Modular Django App Structure (Pluggable Architecture)

1. `accounts`

Purpose: Handles authentication, roles, user profiles.

- Custom `User` model (if not already implemented)
- Role/Permission system (if not using Django Groups)
- 2FA / email OTP logic (if needed)
- Profile management

Reusable in any Django project
Dependencies: None

2. `documents`

Purpose: Core logic for document upload, redaction, and storage.

- Models:
  - `Document`
  - `RedactedDocument`
- File validation
- Encrypted storage (optional)
- Redaction processing utilities

Independent module
�**Dependencies**: `accounts.User`

3. `redaction`

Purpose: Handles redaction logic — useful even outside this platform.

- Handles image/PDF redaction given coordinates
- Utility methods using `Pillow`, `PyMuPDF`, `OpenCV`
- Optionally expose redaction API endpoints

Reusable for any system with redaction (HR, legal, etc.)
Dependencies: Only file path input

4. `sharing`

Purpose: Manages document sharing, access rights, and expiration.

- Models:
    - `DocumentShare`
- Share via user/email
- Expiration handling
- Link generation (optional signed URLs)
- Download/view restriction logic

Reusable for any sharable object (e.g., reports, media, files)
Dependencies: `documents`, `accounts`

5. `viewer`

Purpose: Secure viewing of shared documents.

- Custom secure viewer page
- Watermark overlay engine (username, IP, etc.)
- Disable right-click, F12, print
- Logs access

Reusable for secure viewing in any app
Dependencies: `sharing.DocumentShare`, `accounts.User`

6. `auditlog` (Optional but powerful)

Purpose: General-purpose access log/audit log module.

- Log any object access:
    - `object_id`, `object_type`, `user`, `timestamp`, `action`
- Signals/hooks to other apps
- Admin views for audit reports

Highly reusable across projects
Dependencies: Generic foreign key

7. `api` (DRF APIs)

Purpose: Central app exposing API views for all modules.

- Versioned REST APIs for:
    - Upload
    - Redact

    - Share
    - View document
    - Fetch access logs
- JWT/Session auth

Reusable for building cross-platform apps
Dependencies: `documents`, `sharing`, `viewer`, `accounts`

---

8. `utils` (Optional)

Purpose: Common utilities, mixins, validators.

- File utilities
- Encryption helpers
- Shared constants
- Logging/email helpers

Reusable utility kit
Dependencies: Light, can be imported as needed

Suggested Dependency Flow

```
accounts ┐
         ├──> documents ───┐
         └─────────────┐   │
                       ↓   ↓
                sharing ──> viewer
                   ↓         ↓
                   auditlog (log actions)
```

Optional Add-ons

App Name	Purpose
`notifications`	Email/Push alerts on view/share
`encryption`	File encryption/decryption as a service
`billing`	If you monetize sharing (credits/tokens)
`integration`	Connect to other internal systems (ERP, HRMS, etc.)

Summary

App Name	Description	Can Reuse?	Depends On
`accounts`	Users, roles, profiles		None

App Name	Description	Can Reuse?	Depends On
`documents`	Upload & store documents		`accounts`
`redaction`	Redact files using coordinates		None
`sharing`	Share documents with users		`documents`, `accounts`
`viewer`	Secure viewer with watermarking		`sharing`, `accounts`
`auditlog`	Generic access logging system		Generic ForeignKey
`api`	API interface (DRF)		All functional apps
`utils`	Shared utilities/helpers		None

MODELS

This version aligns with your original goals:

- Upload & manage confidential PDF/images
- Store redacted version separately
- Use `UUIDField` for secure identifiers
- Connect with `accounts.User` (as owner)
- Optional encrypted storage support
- Standalone, reusable, and ready for DRF integration

documents/models.py

```
import uuid

from django.db import models

from django.conf import settings

from django.utils import timezone



def original_document_path(instance, filename):

    return f'documents/originals/{instance.owner.id}/{uuid.uuid4()}_{filename}'



def redacted_document_path(instance, filename):

    return f'documents/redacted/{instance.owner.id}/{uuid.uuid4()}_{filename}'



class Document(models.Model):

    id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)

    owner = models.ForeignKey(

        settings.AUTH_USER_MODEL, on_delete=models.CASCADE,
related_name='uploaded_documents'

    )

    title = models.CharField(max_length=255)

    description = models.TextField(blank=True)

    file = models.FileField(upload_to=original_document_path)
    file_type = models.CharField(max_length=50, choices=[
        ('pdf', 'PDF'),
        ('image', 'Image'),
    ])
    created_at = models.DateTimeField(auto_now_add=True)

    is_encrypted = models.BooleanField(default=False)
    encryption_key = models.CharField(max_length=256, blank=True, null=True)  #
store key securely if needed

    is_redacted = models.BooleanField(default=False)

    def __str__(self):
        return self.title

    class Meta:
        ordering = ['-created_at']


class RedactedDocument(models.Model):
    id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    original = models.OneToOneField(
        Document, on_delete=models.CASCADE, related_name='redacted_version'
    )
    redacted_file = models.FileField(upload_to=redacted_document_path)
    redacted_at = models.DateTimeField(auto_now_add=True)
    redacted_by = models.ForeignKey(
        settings.AUTH_USER_MODEL, on_delete=models.SET_NULL, null=True,
related_name='redactions_done'
    )
    notes = models.TextField(blank=True)

    def __str__(self):
        return f"Redacted: {self.original.title}"
    class Meta:
        ordering = ['-redacted_at']
```

Security Notes

- **Encrypted file storage (optional):**
  - You can encrypt files before saving and decrypt only in secure views.
  - Store the `encryption_key` securely (ideally encrypted or in a key vault).
- **Redaction Separation**:
  - `RedactedDocument` is split as a new model so that:
    - You maintain the original untouched
    - Redaction is clearly audited and managed
    - Easy to plug into workflows where redaction is optional/conditional

Suggestions for Extending This App

Feature	Suggestion
Encrypted file access	Use `cryptography.fernet` to encrypt file bytes before storage
Soft delete	Add `is_deleted` or use a `django-softdelete` package
File type validation	Add `clean()` or `FileValidator` to restrict file types
Virus scanning	Integrate with tools like `clamav` or use a queue worker before saving
Versioning	Add a `DocumentVersion` model if redaction or edits need tracking over time

App: `redaction`

Purpose:

- Decouple redaction logic from `documents`.
- Support redacting text or regions (e.g., rectangle coordinates) from PDFs and images.
- Useful for **HR, legal, compliance, medical records**, etc.
- Can be used as a **utility API or standalone service**.

Dependencies:

- No tight Django app dependency — works with **file paths** or `FileField`.
- No direct foreign key to `Document`.

`models.py`

```
# redaction/models.py


import uuid

from django.db import models


class RedactionTask(models.Model):

    """

    Represents a redaction request/task on a file.

    This model is generic — accepts any file input.

    """

    id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)

    original_file = models.FileField(upload_to='redaction/originals/')

    redacted_file = models.FileField(upload_to='redaction/redacted/', null=True,
blank=True)


    created_at = models.DateTimeField(auto_now_add=True)

    processed = models.BooleanField(default=False)


    def __str__(self):
        return f"RedactionTask {self.id}"


class RedactionRegion(models.Model):
    """
    Specifies a redaction area/region for a given RedactionTask.
    Works for both images and PDF pages.
    """
    id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    task = models.ForeignKey(RedactionTask, on_delete=models.CASCADE,
related_name='regions')

    # For PDF-based redaction (0-indexed)
    page_number = models.PositiveIntegerField(default=0)

    # Coordinates in pixels (top-left x, y, width, height)
    x = models.FloatField()
    y = models.FloatField()
    width = models.FloatField()
    height = models.FloatField()

    # Optional text label for UI/debugging
    label = models.CharField(max_length=100, blank=True, null=True)

    def __str__(self):
        return f"Region on page {self.page_number} for task {self.task.id}"
```

Security Features

- Files are stored in separate folders (`originals/`, `redacted/`)
- No user data embedded → reusable across projects
- UUIDs are used instead of `id` in URLs
- **No direct user reference**, to keep it general-purpose and shareable

How You Can Use This

In your `documents` app or anywhere else:

- When redaction is triggered, create a `RedactionTask`
- Add `RedactionRegion` for each redaction box
- Process using PyMuPDF / Pillow / OpenCV and save the result to `redacted_file`

App: `sharing`

Purpose:
Handles document sharing securely via user or email with expiration control and access restrictions.
Designed to be reusable for sharing **any object (e.g., documents, reports, media)**.

Dependencies:

- `documents.Document` (foreign key)
- `accounts.User` (for authenticated share targets)
- Can support unauthenticated email-based shares

`models.py`

```
# sharing/models.py


import uuid

from datetime import timedelta

from django.conf import settings

from django.db import models

from django.utils import timezone

from django.core.signing import Signer


signer = Signer()


class DocumentShare(models.Model):

    """

    Represents a document shared with a user or email, with optional expiration and
access control.

    """

    id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)


    document = models.ForeignKey('documents.Document', on_delete=models.CASCADE,
related_name='shares')

    shared_by = models.ForeignKey(settings.AUTH_USER_MODEL,
on_delete=models.CASCADE, related_name='shared_documents')
```

```
    # Either share with an authenticated user...
    shared_with_user = models.ForeignKey(
        settings.AUTH_USER_MODEL,
        on_delete=models.SET_NULL,
        null=True,
        blank=True,
        related_name='received_documents'
    )

    # ...or just an email (unregistered)
    shared_with_email = models.EmailField(null=True, blank=True)

    # Expiration
    expires_at = models.DateTimeField(null=True, blank=True)

    # Permissions
    can_view = models.BooleanField(default=True)
    can_download = models.BooleanField(default=False)

    created_at = models.DateTimeField(auto_now_add=True)
    accessed_at = models.DateTimeField(null=True, blank=True)

    def is_expired(self):
        return self.expires_at is not None and timezone.now() > self.expires_at

    def generate_signed_url(self):
        return signer.sign(str(self.id))

    def __str__(self):
        if self.shared_with_user:
            return f"{self.document.name} shared with
{self.shared_with_user.email}"
        elif self.shared_with_email:
            return f"{self.document.name} shared with {self.shared_with_email}"
        return f"Document share {self.id}"
```

Key Features Covered

- Shares are uniquely identified using UUIDs
- Supports sharing with either:
    - A registered user (`shared_with_user`)
    - Or a plain email address (`shared_with_email`)
- Optional expiration via `expires_at`
- Access control: `can_view`, `can_download`
- `generate_signed_url()` can be used in view to create secure access URLs

Notes

- Add access logging by extending this model or via a separate model (e.g. `DocumentShareAccessLog`)
- Use Django signals to send email notifications when a document is shared

5. viewer

Purpose: Secure viewing of shared documents with watermarking and activity logging.

`models.py`

```
# viewer/models.py


from django.db import models

from django.utils import timezone

from django.conf import settings


class DocumentViewLog(models.Model):

    """

    Logs secure view events of shared documents.

    """

    share = models.ForeignKey('sharing.DocumentShare', on_delete=models.CASCADE, related_name='view_logs')

    user = models.ForeignKey(settings.AUTH_USER_MODEL, null=True, blank=True, on_delete=models.SET_NULL)

    ip_address = models.GenericIPAddressField()

    user_agent = models.TextField(blank=True)

    viewed_at = models.DateTimeField(default=timezone.now)

    viewer_session_id = models.CharField(max_length=64, blank=True)  # Optional: track individual sessions


    def __str__(self):

        return f"{self.share} viewed by {self.user or 'anonymous'} at {self.viewed_at}"
```

Features to Implement in View Layer (not model):

- Watermark overlay (username, IP, timestamp) — done dynamically using HTML/CSS
- Disable right-click, F12, print — via JS
- Block screenshot via CSS + warnings (can't fully prevent)
- Log view events via AJAX to this model (`DocumentViewLog`)

6. auditlog

Purpose: General-purpose object access/activity logging system for tracking who did what and when.

models.py

```
# auditlog/models.py


from django.db import models

from django.contrib.contenttypes.models import ContentType

from django.contrib.contenttypes.fields import GenericForeignKey

from django.conf import settings

from django.utils import timezone




class AuditLogEntry(models.Model):

    """

    Generic audit log entry for tracking access and actions on any object.

    """

    user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.SET_NULL, 
null=True, blank=True)

    action = models.CharField(max_length=64)  # e.g., "viewed", "downloaded", 
"updated", "deleted"


    content_type = models.ForeignKey(ContentType, on_delete=models.CASCADE)

    object_id = models.CharField(max_length=64)  # Support UUIDs or integers

    content_object = GenericForeignKey('content_type', 'object_id')


    timestamp = models.DateTimeField(default=timezone.now)

    ip_address = models.GenericIPAddressField(null=True, blank=True)

    user_agent = models.TextField(blank=True)


    extra_data = models.JSONField(null=True, blank=True)


    class Meta:

        ordering = ['-timestamp']
    def __str__(self):
        return f"{self.user} {self.action} {self.content_type} {self.object_id} at {self.timestamp}"
```

Integration Options

- Can be triggered via:
    - Signals (e.g., post_save, post_delete)
    - Manual calls in views
    - Middleware (for read/view logging)
- Can be used for generating detailed audit reports from admin
- Works for any model due to use of **GenericForeignKey**