django

Create Shopify App with Python/Django

thepylot
Jan 27, 2022 • 10 min read

Create Shopify App with Python/Django

In this post, we'll cover how to create a public Shopify app using the Django web framework.

The main purpose of this article is to provide you with the general concept about creating a Shopify app. Before we start, tt would be better if you have a basic understanding of Django and DRF. Also, you can find the source code at the end of this post where you can explore the project or start your own Shopify app easily with this starter pack.

How does it work?

Creating a Shopify app is really simple since Shopify allows to display the original app inside an iframe. So, you don't need to create any specific environment for the development of the Shopify app.

However, Shopify requires to use its own Oauth while installing the app to validate the user and also asks for store permissions such as "read and write" access for products and categories, depending on what the app will utilize. At this point, we only need to set up OAuth flow to authenticate users from Shopify and

allow interaction with our Django app.

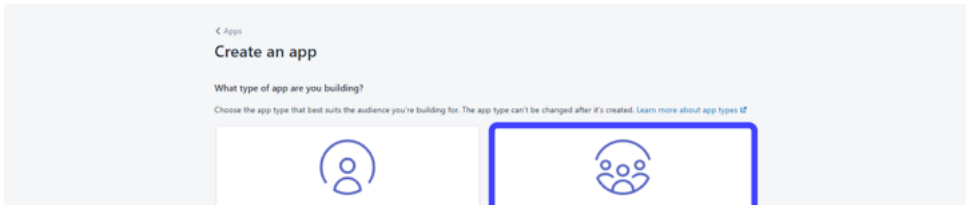
If your app includes in-app purchases then you have to process payments through Shopify billing. For more information please check the official documentation.

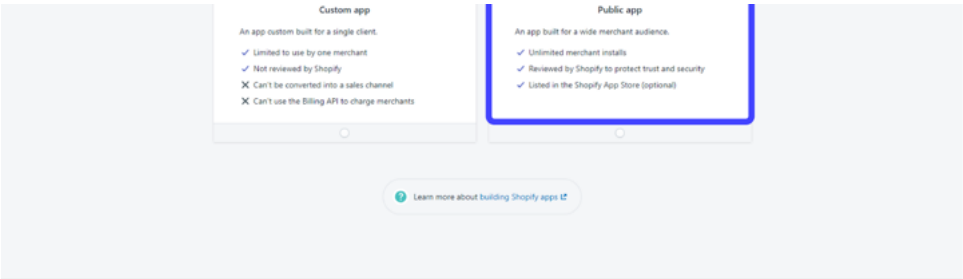
Create Shopify App

To create a Shopify app, you have to register (free) from Shopify Partners where you'll be able to set up your stores, apps and other related functionalities.

First, you need to create a development store. I would highly recommend watching the video below about how to set up a development store in Shopify which will guide your entire process:

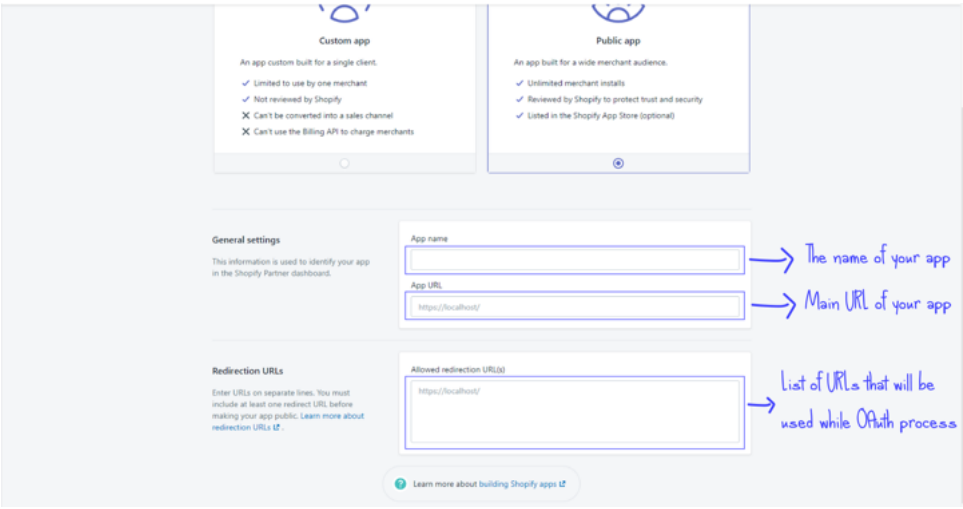
Once you set up the development store, click "*Apps*" in Partners Dashboard and click "*Create app*". Next, choose the type of app which is "*Public app*".





Create Shopify App with Python/Django

It will toggle a form element under the card which expects the *name of the app*, *main URL*, and list of *redirections* that will be used while OAuth process.



Create Shopify App with Python/Django

- **App name** - `testapp`
- **App URL** - It's the main URL of our app where Shopify redirects once the user clicks the "install" button. We're going to create a specific endpoint that will receive requests from Shopify for the authorization process.

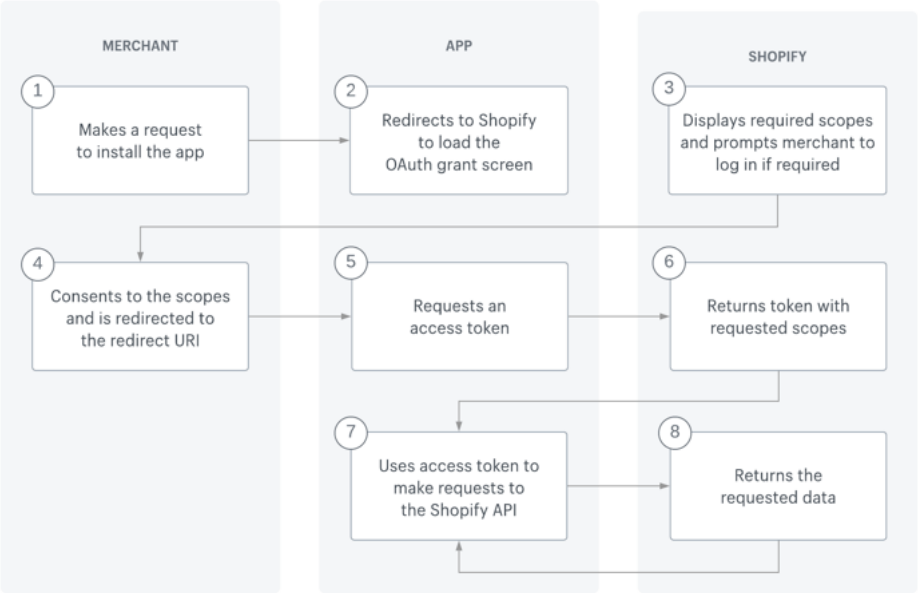
```
http://127.0.0.1:8000/account/oauth/shopify
```

- **Redirection URLs** - Comma separated URLs that will be whitelisted by Shopify to use while OAuth flow. There will be a few redirections to complete the authentication process.

```
http://127.0.0.1:8000/account/oauth/shopify
http://127.0.0.1:8000/account/oauth/shopify/authorize
```

After you set up the items above, click the "*Create app*" button to get **API credentials** that will be used for interacting with Shopify API.

OAuth Flow



Create Shopify App with Python/Django

Now we have our initial app, so it's time to explore the OAuth process. The image above was taken from official documentation which demonstrates the OAuth flow of Shopify.

In total, we'll have 2 main endpoints in our Django app for the entire OAuth flow:

`oauth/shopify/` - Validates the installation request and redirects to grant screen.

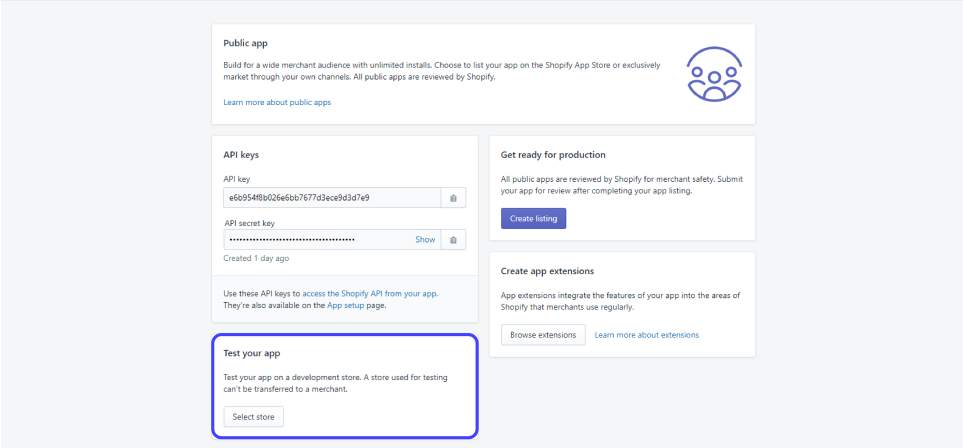
`oauth/shopify/authorize/` - Creates a new user by using fetched data from Shopify API.

Let's break it down by implementing functionalities to understand each step in detail.

User installs app

This is the starting point where the user clicks the "install" button to make the first request for installation. Shopify will redirect to a specific URL (*App URL* - `http://127.0.0.1:8000/account/oauth/shopify`) provided in settings of the app by you which is the actual endpoint to receive installation calls.

You can test it by selecting a development store from the "*Test your app*" section which locates inside the app settings page.



Create Shopify App with Python/Django

The installation will fail because we don't have any endpoints to handle incoming requests yet. You can test your app from this panel while developing the app.

Next, let's see what Shopify sends to the endpoint when a user installs the app:

```
GET '/account/oauth/shopify/?hmac=5480c2b47f7636fbceec315a34572831a9ea2e82a2257c736
```

It sends `GET` request to our `App URL` with extra query parameters that will be used in our app to authorize the request.

- `hmac` - the hashed output generated by Shopify by signing with API secret key (provided in app settings) with the message which is `shop=codepylotdev.myshopify.com&timestamp=1642957638` . We'll verify the signature with the API secret key to validate the request.
- `shop` - the name of your shop.
- `timestamp` - the created time of the request in Unix format.

Now let's start to implement this step by creating a new view and serializer inside `account` app:

account/views.py

```
from django.http.response import HttpResponseRedirect
from rest_framework.generics import GenericAPIView
from rest_framework.permissions import AllowAny
from rest_framework.response import Response


from account.serializers import ShopifyOauthSerializer, ShopifyUserCreati
from account.utils.constants import ShopifyOauth
from account.utils.oauth_client import ShopifyOauthClient


class ShopifyOauthRedirectAPIView(GenericAPIView):
    """Redirects to Shopify to confirm permissions
    """
    permission_classes = [AllowAny]
    serializer_class = ShopifyOauthSerializer

    def get(self, request):
        serializer = self.get_serializer(data=request.query_params)
        if serializer.is_valid(raise_exception=True):
            oauth_client = ShopifyOauthClient(shop_name=request.query_par
            redirect_url = oauth_client.build_oauth_redirect_url(request.
        return HttpResponseRedirect(redirect_to=redirect_url)
```

We're processing request data with serializers and then validated data is going to be passed to `ShopifyOauthClient` to build authorization redirect URL.

account/serializers.py

```
from django.contrib.auth.models import User
from rest_framework import serializers
from account.utils.constants import ShopifyOauth
from account.utils.helpers import  search_string_match, verify_hash_signa


class ShopifyOauthSerializer(serializers.Serializer):
    code = serializers.CharField(required=False)
    hmac = serializers.CharField(required=True)
    host = serializers.CharField(required=False)
    shop = serializers.CharField(required=True)
    timestamp = serializers.CharField(required=True)

    def check_signature(self, attrs):
        secret = ShopifyOauth.SECRET_KEY
        if attrs.get('code'):
            msg = (f"code={attrs['code']}&host={attrs['host']}"
                   f"&shop={attrs['shop']}&timestamp={attrs['timestamp']}"
        else:
            msg = f"shop={attrs['shop']}&timestamp={attrs['timestamp']}"
        is_verified = verify_hash_signature(secret, msg, attrs['hmac'])
        if not is_verified:
            raise serializers.DjangoValidationError(
                {'signature': ["Signature is not valid"]}
            )

        return attrs
```

```
    def validate_shop_url(self, shop_url):
        shop_name_regex = search_string_match(r'[^.\s]+\.myshopify\.com',
        if shop_name_regex != shop_url:
            raise serializers.DjangoValidationError(
                {'shop_name': ["Shop name does not end with 'myshopify.cor
            )
        return shop_url

    def validate(self, attrs):
        self.check_signature(attrs)
        return attrs
```

First, we are checking if the hashed signature (`hmac`) is valid by building the `msg` from query params and using the result with `SHA-256` algorithm for verification.

Then we also need to validate the shop name to make sure it ends with `.myshopify.com`

Redirecting to Grant Screen

In the first step, we successfully validated the request call from Shopify by checking the signature and shop name provided in query params.

Now, we're going to build a redirection URL to show the user OAuth grant screen with a list of permissions. Consider the structure and definition of the redirection URL below which is taken from documentation:

```
https://{shop}.myshopify.com/admin/oauth/authorize?client_id={api_key}&scope={scope
```

Let's break it down:

- **shop** - The name of the merchant's shop.
- **api_key** - The API key for the app.
- **scopes** - A comma-separated list of scopes. For example, to write orders and read customers, use `scope=write_orders,read_customers` . Any permission to write a resource includes the permission to read it.
- **redirect_uri** - The URL to which a merchant is redirected after authorizing the app. The complete URL specified here must be added to your app as an allowed redirection URL, as defined in the Partner Dashboard.
- **nonce** - A randomly selected value provided by your app that is unique for each authorization request. During the OAuth callback, your app must check that this value matches the one you provided during authorization. This mechanism is important for the security of your app.
- **access_mode** - Sets the access mode. For online access mode, set to

per-user. For offline access mode, set to value. If no access mode is defined, then it defaults to offline access mode.

If you want to make your app even more secure, then you should add a unique state or **nonce** as an extra query parameter to make sure redirections are not interrupted by malicious users.

You can use `uuid` module to generate unique hex strings and write them to DB (NoSQL preferred). Try to create it while checking the validation of the installation request and then confirm the existence in the second redirection which is for authorization (after the user grants permissions). This step is not required but adds an extra security layer.

Now, let's remember what we have passed to `ShopifyOauthClient` class after getting validated data from the serializer.

```
            oauth_client = ShopifyOauthClient(shop_name=request.query_par
            redirect_url = oauth_client.build_oauth_redirect_url(request.
```

`ShopifyOauthClient` is initialized with `shop_name` from validated data.

Let's take a look to the `ShopifyOauthClient` class to understand how redirection URL is built and also consider other functions since we'll refer to them later in this post:

account/utils/oauth_client.py

```
import requests

from account.utils.constants import ShopifyOauth
from account.utils.helpers import get_host_url, response_to_dictionary


class ShopifyOauthClient:

    def __init__(self, shop_name: str, access_token: str = ""):
        self.shop_name = shop_name
        self.access_token = access_token

    def get_access_token(self, **data):
        res = requests.post(
            f"https://{self.shop_name}{ShopifyOauth.ACCESS_TOKEN_ENDPOINT
```

```
            data,
            timeout=60,
            verify=False
        )
        self.access_token = response_to_dictionary(res.text)['access_toke
        return self.access_token

    def get_shop_details(self):
        res = requests.get(
            f"https://{self.shop_name}{ShopifyOauth.SHOP_DETAILS_ENDPOINT
            headers={ShopifyOauth.ACCESS_TOKEN_HEADER: self.access_token}
        )
        shop_data = response_to_dictionary(res.text)['shop']

        return shop_data['email'], shop_data['shop_owner']

    def build_oauth_redirect_url(self, absolute_url):
        host = get_host_url(absolute_url)
        redirect_url = (
            f"https://{self.shop_name}{ShopifyOauth.AUTHORIZE_ENDPOINT}?"
            f"client_id={ShopifyOauth.API_KEY}&scope={ShopifyOauth.SCOPES
            f"&redirect_uri={host}{ShopifyOauth.REDIRECT_ENDPOINT}"
            f"&grant_options[]={ShopifyOauth.ACCESS_MODE}"
        )
        return redirect_url
```

In the constructor, we are initializing two main fields of class which are `shop_name` and `access_token` that will be used to interact with Shopify API.

`build_oauth_redirect_url` builds redirection URL by taking pre-defined values from `constants` file below:

account/utils/constants.py

```
from django.conf import settings


class ShopifyOauth:
    API_KEY = settings.SHOPIFY_PUBLIC_APP_KEY
    SECRET_KEY = settings.SHOPIFY_PUBLIC_APP_SECRET_KEY
    SCOPES = (
        "read_orders,write_orders,read_customers,write_customers,"
        "read_products,write_products,read_content,write_content,"
        "read_price_rules,write_price_rules,read_themes,write_themes"
    )
    ACCESS_MODE = "per_user"
    REDIRECT_ENDPOINT = "/account/oauth/shopify/authorize"
    ACCESS_TOKEN_HEADER = "X-Shopify-Access-Token"
    ACCESS_TOKEN_ENDPOINT = "/admin/oauth/access_token"
    AUTHORIZE_ENDPOINT = "/admin/oauth/authorize"
    SHOP_DETAILS_ENDPOINT = f"/admin/api/2021-04/shop.json"

SHOPIFY_API_VERSION = "2021-04"
```

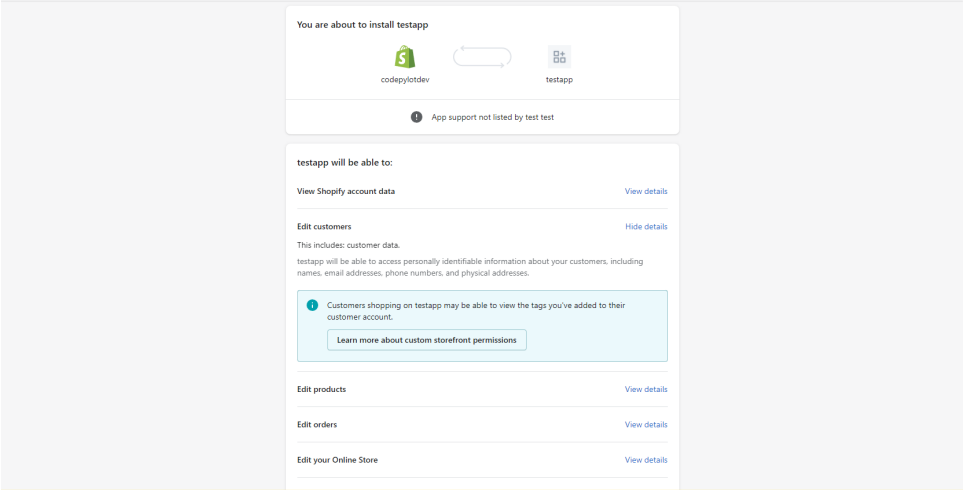
Once redirect URL is built, we are redirecting the user to grant screen from views:

account/views.py

account/views.py

```
return HttpResponseRedirect(redirect_to=redirect_url)
```

The grant screen will look like below:



Create Shopify App with Python/Django

At the end of the page, you should click "Install app" to continue the process.

Get a permanent access token

After the user grants permissions, Shopify makes another redirection to a URL which we assigned to `redirect_uri`:

```
&redirect_uri={host}{ShopifyOauth.REDIRECT_ENDPOINT}
```

Consider the structure that concatenates `host` and value of `ShopifyOauth.REDIRECT_ENDPOINT` constant which produces the absolute URL of the authorization endpoint of our Django app. Since our app is running localhost the full URL will be like below:

```
http://127.0.0.1:8000/oauth/shopify/authorize
```

Now we're allowed to get merchants data by using Shopify API calls. Let's add another view to handle the authorization & user creation process:

account/views.py

```
class ShopifyUserCreationAPIView(GenericAPIView):
    """Creates new user for Shopify Public App
    """
    permission_classes = [AllowAny]
    serializer_class = ShopifyOauthSerializer

    def get(self, request):
        serializer = self.get_serializer(data=request.query_params)
```

```
        if serializer.is_valid(raise_exception=True):
            shop_name = serializer.validated_data['shop']
            oauth_client = ShopifyOauthClient(shop_name)
            token = oauth_client.get_access_token(
                client_id=ShopifyOauth.API_KEY,
                client_secret=ShopifyOauth.SECRET_KEY,
                code=serializer.validated_data['code']
            )
            email, owner = oauth_client.get_shop_details()
            serializer = ShopifyUserCreationSerializer(
                data={
                    "email": email,
                    "full_name": owner,
                    "shop_name": shop_name,
                    'token': token,
                    'state': request.query_params.get("state")
                }
            )
            if serializer.is_valid(raise_exception=True):
                user = serializer.create(validated_data=serializer.valida
                bridge_url = f"https://{shop_name}/admin/apps/testapp"
                return HttpResponseRedirect(redirect_to=bridge_url)

        return Response({"message": "Authentication failed"})
```

The serializer class is same since we have to check the signature for each request from Shopify. In this time, the `msg` will include new parameter named `code` that was sent after user grants access so we have to include it for hash verification. It will be used to get `access token` from Shopify.

So, we're initializing our `ShopifyOauthClient` to get permanent `access token` from the Shopify's endpoint below:

```
POST https://{shop}.myshopify.com/admin/oauth/access_token
```

In our request, `{shop}` is the name of the merchant's shop and the following parameters must be provided in the request body:

- **client_id** - The API key for the app, as defined in the Partner Dashboard.
- **client_secret** - The API secret key for the app, as defined in the Partner Dashboard.
- **code** - The authorization code provided in the redirect.

So, we're passing the required data to `get_access_token` function as shown above and let's take a look inside the function to see the rest of the process:

account/utils/oauth_client.py

```
    def get_access_token(self, **data):
        res = requests.post(
            f"https://{self.shop_name}{ShopifyOauth.ACCESS_TOKEN_ENDPOINT
            data,
```

```
            timeout=60,
            verify=False
        )
        self.access_token = response_to_dictionary(res.text)['access_token']
        return self.access_token
```

After getting `access token` from Shopify, we're ready to pass it as a header for interacting with the API.

Creating new user

The next step is getting the shop details of a merchant with `get_shop_details` function. The response data will include the email and the full name of the owner where we can use it to create a new user.

Then these values are passed to the serializer to create a new user like below:

```
class ShopifyUserCreationSerializer(serializers.Serializer):
    email = serializers.EmailField(required=True)
    full_name = serializers.CharField(required=True, max_length=255)
    shop_name = serializers.CharField(required=True, max_length=255)
    token = serializers.CharField(required=True, max_length=255)

    def create(self, validated_data):
        user, _created = User.objects.get_or_create(
            email=validated_data['email'],
        )
        if _created:
            user.username = validated_data['email']
            password = User.objects.make_random_password(length=10)
            user.is_active = True
            user.set_password(password)

            user.first_name = validated_data['full_name']
            user.save()

        return user
```

Note that, you should also store `access token` in DB since it's a **permanent token** and will be used in **each request with Shopify API**. For now, we're only creating a new user with validated data. Feel free to add your changes in case you have any extra data to be saved.

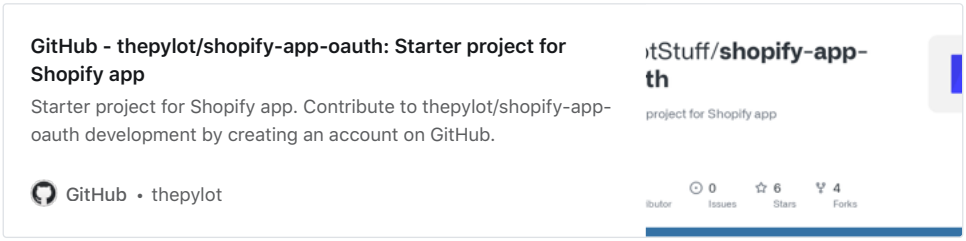
What's next?

We learned how to set up Shopify OAuth to create a new user with `access token` that will be used for interaction with Shopify API. Now here is the list of what you should do after these steps:

- Make sure you're saving `access token` for the created user.
- After the user is created, the final redirection should point to Shopify to display the embedded app.
  `https://{shop}.myshopify.com/admin/apps/{app_name}`.

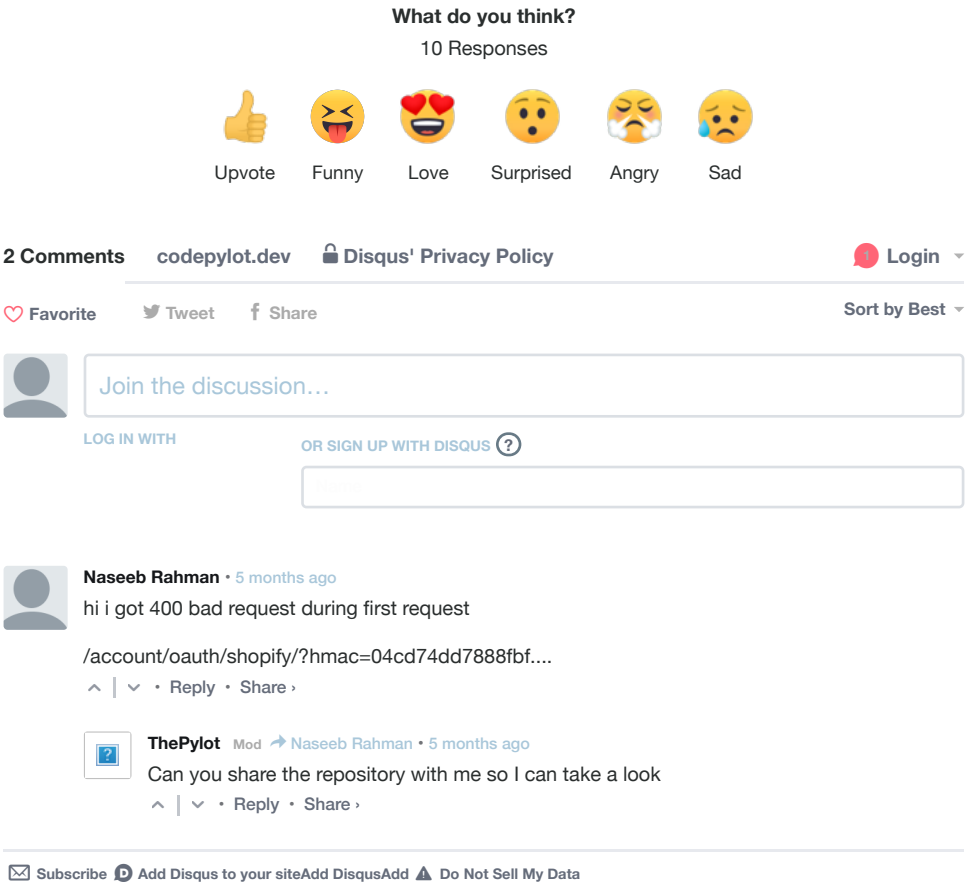
- Django should generate a JWT token for the created users to allow requests from the front-end side (React). You can create it inside `ShopifyUserCreationSerializer`.
- Read about Shopify Bridge to see how you can make your app embedded.
- You'll need to run Ngrok to make your localhost publicly available so the app can show up inside Shopify.

Also, check the source code from the repository below:



Support 🌏

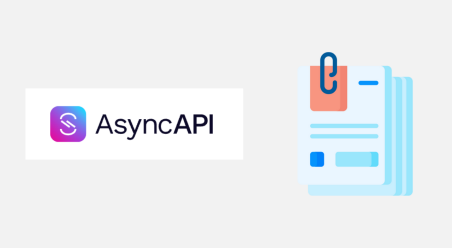
If you feel like you unlocked new skills, please share them with your friends and subscribe to the youtube channel to not miss any valuable information.



Intelligent Computing Servers

Open

Intelligent Computing



Documentation for your Event-Driven API

In this article, we'll cover the tool named AsyncAPI for documenting your event-driven...

Jun 19, 2022 — 4 min read

Setup Grafana with Prometheus for Python projects using Docker

In this article, we will cover how to set up service monitoring for Python projects with...

Jun 11, 2022 — 6 min read

How to optimize the performance of Django admin with millions of data in MongoDB

In this article, I want to share my previous experience where I managed to increase the...

Jun 4, 2022 — 3 min read

Data & License Github