Ingegneria del software avanzata

Presentazione progetto - Elena Mari 144886

Department of Engineering Ferrara
Università degli Studi di Ferrara

Sommario

- Specifiche
- Tecnologie usate
- Implementazione
- Testing
- Containerizzazione e pipeline

Specifiche (informali)

È stata sviluppata un'applicazione web ecommerce per la vendita di prodotti tipici locali di una piccola azienda/agriturismo.
Un **admin** può:

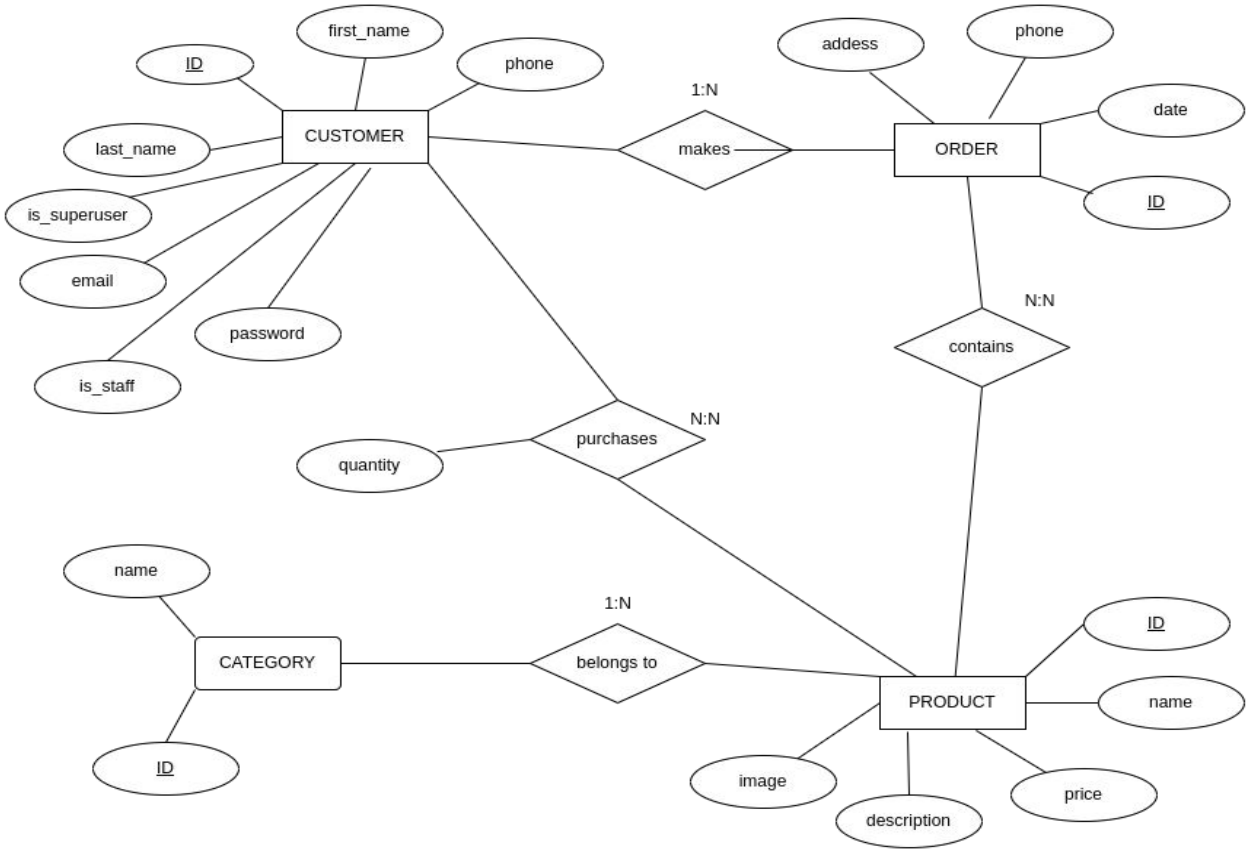
- Effettuare login
- Aggiungere, rimuovere o modificare un prodotto
- Aggiungere, rimuovere o modificare una categoria di prodotto

Un **utente**/**cliente** può:

- Registrarsi ed effettuare il log in
- Modificare il proprio profilo
- Visualizzare i prodotti e il proprio carrello
- Aggiungere/rimuovere prodotti dal carrello e modificare la quantità
- Effettuare il checkout
- Visualizzare lo storico ordini

DE FERRARIAE UNIVERSITAS · 13 91 · EX LABORE FRUCTUS

Diagramma ER



UML - Diagramma dei casi d’uso

Tecnologie usate

Backend e Frontend:
- ➔ Python
- ➔ Django
- ➔ Bootstrap

Package manager:
- ➔ pip

Testing automatizzato:
- ➔ TestCase (Django)

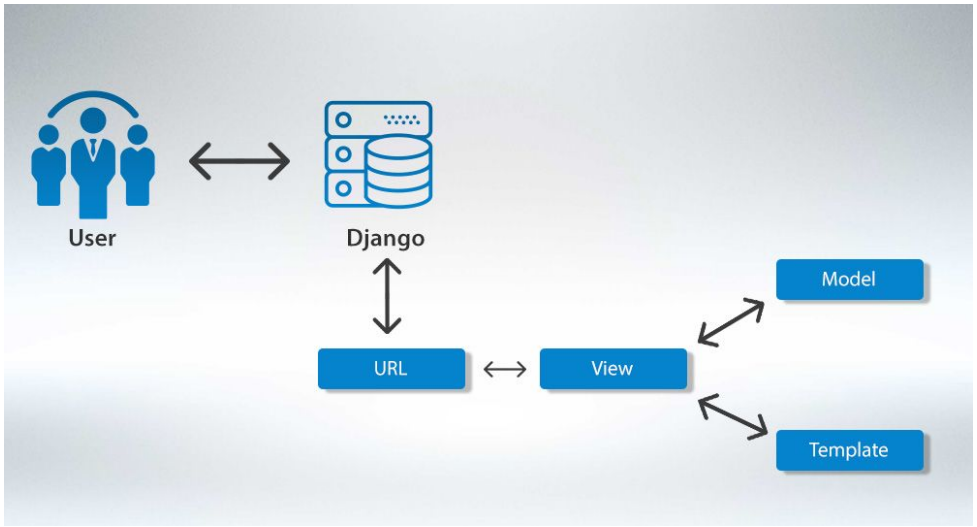
Database:
- ➔ SQLite

Controllo di versione e backup:
- ➔ Git
- ➔ GitHub

Containerizzazione e pipeline:
- ➔ Docker
- ➔ DockerHub
- ➔ GitHub Actions

Implementazione

È stato utilizzato il framework **Django**, il quale segue il modello Model - View - Template (**MVT**):



- **Model**: struttura dei dati
- **View**: logica applicativa e funzionalità
- **Template**: layout e visualizzazione

Modelli e migrations

Modelli creati:

- ➔ **Customer**
- ➔ **Order**
- ➔ **Product**
- ➔ **Category**
- ➔ **Cart**
- ➔ **OrderProduct**

Esempio modello ‘Product’

```python
from django.db import models
from .category import Category

class Product(models.Model):
    name = models.CharField(max_length=255)
    price = models.DecimalField(max_digits=5, decimal_places=2, default=0.00)
    category = models.ForeignKey(Category, on_delete=models.CASCADE, null=True, blank=True)
    description = models.CharField(max_length=255, default='', null=True, blank=True)
    image = models.ImageField(upload_to='uploads/products/')

    @staticmethod
    def get_products_by_id(ids):
        return Product.objects.filter(id__in = ids)

    @staticmethod
    def get_all_products():
        return Product.objects.all()
```

I modelli vengono mappati nel database con tecnica Object Relational Mapping (**ORM**) attraverso le **migration**.

DE: FERRARIAE UNIVERSITAS · 13 91 · EX LABORE FRUCTUS

Testing

È stata creata una suite di test automatizzati attraverso il modulo **test** e la classe **TestCase** fornitI dal Framework Django.

Unit Test:
Test su tutti i metodi dei modelli e sui form

Integration Test:
Test sulle funzionalità, ad esempio effettuare un ordine

Test Coverage:
83 test con copertura del 96%

```
Ran 83 tests in 63.871s
----------------
965     41    96%
```

DE: FERRARIAE UNIVERSITAS · 13 91 · EX LABORE FRUCTUS

Unit Test

Sono stati testati tutti i metodi dei modelli e i form di input

```python
class TestLoginForm(TestCase):
    def setUp(self):
        self.user = Customer.objects.create_user(
            email='test@example.com',
            password='testpassword',
            first_name='Test',
            last_name='User',
            phone='1234567890'
            )

    def test_valid_login(self):
        form_data = {'email': 'test@example.com', 'password': 'testpassword'}
        form = LoginForm(data=form_data)
        self.assertTrue(form.is_valid())

    def test_invalid_login(self):
        form_data = {'email': 'test@example.com', 'password': 'wrongpassword'}
        form = LoginForm(form_data)
        self.assertFalse(form.is_valid())
        self.assertEqual(form.errors['__all__'], ['Invalid email or password'])

    def test_missing_email(self):
        form_data = {'password': 'testpassword'}
        form = LoginForm(data=form_data)
        self.assertFalse(form.is_valid())
        self.assertEqual(form.errors['email'], ['This field is required.'])
```

Test del form di login

```python
class TestProduct(TestCase):
    def setUp(self):
        self.category1 = Category.objects.create(name="Category 1")
        self.category2 = Category.objects.create(name="Category 2")
        self.product1 = Product.objects.create(name="Test Product 1",
        self.product2 = Product.objects.create(name="Test Product 2",
        self.product3 = Product.objects.create(name="Test Product 3",

    def test_get_products_by_id(self):
        products = Product.get_products_by_id([self.product1.id, self
        self.assertEqual(products.count(), 2)
        self.assertIn(self.product1, products)
        self.assertIn(self.product3, products)

    def test_get_all_products(self):
        products = Product.get_all_products()
        self.assertEqual(products.count(), 3)
        self.assertIn(self.product1, products)
        self.assertIn(self.product2, products)
        self.assertIn(self.product3, products)
```

Test del modello ‘product’

DE:

FERRARIAE UNIVERSITAS · 13 91 · EX LABORE FRUCTUS

Integration test

Sono state testate le principali funzionalità dell'applicazione

```python
class CartTestCase(TestCase):
    def setUp(self):
        self.customer = Customer.objects.create_user(email='testuser@email.com', password='12345', first_
        self.product = Product.objects.create(name='Test Product', price=10)

    def test_add_product_to_cart(self):
        self.client.login(email='testuser@email.com', password='12345')
        response = self.client.post(reverse('cart'), {'product_id': self.product.id, 'action': 'add'})
        self.assertEqual(response.status_code, 302)
        self.assertEqual(CartItem.objects.filter(customer=self.customer, product=self.product).count(), 1

    def test_remove_product_from_cart(self):
        CartItem.objects.create(customer=self.customer, product=self.product, quantity=1)
        self.client.login(email='testuser@email.com', password='12345')
        response = self.client.post(reverse('cart'), {'product_id': self.product.id, 'action': 'delete'})
        self.assertEqual(response.status_code, 302)
        self.assertEqual(CartItem.objects.filter(customer=self.customer, product=self.product).count(), 0

    def test_update_product_quantity_in_cart(self):
        CartItem.objects.create(customer=self.customer, product=self.product, quantity=1)
        self.client.login(email='testuser@email.com', password='12345')
        response = self.client.post(reverse('cart'), {'product_id': self.product.id, 'action': 'update',
        self.assertEqual(response.status_code, 302)
        self.assertEqual(CartItem.objects.get(customer=self.customer, product=self.product).quantity, 2)
```

Test della gestione del carrello utente con tutte le funzionalità

DE: FERRARIAE UNIVERSITAS · 13 91 · EX LABORE FRUCTUS

Containerizzazione

L'applicazione è stata containerizzata tramite **Docker**:

```
# Usa un'immagine Python ufficiale come immagine base
FROM python:3.8

# Imposta la directory di lavoro nel container
WORKDIR /usr/src/app

# Copia il file delle dipendenze e installale
COPY requirements.txt ./
# Crea un ambiente virtuale per isolare le dipendenze
RUN python -m venv .venv
RUN .venv/bin/python -m pip install --upgrade pip
RUN .venv/bin/pip install --no-cache-dir -r requirements.txt

# Copia il resto del codice sorgente del progetto nel container
COPY . .

# Espone la porta 8000
EXPOSE 8000

COPY entrypoint.sh /entrypoint.sh
RUN chmod +x /entrypoint.sh
ENTRYPOINT ["/entrypoint.sh"]
```

Dockerfile

```
#!/bin/bash

.venv/bin/python manage.py makemigrations
.venv/bin/python manage.py migrate

if [ ! -f data/db.sqlite3 ]; then
    venv/bin/python manage.py createsuperuser --noinput --email
fi

.venv/bin/python manage.py runserver 0.0.0.0:8000
```

Script entrypoint.sh per gestione migrations e admin

DE: FERRARIAE UNIVERSITAS · 13 91 · EX LABORE FRUCTUS

Docker-compose

È stato utilizzato docker-compose per la gestione dei volumi (database e file statici):

```yaml
services:
  unpodiorigine:
    image: unpodiorigine:latest
    ports:
      - 8000:8000
    volumes:
      - ./data:/usr/src/app/data
      - ./uploads:/usr/src/app/uploads
```

L'applicazione è disponibile alla porta 8000 (porta default di Django).

Pipeline

Sono state utilizzate le GitHub Actions per la pipeline CI/CD. ogni volta che viene eseguita una push su uno dei due branch o una pull request sul branch develop:

```yaml
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Set up Python
        uses: actions/setup-python@v2
        with:
          python-version: "3.8"
      - name: Install dependencies
        run: |
          python -m venv .venv
          .venv/bin/python -m pip install --upgrade pip
          .venv/bin/pip install -r requirements.txt
      - name: Create migrations
        run: |
          .venv/bin/python manage.py makemigrations
      - name: Run tests
        run: |
          .venv/bin/python manage.py test -v2
```

➔ Vengono eseguiti i test

DE: FERRARIAE UNIVERSITAS · 13 91 · EX LABORE FRUCTUS

Caricamento su DockerHub

```yaml
build_and_push:
  needs: test
  runs-on: ubuntu-latest
  if: github.ref == 'refs/heads/master'
  steps:
    - uses: actions/checkout@v2
    - name: Build Docker image
      run: |
        docker build -t ${{ secrets.DOCKER_USERNAME }}/unpodiorigine:${{ github.sha }} .
    - name: Log in to Docker Hub
      uses: docker/login-action@f4ef78c080cd8ba55a85445d5b36e214a81df20a
      with:
        username: ${{ secrets.DOCKER_USERNAME }}
        password: ${{ secrets.DOCKER_PASSWORD }}
    - name: Push Docker image to Docker Hub
      run: |
        docker push ${{ secrets.DOCKER_USERNAME }}/unpodiorigine:${{ github.sha }}
```

➔ Se i test hanno successo, si effettua la push su DockerHub

DE:

FERRARIAE UNIVERSITAS · 13 91 · EX LABORE FRUCTUS

FINE

Grazie per l’attenzione.

Department of Engineering Ferrara

Università degli Studi di Ferrara