Tesina di Tecnologie Web di:
Singh Baljinder
Risotorante

Applicazione web per la gestione di una ristorante

L’applicazione che ho sviluppato riguarda la gestione di una ristorante.

L’applicazione è pensata per essere utilizzata da utenti non registrati e utenti registrati.

Gli **utenti non registrati** hanno la possibilità di registrarsi, visualizzare le informazioni sui prodotti disponibili , visualizzare la gallery e contatti.

Come **utente registrato** posso, in più rispetto agli utenti non registrati, accedere e modificare il mio profilo, aggiungere i prodotti nel carrello per ordinare a casa come(Just Eat). Inoltre l’utente registrato potrà prenotare i posti per mangiare in ristorante.

L’account **amministratore**, utilizzabile solamente dal admin del ristorante. Inoltre egli è l’unico che può aggiungere foto per la gallery,i nuovi prodotti oltre ad avere tutti i permessi per gestire qualsiasi istanza nel db dell’applicazione.

L’account **restaurator**, utilizzabile solamente dal proprietario dell ristorante. Inoltre può aggiungere prodotti e confermare ordini eliminare prenotazione .

Tecnologie Usate

Abbiamo usato un ambiente di sviluppo che comprende un editor e vari tool di analisi, test e debug del codice, Pycharm. Utilizzeremo il framework Django, scrivendo codice principalmente in python, html e javascript,ajax.

Organizzazione logica dell'applicazione a livello di codice

L'applicazione è suddivisa in 3 applicazioni:

1. Ristorante: l'applicazione base da cui tutto parte e che prende il nome della ristorante gestita dall'applicazione.
2. core: applicazione che possiamo definire il nucleo del progetto. Qui vengono gestite le entità più importanti come gli utenti, gli prodotti, prenotazione, gestione pagamento e la gallery.

utente : applicazione che possiamo definire il gestione del utente del progetto. Qui vengono gestite le entità più importanti come gli utenti, gli indirizzi .

Motivazione delle scelte fatte

Le scelte più importanti fatte per il progetto hanno riguardato nel complesso quelle fatte per l'implementazione di funzionalità pensate per rendere la vita dell'utente finale molto semplice.

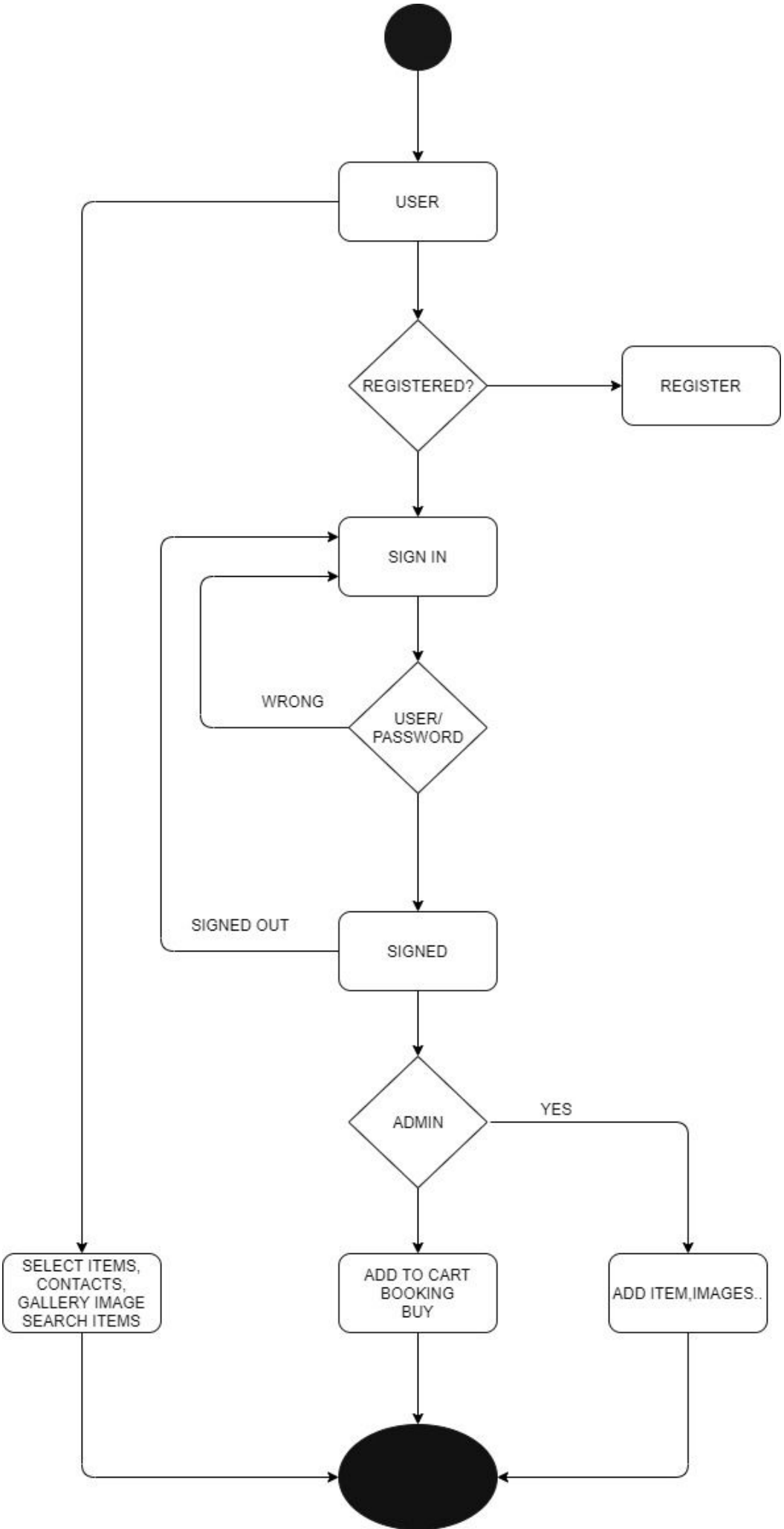
Abbiamo cercato di realizzare un'interfaccia user-friendly basata sul colore e sulla fantasia. La scelta di implementare tutte le app in maniera interna, partendo da zero è stata vincente, in primo perché altrimenti non potevo sperimentare e giocare con il codice, ed in secondo luogo perché non avrei potuto personalizzare a sufficienza l'app già esistente secondo i nostri gusti. Inoltre abbiamo deciso di separare il codice in 2 applicazioni (core e booking): questo perché abbiamo ritenuto più conveniente, dal punto di vista dell'organizzazione del codice, mantenere divisa la gestione di diverse entità, e in ogni applicazione abbiamo quindi raccolto le funzionalità che vanno a lavorare sulle entità contenute

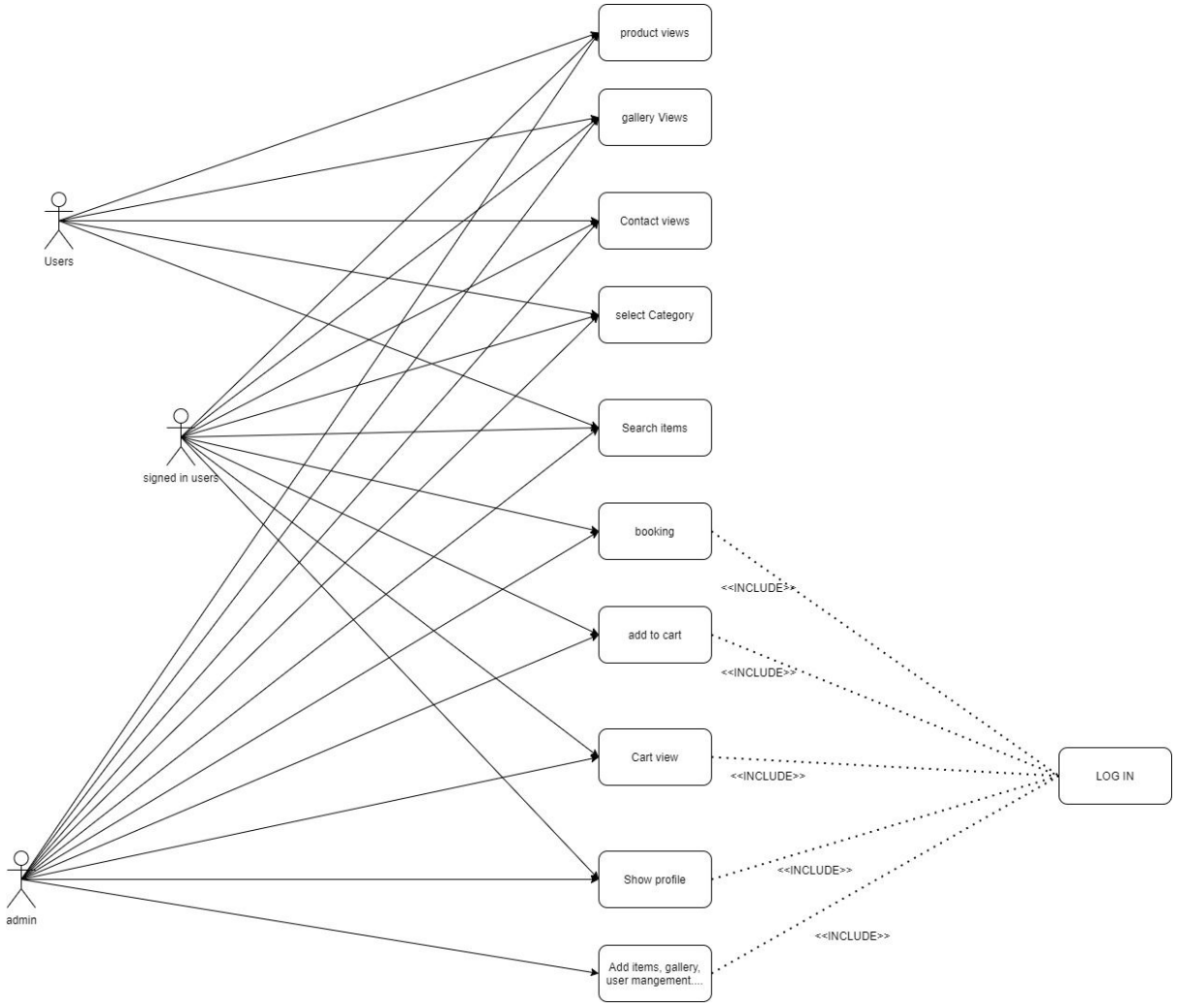
Problemi riscontrati

Ovviamente i problemi più consistenti sono stati quelli relativi all'implementazione dell'app core. Il dover gestire diversi fattori (ordini, pagamenti, gestione e modifica di ordine,conferma ordine) è risultata la parte più complicata del progetto e contestualmente quella che ha portato via più tempo. Ho dovuto fare un bel po di lavoro di ricerca per selezionare un'app che rispecchia il nostro stile da poter inserire, senza riuscirci. Allora ho iniziato un secondo lavoro di ricerca per avere gli strumenti per implementare un'app personale. Ovviamente c'è stato anche un gran lavoro di codifica e programming per sviluppare questa parte, tuttavia nonostante le difficoltà sono riuscito a realizzare un app funzionale.

Diagramma delle classi

Il diagramma mostra la suddivisione del progetto in 2 macro aree che rispecchiano la applicazione dove viene gestite la entità del sistema. Descrive anche le associazioni che esistono tra le diverse entità.


USER
REGISTERED?
REGISTER
SIGN IN
WRONG
USER/
PASSWORD
SIGNED OUT
SIGNED
ADMIN
YES
SELECT ITEMS,
CONTACTS,
GALLERY IMAGE
SEARCH ITEMS
ADD TO CART
BOOKING
BUY
ADD ITEM,IMAGES..


product views
gallery Views
Contact views
select Category
Search items
booking
add to cart
Cart view
Show profile
Add items, gallery, user mangement....
Users
signed in users
admin
<<INCLUDE>>
<<INCLUDE>>
<<INCLUDE>>
<<INCLUDE>>
<<INCLUDE>>
LOG IN

Tests

Ho effettuato tests suddividendoli in due parti quelli valide e quelle non valide.

Prima parte - tests su models item e order questa parte abbiamo effettuato tests che controllano la correttezza degli input nel form per la creazione delle prodotti. Inizialmente abbiamo impostato una fase di setUp dove vengono create tutte le entità necessarie per effettuare correttamente i test.

```python
from django.contrib.auth import get_user_model
from django.contrib.auth.models import User
from django.test import TestCase
from django.urls import reverse


# test view
from core.forms import BookingForm
from core.models import *


class TestViews(TestCase):

    def setUp(self):
        self.user_test = User.objects.create(email='test@test.it', first_name='test_user',
                                             last_name='test_last', password='test')
        self.items=Item.objects.create(title='title', price=5, category='D', label='P',
                                       slug='S', description="ciao")
        self.booking=Booking.objects.create(user=self.user_test, description='ciao',
                                            chair=3, title='nomePersona', date_start='2020-12-1 16:00')
        self.booking.save()
```

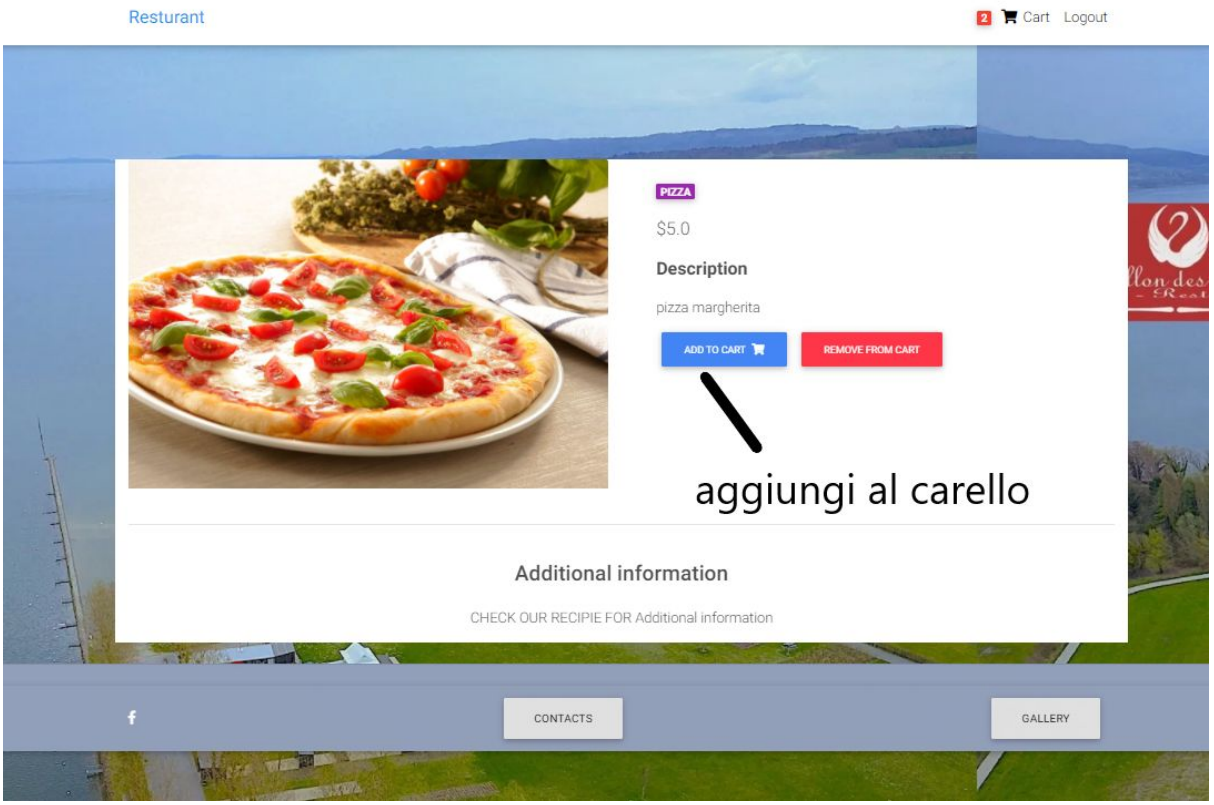
```python
def test_model(self):
    item_test = Item()
    item_test.title = "test da prova"
    item_test.price = 5
    item_test.category = 'A'
    item_test.label = 'P'
    item_test.slug = "sl"
    item_test.description="descrizione"
    item_test.save()
    record = Item.objects.get(id=item_test.id)
    self.assertEqual(record, item_test)


def test_order_item(self):
    order_item_test = OrderItem()
    order_item_test.user = self.user_test
    order_item_test.ordered = True
    order_item_test.item= self.items
    order_item_test.quantity = 4
    order_item_test.save()
    record = OrderItem.objects.get(pk=order_item_test.pk)
    self.assertEqual(record, order_item_test)

def test_address(self):
    address_test = Address()
    address_test.user = self.user_test
    address_test.città="Modena"
    address_test.cap="2000"
```

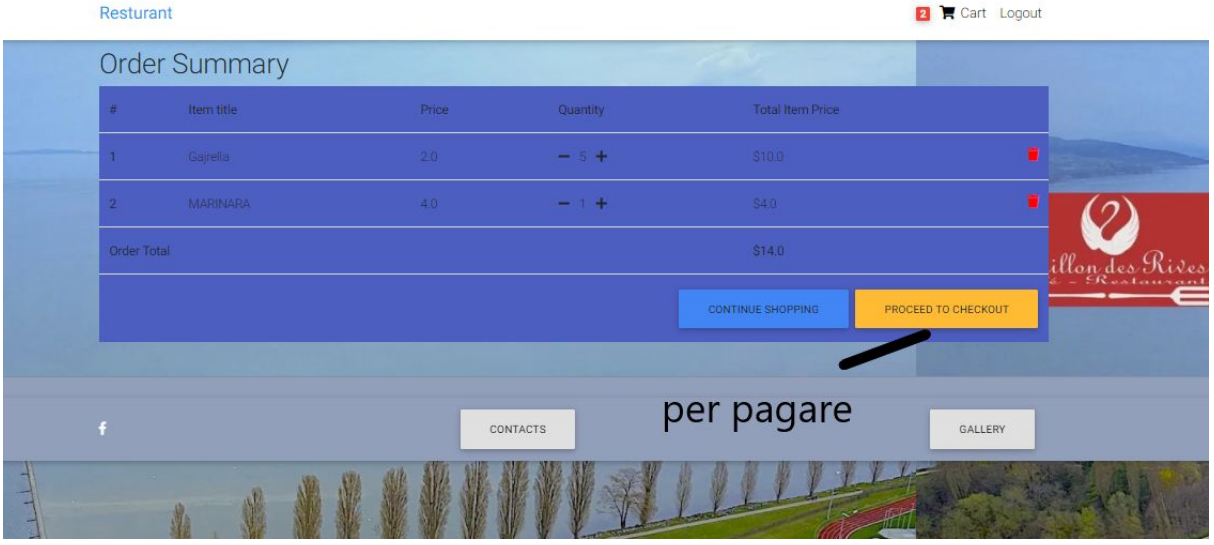
```python
def test_crea_booking_non_valido(self):
    """
        form non valido per la creazione di una Prenotazione , numero dei posti = 0,
        deve ritornare False
    """
    form = BookingForm(data={
        'title': 'prenoto per 5',
        'description': 'descrizione',
        'chair': 0,
        'date_start': '2222-12-05 16:00'
    })
    self.assertEqual(form.is_valid(), False)


def test_crea_booking_valido(self):
    """
        form non valido per la creazione di una Prenotazione , numero dei posti = 0,
        deve ritornare False
    """
    form = BookingForm(data={
        'title': 'prenoto per 5',
        'description': 'descrizione',
        'chair': 2,
        'date_start': '12/12/2021 16:00'
    })
    self.assertEqual(form.is_valid(), True)
```

In fase di creazione di una prenotazione il campo “chair” (posti)deve accettare solamente un intero maggiore di 1e minore di 6. Come parametri, abbiamo prima passato 4(VALIDO), poi 8 (NON VALIDO) e infine un numero negativo -8(NON VALIDO)

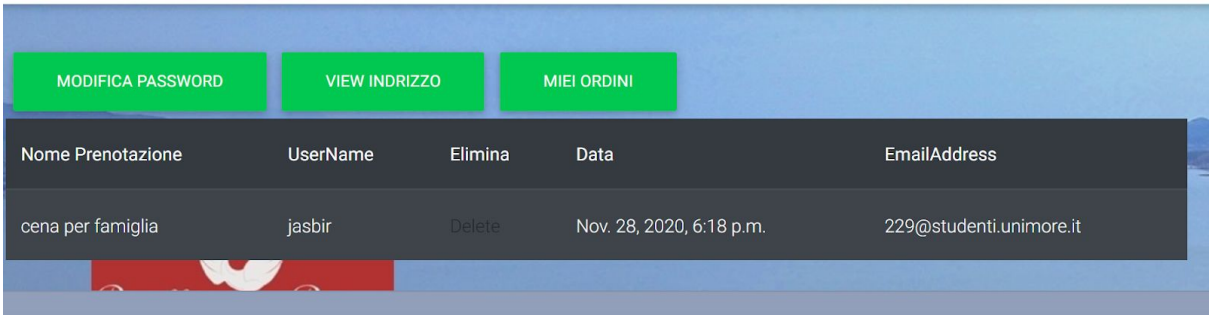
```python
def test_Crea_booking_form(self):
    form= BookingForm(data={'title':'titolo','description':'descrizione per prenotare','chair':'4','date_start':'10/1
    self.assertEqual(form.is_valid(),True)

def test_Crea_booking_form_non_valido(self):
    form = BookingForm(
        data={'title': 'titolo2', 'description': 'descrizione per prenotare','chair':'8', 'date_start': '10/10/2020 2
    self.assertEqual(form.is_valid(), False)

def test_Crea_booking_form_non_valido(self):
    form = BookingForm(
        data={'title': 'titolo', 'description': 'descrizione per prenotare', 'chair': '-8',
              'date_start': '10/10/2020 25:20'})
    self.assertEqual(form.is_valid(), False)
```

Risultati (screenshot per le parti più rilevanti)


Resturant
2 Cart Logout
PIZZA
$5.0
Description
pizza margherita
ADD TO CART
REMOVE FROM CART
aggiungi al carello
Additional information
CHECK OUR RECIPIE FOR Additional information
CONTACTS
GALLERY


Resturant
2 Cart Logout
Order Summary
Item title Price Quantity Total Item Price
1 Gajrella 2.0 − 5 + $10.0
2 MARINARA 4.0 − 1 + $4.0
Order Total $14.0
CONTINUE SHOPPING
PROCEED TO CHECKOUT
per pagare
CONTACTS
GALLERY

interfaccia di profilo utente, utente può disdire appuntamento aggiornare indirizzo di spedizione, modificare password e controllare ordini fatti finora.



in questa pagina proprietario può confermare ordini spediti e controllare grafo di vendite di ogni mese, inoltre può controllare prenotazione di utenti ,e aggiungere nuovo prodotto nel menu.