

Dự án cá nhân về lập trình web-application với django framework



eCweb

eCommerce website



Nguyễn Thành Trung

I Giới thiệu.....	3
II Dự án.....	3
1. Framework, công cụ sử dụng trong dự án.....	3
2. Thành phần chính và chức năng của chúng.....	4
2.1 accounts.....	4
2.2 category.....	6
2.3 store.....	7
2.4 carts.....	12
2.5 orders.....	19
3. Docker.....	24
4. Demo.....	26
III Tổng kết.....	28
1. Các chức năng đã hoàn thiện.....	28
2. Các chức năng sẽ hoàn thiện trong khoảng thời gian tới.....	28

I Giới thiệu

Đi cùng với sự phát triển vượt bậc của ngành logistic thì các website thương mại điện tử như bán hàng online là một dự án rất phổ biến trong lập trình, với lượng khách hàng trải dài từ cá nhân đến các tập đoàn lớn đang buôn bán nhiều loại mặt hàng. Các ông lớn có thể kể đến là các sàn thương mại điện tử Shopee, Lazada,.. Và gần đây là sự gia nhập của Temu đến từ Trung Quốc. Vì vậy, một website bán hàng online là một chủ đề khá hấp dẫn đối với mình. Với dự án này mình tìm hiểu và xây dựng website bán các loại áo như sơ mi, thun và polo với đầy đủ các chức năng cơ bản.

Hiện tại mình đang hướng đến công việc backend developer (sử dụng django, python) vì vậy trong dự án này django, python cho backend là lựa chọn của mình, cùng với đó là html/css, jquery, bootstrap4 cho frontend, MySQL làm hệ quản trị dữ liệu, ngoài ra mình chạy dự án hoàn toàn trên Docker.

II Dự án

1. Framework, công cụ sử dụng trong dự án

- ❖ Front end: **HTML/CSS/Js, jquery, bootstrap4**
- ❖ Back end: **django, python**
- ❖ Database: **MySQL**
- ❖ Environment: **Docker**

2. Thành phần chính và chức năng của chúng

2.1 accounts

2.1.1 Thành phần

a. Models

- **Account:** Mô hình mở rộng từ `AbstractBaseUser` để tùy chỉnh hệ thống người dùng mặc định của django.
 - Các thuộc tính chính: *email, username, first_name, last_name, phone_number, v.v*
 - Các phương thức bổ sung:
 - *has_perm*: Kiểm tra quyền của người dùng
 - *has_module_perms*: Xác nhận quyền truy cập module.
 - *full_name*: Trả về họ và tên đầy đủ của người dùng
- **MyAccountManager:** Lớp quản lý tùy chỉnh để tạo người dùng thông thường (*create_user*) và người dùng quản trị (*create_superuser*).

b. Forms

- **RegistrationForm:** Biểu mẫu đăng ký người dùng với các trường:
 - *first_name, last_name, email, phone_number, password, confirm_password*.
 - Có các phương thức kiểm tra hợp lệ.
 - Xác minh *password* và *confirm_password* khớp nhau.

c. Views

- Các hàm xử lý logic cho các chức năng người dùng
 - **register:** Xử lý đăng ký tài khoản mới và gửi email kích hoạt.
 - **login:** Xử lý đăng nhập người dùng và đồng bộ giỏ hàng.
 - **logout:** Đăng xuất người dùng.
 - **activate:** Kích hoạt tài khoản qua email.
 - **dashboard:** Trang tổng quan cá nhân sau khi đăng nhập.

- **forgotPassword**: Gửi email đặt lại mật khẩu.
- **reset_password_validate**: Xác thực liên kết đặt lại mật khẩu.
- **reset_password**: Đặt lại mật khẩu mới

d. AccountAdmin

AccountAdmin là một lớp kế thừa từ *UserAdmin*, lớp này giúp tùy chỉnh giao diện quản lý cho mô hình người dùng.

- **list_display**:
 - Xác định các trường được hiển thị trong danh sách người dùng trong giao diện admin.
 - Các trường bao gồm: **email**, **username**, **first_name**, **last_name**, **last_login**, **date_joined** và **is_active**.
- **list_display_links**:
 - Cung cấp các liên kết cho các trường danh sách. Các trường **email**, **username**, **first_name** và **last_name** sẽ có thể click vào để chuyển đến trang chi tiết người dùng.
- **readonly_fields**:
 - Các trường này chỉ có thể đọc, không thể chỉnh sửa trong giao diện admin. Ở đây bạn đã đặt **last_login** và **date_joined** vào để không cho phép thay đổi các giá trị này.
- **ordering**: Quy định cách sắp xếp người dùng trong danh sách. *'date_joined'* sẽ sắp xếp theo ngày tham gia, với những người tham gia gần đây nhất xuất hiện đầu tiên.
- **filter_horizontal**, **list_filter**, **fieldsets**: Các thuộc tính này để trống, có thể bổ sung tùy thuộc vào nhu cầu.

2.1.2 Chức năng

Chức năng	Mô tả
Đăng ký tài khoản	Người dùng mới có thể tạo tài khoản và nhận email kích hoạt.
Kích hoạt tài khoản	Xác nhận email để kích hoạt tài khoản thông qua liên kết có mã hóa.
Đăng nhập	Người dùng có thể đăng nhập bằng email và mật khẩu, đồng thời đồng bộ giỏ hàng nếu có.
Đăng xuất	Đăng xuất người dùng và hủy phiên làm việc hiện tại.
Quên mật khẩu	Yêu cầu email để đặt lại mật khẩu và nhận email liên kết.
Đặt lại mật khẩu	Người dùng có thể đặt lại mật khẩu mới sau khi xác thực liên kết email.
Trang cá nhân	Hiển thị thông tin cơ bản của người dùng sau khi đăng nhập.

2.2 category

2.2.1 Thành phần

a. Models

- **category**: Mô hình đại diện cho một danh mục sản phẩm.
 - **category_name**: Tên danh mục.
 - **slug**: Đoạn URL thân thiện (tự động tạo từ *category_name*).
 - **description**: Mô tả ngắn về danh mục.
 - **category_image**: Hình ảnh đại diện cho danh mục.
 - **get_url()**: Trả về URL của trang sản phẩm theo danh mục.

b. Admin

- **CategoryAdmin**: Tùy chỉnh cách quản lý danh mục trong Django Admin.
 - Tự động điền trường slug từ **category_name**.
 - Hiển thị các cột **category_name** và **slug** trong bảng quản trị.

c. Context Processors

- **menu_links**: Trả về danh sách tất cả các danh mục để có thể sử dụng trong các templates, ví dụ để hiển thị danh sách danh mục menu điều hướng.

d. App Config

- Định nghĩa tên của ứng dụng là **category** và cấu hình các tham số mặc định của ứng dụng.

2.2.2 Chức năng

- Quản lý các danh mục sản phẩm
- Hiển thị danh mục sản phẩm trong admin
- Cung cấp danh sách danh mục để hiển thị trong các template

_ Ứng dụng này giúp tổ chức và phân loại các sản phẩm theo mục lục trong cửa hàng trực tuyến của bạn.

2.3 store

2.3.1 Models

a. Product

- Đại diện cho sản phẩm trong cửa hàng.
- **Thuộc tính**:
 - **produce_image, slug**: Tên sản phẩm và slug (URL ngắn gọn).
 - **description**: Mô tả chi tiết sản phẩm.
 - **price, currency**: Giá và loại tiền tệ (mặc định là USD).
 - **images**: Ảnh sản phẩm.
 - **stock, is_available**: Số lượng tồn kho và trạng thái khả dụng.
 - **category**: Liên kết với bảng *Category* (ngoại khóa).
 - **created_date, modified_date**: Ngày tạo và ngày cập nhật.
- **Phương thức**:
 - **get_url**: Tạo URL chi tiết sản phẩm dựa trên slug của danh mục và sản phẩm.

- **__str__**: Trả về tên sản phẩm khi gọi đến đối tượng.

b. Variation

- Đại diện cho các biến thể (ví dụ: màu sắc, kích cỡ) của sản phẩm.
- **Thuộc tính**:
 - **product**: Liên kết với bảng *Product*.
 - **variation_category, variation_value**: Loại biến thể (màu, kích cỡ) và giá trị cụ thể (Ví dụ: "Red", "L").
 - **is_active**: Trạng thái hoạt động của biến thể.
 - **created_date**: Ngày tạo.
- **Quản lý**:
 - **VariationManager** cung cấp các phương thức lấy danh sách biến thể theo loại (**colors, sizes**).
- **Phương thức**:
 - **__str__**: Trả về giá trị biến thể (ví dụ: "Red")

c. ReviewRating

- Lưu trữ đánh giá và xếp hạng sản phẩm từ người dùng.
- **Thuộc tính**:
 - **product, user**: Liên kết đến bảng sản phẩm và tài khoản người dùng.
 - **subject, review**: Chủ đề và nội dung đánh giá.
 - **rating**: Xếp hạng (dạng số thực).
 - **ip**: Địa chỉ IP của người đánh giá.
 - **status**: Trạng thái (hiển thị hay không).
 - **created_date, updated_at**: Ngày tạo và cập nhật.
- **Phương thức**:
 - **__str__**: Trả về chủ đề của đánh giá.
- **Chức năng tổng quát**:
 - **Product**: Quản lý thông tin sản phẩm.

- **Variation:** Quản lý các biến thể (màu sắc, kích cỡ) của sản phẩm.
- **ReviewRating:** Quản lý đánh giá và xếp hạng sản phẩm từ người dùng.

2.3.2 Views

a. store

- Hiển thị danh sách sản phẩm trong cửa hàng, có hỗ trợ lọc theo danh sách mục (nếu **category_slug** được cung cấp).
- **Chức năng:**
 - Lấy danh sách sản phẩm từ cơ sở dữ liệu.
 - Phân trang sản phẩm (mỗi trang hiển thị 6 sản phẩm).
 - Tính tổng thể số lượng sản phẩm.
- **Trả về:**
 - Giao diện về *store/store.html* với các sản phẩm trong danh mục (nếu có) hoặc tất cả sản phẩm.

b. product_detail

- Hiển thị thông tin chi tiết của một sản phẩm cụ thể.
- **Chức năng:**
 - Tìm sản phẩm dựa trên **category_slug** và **product_slug**.
 - Kiểm tra sản phẩm có trong giỏ hàng hay không.
 - Kiểm tra người dùng đã đặt hàng sản phẩm này chưa (**orderproduct**).
 - Lấy danh sách các đánh giá (reviews) của sản phẩm.
- **Trả về:**
 - Giao diện *store/product_detail.html* với thông tin sản phẩm, trạng thái giỏ hàng, đánh giá và trạng thái đặt hàng.

c. search

- Tìm kiếm sản phẩm dựa trên từ khóa

- **Chức năng:**
 - Lấy từ khóa tìm kiếm từ query string “q”.
 - Tìm sản phẩm trong cơ sở dữ liệu dựa trên *product_name* hoặc *description* có chứa từ khóa (không phân biệt hoa thường).
- **Trả về:**
 - Giao diện *store/store.html* với danh sách sản phẩm phù hợp.

d. **submit_review**

- Xử lý việc gửi hoặc cập nhật đánh giá của người dùng đối với một số sản phẩm.
- **Chức năng:**
 - Kiểm tra xem người dùng đã đánh giá chưa.
 - Nếu đã đánh giá, cập nhật nội dung đánh giá.
 - Nếu chưa đánh giá, tạo mới một đánh giá.
 - Lấy địa chỉ IP của người dùng để lưu trữ trong đánh giá.
 - Hiển thị thông báo thành công.
- **Trả về:**
 - Chuyển hướng người dùng về trang trước đó (sử dụng *HTTP_REFERER*).
- **Chức năng tổng quát:**
 - **Hiển thị sản phẩm:** Cửa hàng tổng quát hoặc chi tiết từng sản phẩm.
 - **Tìm kiếm:** Hỗ trợ tìm sản phẩm theo từ khóa.
 - **Đánh giá:** Cho phép người dùng gửi và cập nhật đánh giá cho sản phẩm.

2.3.3 Forms

a. **Mục đích**

- Tạo form cho đánh giá sản phẩm.

- Form này sẽ giúp người dùng dễ dàng gửi đánh giá thông qua giao diện web.

b. Thành phần

ReviewForm

- **Kế thừa:** forms.ModelForm.
- **Liên kết với model:** *ReviewRating* (từ *store/models.py*).
- **Các trường sử dụng:**
 - **subject:** Tiêu đề đánh giá.
 - **review:** Nội dung đánh giá
 - **rating:** Điểm đánh giá (có thể là số thực hoặc số nguyên, tùy thuộc vào cấu hình trong *models.py*)
- **Chức năng:**
 - **Tự động hóa:**
 - Hỗ trợ tạo form dựa trên các trường trong model ***ReviewRating***.
 - Dễ dàng kiểm tra tính hợp lệ của dữ liệu trước khi lưu vào cơ sở dữ liệu.
 - **Giao diện:** Dữ liệu form này sẽ được hiển thị trên giao diện, giúp người dùng gửi đánh giá sản phẩm.
- **Cách sử dụng:**
 - Trong **view**:
 - Tạo một **instance** của ***ReviewForm*** để hiển thị form trống hoặc điền sẵn dữ liệu nếu cần cập nhật.
 - Kiểm tra tính hợp lệ (***form.is_valid()***).
 - Lưu dữ liệu nếu hợp lệ (***form.save()***) hoặc xử lý dữ liệu ***cleaned_data***).
 - Trong **template**:
 - Form này có thể được render trực tiếp bằng ***{%csrf_token%}*** và các trường của form.

2.4 carts

2.4.1 Models

a. Mục đích

- Quản lý giỏ hàng và các sản phẩm trong giỏ hàng.
- Tạo cấu trúc cơ sở dữ liệu để:
 - Lưu trữ thông tin giỏ hàng.
 - Lưu trữ các mục (items) trong giỏ hàng, bao gồm số lượng và các biến thể của sản phẩm.

b. Thành phần chính

- Model: **Cart**

- **Thuộc tính:**

- **cart_id**: ID của giỏ hàng (kiểu *charField*), thường dùng để nhận diện giỏ hàng ẩn danh.
- **date_added**: Ngày tạo giỏ hàng (tự động thêm ngày giờ hiện tại khi tạo giỏ hàng).

- **Phương thức:**

- **__str__**: Trả về chuỗi biểu diễn của **cart_id**.

- Model: **CartItem**

- **Thuộc tính:**

- **user**: Người dùng liên kết với giỏ hàng (dùng *Account*)
 - **null=True**: Hỗ trợ giỏ hàng cho cả người dùng đã đăng nhập và chưa đăng nhập.
- **product**: Sản phẩm liên quan đến mục giỏ hàng (dùng *Product* từ *store/models.py*).
- **variation**: Biến thể của sản phẩm (nếu có), như màu sắc, kích cỡ (dùng *Variation*).
 - **ManyToManyField**: Một mục giỏ hàng có thể có nhiều biến thể.
- **cart**: Liên kết tới giỏ hàng (dùng *Cart*).

- **null=True:** Cho phép mục giỏ hàng tồn tại mà không liên kết tới một giỏ cụ thể.
- **quantity:** Số lượng sản phẩm trong giỏ
- **is_active:** Xác định trạng thái của mục giỏ hàng (đang hoạt động hay không).
- **Phương thức:**
 - **sub_total:** Tính tổng giá trị của mục giỏ hàng dựa trên số lượng và giá sản phẩm.

c. Chức năng

- Model **Cart**:
 - Quản lý thông tin cơ bản của một giỏ hàng.
 - Có thể sử dụng *cart_id* để nhận diện các giỏ hàng của người dùng không đăng nhập.
- Model **CartItem**:
 - Quản lý từng mục sản phẩm trong giỏ hàng.
 - Cho phép lưu trữ thông tin về sản phẩm, số lượng, biến thể và trạng thái hoạt động.

d. Cách sử dụng

- Khi người dùng thêm sản phẩm vào giỏ hàng:
 - Tìm hoặc tạo *Cart* dựa trên *cart_id*.
 - Thêm hoặc cập nhật *CartItem* liên quan đến *Cart* và sản phẩm.
- Khi người dùng đăng nhập:
 - Chuyển giỏ hàng ẩn danh (dựa trên *cart_id*) thành giỏ hàng cá nhân liên kết với tài khoản người dùng.
- Hiển thị giỏ hàng:
 - Lấy tất cả các mục *CartItem* liên quan đến giỏ hàng hiện tại và tính tổng.

2.4.2 Views

a. Tổng quan

- Xử lý các chức năng liên quan đến:
 - Thêm sản phẩm vào giỏ hàng (*add_cart*).
 - Xóa hoặc giảm số lượng sản phẩm trong giỏ hàng (*remove_cart*, *remove_cart_item*).
 - Hiển thị giỏ hàng (*cart*).
 - Thanh toán (*checkout*).

b. Các chức năng chi tiết

- Hàm **_cart_id(request)**
 - Xác định và tạo *cart_id* trong session của người dùng.
 - Quy trình:
 - 1 Kiểm tra nếu session hiện tại có *session_key*.
 - 2 Nếu không có, tạo mới session.
 - 3 Trả về *cart_id*.
 - Ứng dụng: Hỗ trợ giỏ hàng cho người dùng chưa đăng nhập.
- Hàm **add_cart(request, product_id)**
 - Quy trình:
 - 1 Kiểm tra đăng nhập
 - Nếu người dùng đã đăng nhập, liên kết giỏ hàng với tài khoản (*user*).
 - Nếu chưa đăng nhập, liên kết giỏ hàng với *cart_id* trong session.
 - 2 Xử lý biến thể sản phẩm
 - Duyệt qua các tham số POST để xác định biến thể của sản phẩm (nếu có).
 - 3 Kiểm tra sản phẩm đã có trong giỏ hàng
 - Nếu có, tăng số lượng hoặc tạo mục giỏ hàng mới nếu biến thể khác.
 - Nếu không, tạo mục giỏ hàng mới.

4 **Lưu giỏ hàng:** Lưu *CartItem* vào cơ sở dữ liệu.

- Hàm **remove_cart**(*request*, *product_id*, *_cart_item_id*)
 - Quy trình:
 - 1 Tìm sản phẩm dựa vào *product_id* và *cart_item_id*.
 - 2 Nếu số lượng > 1, giảm số lượng và lưu lại.
 - 3 Nếu số lượng = 1, xóa mục giỏ hàng.
- Hàm **remove_cart_item**(*request*, *product_id*, *_cart_item_id*)
 - Quy trình:
 - 1 Tìm sản phẩm dựa vào *product_id* và *cart_item_id*.
 - 2 Xóa mục giỏ hàng.
- Hàm **cart**(*request*)
 - Hiển thị giỏ hàng.
 - Quy trình:
 - 1 Kiểm tra trạng thái đăng nhập
 - Nếu đã đăng nhập, lấy tất cả *CartItem* của người dùng.
 - Nếu chưa đăng nhập, lấy *CartItem* dựa vào *cart_id*.
 - 2 Tính toán
 - Tổng giá trị (*total*) = Giá sản phẩm x Số lượng.
 - Thuế (*tax*) = 2% tổng giá trị.
 - Tổng thanh toán (*grand_total*) = Tổng giá trị + Thuế.
 - 3 Kết xuất thông tin vào template *cart.html*.
- Hàm **checkout**(*request*)
 - Hiển thị trang thanh toán.
 - Quy trình:
 - 1 Lấy các mục giỏ hàng của người dùng đăng nhập.

2 Tính toán tổng giá trị, thuế, và tổng thanh toán (giống như *cart*).

3 Kết xuất thông tin vào template *checkout.html*.

c Cách cải tiến

- **Tối ưu hóa xử lý các biến thể.**
 - Thay vì duyệt qua ***request.POST***, có thể kiểm tra trực tiếp các biến thể cần thiết.
- **Giảm trùng lặp mã.**
 - Kết hợp sử dụng logic của người dùng đăng nhập và chưa đăng nhập vào một hàm chung.
 - Sử dụng helper functions để quản lý logic thêm/xóa mục giỏ hàng.

d Ứng dụng

- **Trang giỏ hàng.**
 - Sử dụng *cart* để hiển thị thông tin.
- **Trang thanh toán.**
 - Sử dụng *checkout* để xử lý thanh toán sau khi người dùng xác nhận đơn hàng.
- **Thao tác thêm, xóa.**
 - Gắn các hàm *add_cart()*, *remove_cart()*, *remove_cart_item()* vào nút hoặc liên kết trong template.

2.4.3 context_processors.py

a. **Mục đích**

- Là một phần của Django cho phép bạn thêm các biến ngữ cảnh (context variables) vào tất cả các templates trong ứng dụng.
- Cung cấp hàm ***counter(request)*** để tính tổng số lượng sản phẩm trong giỏ hàng, được hiển thị trên mọi trang.

b. Hàm **counter(request)**

- **Chức năng:** Tính toán tổng số lượng sản phẩm có trong giỏ hàng (được gọi là **cart_count**).
- **Quy trình hoạt động**

1 Kiểm tra nếu đang truy cập trang quản trị viên

- Nếu đường dẫn chứa từ “**admin**”, trả về một dictionary rỗng **{}** để tránh lỗi không cần thiết khi xử lý giỏ hàng.

2 Trạng thái đăng nhập

- Nếu người dùng đã đăng nhập, lấy các mục giỏ hàng (**CartItem**) liên kết với tài khoản người dùng (**user=request.user**).
- Nếu người dùng chưa đăng nhập.
 - Lấy **cart_id** từ session thông qua **_cart_id(request)**.
 - Sử dụng **cart_id** để lấy giỏ hàng tương ứng (**Cart**).
 - Lấy các mục giỏ hàng (**CartItem**) liên kết với giỏ hàng.

3 Tính toán số lượng sản phẩm

Tổng số lượng (**cart_count**) = Tổng số **quantity** của tất cả các mục giỏ hàng.

4 Xử lý lỗi: Nếu giỏ hàng không tồn tại (**Cart.DoesNotExist**), đặt **cart_count=0**.

5 Trả về kết quả: Trả về một dictionary **{'cart_count' : cart_count}** để sử dụng trong template.

2.4.4 urls

a. **Chức năng**

- **Định nghĩa các đường dẫn URL (routes)** cho các view trong app *carts*. Điều này cho phép ứng dụng định tuyến người dùng đến các chức năng như xem giỏ hàng, xem sản phẩm, xóa sản phẩm và thanh toán.

b. Chi tiết từng đường dẫn

URL Path	View liên kết	Tên (name)	Chức năng
/	views.cart	cart	Hiển thị trang giỏ hàng với thông tin các sản phẩm đã thêm.
add_cart/<int:product_id>/	views.add_cart	add_cart	Thêm sản phẩm vào giỏ hàng dựa trên <i>product_id</i> .
remove_cart/<int:product_id>/<int:cart_item_id>/	views.remove_cart	remove_cart	Giảm số lượng sản phẩm trong giỏ hàng hoặc xóa nếu số lượng là 1.
remove_cart_item/<int:product_id>/<int:cart_item_id>/	views.remove_cart_item	remove_cart_item	Xóa toàn bộ sản phẩm (bao gồm các biến thể) khỏi giỏ hàng.
checkout/	views.checkout	checkout	Hiển thị trang thanh toán với thông tin sản phẩm và tổng giá trị.

2.4.5 admin

a. Chức năng

- Dùng để cấu hình hiển thị và quản lý model **Cart** và **CartItem** trong Django Admin. Giúp quản trị viên dễ dàng theo dõi và chỉnh sửa lại thông tin giỏ hàng trực tiếp từ giao diện quản trị.

b. Chi tiết các thành phần

- *CartAdmin*

- **list_display**: Hiển thị các trường *cart_id* và *date_added* trong bảng quản trị.
- *CartItemAdmin*
 - **list_display**: Hiển thị các trường *product*, *cart*, *quantity*, *is_active*.

c. **Kết quả trong Django Admin**

- *Cart*
 - Có thể tìm kiếm giỏ hàng bằng *cart_id*.
 - Lọc các giỏ hàng theo ngày thêm (*date_added*).
- *CartItem*
 - Hiển thị danh sách các sản phẩm trong giỏ hàng kèm số lượng.
 - Có thể lọc các sản phẩm đang hoạt động (*is_active*).
 - Tìm kiếm nhanh sản phẩm theo tên.

2.5 orders

2.5.1 Models

a. **Mô hình *Payment***

- **Chức năng**: Lưu trữ thông tin về các giao dịch thanh toán của khách hàng.
- **Trường (Fields)**:
 - **user**: Liên kết đến tài khoản người dùng đã thực hiện thanh toán.
 - **payment_id**: ID giao dịch thanh toán.
 - **payment_method**: Phương thức thanh toán.
 - **amount_paid**: Số tiền đã thanh toán.
 - **status**: Trạng thái thanh toán.
 - **created_at**: Thời gian thanh toán được thực hiện.
- **Phương thức**:
 - **__str__**: Trả về **payment_id** khi gọi đối tượng *Payment*.

b. **Mô hình *Order***

- **Chức năng**: Lưu trữ thông tin về đơn hàng của người dùng.

- **Trường (Fields):**

- **user:** Liên kết đến tài khoản người dùng đã đặt hàng.
- **payment:** Liên kết đến mô hình *Payment* để ghi lại giao dịch thanh toán của đơn hàng.
- **order_number:** Số hiệu đơn hàng.
- **first_name, last_name:** Họ tên người đặt hàng.
- **phone, email:** Thông tin liên lạc.
- **address_line_1, address_line_2, country, state, city:** Địa chỉ giao hàng.
- **order_note:** Ghi chú đơn hàng.
- **order_total:** Tổng tiền đơn hàng.
- **tax:** Thuế cho đơn hàng.
- **status:** Trạng thái đơn hàng (mới, đã chấp nhận, đã hoàn thành, đã hủy).
- **ip:** Địa chỉ IP của khách hàng khi đặt hàng.
- **is_ordered:** Cờ cho biết đơn hàng đã được hoàn tất thanh toán và gửi đi hay chưa.
- **created_at, updated_at:** Thời gian tạo và cập nhật đơn hàng.

- **Phương thức:**

- **full_name:** Trả về họ và tên đầy đủ của người đặt hàng.
- **full_address:** Trả về địa chỉ đầy đủ của người đặt hàng.
- **__str__:** Trả về tên người đặt hàng (*first_name*).

c. **Mô hình *OrderProduct***

- **Chức năng:** Lưu trữ các sản phẩm trong đơn hàng.

- **Trường (Fields):**

- **order:** Liên kết đến mô hình *Order* để chỉ rõ hơn đơn hàng mà sản phẩm này thuộc về.
- **payment:** Liên kết đến mô hình *Payment* để chỉ rõ giao dịch thanh toán của sản phẩm.

- **user:** Liên kết đến tài khoản người dùng đã đặt sản phẩm.
- **product:** Liên kết đến sản phẩm đã được đặt trong đơn hàng.
- **variations:** Liên kết đến các biến thể của sản phẩm.
- **quantity:** Số lượng sản phẩm trong đơn hàng.
- **product_price:** Giá sản phẩm tại thời điểm đặt hàng.
- **ordered:** Cờ cho biết sản phẩm đã được giao hay chưa.
- **created_at, updated_at:** Thời gian tạo và cập nhật thông tin sản phẩm trong đơn hàng.
- **Phương thức**
 - **__str__:** Trả về tên sản phẩm đã đặt (*product.product_name*).
- Tóm lại, các mô hình này tạo thành một hệ thống quản lý thanh toán và đơn hàng trong một cửa hàng trực tuyến. Mô hình *Order* lưu trữ thông tin về khách hàng và đơn hàng, trong khi mô hình *OrderProduct* lưu trữ chi tiết về các sản phẩm trong mỗi đơn hàng. Mô hình *Payment* liên kết với các đơn hàng để lưu trữ thông tin giao dịch thanh toán.

2.5.2 Views

Cung cấp các hàm xử lý liên quan đến việc đặt hàng, thanh toán và hoàn thành đơn hàng.

a. **sendEmail(request, order)**

- **Chức năng:** Gửi email cảm ơn khách hàng khi họ đặt hàng thành công.
- **Thành phần chính:**
 - Sử dụng *render_to_string* để tạo nội dung email từ template *orders/order_received_email.html*.
 - Dùng *EmailMessage* để gửi email đến địa chỉ email người dùng.

b. **payments(request)**

- **Chức năng:** Xử lý các yêu cầu AJAX để lưu thông tin thanh toán, cập nhật trạng thái đơn hàng và chuyển các mục trong giỏ hàng thành các mục trong đơn hàng.

- **Quy trình:**

1. **Lấy dữ liệu từ yêu cầu AJAX** (*request.POST*)
2. **Lưu thông tin thanh toán vào bảng** *Payment*.
3. **Cập nhật đơn hàng tương ứng với thanh toán và đánh dấu là đã đặt hàng** (*is_ordered=True*).
4. **Chuyển đổi từng mục trong giỏ hàng** (*CartItem*) thành sản phẩm trong đơn hàng (*OrderProduct*).
5. **Giảm số lượng sản phẩm tồn kho sau khi đặt hàng.**
6. **Xóa tất cả các mục trong giỏ hàng.**
7. **Gửi email cảm ơn và trả về dữ liệu JSON để phản hồi.**

c. **place_order**

- **Chức năng:** Tạo và lưu thông tin đặt hàng, tính toán tổng tiền và thuế, sau đó chuyển đến trang thanh toán.

- **Quy trình:**

1 **Kiểm tra xem giỏ hàng của người dùng có trống không.**

2 **Tính tổng tiền, thuế và tổng cộng** (*grand_total*).

3 **Khi người dùng gửi biểu mẫu đặt hàng:**

- Lưu thông tin khách hàng và đơn hàng vào bảng ***Order***.
- Tạo mã đơn hàng dựa trên ngày hiện tại và ***ID*** của đơn hàng.
- Chuyển hướng đến trang thanh toán (*payments.html*) cùng với dữ liệu đơn hàng và giỏ hàng.

d. **order_complete**

- **Chức năng:** Hiển thị trang hoàn tất đơn hàng với các thông tin chi tiết về đơn hàng và thanh toán.

- **Quy trình:**

1 **Lấy mã đơn hàng** (*order_number*) và mã giao dịch thanh toán (*transID*) từ URL.

2 **Truy vấn thông tin đơn hàng và sản phẩm đã đặt.**

3 Tính tổng phụ (subtotal) dựa trên giá và số lượng sản phẩm.

4 Truy xuất thông tin thanh toán và hiển thị trên giao diện
order_complete.html.

2.5.3 urls

Định nghĩa các đường dẫn URL cho các chức năng liên quan đến việc đặt hàng và thanh toán.

a. place_order/

- **Ảnh xạ:** *views.place_order*
- **Tên URL:** *place_order*
- **Chức năng:**
 - Xử lý yêu cầu từ trang thanh toán khi người dùng đặt hàng.
 - Chuyển thông tin đơn hàng đến trang thanh toán.

b. payments/

- **Ảnh xạ:** *views.payments*
- **Tên URL:** *payments*
- **Chức năng:** Xử lý yêu cầu AJAX để lưu thông tin thanh toán và hoàn tất đơn hàng.

c. order_complete/

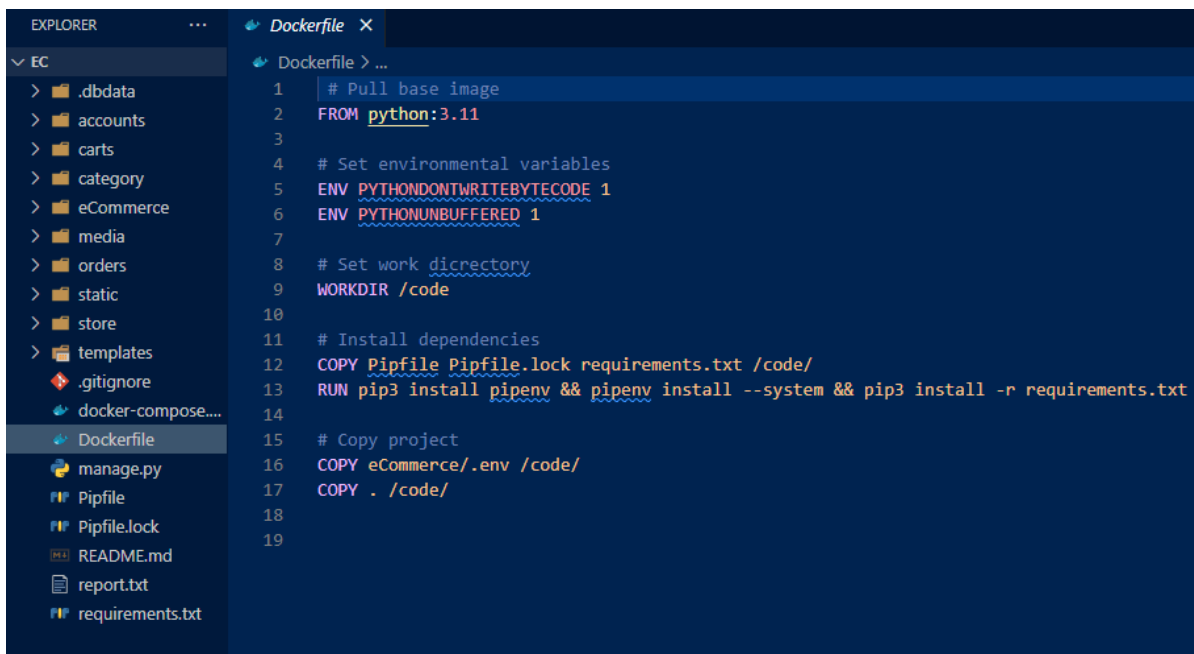
- **Ảnh xạ:** *views.order_complete*
- **Tên URL:** *order_complete*
- **Chức năng:** *Hiển thị trang hoàn tất đơn hàng với thông tin chi tiết về đơn hàng và thanh toán.*

2.5.4 forms.py

- **Chức năng:**
 - Cung cấp một biểu mẫu để người dùng nhập thông tin chi tiết đơn hàng.
 - Tự động ánh xạ các trường từ model *Order* vào biểu mẫu, tiết kiệm thời gian và đảm bảo tính đồng nhất.
- **Các trường (fields):**
 - **first_name, last_name:** Họ và tên người đặt hàng.
 - **phone:** số điện thoại liên lạc.
 - **email:** Email của người đặt hàng.
 - **address_line_1, address_line_2:** Địa chỉ giao hàng.
 - **country, state, city:** Quốc gia, tỉnh, thành phố.
 - **order_note:** Ghi chú đặc biệt đơn hàng.

3. Docker

Dự án sẽ được chạy trên Docker, vì vậy tôi chuẩn bị 2 file để config môi trường là **Dockerfile** và **docker-compose.yml**.

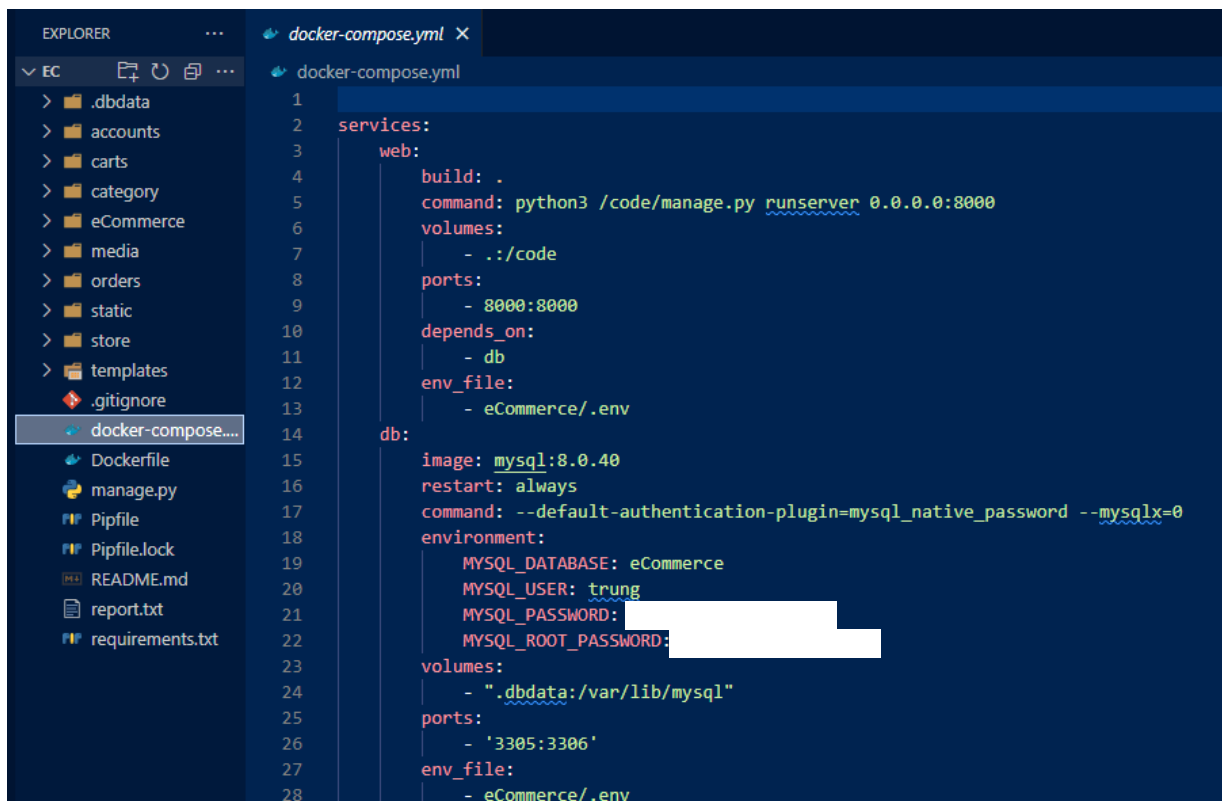


```
1  # Pull base image
2  FROM python:3.11
3
4  # Set environmental variables
5  ENV PYTHONDONTWRITEBYTECODE 1
6  ENV PYTHONUNBUFFERED 1
7
8  # Set work directory
9  WORKDIR /code
10
11 # Install dependencies
12 COPY Pipfile Pipfile.lock requirements.txt /code/
13 RUN pip3 install pipenv && pipenv install --system && pip3 install -r requirements.txt
14
15 # Copy project
16 COPY eCommerce/.env /code/
17 COPY . /code/
18
19
```

Hình 1. Dockerfile

Sử dụng Dockerfile nhằm xác định các bước cần thiết để dựng và cấu hình môi trường container cho ứng dụng.

- Pull base image: sử dụng python 3.11 làm image cơ sở.
- Set environmental variables:
 - `PYTHONDONTWRITEBYTECODE`: Ngăn Python ghi các file bytecode (.pyc), giảm lượng dữ liệu thừa trong container.
 - `PYTHONUNBUFFERED`: Đảm bảo log output từ Python được ghi trực tiếp (unbuffered) để dễ dàng theo dõi log trong container.
- Set work directory: mọi lệnh tiếp theo sẽ được thực thi trong thư mục (/code).
- Install dependencies.
- Copy project.



```
1
2 services:
3   web:
4     build: .
5     command: python3 /code/manage.py runserver 0.0.0.0:8000
6     volumes:
7       - ../code
8     ports:
9       - 8000:8000
10    depends_on:
11      - db
12    env_file:
13      - eCommerce/.env
14  db:
15    image: mysql:8.0.40
16    restart: always
17    command: --default-authentication-plugin=mysql_native_password --mysqlx=0
18    environment:
19      MYSQL_DATABASE: eCommerce
20      MYSQL_USER: trung
21      MYSQL_PASSWORD:
22      MYSQL_ROOT_PASSWORD:
23    volumes:
24      - ".dbdata:/var/lib/mysql"
25    ports:
26      - '3305:3306'
27    env_file:
28      - eCommerce/.env
```

Hình 2. Docker-compose.yml

Docker-compose.yml định nghĩa cấu trúc và cấu hình cho các container Docker trong dự án. Mỗi service trong này đại diện cho một thành phần của ứng dụng.

- Service **web**:
 - Xây dựng container từ **Dockerfile**.
 - Chạy server Django trên cổng 8000.
 - Mount mã nguồn vào container để hỗ trợ phát triển.
- Service **db**:
 - Tạo container MySQL với cấu hình cụ thể.
 - Lưu trữ dữ liệu trên host bằng volume.
 - Sử dụng file **.env** để đảm bảo bảo mật.
- Kết nối giữa **web** và **db**:
 - **depends_on** đảm bảo database khởi động trước web.
 - Biến môi trường từ file **.env** giúp định cấu hình kết nối database.

Tóm gọn thì sử dụng Docker nhằm đồng bộ môi trường chạy dự án, tránh xung đột khi cài đặt môi trường ở từng máy khác nhau khi chạy dự án, và với cá nhân mình thì đánh là bài thực hành chi tiết về Docker sau chuỗi ngày đọc lý thuyết về nó.

4. Demo

1. Chạy dự án trên Docker Desktop - Demo

2. Khách vãng lai

a. Truy cập và tìm kiếm sản phẩm - Demo

b. Thêm sản phẩm cần mua vào giỏ hàng - Demo

c. Đăng nhập/ Đăng ký mới - Demo

Dù có thể thêm sản phẩm giỏ hàng tuy nhiên để đến được trang đặt hàng thì mọi người dùng đều phải đăng nhập.

3. User

- Truy cập trang web, đăng nhập/ đăng ký, thêm sản phẩm vào giỏ hàng giống với mục 2.

- Mua hàng, thanh toán.
- Đánh giá, bình luận sản phẩm sau khi mua hàng thành công.
- Lấy lại mật khẩu đã quên.
- **Demo**

4. Admin

- Truy cập store và có các thao tác tương tự như một tài khoản người dùng bình thường - **Demo**.
- Ngoài ra, tài khoản admin còn có thể truy cập trang quản trị để quản lý hệ thống của trang web, với các thao tác - **Demo**:

a. Quản lý tài khoản người dùng (*Account*).

- Thêm, cập nhật thông tin, cấp quyền và xóa tài khoản người dùng.

b. Quản lý các thông tin liên quan đến trang Store (*Product, category, carts*).

- Thêm, cập nhật thông tin và xóa **category** hay **carts**.
- Thêm, cập nhật và xóa các thông tin liên đến **Product**:
product, variations (size, color), đánh giá & bình luận.

c. Quản lý thông tin đơn hàng (*order, payment*).

III Tổng kết

1. Các chức năng đã hoàn thiện

- Đăng nhập/Đăng ký/Lấy lại mật khẩu đã quên thông qua email.
- Hoàn thiện được một trang bán hàng trực tuyến với trang hiển thị sản phẩm và thông tin sản phẩm, cho phép người dùng tìm kiếm, thêm sản phẩm vào giỏ hàng, thanh toán giỏ hàng qua Paypal, đánh giá và bình luận về sản phẩm sau khi mua hàng.
- Trang quản trị riêng cho tài khoản admin để quản lý thông tin về:
 - Tài khoản người dùng trong hệ thống (Account): thêm, xóa, chỉnh sửa thông tin và cấp quyền cho từng tài khoản trong hệ thống.
 - Các thông tin liên quan đến sản phẩm như thêm, xóa và chỉnh sửa products, category, carts, variations trong product.
 - Các thông tin liên đến order: thêm, xóa và chỉnh sửa các đơn đặt hàng, xóa hoặc thêm mới một hóa đơn đã thanh toán.

2. Các chức năng sẽ hoàn thiện trong khoảng thời gian tới

- Thêm các tính năng cho dashboard để mỗi tài khoản có thể quản lý tài khoản riêng của mình về mật khẩu, tên người dùng, lịch sử đặt hàng, phương thức thanh toán, tài khoản thanh toán.