**HƯỚNG DẪN CÀI ĐẶT**

1. Clone source từ github về local
2. Import database fashionshopdb
3. Open source code với PyCharm
4. Cài đặt môi trường cho project: File → Settings → Project: FashionShopDjango → chọn biểu tượng → add → chọn New environment → OK
5. Sau khi tạo mới environment thì ta mở tab terminal và chạy lệnh sau để cài đặt các thư viện cần thiết: pip install -r requirements.txt
6. Tiếp tục tại tab terminal chạy lệnh: python manage.py migrate
7. Sau khi cài đặt mọi thứ thành công thì chạy lệnh “python manage.py runserver” để chạy chương trình.

**GIỚI THIỆU CÔNG NGHỆ**

1. **Giới thiệu Django**

- Django là một framework mạnh mẽ được viết hoàn toàn bằng ngôn ngữ Python .

- Django hoạt động theo mô hình MVT (Model View Template) giúp phát triển dễ dàng , thuận lợi các ứng dụng Web phức tạp.

- Ưu điểm:

* Django giúp các lập trình viên hoàn thành thiết kế một cách nhanh chóng và hiệu quả.
* Django là một framework có tính bảo mật rất mạnh với hệ thống phân quyền và chứng thực.
* Django có cộng đồng người dùng đông đảo vì vậy nhận được sự trợ giúp của cộng đồng một cách dễ dàng.
* Django có bộ tài liệu luôn được cập nhật liên tục.
* Django dễ sử dụng, dễ học.

- Nhược điểm:

* Không phù hợp đối với những dự án nhỏ.
* Không có quy ước, bộ nguyên tắc để các lập trình viên tuân theo.
* Khi ứng dụng phát triển ở quy mô nhỏ có thể gặp một số vấn đề.
* Tương đối gặp khó khăn trong quá trình định tuyến ở django.
* Không có cảnh báo khi lỗi xảy ra trong mẫu.

- Mặc định khi tạo project Django sẽ sử dụng cơ sở dữ liệu sqlite3 Params:

* ENGINE: công nghệ được sử dụng để kết nối cơ sở dữ liệu
* NAME: tên database được tạo sử dụng cho ứng dụng
* USER: tài khoản đăng nhập cơ sở dữ liệu
* PASSWORD: mật khẩu đăng nhập cơ sở dữ liệu
* HOST: địa chỉ kết nối tới cơ sở dữ liệu
* PORT: port kết nối
* OPTION: tùy chọn kết nối

- Truy vấn dữ liệu:

* create(): Tạo đối tượng và lưu vào cơ sở dữ liệu.
* update(): Cập nhật đối tượng vào cơ sở dữ liệu.
* delete(): Xoá đối tượng khỏi cơ sở dữ liệu.
* save(): Lưu đối tượng vào cơ sở dữ liệu.
* get\_or\_create(defaults, \*\*kwargs): Tạo đối tượng và lưu vào cơ sở dữ liệu. Nếu đối tượng đó bị trùng lắp sẽ thì việc tạo đối tượng đó sẽ không được thực thi. Dữ liệu trả về là một dạng tuple (instance,created), Trong đó phần tử created là kiểu bool, Nếu đối tượng được tạo sẽ trả về True, và ngược lại trả về False
* updated\_or\_create(defaults, \*\*kwargs): Cập nhật đối tượng thoả điều kiện kwargs. Hoặc tạo mới đối tượng nếu như đối tượng đó không tồn tại. Dữ liệu trả về giống hàm get\_or\_create
* count(): Số đối tượng trong QuerySet.
* latest(): Trả về đối tượng cuối trong QuerySet.
* earliest(): Trả về đối tượng đầu trong QuerySet.
* first(): Trả về đối tượng đầu tiên trong QuerySet.
* last(): Trả về đối tượng cuối cùng trong QuerySet.
* exists(): Kiểm tra QuerySet có tồn tại.
* aggregate(): Thống kê cho QuerySet (sum, max)

1. **Giới thiệu REST**

- REST (Representational State Transfer) là một kiểu kiến trúc phần mềm định nghĩa một tập hợp các ràng buộc được sử dụng để tạo các dịch vụ Web.

- REST hoạt động chủ yếu dựa vào giao thức HTTP. Mỗi hoạt động cơ bản trên sẽ sử dụng những phương thức HTTP riêng (HTTP method):

• GET (READ) : Trả về một resource hoặc một danh sách resources.

• POST (CREATE) : Tạo mới một resource.

• PUT (UPDATE) : Cập nhật, thay thế thông tin cho resource.

• DELETE (DELETE) : Xoá một resource.

- REST tạo ra một kiến trúc đơn giản, có thể mở rộng, hiệu quả, an toàn và đáng tin cậy.

- Các ràng buộc trong REST:

* Client-server: máy khách và máy chủ phải tách biệt với nhau và được phép phát triển riêng lẻ.
* Stateless: REST APIs là không trạng thái, các lệnh gọi có thể được thực hiện độc lập với nhau và mỗi lệnh gọi chứa tất cả dữ liệu cần thiết để tự hoàn thành thành công.
* Cache: một API không trạng thái có thể làm tăng chi phí yêu cầu bằng cách xử lý rất nhiều lênh gọi đến và đi, một REST API nên được thiết kế để khuyến khích lưu trữ dữ liệu có thể lưu vào bộ nhớ cache.
* Uniform Interface: Chìa khóa để tách máy khách khỏi máy chủ là cần có một giao diện thống nhất cho phép ứng dụng phát triển độc lập mà không cần các dịch vụ hoặc mô hình và hành động của ứng dụng được kết hợp chặt chẽ với lớp API.
* Layered System: REST APIs có các lớp khác nhau trong cùng kiến trúc, mỗi kiến trúc có thể xây dựng thành hệ thống phân cấp giúp tạo ra một ứng dụng mô-đun và có thể mở rộng hơn.
* Code on Demand: cho phép mã code hoặc ứng dụng được truyền qua API để sử dụng trong ứng dụng.

1. **Giới thiệu RESTFUL WEB SERVICE**

**-** RESTful Web service cho phép các hệ thống yêu cầu truy cập và thao tác các biểu diễn dạng văn bản của tài nguyên Web bằng cách sử dụng một tập hợp các giao thức không trạng thái (stateless protocol) thống nhất và được xác định trước.

- Để hiện thực Restful API cần thiết kế các thông tin sau: Resource; Request (Verb, Header, Body); Response (Body, Status code).

- API cho phép giao tiếp giữa các hệ thống phần mềm khác biệt nhau về địa lý.

1. **Giới thiệu RESTFUL API**

- 4 hoạt động thiết yếu khi làm việc với server gồm:

• Lấy dữ liệu ở một định dạng nào đó (thường là JSON)

• Tạo mới dữ liệu

• Cập nhật dữ liệu

• Xóa dữ liệu

- Request Headers: Chứa những quy định gửi kèm với request như loại dữ liệu response hoặc các thông tin chứng thực.

- Request Body: Chứa dữ liệu request gửi lên, nó được gửi cho các request sử dụng phương thức POST, PUT, PATCH.

- Response Body: Dữ liệu phản hồi về cho phía client.

- Response status code:

* 200 OK: Request thực hiện hành công.
* 201 Created: Tài nguyên tạo thành công.
* 204 No Content: Tài nguyên xoá thành công.
* 400 Bad Request: Request không hợp lệ.
* 401 Unauthorized: Request chứa được chứng thực.
* 403 Forbidden: Request không được quyền truy cập.
* 404 Not Found: Request không được tìm thấy.
* 500 Internal Server Error: Lỗi server.

1. **Django Rest API Framework**

**-** Django REST framework là bộ công cụ linh hoạt để phát triển API.

- Hỗ trợ giao diện duyệt API hiệu quả cho developer

- Hỗ trợ nhiều authentication policies, chứng thực bao gồm OAuth1a và OAuth2.

- Serialization hỗ trợ cả các data source theo ORM và không ORM (non-ORM).

- Tài liệu phong phú và cộng đồng lớn.

- ***Request:***

• Request của REST framework kế thừa từ HttpRequest chuẩn, nhưng linh hoạt hơn trong xử lý và chứng thực request (request parsing và request authentication).

• Request giúp cho việc xử lý dữ liệu request dạng json hay một media type nào khác theo cùng cách thức làm việc với dữ liệu form.

• Các thuộc tính của request:

+ request.data: dữ liệu của request body.

+ request.query\_params: các tham số truyền từ request get. + request.method: trả về phương thức (chữ in hoa) thực hiện request như GET, POST, PUT, DELETE, PATCH.

+ request.content\_type: trả về chuỗi đại diện cho media type của request body.

+ request.user: các user đã chứng thực thì nó trả về thể hiện của django.contrib.auth.models.User

+ request.auth: chứa thông tin bổ sung chứng thực, nó phụ thuộc cách thức chứng thực, nhưng nó thường chứa thông tin token của request.

+ request.authenticators: các chính sách chứng thực, trong các lớp APIView hoặc @api\_view thuộc tính được thiết lập thông qua thuộc tính authentication\_classes hoặc giá trị thuộc tính cấu hình DEFAULT\_AUTHENTICATORS.

- ***Phân trang:*** cho phép chỉ định số phần tử hiển thị trên từng trang.

- ***Response:***

+ REST framework cung cấp lớp Response trả về nội dung với nhiều dạng media type phụ thuộc client request. Lớp Response là lớp con của Django SimpleTemplateResponse

+ Các thuộc tính của response:

• data: dữ liệu đã được serialize.

• status: status code cho response, mặc định 200.

• headers: từ điển HTTP headers cho response.

• content\_type: content type cho response.

• template\_name: tên template sử dụng nếu HTMLRenderer được chọn.

- ***Serializer:***

• Serializer cho phép chuyển các dữ liệu phức tạp như querysets hoặc thể hiện của model thành các kiểu dữ liệu python và dễ dàng kết xuất thành dữ liệu JSON, XML hay một content type nào khác.

• Nó cũng cung cấp khả năng deserialization để chuyển dữ liệu đã được chuyển về kiểu dữ liệu phức tạp ban đầu sau khi đã kiểm tra (validate).

• Lớp Serializer cung cấp các phương thức chung để xử lý output cho response.

• Lớp ModelSerializer giúp nhanh chóng tạo lớp Serializer với các trường tương ứng với các trường của Model.

• Lớp HyperlinkedModelSerializer tương tự ModelSerializer, nhưng nó sử dụng siêu liên kết đại diện cho các quan hệ.

- ***APIView:*** REST framework cung cấp lớp APIView là lớp con của lớp View trong Django.Trong đó:

+ Đối tượng request là thể hiện Request của REST framework.

+ Các phương thức trả về reponse là thể hiện Response của REST framework.

+ Sử dụng cũng tương tự View sẽ có các phương thức request tương ứng như get(), post() và nhiều thuộc tính cho API policy.

- Generic View:

+ Generic View giúp phát triển nhanh API view.

+ GenericAPIView kế thừa APIView, bổ sung thêm một số hành vi chuẩn như list và detail.

- ***Các lớp Mixins(rest\_framework.mixins):*** cung cấp các action hiện thực sẵn các hành vi view cơ bản.

- ***ViewSet:***

+ Django REST framework cho phép kết hợp logic của các view liên quan trong một lớp duy nhất gọi là ViewSet.

+ ViewSet không cung cấp các phương thức get(), post(), thay vào đó là các phương thức như list(), create(), retrieve(), destroy(), update().

- ***Routers:*** Định tuyến cho các url mapping tới các action

- ***Authentication:***

+ Authentication là cơ chế kết hợp các request nhận được với tập các thông tin chứng thực.

+ Các chính sách permission và throttling sử dụng các thông tin chứng thực để quyết định request có quyền phép truy cập không.

+ Authentication luôn chạy khi bắt đầu view và trước khi kiểm tra permisson và throttling.

- ***OAuth2:***

+ OAuth là một giao thức mở và an toàn để chứng thực người dùng giữa các dịch vụ liên quan.

+ OAuth uỷ quyền một dịch vụ được phép truy cập những tài nguyên từ một dịch vụ khác đại diện cho người dùng, mà không cần cung cấp thông tin như username, password.

+ OAuth2 là phiên bản mới nhất của giao thức OAuth.

+ Các bước thực hiện giao tiếp giữa client và server theo giao thức OAuth2:

• Authorize: chứng thực yêu cầu quyền truy cập từ client.

• Yêu cầu access token.

• Truy cập các tài nguyên được bảo vệ.

- ***Django OAuth Toolkits:*** cung cấp tất cả các endpoint, dữ liệu và logic cần thiết để thực hiện giao thức OAuth2 cho django project.

- ***CORS:***

+ CORS (Cross-origin resource sharing) là cơ chế cho phép giới hạn các tài nguyên trên trang web được yêu cầu truy cập từ domain khác.

+ CORS cho phép trình duyệt và server xác định được request cross-origin có an toàn không.

- ***Tích hợp Debug Toolbar:*** chủ yếu được thực hiện trong middleware.Debug Toolbar chỉ được hiển thị trên các địa chỉ IP có trong danh sách INTERNAL\_IPS của tập tin cấu hình.

- ***Tích hợp Swagger:*** thiết lập ẩn một api để không đưa vào swagger bằng cách thiết lập thuộc tính swagger\_schema = None

1. **GraphQL**

Được ông lớn công nghệ Facebook phát triển nội bộ vào năm 2012 trước khi được phát hành công khai vào năm 2015. Nó cung cấp một giải pháp thay thế cho kiến trúc web service REST và ad-hoc.

1. **Graphene và Graphene-Django**

- Graphene là thư viện được phát triển dành riêng cho ngôn ngữ python, để phát triển hệ thống APIs cho GraphQL. Là thư viện Open Source được cộng đồng tin tưởng và sử dụng nhiều nhất. Ngoài ra còn những thư viện khác để phát triển APIs GraphQL như Ariadne, Strawberry, Tartiflette…

- Graphene-Django là thư viện được xây dựng dựa trên thư viện Graphene. Cung cấp phương thức chuyển đổi và cấu hình để phù hợp với các dự án web python dùng Django. Như phương thức chuyển đổi các field model thành các ObjectType …

1. **Giới thiệu cơ sở dữ liệu MySQL:**

- MySQL là hệ quản trị cơ sở dữ liệu có mã nguồn mở hoạt động dựa theo mô hình client-server được phát hành từ những thập niên 90s.

- Cách quản lý dữ liệu trong MySQL dựa trên cơ sở dữ liệu. Có thể có nhiều bảng quan hệ chứa dữ liệu trong cơ sở dữ liệu.

- MySQL cũng có phương thức và mã truy cập giống ngôn ngữ SQL.

- Ưu điểm:

+ Sử dụng đơn giản: MySQL là một hệ quản trị cơ sở dữ liệu có tốc độ nhanh và cực kì ổn định.

+ Bảo mật tốt: MySQL có tốc độ bảo mật rất cao vì vậy rất phù hợp khi người sử dụng phát triển các ứng dụng cần sự truy cập cơ sở dữ liệu.

+ Đa tính năng, xử lý nhiều dữ liệu: Rất nhiều chức năng của SQL được tính hợp trong MySQL.

+ Dễ đàng mở rộng và nâng cấp.

+ Tốc độ nhanh: sử dụng MySQL làm giảm chi phí, thực thi lại rất nhanh.

- Nhược điểm:

+ Còn hạn chế về chức năng, dung lượng còn ít.

+ Độ tin cậy còn kém hơn so với nhiều hệ quản trị cơ sở dữ liệu khác.