.assertEqual(response.status_code, 200) # codice 200: la richiesta ha avuto successo

    response = self.client.post('/accounts/registrazione/', {'username':self.dummy_user.username, 'email':self.dummy_user.email,
                                                                'password1':'ciaoone', 'password2':'wrong'})

    self.assertFormError(response, 'form', 'password2', 'I due campi password non corrispondono.')

    form_data = {'username':'user', 'email':'mail', 'password1':'vigorsol', 'password2':'vigorsol'}
    form = FormRegistrazione(data=form_data)
    self.assertTrue(form.is_valid())
```

Ho testato il login di un utente controllando che le credenziali inserite siano corrette, se sono corrette il login viene effettuato correttamente.

```
def test_login(self):
    fake_credential = {'username':'ciao', 'password':'ciao'}
    true_credential = {'username':'dummy', 'password':'nothings'}
    t_cred = self.client.login(**true_credential)
    f_cred = self.client.login(**fake_credential)

    self.assertTrue(t_cred)
    self.assertFalse(f_cred)
```

Test nell'app core:

Ho testato la corretta visualizzazione di un item che ritorna una risposta http 200 (OK)

```
def test_visualizzaItem(self):  
    """  
    controllo che l'utente riesca a visualizzare il suo annuncio dell'item appena creato  
    """  
    self.client.login(**self.credential)  
    response = self.client.get('/products/' + self.item.slug)  
    self.assertTemplateUsed(response, 'core/item.html')  
    self.assertEqual(response.status_code, 200)
```

La creazione di un item verificando che i campi obbligatori siano rispettati

```
def test_CreaItem(self):  
    """  
    Verifico che nella creazione dell'item i campi obbligatori siano rispettati  
    """  
    self.client.login(**self.credential)  
    response = self.client.post('/nuovo-item/', {})  
    self.assertFormError(response, 'form', 'nome', 'Questo campo è obbligatorio.')  
    self.assertFormError(response, 'form', 'prezzo', 'Questo campo è obbligatorio.')  
    self.assertFormError(response, 'form', 'categoria', 'Questo campo è obbligatorio.')  
    self.assertFormError(response, 'form', 'descrizione', 'Questo campo è obbligatorio.')  
    self.assertFormError(response, 'form', 'immagine', 'Questo campo è obbligatorio.')  
    self.assertFormError(response, 'form', 'tagliaEU', 'Questo campo è obbligatorio.')  
    self.assertFormError(response, 'form', 'condizioni', 'Questo campo è obbligatorio.')  
  
    self.assertTemplateUsed(response, 'core/aggiungi_prodotto.html')  
    self.assertEqual(response.status_code, 200) # verifica per capire se il template utilizzato è quello corretto  
  
    response = self.client.post('/nuovo-item/',  
                                {'nome': 'J4', 'prezzo': '50', 'categoria': 'A', 'descrizione': 'v',  
                                 'immagine': 's',  
                                 'tagliaEU': '45', 'condizioni': 'U'})  
    self.assertTemplateUsed(response, 'core/aggiungi_prodotto.html')  
    self.assertEqual(response.status_code, 200)
```

Ho testato la possibilità di modificare alcuni campi di un item creato. Le modifiche possono essere apportate solo dal creatore di quell'item e quindi se un utente non è il creatore, se proverà ad accedere alla pagina della modifica verifico che venga reindirizzato alla homepage.

```
def test_item_change(self):  
    # con utente autenticato  
    self.client.login(**self.credential)  
    response = self.client.get('/item/' + str(self.item.id) + '/modifica/')  
    self.assertEqual(response.status_code, 200)  
    response = self.client.post('/item/' + str(self.item.id) + '/modifica/', {})  
    self.assertFormError(response, 'form', 'nome', 'Questo campo è obbligatorio.')  
    self.assertFormError(response, 'form', 'prezzo', 'Questo campo è obbligatorio.')  
    self.assertFormError(response, 'form', 'tagliaEU', 'Questo campo è obbligatorio.')  
    response = self.client.post('/item/' + str(self.item.id) + '/modifica/',  
                                {'nome': 'mod', 'prezzo': '50', 'tagliaEU': '38'})  
    self.assertRedirects(response, '/user/' + self.user.username + '/')  
    self.client.logout()  
  
    # con utente autenticato ma non creatore  
    self.client.login(**self.credential2)  
    response = self.client.get('/item/' + str(self.item.id) + '/modifica/')  
    self.assertTemplateUsed(response, 'core/homepage.html')  
    self.assertTemplateNotUsed(response, 'core/modifica_item.html')
```

Ho testato la possibilità di eliminare un item. L'item può essere eliminato solo dal suo creatore e quindi se un utente non è il creatore, se proverà ad accedere alla pagina per eliminare l'item, verifico che venga reindirizzato alla homepage.

```
def test_item_delete(self):
    # con utente autenticato
    self.client.login(**self.credential)
    response = self.client.get('/item/' + str(self.item.id) + '/delete/')
    self.assertEqual(response.status_code, 200)
    response = self.client.post('/item/' + str(self.item.id) + '/delete/', {})
    self.assertRedirects(response, '/user/' + self.user.username + '/')
    self.client.logout()

    # con utente autenticato ma non creatore
    self.client.login(**self.credential2)
    response = self.client.get('/item/' + str(self.item.id) + '/delete/')
    self.assertTemplateNotUsed(response, 'core/deleteitem.html')
```

Ho verificato che per creare un item è necessario essere loggati al sito. Viene ritornata una risposta http 302.

```
def test_login_required(self):
    '''test on login_required sulla creazione di un item'''
    response = self.client.get('/nuovo-item/')
    self.assertRedirects(response, '/accounts/login/?next=/nuovo-item/')
    # 302 --> FOUND: pagina esiste ma non puoi entrarci
    self.assertEqual(response.status_code, 302)
```

Invece se è stato fatto il login, si può creare un item e viene ritornata una risposta http 200.

```
def test_new_item(self):
    self.client.login(**self.credential)
    response = self.client.get('/nuovo-item/')
    self.assertEqual(response.status_code, 200)
```


Ho testato il corretto reindirizzamento nella pagina del profilo di un utente. Se l'utente è loggato e vuole visualizzare il suo profilo verrà reindirizzato nella pagina corretta e che utilizza uno specifico template. Se l'utente vuole accedere alla pagina profilo di un altro utente verrà reindirizzato nella pagina corretta ma che usa un template diverso.

```
def test_userProfileView(self):
    # con utente autenticato
    # su il tuo profilo
    self.client.login(**self.credential)
    response = self.client.get('/user/' + self.user.username + '/')
    self.assertTemplateUsed(response, 'core/profilo.html')
    self.assertTrue(response.status_code, 200)

    # sul profilo degli altri
    response = self.client.get('/altriuser/' + self.user2.username + '/')
    self.assertTemplateNotUsed(response, 'core/profilo.html')
    self.assertTemplateUsed(response, 'core/user_profile.html')

    # con utente non autenticato
    self.client.logout()
    response = self.client.get('/altriuser/' + self.user.username + '/')
    self.assertTemplateUsed(response, 'core/user_profile.html')
    self.assertTrue(response.status_code, 200)
```

Ho testato il corretto cambiamento di alcuni campi di un indirizzo di consegna di un utente, le modifiche possono essere apportate solo dall'utente creatore dell'indirizzo.

```
def test_address_change(self):
    # con utente autenticato
    self.client.login(**self.credential)
    response = self.client.get('/user/address/' + str(self.address.id) + '/modifica/')
    self.assertEqual(response.status_code, 200)
    response = self.client.post('/user/address/' + str(self.address.id) + '/modifica/', {})
    self.assertFormError(response, 'form', 'città', 'Questo campo è obbligatorio.')
    self.assertFormError(response, 'form', 'via', 'Questo campo è obbligatorio.')
    self.assertFormError(response, 'form', 'stato', 'Questo campo è obbligatorio.')
    self.assertFormError(response, 'form', 'cap', 'Questo campo è obbligatorio.')
    response = self.client.post('/user/address/' + str(self.address.id) + '/modifica/',
                                {'città': 'modena', 'via': 'S50', 'stato': 'it', 'cap': '3434'})
    self.assertRedirects(response, '/user/' + self.user.username + '/address_page/')
    self.client.logout()

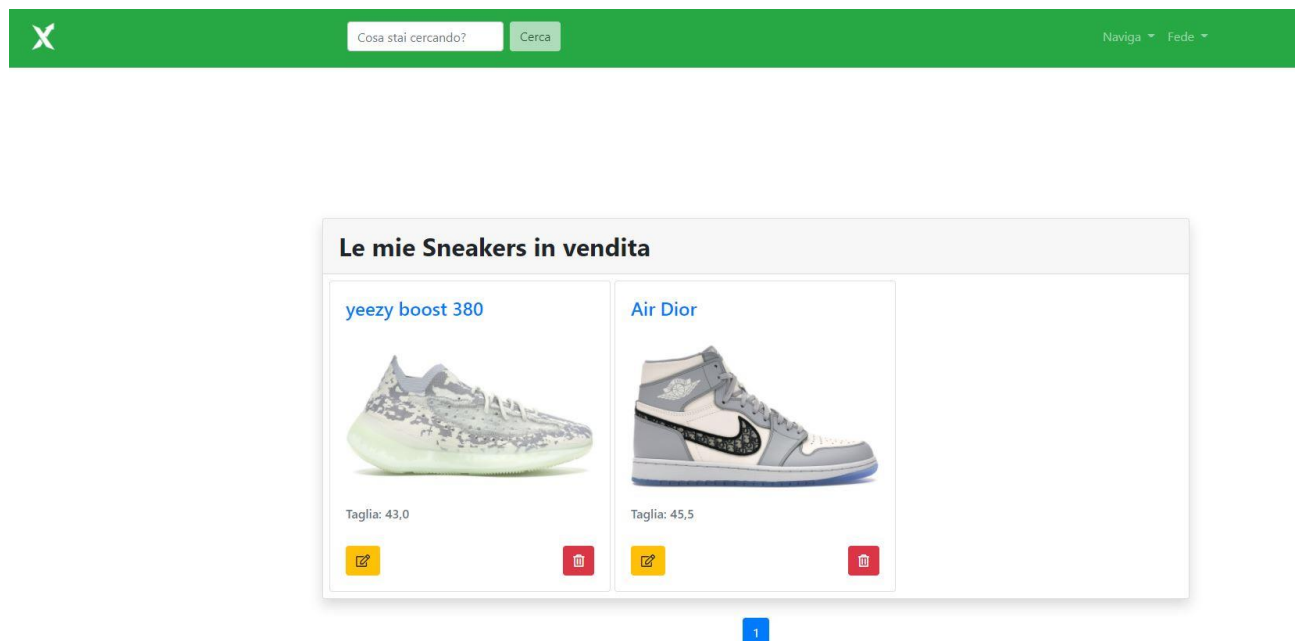
    # con utente autenticato ma non creatore
    self.client.login(**self.credential2)
    response = self.client.get('/user/address/' + str(self.address.id) + '/modifica/')
    self.assertTemplateUsed(response, 'core/homepage.html')
    self.assertTemplateNotUsed(response, 'accounts/modifica_address.html')
```

Ho testato l'eliminazione di un indirizzo di consegna, che può essere eliminato solamente dall'utente che lo ha creato.

```
def test_address_delete(self):  
    # con utente autenticato  
    self.client.login(**self.credential)  
    response = self.client.get('/user/address/' + str(self.address.id) + '/delete/')  
    self.assertEqual(response.status_code, 200)  
    response = self.client.post('/user/address/' + str(self.address.id) + '/delete/', {})  
    self.assertRedirects(response, '/user/' + self.user.username + '/address_page/')  
    self.client.logout()  
  
    # con utente autenticato ma non creatore  
    self.client.login(**self.credential2)  
    response = self.client.get('/user/address/' + str(self.address.id) + '/delete/')  
    self.assertTemplateNotUsed(response, 'accounts/address_delete.html')
```

5. Alcuni Screenshot

Pagina del profilo dell'utente Fede, vista dall'utente Fede




Pagina di un item in vendita

Indietro

Air Dior

Venditore: [@Fede](#) || Categoria: [Air Jordan](#)



Prezzo : 140,0 €

Taglia(EU): 45,5


Deadstock(New)

Descrizione: jordan 1 uscite nel 2020 , feet standard di una classica jordan 1.

Aggiungi al carrello

Carrello di un utente

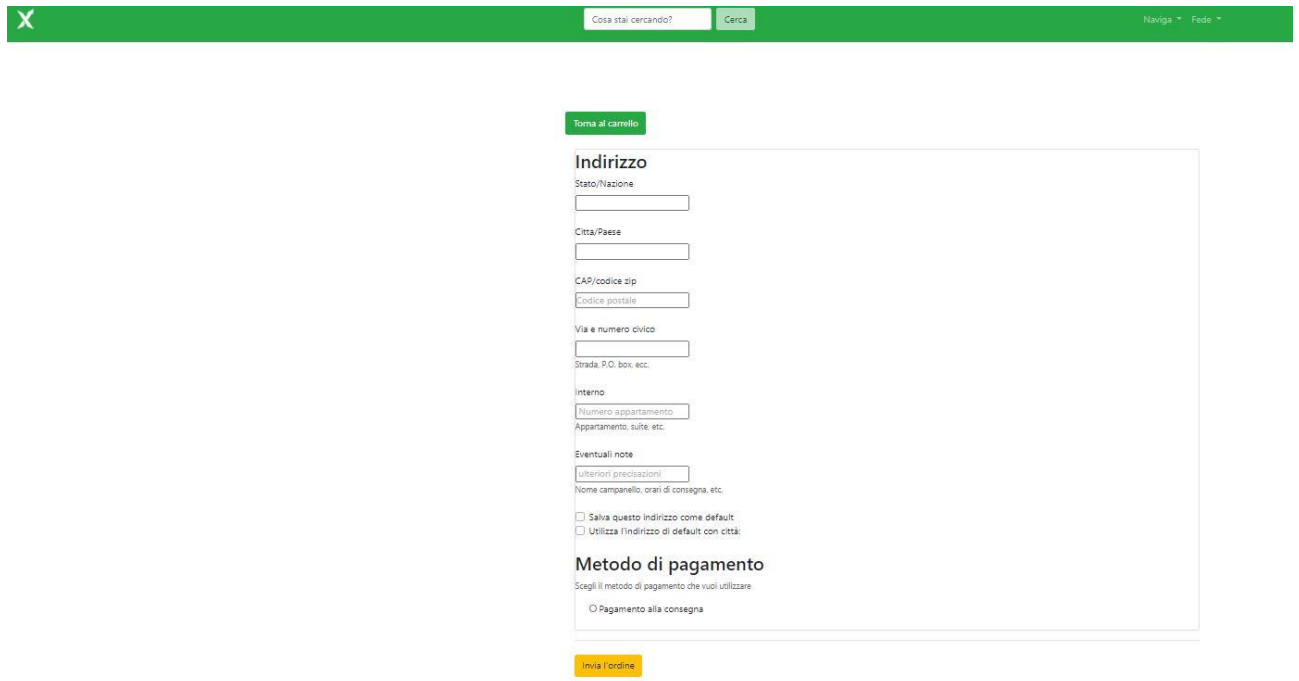
Riepilogo dell'ordine

Nome	Taglia	Condizione	Prezzo	Elimina
Air Dior	45,5	Deadstock(New)	140,0 €	
Totale			140,0 €	

Continua lo shopping

Checkout

Pagina del checkout (è la pagina dove bisogna inserire l'indirizzo di spedizione e il metodo di pagamento prima di inviare l'ordine)



The screenshot shows a web application interface for a checkout page. At the top, there is a green header bar containing a logo 'X' on the left, a search bar with the placeholder text 'Cosa stai cercando?' and a 'Cerca' button, and navigation links 'Naviga' and 'Fede' on the right. Below the header, there is a green button labeled 'Torna al carrello'. The main content area is a form titled 'Indirizzo' and 'Metodo di pagamento'. The 'Indirizzo' section includes input fields for 'Stato/Nazione', 'Città/Paese', 'CAP/codice zip', 'Codice postale', 'Via e numero civico', and 'Strada, P.O. box, ecc.'. There is also a section for 'Interno' with a 'Numero appartamento' field and 'Appartamento, suite, etc.'. Below these are 'Eventuali note' and 'Ulteriori precisazioni' fields, followed by a note about 'Nome campanello, orari di consegna, etc.'. At the bottom of the address section are two checkboxes: 'Salva questo indirizzo come default' and 'Utilizza l'indirizzo di default con città:'. The 'Metodo di pagamento' section has a heading and a subtext 'Scegli il metodo di pagamento che vuoi utilizzare.', followed by a radio button option 'Pagamento alla consegna'. At the bottom of the form is a yellow button labeled 'Invia l'ordine'.

6. Conclusioni

Si potrebbero aggiungere per migliorare l'app nuovi metodi di pagamento, e aggiungere una procedura di rimborso. In conclusione mi ritengo soddisfatto del lavoro e dell'esperienza ottenuta imparando ad usare Django per la creazione di questo progetto.