

MangaWorld

Progetto a cura di Scoppitto Matteo

Indice

1. Descrizione generale del progetto
2. Tecnologie Utilizzate
3. Test
4. Diagrammi
5. Screenshot

Descrizione generale del progetto

Il progetto è basato su un sito di compravendite. Il sito si occupa di compravendite di manga (fumetti di piccolo formato di origini giapponesi), che ultimamente sta diventando sempre più una moda e ormai la quantità di vendite di quest'ultimi è elevatissima.

Il progetto è diviso:

App Django:

- Accounts , dove viene gestita la parte di registrazione e di modifica dell'utente

- Core, dove viene gestita la parte di compravendita e quindi di tutte le interazione che riguardano gli oggetti in vendita.

Gli utenti possono registrarsi ma possono anche rimanere **Guest**, nel caso in cui non siano registrati potranno navigare sul sito ma non in tutte le pagine. Può utilizzare la barra di ricerca e cercare i prodotti, può anche vedere gli altri utenti e vedere i prodotti che quest'ultimo vende. Non avrà accesso ai prodotti consigliati siccome non potrà effettuare acquisti.

Utente registrato e loggato ha fatto la registrazione dall'apposito form, inserendo i suoi dati e dopo essersi registrato, può navigare sul sito, mettendo i propri articoli in vendita, comprare degli altri e avere l'accesso alla sezione consigliati, diversamente da Guest. Potrà riaccedere al proprio account inserendo le proprie credenziali.

Admin può svolgere tutte le azioni che possono svolgere gli altri utenti, in più potrà eliminare gli annunci e eliminare gli utenti(cancellando anche tutti gli annunci di quel utente).

Gestione degli ordini:

Se un utente non ha ancora nessun prodotto nel carrello, non ha un ordine a suo conto. Aggiungendo un prodotto nel carrello viene creato l'ordine. Nella pagina del carrello (accessibile tramite apposito tasto posizionato in alto a destra) si potrà eliminare un prodotto, avanzare verso il checkout oppure continuare a vedere il sito per effettuare altri acquisti. Nella pagina del carrello ci sarà già il totale delle dei prodotti al suo interno.

Quando l'utente deciderà di avanzare con il checkout, verrà reindirizzato verso la pagina di inserimento dei dati di indirizzo della spedizione. Si potrà decidere, dopo aver inserito i dati, di salvarli e/o impostarli come default per eventuali nuovi acquisti. Questi indirizzi salvati potranno essere modificati (togliendo o aggiungendo il default) e si potrà eliminarli. Bisognerà selezionare il metodo di pagamento (svilupato solo il metodo pagamento alla consegna). Dopo aver compilato tutto si potrà terminare l'ordine oppure ritornare indietro per svolgere altre operazioni. Ogni utente avrà la propria cronologia degli ordini effettuati, con tutti i dati presenti.

Vendita prodotto:

Un utente registrato può vendere i propri prodotti. Per mettere in vendita i propri prodotti e creare l'annuncio bisogna premere il bottone nella homepage **Aggiungi**

Manga. Bisognerà, successivamente, compilare un form con Nome, prezzo, condizione(nuovo o usato), descrizione del prodotto e una sua immagine. Il prodotto sarà disponibile sul sito e sul profilo del venditore. L'utente può decidere di apportare modifiche ai propri prodotti, come il nome, il prezzo, la descrizione e la condizione oppure eliminare il prodotto. Quando un prodotto viene venduto, questi saranno visibili nella sezione 'le mie vendite'.

Raccomandazioni:

Il sistema di raccomandazioni è basato sugli ordini che effettua un utente. È stato necessario creare il modello 'Itemconsigliato'

```
130         return self.user.username
131
132
133     class ItemConsigliato(models.Model):
134         num_item = models.PositiveIntegerField(default=0)
135         condizioneN = models.PositiveIntegerField(default=0)
136         condizioneU = models.PositiveIntegerField(default=0)
137         user = models.ForeignKey(User, on_delete=models.CASCADE)
138         prezzo = models.FloatField(validators=[MinValueValidator(0.0)], default=0)
139         sum_prezzo = models.FloatField(default=0)
140
141
142
143     def __str__(self):
144         return self.user.username
145
146     class Meta:
147         verbose_name_plural = 'ItemConsigliati'
148
```

Il modello in questione salva i vari dati di ogni ordine, tiene conto di quanti item ha ordinato un utente, del numero di prodotti con condizione 'nuova' e del numero di item con condizione '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 del prodotto. Prima di tutto in base alla media del prezzo salvata nel modello 'ItemConsigliato' viene controllato se il prodotto ha un prezzo che è vicino a quello della media, cioè può essere al massimo la media + 10 oppure non può essere minore alla media -10. Poi viene controllata la condizione, se un utente ha ordinato più di due prodotti con la stessa condizione e se i prodotti ordinati con questa condizione sono maggiori rispetto a quelli ordinati che hanno un'altra condizione. Se vengono soddisfatte queste richieste il prodotto viene aggiunto alla lista di raccomandazione. I prodotti 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 i prodotti. In una sezione vengono mostrati gli item più convenienti con condizione uguale a 'nuova', in un'altra i prodotti più convenienti con condizione uguale a 'usata'. Con più convenienti si intendono quelli che hanno un prezzo minore o uguale a 6 euro. L'ultima sezione è quella contenente gli item consigliati che fa riferimento al sistema di raccomandazione. Quest'ultima è visibile solamente dagli utenti loggati.

2. Tecnologie

Django:

Come base è stato utilizzato Django, un framework MVC(Model View Controller) per lo sviluppo. 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 Web.

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 una particolare evento da parte dell'utente.

Ajax:

A 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.

3.Test

Test app Accounts:

Il test consiste nel controllare se nella registrazione di un utente vengano inseriti, correttamente, tutti i campi e che le due password sia stata inserite coincidano,altrimenti si ha un errore.

```

class LoginTest(TestCase):
    """
    Per eseguire i test abbiamo bisogno di un utente con credenziali valide
    L'utente viene creato con questa funzione di setup
    """
    def setUp(self):
        self.dummy_user = User.objects.create_user(username='pippo', password='topolino', email='pippo@pippo2.com')

    """
    Test se tutti i campi per la registrazione sono obbligatori
    e che si rimanga sulla pagina di registrazione
    Inserisco due password diverse per vedere se il form ritorna un errore
    Infine inserisco dei valori validi e controllo che vengano accettati
    """
    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())

```

Testo il login controllando che le credenziali siano esatte, se sono corrette il login avviene con successo.

```

41 """
42 In questo test si controlla che l'utente creato nel setUp possa fare un login
43 Poi inserisco delle credenziali false per vedere se il sistema rifiuta il login
44 """
45 def test_login(self):
46     true_credential = {'username':'pippo', 'password':'topolino'}
47     fake_credential = {'username':'ciao', 'password':'ciao'}
48     t_cred = self.client.login(**true_credential)
49     f_cred = self.client.login(**fake_credential)
50     self.assertTrue(t_cred)
51     self.assertFalse(f_cred)
52

```

Test app core:

Creiamo l'oggetto da controllare.

```
16 ▶ class CoreCase(TestCase):
17
18     def setUp(self):
19         self.user = User.objects.create_user(username='pippo', email='pippo@pippo.com', password='topolino')
20         self.credential = {'username': 'pippo', 'password': 'topolino'}
21         self.item = Item.objects.create(nome="Nome Test",
22                                         prezzo="30",
23                                         categoria="S",
24                                         descrizione="Descrizione Test",
25                                         immagine="n",
26                                         condizioni='N',
27                                         autore_vendita=self.user)
28
29         self.address = ShoppingAddress.objects.create(user=self.user,
30                                                     città="Monte sant'angelo",
31                                                     stato="0",
32                                                     cap="41127",
33                                                     via="n",
34                                                     interno="3",
35                                                     note="ciao")
36
37         self.user2 = User.objects.create_user(username='pippo2', email='pippo@pippo2.com', password='topolino')
38         self.credential2 = {'username': 'pippo2', 'password': 'topolino'}
```

Testo la visualizzazione di un prodotto che se avviene senza problemi da come risposta http 200

```
42 ▶ def test_visualizzaItem(self):
43     """
44     controllo che l'utente riesca a visualizzare il suo annuncio dell'item appena creato
45     """
46     self.client.login(**self.credential)
47     response = self.client.get('/products/' + self.item.slug)
48     self.assertTemplateUsed(response, 'core/item.html')
49     self.assertEqual(response.status_code, 200)
```

Ho testato la creazione di un item verificando che i campi obbligatori siano rispettati.

```
61 ▶ def test_CreaItem(self):
62     """
63     Verifica che nella creazione dell'item i campi obbligatori siano rispettati
64     """
65     self.client.login(**self.credential)
66     response = self.client.post('/nuovo-item/', {})
67     self.assertFormError(response, 'form', 'nome', 'Questo campo è obbligatorio.')
68     self.assertFormError(response, 'form', 'prezzo', 'Questo campo è obbligatorio.')
69     self.assertFormError(response, 'form', 'categoria', 'Questo campo è obbligatorio.')
70     self.assertFormError(response, 'form', 'descrizione', 'Questo campo è obbligatorio.')
71     self.assertFormError(response, 'form', 'immagine', 'Questo campo è obbligatorio.')
72     self.assertFormError(response, 'form', 'condizioni', 'Questo campo è obbligatorio.')
73
74     self.assertTemplateUsed(response, 'core/aggiungi_prodotto.html')
75     self.assertEqual(response.status_code, 200) # verifica per capire se il template utilizzato è quello corretto
76
77     response = self.client.post('/nuovo-item/',
78                                 {'nome': 'J4', 'prezzo': '50', 'categoria': 'S', 'descrizione': 'V',
79                                 'immagine': 's', 'condizioni': 'U'})
80     self.assertTemplateUsed(response, 'core/aggiungi_prodotto.html')
81     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.

```
73
74 ▶ def test_item_change(self):
75     # con utente autenticato
76     self.client.login(**self.credential)
77     response = self.client.get('/item/' + str(self.item.id) + '/modifica/')
78     self.assertEqual(response.status_code, 200)
79     response = self.client.post('/item/' + str(self.item.id) + '/modifica/', {})
80     self.assertFormError(response, 'form', 'nome', 'Questo campo è obbligatorio.')
81     self.assertFormError(response, 'form', 'prezzo', 'Questo campo è obbligatorio.')
82     response = self.client.post('/item/' + str(self.item.id) + '/modifica/',
83                                {'nome': 'mod', 'prezzo': '50', 'categoria': 'A', 'descrizione': 'c', 'condizione': 'N'})
84     self.client.logout()
85
86     # con utente autenticato ma non creatore
87     self.client.login(**self.credential2)
88     response = self.client.get('/item/' + str(self.item.id) + '/modifica/')
89     self.assertTemplateUsed(response, 'core/homepage.html')
90     self.assertTemplateNotUsed(response, 'core/modifica_item.html')
```

Ho testato l'eliminazione di un item controllando che possa essere cancellato solo dal suo creatore, e quindi se l'utente non è il creatore, se proverà ad accedere alla pagina per eliminare l'item, verrà reindirizzato alla homepage.

```
91
92 ▶ def test_item_delete(self):
93     # con utente autenticato
94     self.client.login(**self.credential)
95     response = self.client.get('/item/' + str(self.item.id) + '/delete/')
96     self.assertEqual(response.status_code, 200)
97     response = self.client.post('/item/' + str(self.item.id) + '/delete/', {})
98     self.assertRedirects(response, '/user/' + self.user.username + '/')
99     self.client.logout()
100
101     # con utente autenticato ma non creatore
102     self.client.login(**self.credential2)
103     response = self.client.get('/item/' + str(self.item.id) + '/delete/')
104     self.assertTemplateNotUsed(response, 'core/deleteitem.html')
105
```

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

```
106 ▶ def test_login_required(self):
107     '''test on login_required sulla creazione di un item'''
108     response = self.client.get('/nuovo-item/')
109     self.assertRedirects(response, '/accounts/login/?next=/nuovo-item/')
110     # 302 --> FOUND: pagina esiste ma non puoi entrarci
111     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.

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

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.

```
118 ▶ def test_userProfileView(self):
119     # con utente autenticato
120     # su il tuo profilo
121     self.client.login(**self.credential)
122     response = self.client.get('/user/' + self.user.username + '/')
123     self.assertTemplateUsed(response, 'core/profilo.html')
124     self.assertTrue(response.status_code, 200)
125
126     # sul profilo degli altri
127     response = self.client.get('/altriuser/' + self.user2.username + '/')
128     self.assertTemplateNotUsed(response, 'core/profilo.html')
129     self.assertTemplateUsed(response, 'core/user_profile.html')
130
131     # con utente non autenticato
132     self.client.logout()
133     response = self.client.get('/altriuser/' + self.user.username + '/')
134     self.assertTemplateUsed(response, 'core/user_profile.html')
135     self.assertTrue(response.status_code, 200)
```

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

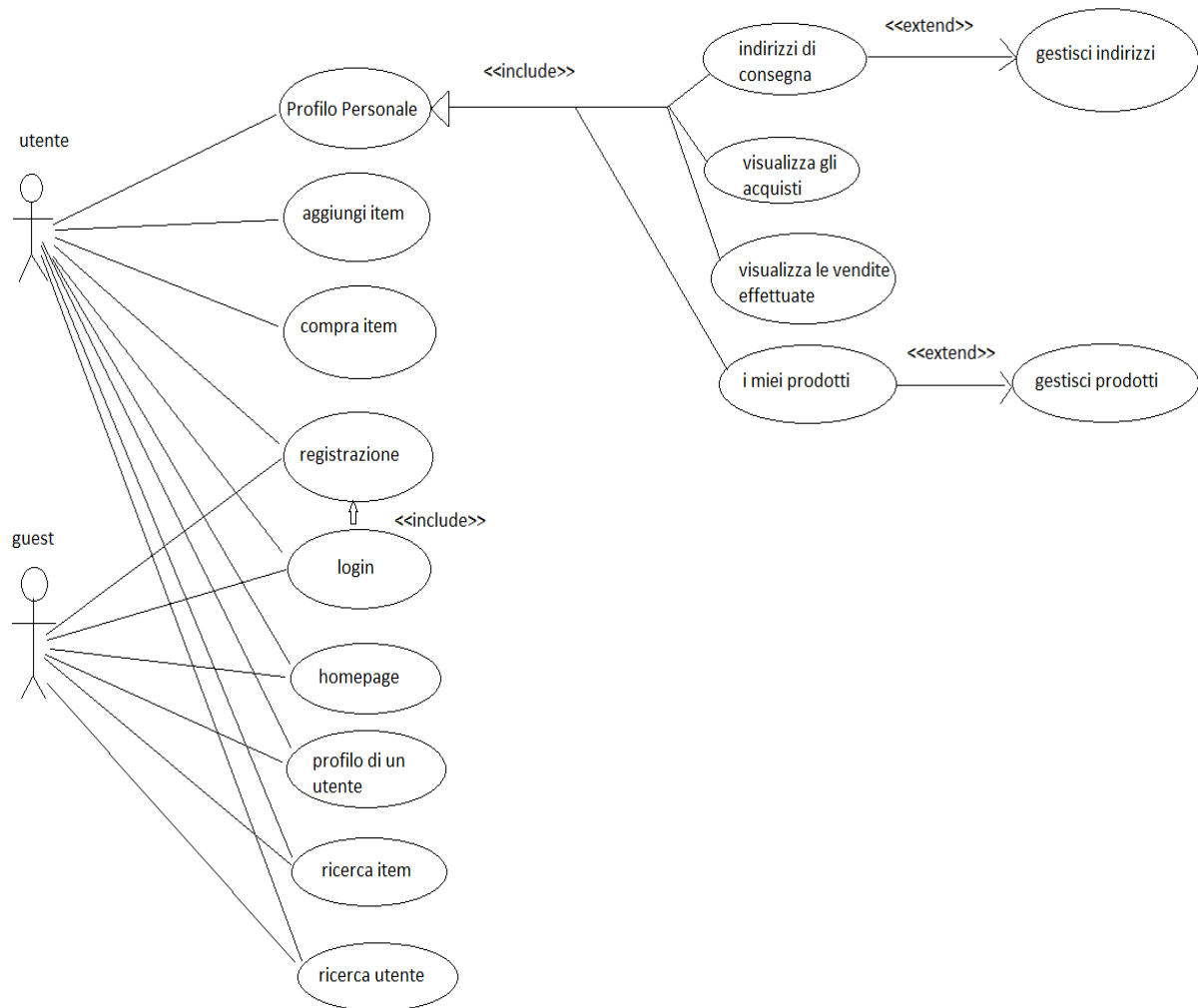
```
137 ▶ def test_address_change(self):
138     # con utente autenticato
139     self.client.login(**self.credential)
140     response = self.client.get('/user/address/' + str(self.address.id) + '/modifica/')
141     self.assertEqual(response.status_code, 200)
142     response = self.client.post('/user/address/' + str(self.address.id) + '/modifica/', {})
143     self.assertFormError(response, 'form', 'città', 'Questo campo è obbligatorio.')
144     self.assertFormError(response, 'form', 'via', 'Questo campo è obbligatorio.')
145     self.assertFormError(response, 'form', 'stato', 'Questo campo è obbligatorio.')
146     self.assertFormError(response, 'form', 'cap', 'Questo campo è obbligatorio.')
147     response = self.client.post('/user/address/' + str(self.address.id) + '/modifica/',
148                                {'città': 'modena', 'via': 'S50', 'stato': 'it', 'cap': '3434'})
149     self.assertRedirects(response, '/user/' + self.user.username + '/address_page/')
150     self.client.logout()
151
152     # con utente autenticato ma non creatore
153     self.client.login(**self.credential2)
154     response = self.client.get('/user/address/' + str(self.address.id) + '/modifica/')
155     self.assertTemplateUsed(response, 'core/homepage.html')
156     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.

```
158 ▶ def test_address_delete(self):  
159     # con utente autenticato  
160     self.client.login(**self.credential)  
161     response = self.client.get('/user/address/' + str(self.address.id) + '/delete/')  
162     self.assertEqual(response.status_code, 200)  
163     response = self.client.post('/user/address/' + str(self.address.id) + '/delete/', {})  
164     self.assertRedirects(response, '/user/' + self.user.username + '/address_page/')  
165     self.client.logout()  
166  
167     # con utente autenticato ma non creatore  
168     self.client.login(**self.credential2)  
169     response = self.client.get('/user/address/' + str(self.address.id) + '/delete/')  
170     self.assertTemplateNotUsed(response, 'accounts/address_delete.html')
```

4. Diagrammi

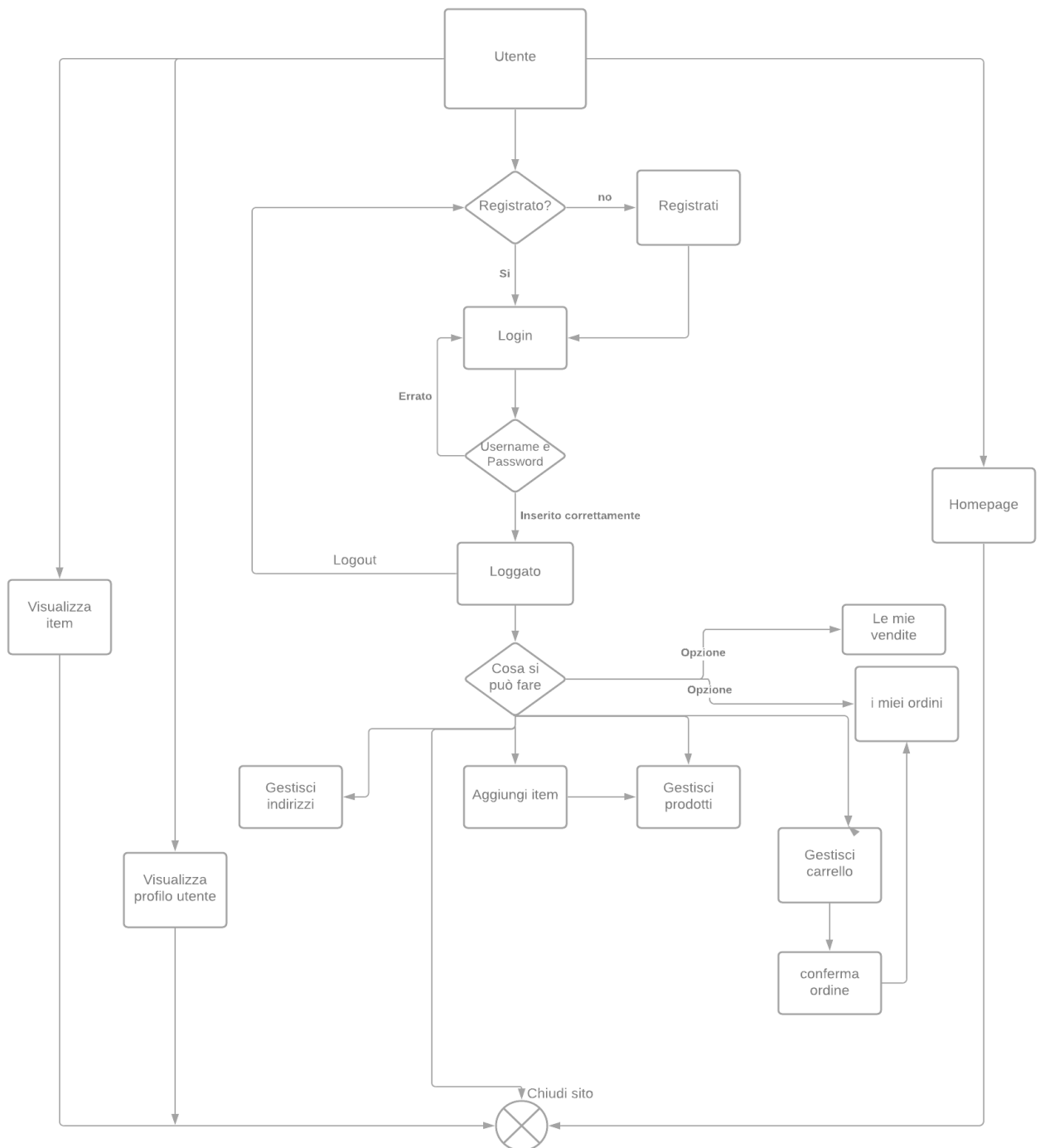
Diagramma dei casi d'uso



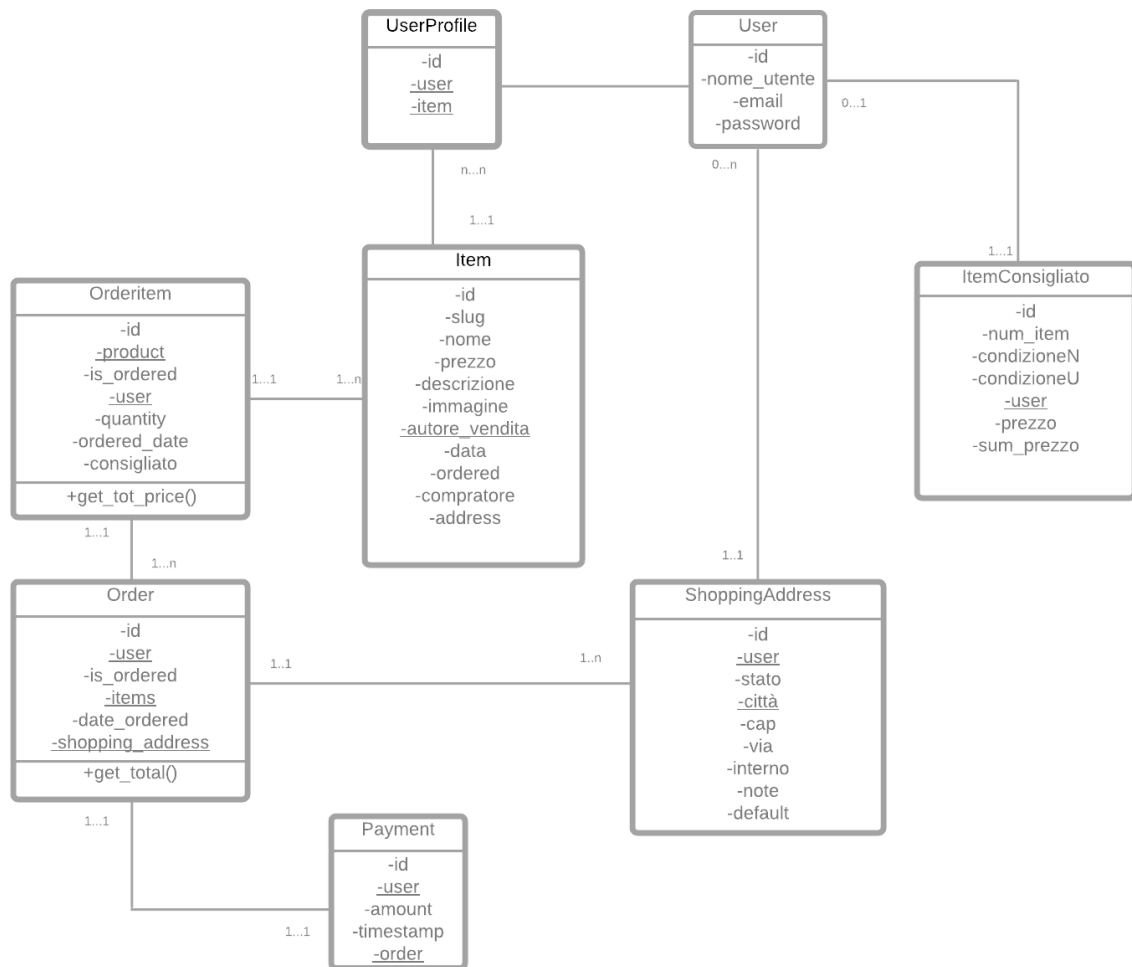
Scenario:

Mario Rossi, il 3 marzo decide di entrare nel sito MangaWorld per vedere se qualcuno vende un manga molto raro. Si logga correttamente mettendo il suo username e la sua password. Nella homepage vede se è presente quello che cerca, trova il prodotto ma è molto scettico sul sito. Allora decide di cercare qualcuno che aveva già effettuato acquisti, cercando della 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.

3.2 Diagramma attività

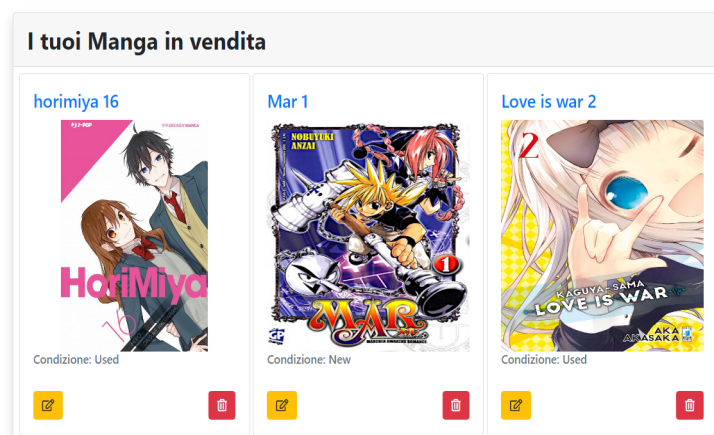


3.3 Diagramma classi





5. Screenshot

Profilo dell'utente gigi, vista dall'utente gigi




Pagina di registrazione del sito

Categorie  Registrazione Login

mangaworld

Cosa stai cercando?

Cerca 

Crea un Account!

Nome utente*

Obbligatorio. 150 caratteri o meno.Solo lettere, cifre e @/./+/-/_

Password*

- La tua password non può essere troppo simile alle altre tue informazioni personali.
- La tua password deve contenere almeno 8 caratteri.
- La tua password non può essere una password comunemente usata.
- La tua password non può essere interamente numerica.


Conferma password*

Inserisci la stessa password inserita sopra, come verifica.

Email*

Crea Account

[Hai già un account? Vai al Login!](#)




Pagina item del proprio profilo

Indietro

Mar 1

Venditore: [@gigi](#) | Categoria: [Shonen](#)



Prezzo : 20,0 €

New


Descrizione: Mar 1 , nuovo introvabile edizione 2009

pagina item in vendita

[Indietro](#)

Love is war 1

Venditore: @admin || Categoria: Seinen



Prezzo : 5,6 €
New
Descrizione: Love is war 1, nuovo

[Aggiungi al carrello](#)

Pagina del carrello di un utente

Riepilogo dell'ordine

Nome	Condizione	Prezzo	Elimina
Love is war 1	New	5,6 €	Elimina
Totale			5,6 €

[Continua lo shopping](#) [Checkout](#)

Pagina di Checkout

[Torna al carrello](#)

Indirizzo

Stato/Nazione

Città/Paese

CAP/codice zip
Codice postale

Via e numero civico

Strada, P.O. box, ecc.

Interno
Numero appartamento
Appartamento, suite, etc.

Eventuali note
ulteriori precisazioni
Nome campanello, orari di consegna, etc.

☐ Salva questo indirizzo come default
☐ Utilizza l'indirizzo di default con città:

Metodo di pagamento

Scegli il metodo di pagamento che vuoi utilizzare

☐ Pagamento alla consegna

[Invia l'ordine](#)

6.Conclusioni

Il sito è parecchio migliorabile sotto molti ambiti, tra cui la facilità di accesso all'interfaccia e alla gestione dei pagamenti, aggiungendo più metodi e il monitoraggio delle spedizioni.