how would i go about adding a duo login for my website so that way users can make orders or report issues

Got you, Zeke. You've basically got two clean paths to add Duo:

Option A: Add Duo 2FA to your existing Django login (fastest to wire up)

This keeps your current username plus password flow, then sends users to Duo's Universal Prompt for the second factor.

What you set up in Duo

1. In the Duo Admin Panel, create a Web SDK, Universal Prompt app. Copy the client_id, client_secret, api_hostname. Add your callback URL, for example **`https://your-domain.com/duo/callback`**. Duo Security
2. Universal Prompt uses an OIDC style flow behind the scenes, which is what the current SDKs implement. Duo Security

What you add to Django

1. Install Duo's Universal Python SDK:
   `pip install duo-universal` GitHub
   Optional, there is a drop-in Django middleware package if you prefer: **`pip install django-duo-universal-auth`**. PyPI
2. Configure settings:

   python

   ```
   # settings.py
   DUO_CLIENT_ID = "DIXXXXXXXXXXXXXXXXXX"
   DUO_CLIENT_SECRET = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
   DUO_API_HOSTNAME = "api-XXXXXXXX.duosecurity.com"
   DUO_REDIRECT_URI = "https://your-domain.com/duo/callback"
   LOGIN_URL = "/login/"
   LOGIN_REDIRECT_URL = "/"
   ```

3. Views: after a normal Django auth login succeeds, start Duo. On the callback, verify and mark the session Duo-verified.

   python

   ```
   # views.py
   from django.conf import settings
   from django.contrib.auth.decorators import login_required
   from django.shortcuts import redirect, render
   from duo_universal.client import Client
   from datetime import timedelta
   ```

```
from django.utils import timezone

def _duo():
    return Client(
        client_id=settings.DUO_CLIENT_ID,
        client_secret=settings.DUO_CLIENT_SECRET,
        api_hostname=settings.DUO_API_HOSTNAME,
        redirect_uri=settings.DUO_REDIRECT_URI,
    )

@login_required
def start_duo(request):
    username = request.user.username
    state = _duo().generate_state()
    request.session["duo_state"] = state
    auth_url = _duo().create_auth_url(username=username, state=state)
    return redirect(auth_url)

@login_required
def duo_callback(request):
    state = request.session.get("duo_state")
    duo = _duo()
    token = duo.exchange_authorization_code_for_2fa_result(
        request.GET.get("duo_code"), request.GET.get("state"), state
    )
    # If we get here, 2FA succeeded. Set a short TTL marker.
    request.session["duo_mfa_ok_until"] = (timezone.now() + timedelta(hours=12)).isoformat()
    return redirect(settings.LOGIN_REDIRECT_URL)
```

4. Gate the sensitive routes with a “fresh Duo” check. For example, require a recent Duo step-up before users can place orders or file issues:

```
python

def require_recent_duo(view):
    def wrapper(request, *args, **kwargs):
        ts = request.session.get("duo_mfa_ok_until")
        fresh = ts and timezone.now().isoformat() <= ts
        if not fresh:
            return redirect("/duo/start")
        return view(request, *args, **kwargs)
    return login_required(wrapper)
```

Then decorate your critical views:

```
python

@require_recent_duo
def create_order(request): ...
@require_recent_duo
def report_issue(request): ...
```

Why this works: the Universal Prompt is Duo’s current, supported path. The Python SDK handles the OIDC-style handshake and signature checks so you do not need to hand-roll the protocol. Duo Security+2

Option B: Use Duo Single Sign-On as your IdP (centralized login)

If you want Duo to be your identity provider, you can register your app as an OIDC or SAML service and outsource both primary and second-factor auth to Duo SSO. Then your Django app trusts Duo’s assertions. This is great if you want campus-wide SSO policies or to plug in AD/LDAP later. Duo Security+1

High level:

1. In Duo SSO, create a Generic OIDC or Generic SAML application and set your redirect or ACS URLs from Django. Duo Security+1
2. In Django, add an OIDC or SAML client (for example, **`mozilla-django-oidc`** or **`python3-saml`**) and trust Duo's metadata. Duo enforces MFA and sends you the authenticated user. Duo Security

Which should you pick?

• You already have Django auth and just want Duo as a second factor for "Place order" and "Report issue": choose Option A. It is quick, minimal code, and you can "step-up" only on sensitive pages. Duo Security
• You want one login for many apps now or later: choose Option B. Centralized policies, SSO, and MFA all in one place. Duo Security

Small rollout checklist

1. Create a Duo test user and enroll them in Universal Prompt. Duo Guide+1
2. Add the app in Duo, copy the client values, and set your redirect URI. Duo Security
3. Wire the SDK and callback in Django, protect only one test route first. GitHub
4. Enable policies like "Duo required only for order or issue routes" and tune session lifetime. Duo Security
5. Flip it on for your account, then Ed's, then the student worker group, and so on.

If you want, I can tailor the exact code to your project's URL structure and drop the decorator on your "Order" and "Report issue" views.

i would want option B so it can be integrated more cleanly so can you give me a rollout plan for that

Awesome, Option B it is. Here's a clean, pragmatic rollout plan to put Duo Single Sign-On in front of your Django app using OIDC, with minimal downtime and a safe rollback.

Phase 0: Decide scope and protocol

1. Use Duo SSO as the IdP with a Generic OIDC app. This gives you Duo MFA plus centralized policies and an OIDC discovery URL you plug into Django. Duo Security+1
2. Prefer OIDC over SAML for Django. It is simpler to wire with a mature package like mozilla-django-oidc. mozilla-django-oidc.readthedocs.io+1

Phase 1: Create Duo SSO OIDC app

1. In the Duo Admin Panel, add the “Generic OIDC Relying Party – Single Sign-On” application. Copy Client ID, Client Secret, Issuer and the OIDC discovery endpoints. Add your Redirect URI, for example https://your-domain.com/oidc/callback↗. Duo Security+1
2. Universal Prompt is the default experience with Duo SSO, which replaces the old iframe prompt. Good for long-term support and UX. Duo Guide+1

Phase 2: Django wiring in a feature branch

1. Install and enable an OIDC client library. Common choice: mozilla-django-oidc. mozilla-django-oidc.readthedocs.io+1
2. Configure settings using Duo’s OIDC endpoints from discovery metadata. You will set at minimum: OIDC_RP_CLIENT_ID, OIDC_RP_CLIENT_SECRET, OIDC_OP_AUTHORIZATION_ENDPOINT, OIDC_OP_TOKEN_ENDPOINT, OIDC_OP_USER_ENDPOINT, OIDC_OP_JWKS_ENDPOINT, plus SITE_URL. mozilla-django-oidc.readthedocs.io+1
3. Map the Duo SSO claim you want to use as the Django username. Typical is email or upn, exposed in the userinfo endpoint. Plan to normalize it to your existing usernames to avoid duplicates. Duo Security
4. Add login routes. For example:
   - /oidc/auth to start OIDC
   - /oidc/callback for the redirect
   - Update LOGIN_URL and LOGIN_REDIRECT_URL accordingly. mozilla-django-oidc.readthedocs.io
5. Keep your current superuser path active behind a secret URL for emergency access during cutover.

Phase 3: Policy and groups in Duo

1. In Duo SSO, scope an application policy for your Django app. Require Duo MFA and set session duration. You can set stricter policies for sensitive actions later. Duo Security
2. Decide who can log in on day one. Start with you, Ed and a small pilot group, then expand to student workers and general users.

Phase 4: Attribute and provisioning plan

1. Attribute choice. Decide the authoritative identifier that links Duo identities to Django users. Email is simplest. Document the rule, for example “Django username equals email local part.” Duo Security
2. New user provisioning. Choose auto-create on first login or pre-seed users. mozilla-django-oidc supports creating users and mapping claims in a custom pipeline. mozilla-django-oidc.readthedocs.io

Phase 5: Test matrix in a staging environment

1. Validate the OIDC flow end to end: authorization, token, userinfo and session creation. mozilla-django-oidc.readthedocs.io
2. Try multiple factors in Universal Prompt: Duo Push, passkeys, phone, etc. Confirm “remembered devices” and policy behavior. Duo Guide

3. Error handling drills. Duo has a common OIDC errors guide; provoke bad redirect_uri, wrong client secret and stale state to ensure you show a friendly error and log details. Duo Security

Phase 6: Step-up protection for critical actions

1. Even with SSO, you can add a “fresh auth” step before creating orders or submitting issue reports. Implement a short re-auth window using your existing session plus an ACR or prompt=login round-trip to Duo if needed. Policy-side, you can shorten the SSO session for this app. Duo Security

Phase 7: Cutover plan

1. Announce the change and provide a one-pager: where the new “Log in with Duo” button is, what factors they can use, and where to enroll if Duo prompts them. Duo Security
2. Switch the production LOGIN_URL to the OIDC login. Keep the secret local-admin URL for 24 to 48 hours.
3. Monitor logs for OIDC and app errors and keep a quick rollback: revert LOGIN_URL and temporarily disable the Duo SSO app if needed.

Phase 8: Post-launch hardening

1. Set final session and device trust policies in Duo SSO for the app. Tune based on real use. Duo Security
2. If you plan API access later, note that Duo SSO also supports OAuth features like token introspection for machine-to-machine flows. Not needed now, but good to know. Duo Security
3. Document the attribute mapping, emergency access, and support steps in your repo’s docs.

Minimal config sketch for Django

You will plug Duo’s discovery data into these settings from mozilla-django-oidc docs:
- OIDC_OP_AUTHORIZATION_ENDPOINT
- OIDC_OP_TOKEN_ENDPOINT
- OIDC_OP_USER_ENDPOINT
- OIDC_OP_JWKS_ENDPOINT
- OIDC_RP_CLIENT_ID and OIDC_RP_CLIENT_SECRET
- SITE_URL and LOGIN_URL paths

Then implement user creation and claim mapping in the OIDC callback per the library’s quick start. mozilla-django-oidc.readthedocs.io+2

If you want, I can drop in a ready-to-paste settings.py and a tiny OIDC auth view module aligned to your current URL structure and your desired username rule.

so i have decided to not do DUO as it would be an extra step that is not neccisarily needed but how do you suggest i go about emailing users for things such as password changes because my college will not let me make an account to put in their system to do it so i would have to make a personal one that can send out items such as this

Got it. Since Berea won't let you hook into their official email system, you'll need to handle transactional emails (password resets, order confirmations, issue receipts, etc.) from your own account. Here are a few paths you can take:

1. Use a personal email account via SMTP

- Gmail, Outlook, Yahoo, etc. all expose SMTP servers.
- In Django, you configure something like this in `settings.py`:

```python
EMAIL_BACKEND = "django.core.mail.backends.smtp.EmailBackend"
EMAIL_HOST = "smtp.gmail.com"
EMAIL_PORT = 587
EMAIL_USE_TLS = True
EMAIL_HOST_USER = "youraccount@gmail.com"
EMAIL_HOST_PASSWORD = "your_app_password"
DEFAULT_FROM_EMAIL = "youraccount@gmail.com"
```

- **Caution**: for Gmail/Outlook you should create an *app password* (not your real login) if 2FA is on.

2. Create a project-only mailbox

- Register a free mailbox just for this system (e.g., a Gmail account like `printer-system-notify@gmail.com` or a cheap domain + mailbox).
- Keeps things separate from your personal inbox.

3. Use a transactional email service (cleaner, more reliable)

- Free tiers exist for SendGrid, Mailgun, Postmark, Amazon SES.
- These are designed for app-generated emails, give you better deliverability, and don't tie the system to your personal email.
- Example with SendGrid:

```python
EMAIL_BACKEND = "django.core.mail.backends.smtp.EmailBackend"
EMAIL_HOST = "smtp.sendgrid.net"
EMAIL_PORT = 587
EMAIL_USE_TLS = True
EMAIL_HOST_USER = "apikey"
EMAIL_HOST_PASSWORD = "<your sendgrid api key>"
DEFAULT_FROM_EMAIL = "no-reply@yourdomain.com"
```

4. Best practices

- **Separate mailbox**: even if you don't buy a domain, create a dedicated Gmail or Outlook account just for this project.
- **No-reply address**: use something like `no-reply@...` so people don't accidentally send responses.

- **SPF/DKIM (if domain)**: if you buy a cheap domain, add DNS records so your mails don't hit spam.
- **Templates**: Django has `django.core.mail.send_mail` and `EmailMessage` for plain text, and `django-templated-mail` for HTML templates.

Do you want me to draft a rollout plan for setting up a **dedicated Gmail account with app password** (simplest), or do you want me to show you the **SendGrid free-tier setup** (cleaner long term)?

so i have created a fresh outlook account how do i need to format it so my site can send from it

Perfect, Outlook works fine with Django's built-in email backend. Here's exactly how to wire it:

1. Enable SMTP on Outlook

- Go to your new Outlook account → **Security** → **Advanced Security Options**.
- Turn on **Two-step verification** (recommended).
- Generate an **App Password** (special password just for your Django app).
  You'll use this instead of your real Outlook password.

2. Configure Django settings

In your `settings.py`, add this:

```python
EMAIL_BACKEND = "django.core.mail.backends.smtp.EmailBackend"
EMAIL_HOST = "smtp.office365.com"     # Outlook/Office365 SMTP server
EMAIL_PORT = 587                      # TLS port
EMAIL_USE_TLS = True
EMAIL_HOST_USER = "yournewaccount@outlook.com"
EMAIL_HOST_PASSWORD = "your_app_password"    # not your real login password
DEFAULT_FROM_EMAIL = "Printer System <yournewaccount@outlook.com>"
```

3. Send a test email

In the Django shell (`python manage.py shell`):

```python
from django.core.mail import send_mail

send_mail(
    subject="Test Email",
    message="This is a test email from the Capstone system.",
    from_email=None,  # uses DEFAULT_FROM_EMAIL
    recipient_list=["yourpersonal@email.com"],
    fail_silently=False,
)
```

You should see it in your inbox within a minute.

4. Production best practices

- Keep the app password in a `.env` file or environment variable, not hardcoded. Example with `django-environ`:

  ```python
  EMAIL_HOST_PASSWORD = env("OUTLOOK_APP_PASSWORD")
  ```

- For system messages like password resets, use a consistent `DEFAULT_FROM_EMAIL`, e.g., `no-reply-printersystem@outlook.com` so users don't try to reply.
- Later, if you want fancier HTML templates, add `django-templated-mail` or just use `EmailMultiAlternatives`.

Do you want me to also show you how to hook this into Django's **password reset flow** so Outlook handles those emails automatically?

> how do i do this: The account isn't actually allowed to use basic SMTP. In Microsoft 365/Admin center you need Authenticated SMTP enabled for that mailbox. If your tenant uses Security Defaults, authenticated SMTP is off for all accounts unless you explicitly turn it on for just this mailbox.

Good catch. Outlook/Office365 blocks SMTP by default now, so if you just created a fresh Outlook.com or Microsoft 365 mailbox, you need to explicitly allow **Authenticated SMTP**. Here's how to do it, step by step:

---

1. If this is a free personal Outlook.com account

- Microsoft has been gradually disabling "basic auth SMTP" on personal accounts too.
- The trick: you can't just use your normal password, you need to generate an **App Password** (requires MFA turned on).
- If SMTP is still blocked, you'll need to use **Microsoft Graph API** or an SMTP relay service instead.

---

2. If this is a Microsoft 365 account (work/school tenant)

1. Log in to the Microsoft 365 Admin Center ↗.
2. Go to **Users** → **Active Users**.
3. Select the account you created for the app.
4. Under **Mail** settings, look for **Mail apps** → **Manage email apps**.
5. Turn **ON** the toggle for **Authenticated SMTP**.
6. Save changes.

3. If your tenant has Security Defaults or Modern Authentication

- With **Security Defaults** on, basic auth protocols (like SMTP AUTH) are blocked globally.
- To work around that:
    - Go to **Azure Active Directory Admin Center** → **Users** → pick your mailbox.
    - Under **Mail** → **Mail Apps**, enable “Authenticated SMTP.”
    - If Security Defaults block it, you may need to create a **Conditional Access policy** that allows just this one account to use SMTP.

---

4. Alternative if SMTP is blocked entirely

If you can’t get Authenticated SMTP working:

- Use **Microsoft Graph API** for sending mail. Django doesn’t support it out of the box, but you can install `O365` or `msal` Python libraries and authenticate with OAuth2.
- Or use a relay service (SendGrid, Mailgun, Amazon SES) and point your Django app there. This avoids Outlook limitations completely.

⚠️ **Important security note**: Don’t give this mailbox global privileges. Make it a restricted, mail-only account with a strong password and MFA (then generate an App Password for SMTP).

Do you want me to show you the **exact Outlook.com vs Microsoft 365 check** (to confirm which one you made), and then I can write the correct setup path for that case?