

Progetto per il corso di Tecnologie Web

Byte-Shop

Portale di e-commerce per prodotti tech

*Filippo Reggiani
Matricola: 148084*

Indice:

- 1. *Requisiti***
- 2. *Obiettivi***
- 3. *Casi d'uso***
- 4. *Modellazione entità e diagramma delle classi***
- 5. *Diagramma delle classi***
- 6. *Tecnologie adottate***
- 7. *Meccanismo di recommendation***
- 8. *Unit Test***
- 9. *Risultati***

1. Requisiti

Si vuole realizzare un portale di e-commerce per la vendita di dispositivi elettronici. Oltre agli utenti anonimi, possono essere presenti due tipologie di utenti: Utenti clienti e utenti admin.

Gli utenti admin caricano i prodotti sul sito e gestiscono le vendite dei prodotti.

Gli utenti clienti acquistano i prodotti con la possibilità di lasciare una recensione su questi.

- **Utenti anonimi:**

- Possono navigare il sito e visualizzare prodotti con relativo prezzo e recensioni.
- Non possono effettuare acquisti.
- Possono effettuare ricerche sui prodotti.
- Possono registrarsi mediante form di registrazione.

- **Utenti registrati clienti:**

- Possono effettuare l'acquisto dei prodotti.
- Possono lasciare recensioni sui prodotti che hanno acquistato.
- Hanno suggerimenti sui prodotti da poter acquistare tramite recommendation system.
- Possono effettuare ricerche sui prodotti.
- Possono modificare i propri dati.

- **Utenti registrati admin:**

- Possono mettere in vendita i prodotti sul portale.
- Possono aggiornare i prodotti caricati sul portale.
- Possono vedere informazioni sugli acquisti dei prodotti (ordini effettuati).
- Possono vedere statistiche sui prodotti più venduti.
- Possono modificare i propri dati.

2. Obiettivi

Si vuole mettere a disposizione una piattaforma online di e-commerce al negozio Byte-Shop per consentirgli la vendita dei suoi prodotti tech online.

I membri dello staff del negozio, registrati come admin nel portale, hanno la possibilità di caricare prodotti di diverse categorie sul portale.

Le informazioni sui prodotti caricati sono modificabili e i prodotti caricati sono eliminabili.

Si ha inoltre la possibilità di vedere gli ordini relativi ai prodotti acquistati e vedere statistiche sui prodotti venduti.

I clienti hanno la possibilità di acquistare i prodotti da loro desiderati aggiungendoli al carrello ed effettuando l'ordine.

Sui prodotti acquistati hanno inoltre la possibilità di effettuare una recensione in modo da aiutare futuri clienti nei loro acquisti.

3. Casi d'uso

Utente anonimo:

L'utente anonimo può visualizzare i prodotti caricati sul portale, visualizzare le recensioni presenti su essi ed effettuare ricerche di prodotti.

Può effettuare la registrazione.

Utente client:

L'utente registrato come client, dopo aver effettuato il login, oltre a ciò che può fare un utente anonimo, può visualizzare il proprio carrello.

All'interno del carrello l'utente può aggiungere, eliminare o effettuare l'ordine di prodotti.

Sui prodotti che ha acquistato può successivamente lasciare una recensione.

Attraverso gli acquisti che l'utente ha effettuato, confrontandoli con gli acquisti effettuati da tutti gli utenti registrati, il recommendation system fornisce all'utente dei prodotti consigliati.

L'utente ha inoltre la possibilità di modificare i dati del suo profilo e modificare anche la password.

Per terminare la propria sessione l'utente può effettuare il logout.

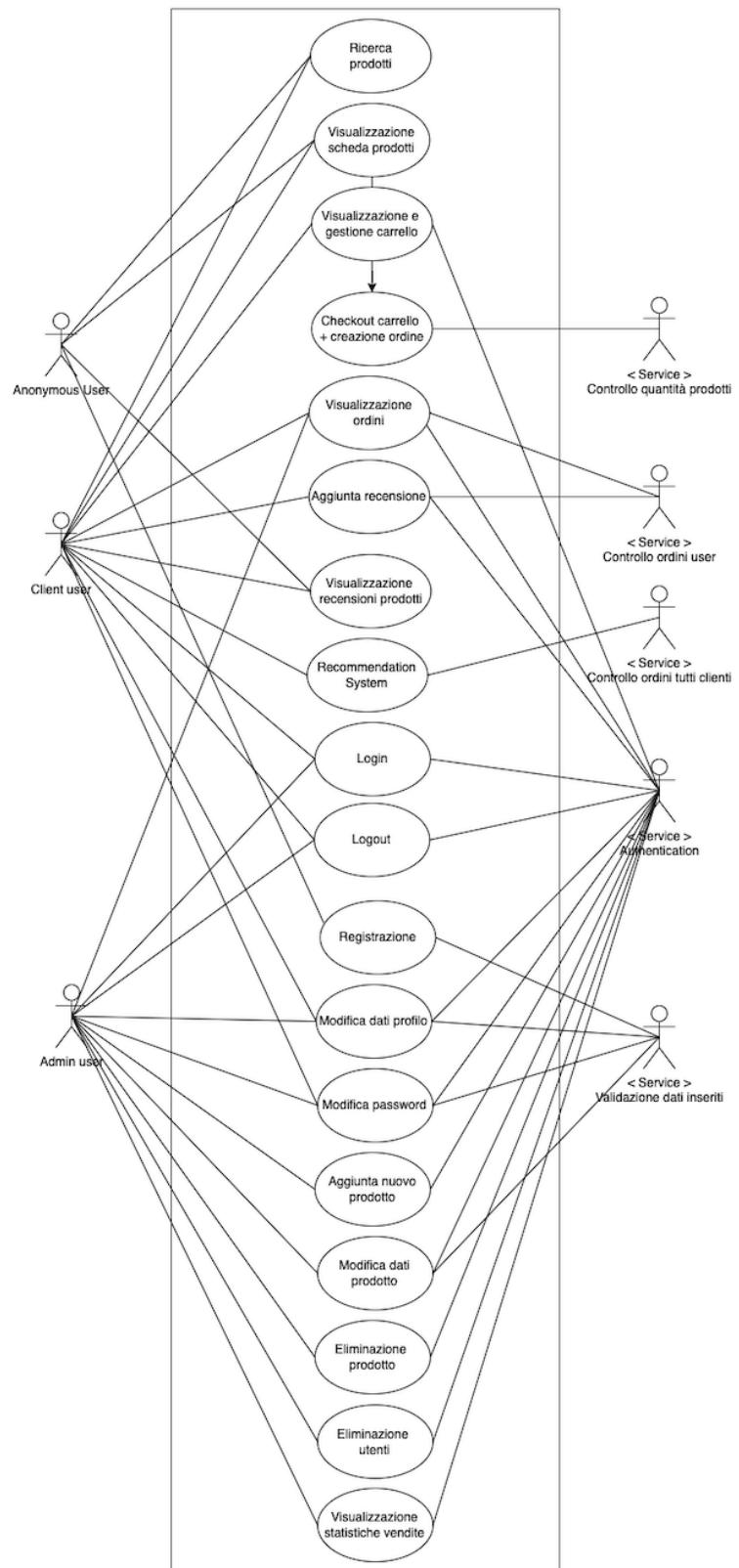
Utente admin:

L'utente registrato come admin, dopo aver effettuato il login, può visualizzare le statistiche relative alle vendite dei prodotti che ha inserito.

Può inserire un nuovo prodotto, modificare quelli già presenti o eliminarli. Può eliminare gli utenti cliente registrati.

Può modificare i dati relativi al suo profilo e modificare la password.

Per terminare la propria sessione l'utente può effettuare il logout.



4. Organizzazione logica e modellazione entità

Organizzazione logica

Si è deciso di suddividere il progetto in 5 applicazioni:

- **Ecommerce:** Rappresenta il progetto Django, contiene le impostazioni generali del progetto, i percorsi che implementano il routing della web app e le applicazioni installate.
- **products:** Contiene al suo interno tutto ciò che riguarda la gestione dei prodotti. La visualizzazione dei prodotti, sia in maniera dettagliata che generale, l'aggiunta di nuovi prodotti, l'aggiornamento e la cancellazione di quelli presenti e il meccanismo di ricerca su di essi. Include i forms per l'inserimento e aggiornamento dei prodotti. Contiene inoltre la creazione di una nuova recensione relativa ad un prodotto con relativo forms.
- **cart:** Racchiude i meccanismi per la gestione del carrello da parte degli utenti. Consente la visualizzazione del carrello, la modifica di quantità e l'eliminazione dei prodotti in esso.
- **orders:** Contiene al suo interno i meccanismi per la gestione degli ordini, quali la visualizzazione degli ordini, la creazione di nuovi ordini, lo svuotamento del carrello e la modifica della quantità dei prodotti presenti una volta completato un ordine.
- **authentication:** Racchiude i meccanismi di registrazione, autenticazione, logout per una maggiore chiarezza e compattezza del codice. La registrazione avviene mediante l'utilizzo di forms. Gestisce la modifica dei propri dati e della password da parte degli utenti (client e admin). Consente inoltre la cancellazione di utenti clienti da parte degli admin.

Modellazione entità

Per la gestione dei dati si è deciso di modellare 12 entità:

Models in products:

Il model *Product* rappresenta un prodotto generico, contiene al suo interno informazioni che riguardano prodotti appartenenti a qualsiasi tipo di categoria.

I model *Computer*, *Smartphone*, *Cover*, *Cuffie*, *Televisore* estendono il model *Product*. Estendendo da esso si ha la possibilità di avere model più specifici con dati relativi alla categoria di prodotto in questione.

Il model *Product* ha al suo interno una relazione ManyToMany con il model *Recensione* in quanto per ogni prodotto possono esserci più recensioni e una Foreign Key con riferimento all'utente admin che ha caricato il prodotto.

I model *Recensione* consente di rappresentare una recensione lasciata da un cliente su un prodotto che ha acquistato. Ha al suo interno una ForeignKey che fa da riferimento al Client che ha scritto la recensione.

Models in authentication:

Per quanto riguarda la gestione degli utenti si è deciso di utilizzare come base lo *User* di default di Django.

Gli utenti admin sono rappresentati attraverso lo *User*, mentre gli utenti clienti sono rappresentati tramite il model *Client* che ha una relazione OneToOne con la classe *User*.

Per effettuare la distinzione tra utenti clienti e utenti staff si utilizza il field “*is_staff*” booleano presente all’interno di *User*.

In caso di valore True l’utente è di tipo admin.

In caso di valore False l’utente è di tipo cliente.

Models in cart:

Il model *Carrello* consente la rappresentazione del carrello dell’utente cliente.

Il model *Carrello_Item* consente di rappresentare un singolo prodotto all’interno del carrello. Si è deciso di creare questa nuova classe invece di utilizzare *Product* in quanto in questo modo è possibile acquistare prodotti diversi ognuno con una diversa quantità.

Il model *Carrello* ha dunque una relazione ManyToMany con il model *Carrello_Item*, il quale ha una ForeignKey per far riferimento ad un prodotto.

Il Model *Carrello* ha poi una relazione OneToOne con il model *Client*, in questo modo per ogni cliente si ha un solo carrello.

Models in orders

Il model *Ordine* consente la rappresentazione di un ordine effettuato dall’utente.

Viene creato quando viene fatto il checkout del carrello dell’utente.

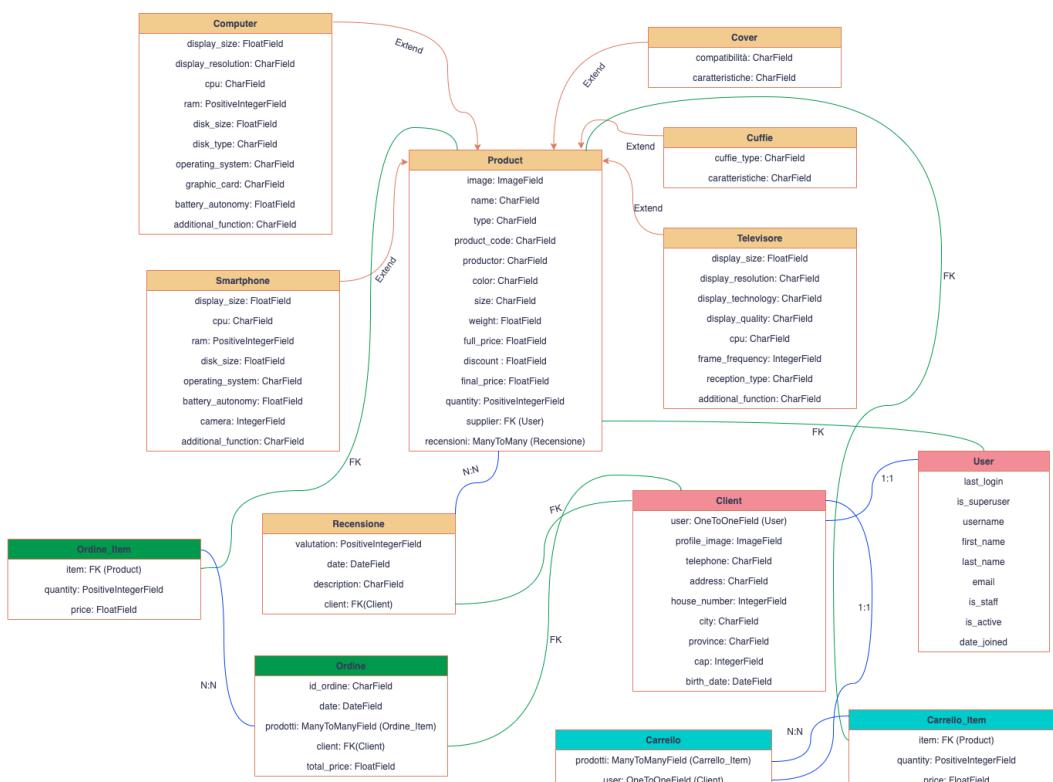
Ha una relazione ManyToMany con il model *Ordine_Item*, che a sua volta ha al suo interno una ForeignKey per riferirsi al prodotto.

In questo modo, come per il carrello, per un ordine si possono avere prodotti diversi con diverse quantità.

All’interno del model *Ordine* si ha poi un riferimento tramite ForeignKey al Client che ha fatto l’ordine.

5. Diagramma delle classi

Il seguente diagramma delle classi mostra l’organizzazione del progetto ad alto livello.



6. Tecnologie adottate

L'implementazione dell'applicazione avviene tramite framework Django, una libreria python open-source supportata da una vasta community.

Offre nativamente un sistema di autenticazione sicuro, robuste API di accesso al database, strumenti di testing, forms di inserimento e molto altro.

Supporta inoltre la filosofia DRY(Don't repeat yourself) la quale permette agli sviluppatori di risparmiare tempo e concentrare le proprie risorse sul codice da implementare.

Il front end dell'applicazione è stato creato attraverso crispy filter con Bootstrap o solo Bootstrap.

Bootstrap è un framework production-ready che permette agli sviluppatori di risparmiare molto tempo nella scrittura del css.

Si ha infatti la possibilità di utilizzare direttamente i componenti forniti dal toolkit.

Un altro punto di forza è il potente sistema di ghiriglie che permette agli sviluppatori di creare design responsive rendendo la visualizzazione dei contenuti del sito adeguata per ogni grandezza di schermo possibile.

In alcuni punti è stato fatto uso di Javascript per manipolare elementi html.

Per la visualizzazione delle statistiche delle vendite dell'utente è stato utilizzato chart.js.

7. Meccanismo di recommendation

Come meccanismo di recommendation si è deciso di utilizzare un approccio **User-based collaborative filtering**.

Cercando di predire quello che piacerà all'utente in base alle sue somiglianze con altri utenti.

Il meccanismo di recommendation effettua diversi controlli per essere in grado di restituire ogni volta la miglior soluzione.

Ci sono 4 tipologie di risultati:

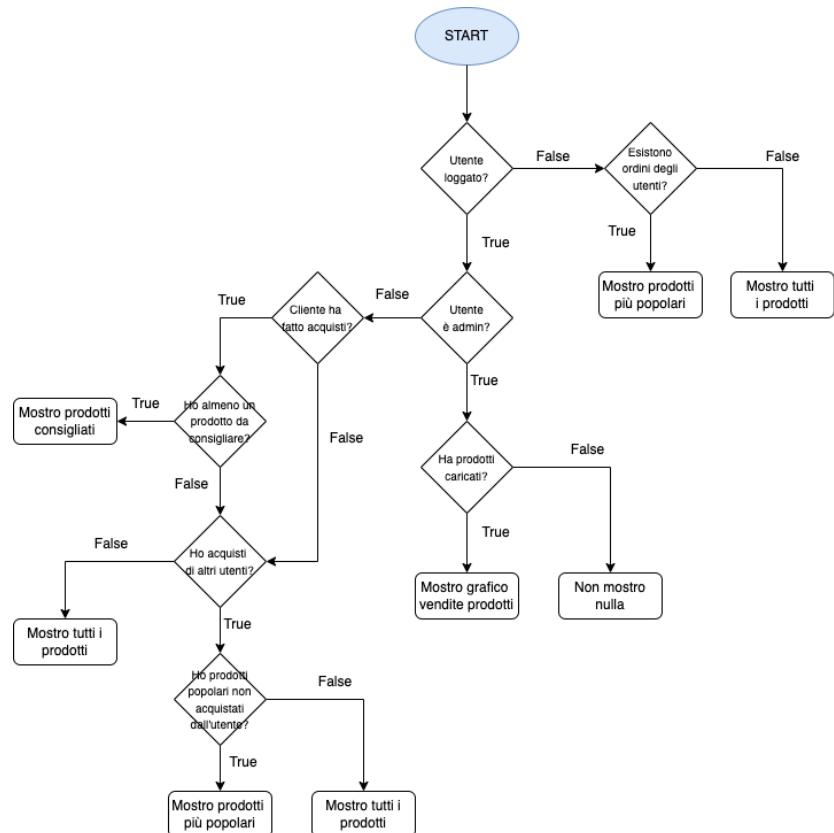
- **Prodotti consigliati**
- **Prodotti più popolari**
- **Tutti i prodotti**
- **Grafico statistiche**

Prodotti consigliati:

Vengono mostrati nel caso in cui l'utente loggato di tipo cliente ha effettuato acquisti e sono presenti altri acquisti fatti da altri utenti.

I prodotti consigliati sono ottenuti acquisendo i prodotti acquistati dall'utente.

Ognuno di questi viene cercato negli ordini effettuati dagli altri utenti. Se un utente ha acquistato lo stesso prodotto assieme ad un altro, viene consigliato quell'altro prodotto.



Esempio:

Daniele ha acquistato *Iphone 13*

Prodotto: Iphone 13
Quantità: 1
Prezzo: 798.15€



Jennifer ha acquistato *Iphone 13 + cover Iphone 13*

Prodotto: Iphone 13
Quantità: 1
Prezzo: 798.15€



Prodotto: Cover Iphone 13
Quantità: 1
Prezzo: 48.97€



Il sistema di recommendation consiglierà a Daniele il prodotto *cover iphone 13*.

Prodotti consigliati in base ai tuoi acquisti:

Cover Iphone 13

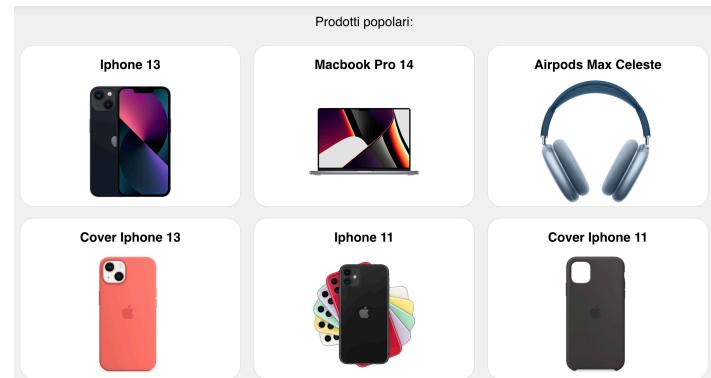


Prodotti più popolari:

Vengono mostrati solo se esistono ordini di altri utenti nel caso di utente non autenticato e se non ci sono prodotti da consigliare all'utente.

Sono ottenuti acquisendo dagli ordini i prodotti che sono stati acquistati in maniera distinta. Per ogni prodotto distinto si ottiene la quantità complessiva che è stata acquistata.

I prodotti restituiti sono ordinati per quantità acquistata complessivamente, quelli più in alto sono quelli che sono stati acquistati più volte.



Tutti i prodotti:

Nel caso in cui non sia possibile restituire i prodotti consigliati e nemmeno i prodotti più popolari vengono semplicemente restituiti tutti i prodotti.

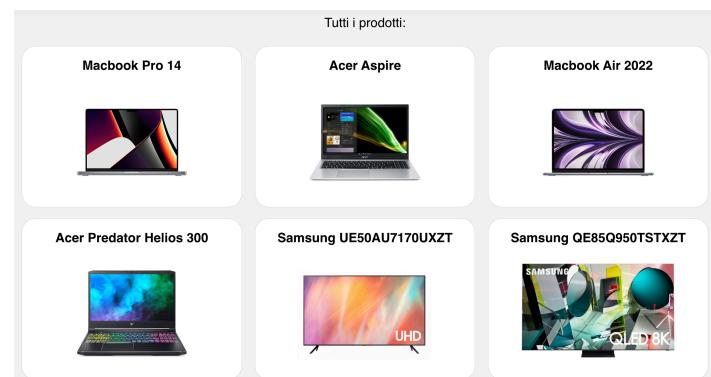
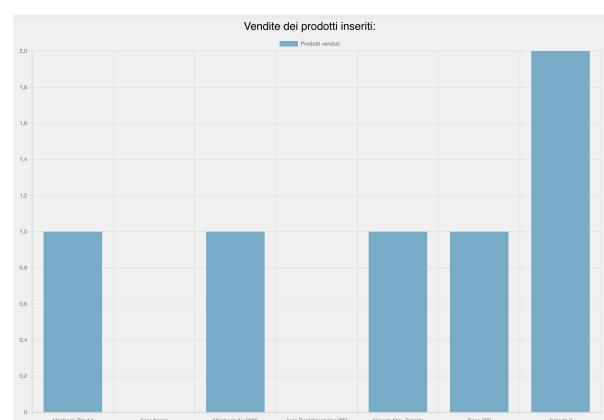


Grafico statistiche:

Nel caso in cui sia autenticato un utente admin che ha caricato dei prodotti, vengono recuperati i prodotti inseriti da questo utente e controllando gli ordini si ottengono le quantità vendute dei prodotti che ha inserito.

Nel caso in cui l'utente admin non abbia inserito prodotti, viene semplicemente avvisato l'utente che non ci sono prodotti caricati.



8. Unit Test

Django mette a disposizione la classe python **TestCase**.

Per iniziare è necessario creare una classe ereditata da *django.test.TestCase*.

Occorre poi definire alcune funzioni (queste devono iniziare con *test* in modo che unittest le esegua) all'interno delle quali vengono asserte alcune condizioni considerate.

La classe *django.test.TestCase* svuota il database di test ogni volta che deve essere eseguito un nuovo *TestCase*.

Occorre dunque inizializzare i dati nel database dove necessario.

L'utilità *setUp* è pensata a tale scopo.

Il comportamento di *setUp* è quello di essere eseguito ogni volta che deve essere costruito un test.

I test sono stati svolti all'interno dell'applicazione *authentication* e dell'applicazione *products*.

All'interno dell'app *authentication* sono presenti due classi, ognuna con un test:

- La classe **ClientRegistrationTest** estende da *TestCase* ed effettua dei test sulla registrazione da parte dell'utente cliente.

Al suo interno è definito il **metodo *setUp*** che imposta l'url su cui verranno poi effettuati i test.

È poi stato definito il **metodo *test_client_registration(self)***, all'interno del quale viene creato un utente cliente di test con dati fintizi.

All'interno di questo test vengono passati i dati dello user di test al form di registrazione degli utenti clienti, viene controllata:

- Validità form di registrazione.
- Validità singoli campi del form
- Correttezza requisiti dei campi del form.

```
#Test registrazione cliente
class ClientRegistrationTest(TestCase):
    def setUp(self): ...

    def test_client_registration(self): ...
```

Viene poi effettuata una richiesta POST alla view di registrazione del cliente e si controlla che la registrazione sia avvenuta con successo controllando che lo status code sia uguale a 200.

Dato che la view di registrazione reindirizza alla view di login, viene effettuato anche un controllo sul redirezionamento.

Successivamente vengono effettuati controlli sul corretto inserimento nel database dei dati dell'utente, sia all'interno della classe *Client*, che nella classe *User*.

Viene infine controllata la corretta creazione del carrello dell'utente.

- La classe **AdminRegistrationTest** estende da *TestCase* ed effettua dei test sulla registrazione da parte dell'utente admin.

Al suo interno è definito il **metodo *setUp*** che imposta l'url su cui verranno poi effettuati i test.

```
#Test registrazione admin
class AdminRegistrationTest(TestCase):
    def setUp(self): ...

    def test_admin_registration(self): ...
```

È poi stato definito il **metodo *test_admin_registration (self)***, all'interno del quale viene creato un utente admin di test con dati fintizi.

All'interno di questo test vengono passati i dati dello user di test al form di registrazione degli utenti admin, viene controllata:

- Validità form di registrazione.
- Validità singoli campi del form
- Correttezza requisiti dei campi del form.

Viene poi effettuata una richiesta POST alla view di registrazione dell'admin e si controlla che la registrazione sia avvenuta con successo controllando che lo status code sia uguale a 200.

Dato che la view di registrazione reindirizza alla view di login, viene effettuato anche un controllo sul redirezionamento.

Successivamente vengono effettuati controlli sul corretto inserimento nel database dei dati dell'utente all'interno della classe User.

All'interno dell'applicazione *products* è presente una classe con due test:

- La classe ***DeleteProductViewTest*** estende da *TestCase* ed effettua dei test sulla cancellazione di un prodotto.

Attraverso il **metodo *setUp*** vengono creati un prodotto e un utente admin con dati fintizi.

- Il **metodo *test_delete_product*** effettua dei test sulla cancellazione di un prodotto.

Viene effettuato prima il login da parte dell'utente admin e successivamente si accede all'url da testare.

Una volta effettuato l'accesso allo url da testare si effettua un controllo sullo status code, che deve essere 302. Si controlla poi lo url a cui viene effettuato il redirezionamento.

```
#Test sulla cancellazione di un prodotto
class DeleteProductViewTest(TestCase):

    def setUp(self):
        pass

    def test_delete_product(self):
        self.client.login(username='testuser', password='testpassword') #Effettuo login
        response = self.client.post(reverse('products:delete_product', args=[self.product.product_code]))

        self.assertEqual(response.status_code, 302) #Controllo status code
        self.assertContains(response, '/allproducts') #Controllo redirect
        self.assertFalse(Product.objects.filter(product_code='12345').exists()) #Controllo cancellazione prodotto

    def test_delete_nonexistent_product(self):
        self.client.login(username='testuser', password='testpassword') #Effettuo login
        response = self.client.post(reverse('products:delete_product', args=['nonexistent']))

        self.assertEqual(response.status_code, 200) #Controllo status code
        self.assertContains(response, "ERROR: product_code non valido") #Controllo che dia errore di product_code non valido
        self.assertTrue(Product.objects.filter(product_code='12345').exists()) #Controllo prodotto non sia stato cancellato
```

- Il metodo **metodo *test_delete_nonexistent_product*** effettua il test nel caso in cui il prodotto non sia presente.

Dopo aver effettuato il login e l'accesso allo url con product code non valido, effettua un controllo sullo status code, che in questo caso deve essere 200, controlla che l'errore restituito sia corretto e che il prodotto creato in precedenza non sia stato cancellato.

9. Risultati

Homepage

The screenshot shows the homepage of the Byte Shop Computer Store. At the top, there is a navigation bar with links for phone number (+39 349 1823467), email (contact@byteshop.com), language (Italiano), currency (€ EUR), registration, login, and search. Below the navigation is the store logo 'BYTE SHOP COMPUTER STORE' and a search bar. The main content area features a section titled 'Prodotti popolari:' with a 3x3 grid of product cards. The products include a Samsung UE50AU7170UXZT TV, an iPhone 13, Airpods 3, a Macbook Pro 14, Airpods Max Celeste, a Cover Iphone 13, an iPhone 11, a Cover Iphone 11, and a Samsung QE85Q950TSTXZT TV.

Dettagli prodotto con user autenticato cliente

The screenshot shows the product detail page for an iPhone 13. At the top, there is a navigation bar with links for phone number (+39 349 1823467), email (contact@byteshop.com), language (Italiano), currency (€ EUR), and user info (Benvenuta, Jennifer). Below the navigation is the store logo and a search bar. The main content area shows a large image of the iPhone 13 and its specific product details. It includes a summary table with the following information:

Prodotto:	Apple
Codice:	Stock
Dimensioni:	7x14x0.7 cm
Peso:	0.17 Kg
Processore:	A15 Bionic
Memoria:	128 GB
Dimensione display:	13.3 x 6.8 cm
Sistema Operativo:	iOS
Autonomia:	15.5 h
Fotocamera:	12 Mp
Funzioni Aggiuntive:	Doppia fotocamera

Below the table, it says 'Disponibilità: Prodotto disponibile' and 'Quantità disponibile: 13'. The price is listed as 'Prezzo: 999,00€ -15% 798,15€'. A quantity input field is shown with a value of '1'. A blue 'Aggiungi al carrello' button is at the bottom. Below the product details, there is a section for 'Recensioni prodotto:' with a 5-star rating from 'Jennifer' dated 'Feb. 20, 2023' with the review 'Il miglior phone di sempre!'.

Visualizzazione ordini con user autenticato cliente

The screenshot shows the order history page for a logged-in user ('Benvenuta, Jennifer'). At the top, there is a navigation bar with links for phone number (+39 349 1823467), email (contact@byteshop.com), language (Italiano), currency (€ EUR), and user info (Benvenuta, Jennifer). Below the navigation is the store logo and a search bar. The main content area shows a section titled 'I miei ordini:' with two order entries. The first order is for an iPhone 13 (Ordine numero: GAyayzWw) with details: Data: April 20, 2023, Indirizzo di consegna: via marconi 15, Stato: Confermato, Importo totale: 764,98€. The second order is for a Cover Iphone 13 (Ordine numero: BJUIF1XR) with details: Data: Feb. 10, 2023, Indirizzo di consegna: via marconi 15, Stato: Confermato, Importo totale: 847,12€. Both orders have a small image of the product next to the order number.

Visualizzazione carrello con user autenticato cliente

Il mio carrello:

Totale parziale: 1563.13€ [Acquista](#)

	Prodotto: Iphone 13 Quantità selezionata: 1 <input type="button" value="Aggiorna"/> Elimina Prodotto	Prezzo totale: 798.15€
	Prodotto: Samsung UE50AU7170UXZT Quantità selezionata: 2 <input type="button" value="Aggiorna"/> Elimina Prodotto	Prezzo totale: 764.98€

Gestione prodotti user autenticato admin

Prodotti che hai inserito:

Gestione utenti user autenticato admin

3 utenti registrati:

Username: Jennifer Nome: Jennifer Cognome: Reggiani Email: reggiani@live.it Data Nascita: Dec. 30, 1990 Via: via dei fiori, 123 Indirizzo: via marconi Numero Civico: 15 Città: Modena Provincia: Modena CAP: 41019 Elimina Utente	Username: Daniele Nome: Daniele Cognome: Zanini Email: daniele09@gmail.com Data Nascita: June 7, 1999 Via: via dei fiori, 123 Indirizzo: via san michele Numero Civico: 35 Città: Modena Provincia: Modena CAP: 41019 Elimina Utente	Username: Lorenzo Nome: Lorenzo Cognome: Comardi Email: lorenzo21@gmail.com Data Nascita: March 4, 1997 Via: via dei fiori, 123 Indirizzo: via pagani Numero Civico: 12 Città: Modena Provincia: Modena CAP: 41019 Elimina Utente
---	---	--