Product Review System using Content-based Recommender Systems

SOFTWARE ENGINEERING LABORATORY
STUDENT: CORNEA HOREA – IONUT
GROUP: 30433

Abstract:

The Product Review System is a Django-based web application designed to enhance user engagement by providing personalized product recommendations and a user-friendly review platform. Authenticated users can browse products, submit detailed reviews, and receive tailored suggestions based on their past interactions. The system implements a content-based recommendation approach, using natural language processing to analyze product descriptions and user preferences. This document presents a detailed overview of the system's architecture, implementation process and bibliographic research.

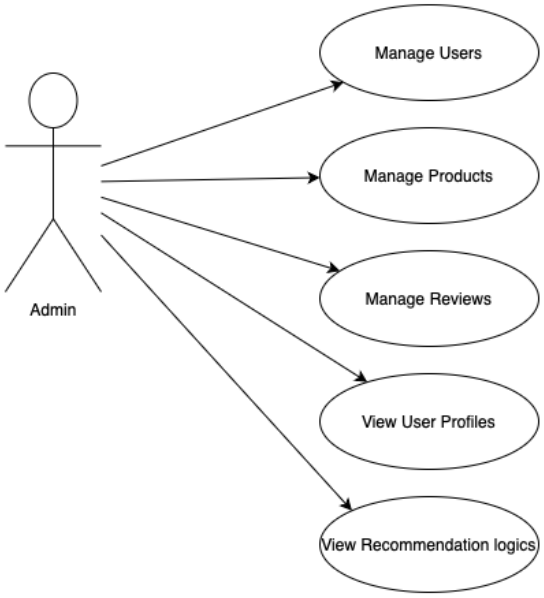
Table of Contents

1. Design

*The system is designed using the **Model-View-Controller (MVC)** pattern as follows:*

- ***Model**: Defines the structure of the data (e.g., Product, Review).*
- ***View**: Contains the logic for presenting data (e.g., product pages, review forms).*
- ***Controller**: Handles the interactions and manages requests.*

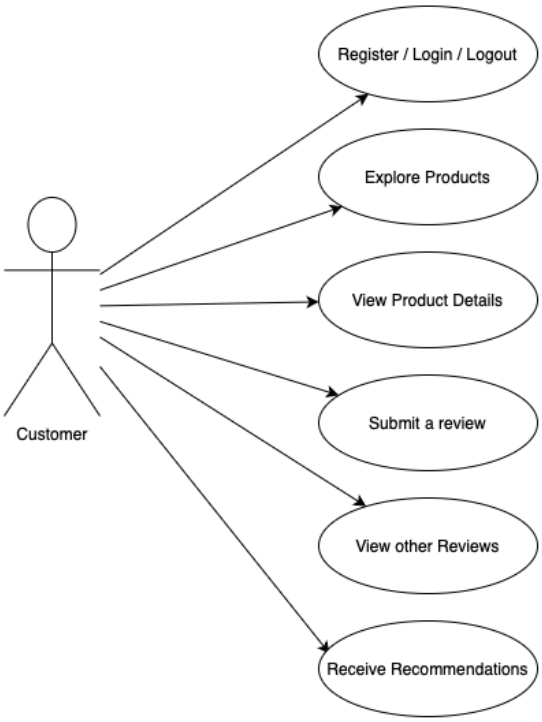
Use Case Diagram



*The **Standard User** represents a customer or registered user who interacts with the system to browse, review, and receive recommendations.*

Use cases for the Standard User:

- *Register / Login / Logout*
- *Explore Products*
- *View Product Details*
- *Submit a review*
- *View other Reviews*
- *Receive Recommendations*

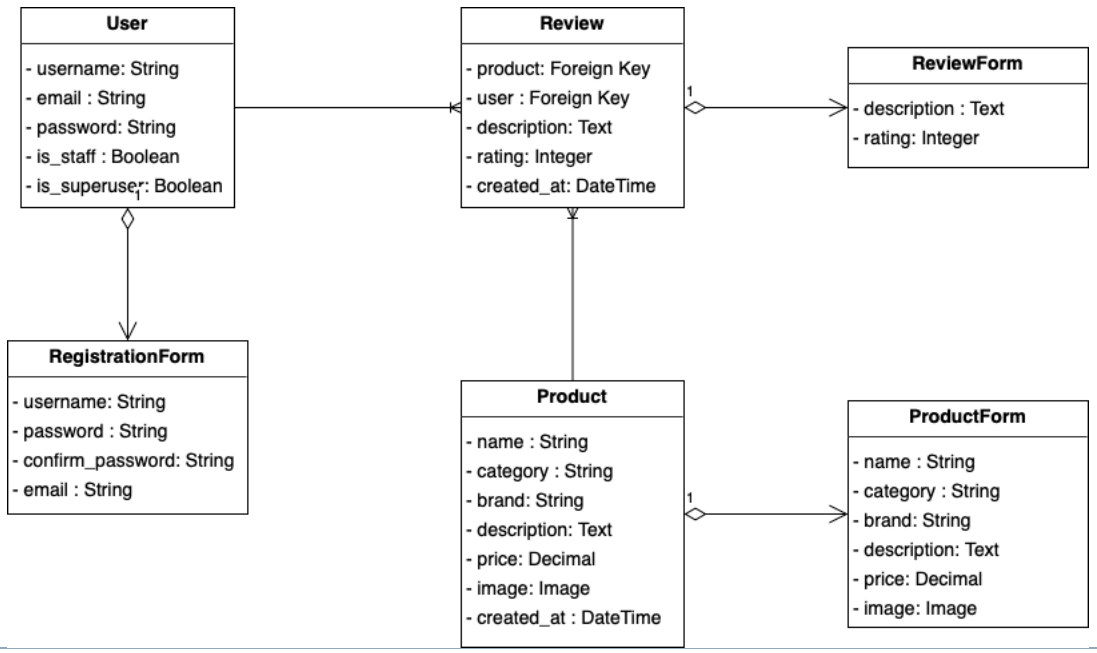


The **Admin** is responsible for managing the content and ensuring that inappropriate reviews are moderated.

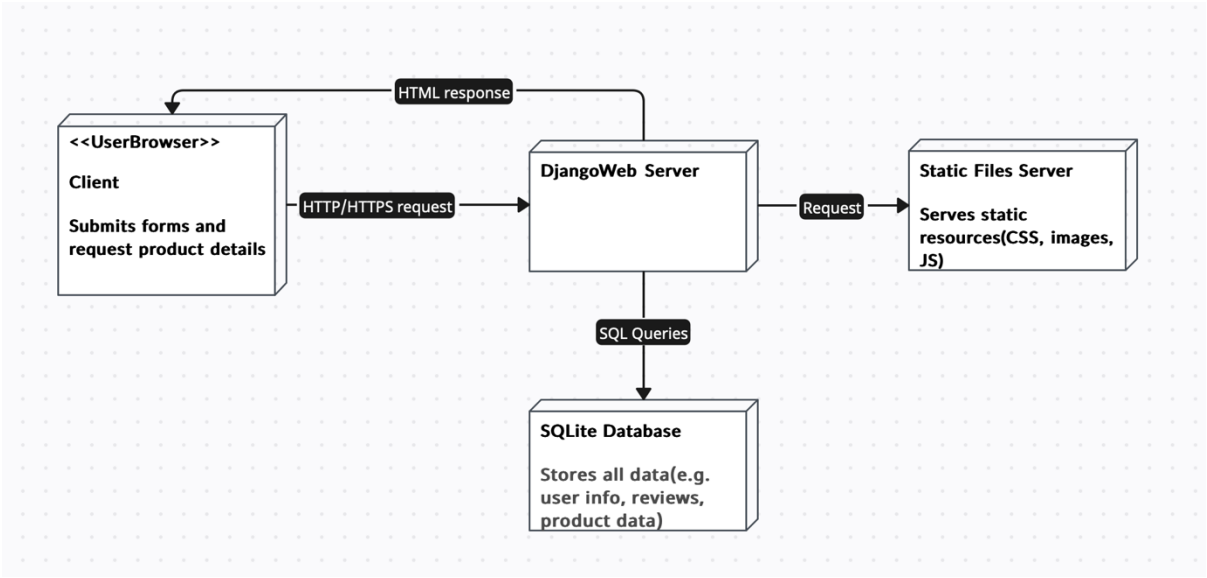
Use cases for the Admin:

- Manage Users
- Manage Products
- Manage Reviews
- View User Profiles
- View Recommendations Logics

Class Diagram



Deployment Diagram



2. Implementation

This section outlines the details of how the system was implemented, covering key components, system flows, important views, forms, and the models. Each aspect is described with code examples and explanations.

System Components Overview

*The application follows the **Django MVC pattern** (Model-View-Controller). Below is an overview of the components:*

1. ***Models (**models.py**)** – Define the data structure for products and reviews.*
2. ***Forms (**forms.py**)** – Handle user input and validation for various forms.*
3. ***Views (**views.py**)** – Contain the business logic for processing requests and rendering responses.*
4. ***Templates (HTML pages)** – Provide the front-end interface for users.*
5. ***Static Files (CSS, JavaScript)** – Add styling and interactivity to the pages.*
6. ***Recommender System (**content_based.py**)** – Generates personalized recommendations using **TF-IDF** and **cosine similarity**.*

Models Explanation

The Product and Review models define the database structure for the system.

1. Product Model:

```
class Product(models.Model):
    name = models.CharField(max_length=255)
    category = models.CharField(max_length=255, default='Unknown')
    brand = models.CharField(max_length=255, default='Unknown')
    description = models.TextField()
    price = models.DecimalField(max_digits=10, decimal_places=2)
    image = models.ImageField(upload_to='static/product_images', null=True, blank=True)
    created_at = models.DateTimeField(auto_now_add=True)  # Auto set the time when
the product is created
```

- *name: The name of the product.*
- *category **and** brand: Categorical data for filtering.*
- *description: Text-based product details used in recommendation logic.*
- *image: An optional image file for each product.*

2. Review Model

```
class Review(models.Model):
    product = models.ForeignKey(Product, related_name='reviews',
on_delete=models.CASCADE)
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    description = models.TextField()
    rating = models.PositiveIntegerField(choices=[(i, str(i)) for i in range(1, 6)])  # 1 to 5 stars
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f"Review by {self.user.username} for {self.product.name}"
```

- *product: A foreign key linking to the associated product.*
- *user: A foreign key linking to the user who created the review.*
- *rating: A 1 to 5-star rating.*
- *description: User's text-based review.*

Forms Explanation

Forms handle the front-end interface for user input and perform validation.

- *RegistrationForm – Handles user sign-up.*
- *ReviewForm – Handles user reviews.*
- *PasswordResetForm – Handles password reset functionality.*

```
class RegistrationForm(forms.ModelForm):
    password = forms.CharField(widget=forms.PasswordInput(attrs={'class': 'form-control'}))
    confirm_password = forms.CharField(widget=forms.PasswordInput(attrs={'class': 'form-
control'}))

    def clean(self):
        cleaned_data = super().clean()
        if cleaned_data.get("password") != cleaned_data.get("confirm_password"):
            raise forms.ValidationError("Passwords do not match.")
```

Views Explanation

1. User Authentication and Session Management

- *custom_register_view(request): Handles user registration.*
- *custom_login_view(request): Handles user login.*
- *custom_logout_view(request): Logs out the user.*

```
def custom_register_view(request):
    if request.method == 'POST':
        form = RegistrationForm(request.POST)
        if form.is_valid():
            user = form.save(commit=False)
            user.set_password(form.cleaned_data['password'])
            user.save()
            return redirect('login')  # Redirects to login page after successful registration
    else:
        form = RegistrationForm()
    return render(request, 'registration/register.html', {'form': form})
```

2. *Product Management*

- *add_product(request): Allows admins to add new products.*
- *edit_product(request, product_id): Enables admins to update product information.*
- *delete_product(request, product_id): Removes a product from the database.*

```
@user_passes_test(is_admin)
def add_product(request):
    if request.method == "POST":
        form = ProductForm(request.POST, request.FILES)
        if form.is_valid():
            form.save()
            return redirect('explore_products')  # Redirect to the product list after adding
    else:
        form = ProductForm()
    return render(request, 'products/add_product.html', {'form': form})
```

3. *Review System*

- *product_detail(request, product_id): Displays product details and reviews.*
- *delete_review(request, review_id): Enables admins to delete reviews.*

```
def product_detail(request, product_id):
    product = get_object_or_404(Product, id=product_id)
    reviews = product.reviews.all()  # Fetches all reviews for this product

    if request.method == 'POST':
        form = ReviewForm(request.POST)
        if form.is_valid():
            review = form.save(commit=False)
            review.product = product
            review.user = request.user
            review.save()
            return redirect('product_detail', product_id=product.id)
    else:
        form = ReviewForm()

    return render(request, 'products/product_detail.html', {
        'product': product,
        'reviews': reviews,
        'form': form,
    })
```

3. Bibliographic Research on Recommender Systems

Introduction

Recommender systems are advanced software tools designed to predict user preferences and suggest relevant items such as products, books, or movies. They play a crucial role in reducing information overload by personalizing user experiences and assisting in decision-making. Recommender systems have become indispensable in e-commerce (e.g., Amazon), streaming platforms (e.g., Netflix, Spotify), and educational content providers. The core purpose of these systems is to analyze large datasets and offer user-specific content, thus enhancing engagement and user satisfaction.

Categories of Recommender Systems

Recommender systems can be categorized into three main types: content-based filtering, collaborative filtering, and hybrid systems.

1. Content – Based filtering

Content-based recommender systems predict user preferences by analyzing item features and comparing them to the user's past interactions. Key features (e.g., genre, keywords, tags) of the recommended items match the user's history.

- **Advantages:**
    - Personalized recommendations based solely on the individual user.
    - No need for data from other users.
- **Disadvantages**:
    - Over-specialization: The system tends to recommend items that are too similar to what the user has already interacted with.
    - Limited exploration of diverse items.

2. Collaborative Filtering

Collaborative filtering identifies relationships between users and items by examining their interactions. It assumes that users with similar behavior (likes or purchases) will prefer similar items.

- **User-Based Collaborative Filtering:**

Recommends items based on similarities between user preferences.

- **Item-Based Collaborative Filtering:**

Suggests items that are similar to those the user has interacted with.

- **Advantages:**
    - High-quality recommendations when there is sufficient user interaction data.
- **Disadvantages:**

- **Cold-Start Problem:** The system struggles to make recommendations for new users or items.
- Requires large datasets to detect meaningful patterns.

3. *Hybrid Recommender Systems*

Hybrid systems combine content-based and collaborative filtering approaches to overcome their respective limitations. For example, Netflix employs a hybrid recommender system that combines item-based collaborative filtering with personalized content-based suggestions.

- **Advantages:**
  - Reduces the cold-start problem by incorporating content features for new items.
  - Provides more diverse recommendations.
- **Disadvantages:**
  - Can be computationally expensive and complex to implement.

Techniques and Algorithms

1. Nearest Neighbor Methods

These algorithms recommend items that are "nearest" (similar) to the ones the user has previously interacted with based on cosine similarity or Euclidean distance.

2. Matrix Factorization (SVD, SVD++)

Matrix factorization techniques, such as Singular Value Decomposition (SVD), reduce data into a lower-dimensional space to uncover latent features and patterns in user-item interactions.

3. Bayesian Classifiers

These probabilistic models predict preferences by computing the likelihood of a user preferring an item based on prior data.

4. Content-Based Filtering with TF-IDF

TF-IDF (Term Frequency-Inverse Document Frequency) is a common method for converting text into numerical data by weighting unique terms more heavily. It's effective for comparing product descriptions to user preferences in content-based recommendations but can lead to over-specialized results and high memory usage for large datasets.

4. Recommendation System Explanation

The recommendation system in this project uses **content-based filtering**. It constructs a **user profile** by analyzing the reviews submitted by the user and then compares this profile to product descriptions using **TF-IDF (Term Frequency-Inverse Document Frequency)** and **cosine similarity** to recommend the most relevant products.

Building the User Prodfile

A **user profile** is built by concatenating the features (e.g., product category, brand, and description) of all the products the user has reviewed. The process is as follows:

1. Retrieve Reviews:

The system fetches all the reviews written by the user.

```
reviews = Review.objects.filter(user_id=user_id)
```

2. Construct Feature Vector:

For each reviewed product, the system constructs a text-based feature vector consisting of:

- Category of the product (e.g., "Electronics").
- Brand (e.g., "Samsung").
- Description (e.g., "Smartphone with 128GB storage").

3. Combine Features:

All the extracted features from the user's reviews are concatenated into a single text string to form the **user profile**.

TF – IDF Matrix Construction

After building the user profile, the system compares it with all available product descriptions in the database. This is done using **TF-IDF vectorization**:

1. Input Texts:

The system creates a list of text inputs:

- Product descriptions (from the database).
- The user profile text (created from reviews).

```
descriptions = product_descriptions + [user_profile_text]
```

2. TF-IDF Vectorizer:

The **TfidfVectorizer** converts the texts into numerical vectors where:

- Higher weights are assigned to unique terms in the document.

- Common terms (like "the", "a") have low weights.

```
vectorizer = TfidfVectorizer(stop_words='english')
tfidf_matrix = vectorizer.fit_transform(descriptions)
```

Computing Cosine Similarity

The system computes the **cosine similarity** between the user profile vector and each product vector. **Cosine similarity** measures the angle between two vectors:

- **Value 1:** Perfectly similar.
- **Value 0:** Completely dissimilar.

```
similarity_scores = cosine_similarity(tfidf_matrix[-1], tfidf_matrix).flatten()
```

  - The [-1] index refers to the user profile vector.
  - The similarity_scores array contains similarity scores for all products.

Selecting Top Recommendations

The system sorts the products based on their similarity scores and selects the top 5:

```
def get_similar_products_with_explanations(user_id):
    user_profile = build_user_profile(user_id)
    product_descriptions = list(Product.objects.values_list('description', flat=True))
    descriptions = product_descriptions + [user_profile]

    tfidf = TfidfVectorizer(stop_words='english')
    tfidf_matrix = tfidf.fit_transform(descriptions)
    cosine_similarities = cosine_similarity(tfidf_matrix[-1], tfidf_matrix).flatten()

    similar_indices = cosine_similarities.argsort()[-6:-1]  # Top 5 recommendations
    recommendations = [(Product.objects.get(pk=i + 1), f"Score: {cosine_similarities[i]:.2f}")
for i in similar_indices]

    return recommendations
```

Example Recommendation Flow

Example Scenario:

- The user reviews a **laptop** with "Intel processor" and "16GB RAM".
- The user profile includes these keywords.
- The system compares the user profile with all products and finds that **another laptop** with similar features ("Intel i7", "16GB RAM") has a high similarity score.
- The system recommends this laptop to the user with the explanation: *"Intel processor (score: 0.85), 16GB RAM (score: 0.75)"*.

5. Refrences

1. **Django Documentation (Web Framework) -** https://docs.djangoproject.com/en/stable/

2. **TF-IDF Implementation -** **https://www.kaggle.com/code/yassinehamdaoui1/creating-tf-idf-model-from-scratch**

3. **Recommender Systems Handbook (Bibliographic Research) - Francesco Ricci, Lior Rokach, Bracha Shapira, Paul B. Kantor "Recommender Systems Handbook" , Springer**

4. **Online Tutorials - "Building a User Authentication System with Django"** **https://realpython.com**

Appendix

content_based.py

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from App_Product_Review.models import Product, Review
import pandas as pd

def get_similar_products_with_explanations(user_id):
    user_profile_text = build_user_profile(user_id)

    if not user_profile_text:
        return []  # If no reviews, return an empty list

    # Get all product descriptions
    products = Product.objects.all()
    product_descriptions = []

    for product in products:
        product_description = f"{product.category} {product.brand} {product.description}"
        product_descriptions.append(product_description)

    # Append the user profile as the last item in the list of descriptions
    descriptions = product_descriptions + [user_profile_text]

    # Use TF-IDF Vectorizer to convert the text into vectors
    tfidf = TfidfVectorizer(stop_words='english')
    tfidf_matrix = tfidf.fit_transform(descriptions)
    save_tfidf_matrix_to_csv(tfidf, tfidf_matrix, descriptions)

    # Compute cosine similarity between the user profile (last row) and all products (all rows
except the last one)
    cosine_sim = cosine_similarity(tfidf_matrix[-1], tfidf_matrix[:-1])

    # Get the indices of the most similar products
    similar_indices = cosine_sim.argsort()[0, -6:-1]  # Top 5 similar products

    # Extract feature contributions for explanation
    feature_names = tfidf.get_feature_names_out()
    user_profile_vector = tfidf_matrix[-1].toarray()[0]
    recommendations = []

    for idx in similar_indices:
        product_vector = tfidf_matrix[idx].toarray()[0]
        product_features = []

        for i, value in enumerate(product_vector):
            if value > 0 and user_profile_vector[i] > 0:
                product_features.append((feature_names[i], user_profile_vector[i] * value))

        # Sort features by contribution to similarity
        product_features.sort(key=lambda x: x[1], reverse=True)
```

```
        # Format explanation as a string
        explanation = ", ".join([f"{feature} (score: {score:.2f})" for feature, score in
product_features[:5]])

    # Convert idx to Python int before using it
        recommendations.append((products[int(idx)], explanation))

    return recommendations

def build_user_profile(user_id):
    """
    Build a user profile based on the products they have reviewed.
    """
    reviews = Review.objects.filter(user_id=user_id)  # Retrieve all reviews by the user
    user_profile = []  # Initialize a blank user profile

    for review in reviews:
        product = review.product
        features = f"{product.category} {product.brand} {product.description}"
        user_profile.append(features)
        # Create user profile based on category, brand, and description

    return " ".join(user_profile)

def save_tfidf_matrix_to_csv(tfidf, tfidf_matrix, descriptions):
    """
    Save the TF-IDF matrix to a CSV file for inspection.
    """
    # Get feature names (unique words) from the vectorizer
    feature_names = tfidf.get_feature_names_out()

    # Convert the sparse matrix to a dense array
    dense_matrix = tfidf_matrix.toarray()

    # Create a DataFrame for better visualization
    df = pd.DataFrame(dense_matrix, columns=feature_names, index=[f"Product {i+1}" for i
in range(len(descriptions) - 1)] + ["User Profile"])

    # Save the DataFrame to a CSV file
    df.to_csv("tfidf_matrix.csv", index=True)

    print("TF-IDF matrix saved to tfidf_matrix.csv")
```

models.py

```
import datetime
from datetime import datetime
from django.contrib.auth.models import User
from django.db import models


class Product(models.Model):
    name = models.CharField(max_length=255)
    category = models.CharField(max_length=255, default='Unknown')
    brand = models.CharField(max_length=255, default='Unknown')
    description = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)  # Automatically sets date
when created
    price = models.DecimalField(max_digits=10, decimal_places=2)
    image = models.ImageField(upload_to='static/product_images', null=True, blank=True)

    def __str__(self):
        return self.name


class Review(models.Model):
    product = models.ForeignKey(Product, related_name='reviews',
on_delete=models.CASCADE)
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    description = models.TextField()
    rating = models.PositiveIntegerField(choices=[(i, str(i)) for i in range(1, 6)])  # 1 to 5 stars
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f"Review by {self.user.username} for {self.product.name}"
```

forms.py

```
from django import forms
from django.core.validators import MinLengthValidator
from .models import Product, Review
from django.contrib.auth.models import User


class RegistrationForm(forms.ModelForm):
    password = forms.CharField(
        widget=forms.PasswordInput(attrs={'class': 'form-control'}),
        label="Password"
    )
    confirm_password = forms.CharField(
        widget=forms.PasswordInput(attrs={'class': 'form-control'}),
        label="Confirm Password"
    )
    username = forms.CharField(
        widget=forms.TextInput(attrs={'class': 'form-control'}),
        label="Username",
        validators=[MinLengthValidator(8, message="Username must be at least 8 characters
long.")]
    )

    class Meta:
        model = User
        fields = ['username', 'email', 'password']
        labels = {
            'username': 'Username',
            'email': 'Email address',
        }
        help_texts = {
            'username': 'Minimum 8 characters',
            'email': None,  # Removes the default help text
        }
        widgets = {
            'username': forms.TextInput(attrs={'class': 'form-control'}),
            'email': forms.EmailInput(attrs={'class': 'form-control'}),
        }

    def clean(self):
        cleaned_data = super().clean()
        password = cleaned_data.get("password")
        confirm_password = cleaned_data.get("confirm_password")

        if password != confirm_password:
            raise forms.ValidationError("Passwords do not match.")


class PasswordResetForm(forms.Form):
    email = forms.EmailField(label="Email", required=True)
    old_password = forms.CharField(label="Old Password", widget=forms.PasswordInput,
required=True)
    new_password = forms.CharField(label="New Password", widget=forms.PasswordInput,
required=True)
    confirm_new_password = forms.CharField(label="Confirm New Password",
widget=forms.PasswordInput, required=True)
```

```
 def clean(self):
        cleaned_data = super().clean()
        new_password = cleaned_data.get("new_password")
        confirm_new_password = cleaned_data.get("confirm_new_password")

        if new_password != confirm_new_password:
            raise forms.ValidationError("New passwords do not match!")
        return cleaned_data


class ProductForm(forms.ModelForm):
    class Meta:
        model = Product
        fields = ['name', 'description', 'brand', 'category', 'price', 'image']


class ReviewForm(forms.ModelForm):
    description = forms.CharField(
        widget=forms.Textarea(attrs={'class': 'form-control', 'rows': 5, 'placeholder': 'Write your
review here'}),
        required=True
    )

    class Meta:
        model = Review
        fields = ['description', 'rating']
```