

# InvestRand Platform Backend - Comprehensive Code Review

---

**Generated:** 2025-11-17

**Repository:** platform-back-end

**Branch:** main

---

## Table of Contents

1. [Project Overview](#)
  2. [Frameworks & Technologies](#)
  3. [Programming Languages](#)
  4. [Application Structure](#)
  5. [Credentials Found](#)
  6. [Image Upload Implementation](#)
  7. [S3 Upload Mechanism & Authentication](#)
  8. [Deployment Configuration](#)
  9. [Security Recommendations](#)
- 

## Project Overview

**Type:** Django-based REST & GraphQL API for a property investment platform

**Primary Function:** Property listing and investment management system

**Geographic Focus:** South Africa (AWS region: af-south-1)

**Architecture:** Modular Django apps with serverless deployment capability

---

## Frameworks & Technologies

### Core Framework

- **Django 4.2.15** - Main web framework

- **Django REST Framework 3.15.2** - RESTful APIs
- **Graphene 3.3 + Graphene-Django 3.2.2** - GraphQL APIs

## Authentication & Security

- `django-rest-framework-simplejwt` - JWT tokens
- `django-allauth` - Social authentication
- `djoser` - User authentication endpoints
- `django-axes` - Brute-force login protection (5 attempts, 2-hour lockout)

## AWS & Storage

- `boto3 1.35.0` + `botocore 1.35.0` - AWS SDK
- `django-s3-storage 0.15.0` - S3 file storage
- `django-storages 1.14.4` - Cloud storage abstraction

## Financial & Data

- `django-money 3.5.3` - Multi-currency support
- `django-import-export 4.1.1` - Excel/CSV import-export
- `django-countries 7.6.1` - Country fields

## Integrations

- **Active Campaign** ( `activecampaign-python 1.0.10` ) - CRM integration
- **Telegram Bot** ( `python-telegram-bot 21.4` ) - Messaging integration
- **Sentry** ( `sentry-sdk 2.13.0` ) - Error tracking

## Admin & Tools

- `django-material 1.12.0` - Material design admin
- `django-viewflow 2.2.7` - Workflow/BPM
- `drf-spectacular 0.27.2` - OpenAPI/Swagger docs

---

## Programming Languages

- **Python 3.9/3.12** - Primary backend language
- **SQL** - Database queries and migrations

- **Bash/PowerShell** - Build and deployment scripts
  - **YAML** - Docker and CI/CD configuration
  - **JSON** - Configuration files
- 

## Application Structure

### Core Business Apps (10 total)

1. **property** - Real estate listings (1,345 lines in models.py)
  - File: `property/models.py`
  - Manages property listings, statuses, and metadata
  - Active Campaign and Telegram integration
2. **investment** - Investment vehicles and financing (594 lines)
  - File: `investment/models.py`
  - Investment models, rental models, transaction costs, financing
3. **user\_account** - User account management
  - File: `user_account/models.py`
  - Core account functionality
4. **user\_profile** - Extended user profiles
  - File: `user_profile/models.py`
  - Additional user information
5. **service\_providers** - External service provider management
  - File: `service_providers/models.py`
  - Provider information and pictures
6. **rating** - Polymorphic rating system
  - File: `rating/models.py`
  - Flexible rating system for multiple model types
7. **approvals** - Generic approval workflow engine
  - File: `approvals/models.py`
  - Workflow-based approval system
8. **contracts** - Contract/agreement management
  - File: `contracts/models.py`
  - Legal agreements and contracts
9. **notification** - Multi-channel notification system
  - File: `notification/models.py`
  - Event-driven notifications

## 10. **auth\_extension** - Custom authentication enhancements

- File: `auth_extension/models.py`
- Extended authentication functionality

## Infrastructure Apps

- **baseapp** - Main Django project, settings, integrations
  - **permission\_manager** - Permission utilities
  - **events** - Event system framework
  - **job\_manager** - Async job/task management
  - **webhooks** - Webhook handling
- 

## Credentials Found

### CRITICAL - Exposed in Source Code

#### 1. Sentry DSN (Production Error Tracking)

**File:** `baseapp/settings/components/sentry_settings.py:7`

**Value:**

```
dsn="https://dc6338422fe34d9fa9857289eefe6e7a@o4505420590678016.ingest.sentry.io/4505420592644096"
```

**Issue:** Public key exposed, used across dev/uat/prod environments

**Recommendation:** Move to environment variables immediately

---

### HIGH PRIORITY - Test Credentials

#### 2. Test User Passwords (Multiple Files)

**Files:**

- `permission_manager/tests.py:178, 183, 188, 353, 358, 363`
- `baseapp/tests.py:24`

**Username:**

- `test_user@test.com`

- test\_user2@test.com
- test\_user3@test.com

**Password:** "topsecret\_password" (hardcoded in all test cases)

**Issue:** Same password used across all test users

**Recommendation:** Use environment variables or Django test fixtures

---

## MEDIUM PRIORITY - Infrastructure IDs

### 3. AWS Infrastructure (Hardcoded in zappa\_settings.json)

- **AWS Account ID:** 629836046130
- **Certificate ARN:** arn:aws:acm:us-east-1:629836046130:certificate/a2b81380-67d4-4597-883d-32592a24cb0b
- **Subnet IDs:**
  - subnet-0d1dd7887841512ed
  - subnet-075f2f8925ca62306
- **Security Group:** sg-0ad0cad02e74e26b8

**Recommendation:** Move to environment-specific configuration

---

## GOOD PRACTICES - Properly Externalized

The following credentials are **correctly externalized** to baseapp/\_secrets/ :

- Database credentials (dev, uat, prod)
- Email credentials (host, user, password)
- Active Campaign API key
- Telegram bot token
- AWS access keys (AWS\_ACCESS\_KEY\_ID, AWS\_SECRET\_ACCESS\_KEY)

**Protection:** .gitignore correctly excludes \*/ directories containing secrets

**Example from settings:**

```
from baseapp._secrets import _secrets

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': _secrets.SECRETS_PROD_DB_NAME,
        'USER': _secrets.SECRETS_PROD_USER_NAME,
        'PASSWORD': _secrets.SECRETS_PROD_DB_PASSWORD,
        'HOST': _secrets.SECRETS_PROD_DB_HOST,
        'PORT': _secrets.SECRETS_PROD_DB_PORT,
    }
}
```

---

## Image Upload Implementation

### Upload Technology Stack

- **Storage:** AWS S3 (af-south-1 region)
- **Backend:** `django-storages` with `S3Boto3Storage`
- **AWS Client:** `boto3` (v1.35.0)
- **GraphQL Upload:** `graphene-file-upload` (v1.3.0)
- **REST Parsers:** `MultiPartParser` , `FormParser`

### Upload Endpoints

#### 1. Service Provider Pictures

- **Endpoint:** `/api/v1/service-providers/pictures/upload/`
- **ViewSet:** `ServiceProviderPictureViewSet` ( `service_providers/views.py:34-36` )
- **Model:** `ServiceProviderPicture` ( `service_providers/models.py:196-204` )
- **Storage:** `PictureStorage` → S3 bucket with prefix `service_provider_pictures/`

#### 2. Property Artifacts

- **Endpoint:** `/api/v1/property/upload/`
- **ViewSet:** `PropertyArtifactViewSet` ( `property/views.py:48-50` )
- **Model:** `Artifact` ( `property/models.py:998-1031` )
- **Storage:** `ArtifactStorage` → S3 bucket with prefix `property_artifacts/`

- **Types:** Listing photos, financial reports, leases, offers, proof of income, custom

### 3. GraphQL Upload (Incomplete)

- **Mutation:** `uploadFile` ( `property/schema.py:606-617` )
- **Status:** Stub implementation (just prints filename)
- **Issue:** Not production-ready

## Upload Flow

```

Client (multipart/form-data)
  ↓
REST API Endpoint
  ↓
ViewSet (ModelViewSet)
  ↓
Serializer (FileField validation)
  ↓
Model FileField
  ↓
build_filename() → UUID-based filename
  ↓
Custom Storage Class (PictureStorage/ArtifactStorage)
  ↓
AWS S3 (af-south-1)
  ↓
Public URL (public-read ACL)

```

## S3 Configuration

Setting	Value
<b>Region</b>	af-south-1 (South Africa)
<b>ACL</b>	public-read (files publicly accessible)
<b>Dev Bucket</b>	investrand-backend-dev
<b>Prod/UAT Bucket</b>	investrand-backend-prod
<b>Filename Format</b>	{prefix}/{uuid}.{extension}

## Security Concerns with Image Upload

1. **No File Type Validation** - Any file type accepted (potential malware upload)
2. **No File Size Limits** - No max upload size configured (DoS vulnerability)
3. **No Content Inspection** - No MIME type verification
4. **Public S3 Access** - All files set to `public-read`
5. **No Rate Limiting** - Unlimited uploads possible
6. **Incomplete GraphQL Implementation** - Mutation is a placeholder
7. **No User Filtering** - ViewSets lack queryset restrictions
8. **No Virus Scanning** - Files not scanned before storage

## Upload Validation (Current State)

### Serializers:

```
# service_providers/serializers.py:27-32
class ServiceProviderPictureSerializer(serializers.ModelSerializer):
    file = serializers.FileField()
    class Meta:
        model = ServiceProviderPicture
        fields = ['file']
```

### No validation includes:

- File extension checking
  - MIME type verification
  - Image dimension validation
  - File size enforcement
  - Virus/malware scanning
- 

## S3 Upload Mechanism & Authentication

### AWS Authentication Method

The application uses **IAM Access Keys (Access Key ID + Secret Access Key)** for S3 authentication.

### Authentication Configuration

#### Credentials Source:

- Stored in `baseapp/_secrets/_aws_secrets.py` (not in version control)

- Imported in environment settings via:

```
python from baseapp._secrets._aws_secrets import *
```

#### Key Variables (from dev\_local.py:122-124):

```
AWS_S3_ACCESS_KEY_ID = AWS_ACCESS_KEY_ID
AWS_S3_SECRET_ACCESS_KEY = AWS_SECRET_ACCESS_KEY
AWS_S3_REGION_NAME = AWS_REGION
```

These credentials are **externalized** and loaded from the `_secrets` module, which is properly excluded from git via `.gitignore`.

## AWS Profile for Zappa Deployment

For serverless Lambda deployments (zappa\_settings.json:9):

```
"profile_name": "ir"
```

This uses the AWS CLI profile named `"ir"` which references credentials stored in `~/.aws/credentials` on the deployment machine.

---

## S3 Upload Flow - Step by Step

### 1. Client Request

```
POST /api/v1/property/upload/
Content-Type: multipart/form-data

file: [binary data]
listing: [listing_id]
```

### 2. Django Request Processing

#### ViewSet Configuration (property/views.py:48-50):

```
class PropertyArtifactViewSet(ModelViewSet):
    serializer_class = PropertyArtifactSerializer
    parser_classes = (MultiPartParser, FormParser) # Parse multipart data
```

The parsers extract the file from the multipart request.

### 3. Serializer Validation

**PropertyArtifactSerializer (property/serializers.py:27-32):**

```
class PropertyArtifactSerializer(serializers.ModelSerializer):
    file = serializers.FileField()
    class Meta:
        model = Artifact
        fields = ['file', 'listing']
```

The serializer validates the file field (basic validation only - no size/type checks).

### 4. Model Save Process

**Artifact Model (property/models.py:1028-1031):**

```
file = models.FileField(
    storage=ArtifactStorage(),      # Custom S3 storage backend
    upload_to=build_filename        # Filename generator function
)
```

### 5. Filename Generation

**build\_filename function (property/models.py:992-995):**

```
def build_filename(instance, filename):
    file_name = str(instance.id)
    file_extension = filename.split(".")[1]
    return f"property_artifacts/{uuid.uuid4()}.{file_extension}"
```

**Result:** `property_artifacts/550e8400-e29b-41d4-a716-446655440000.jpg`

### 6. Custom Storage Backend

**ArtifactStorage Class (property/storage.py:23-25):**

```
class ArtifactStorage(S3Boto3Storage):
    bucket_name = settings.AWS_S3_BUCKET_NAME
    location = 'property_artifacts'
```

**Inheritance:** Extends `storages.backends.s3boto3.S3Boto3Storage`

### 7. S3Boto3Storage Upload Process

The `django-storages` library handles the actual S3 upload:

## What happens internally:

1. **Creates boto3 S3 client** using credentials:

```
python import boto3 s3_client = boto3.client( 's3',  
aws_access_key_id=settings.AWS_S3_ACCESS_KEY_ID,  
aws_secret_access_key=settings.AWS_S3_SECRET_ACCESS_KEY, region_name='af-  
south-1' )
```

2. **Uploads file to S3:**

```
python s3_client.put_object( Bucket='investrand-backend-prod',  
Key='property_artifacts/550e8400-e29b-41d4-a716-446655440000.jpg',  
Body=file_content, ACL='public-read' )
```

3. **Returns file URL:**

```
https://investrand-backend-prod.s3.af-south-1.amazonaws.com/property_artifacts/  
550e8400-e29b-41d4-a716-446655440000.jpg
```

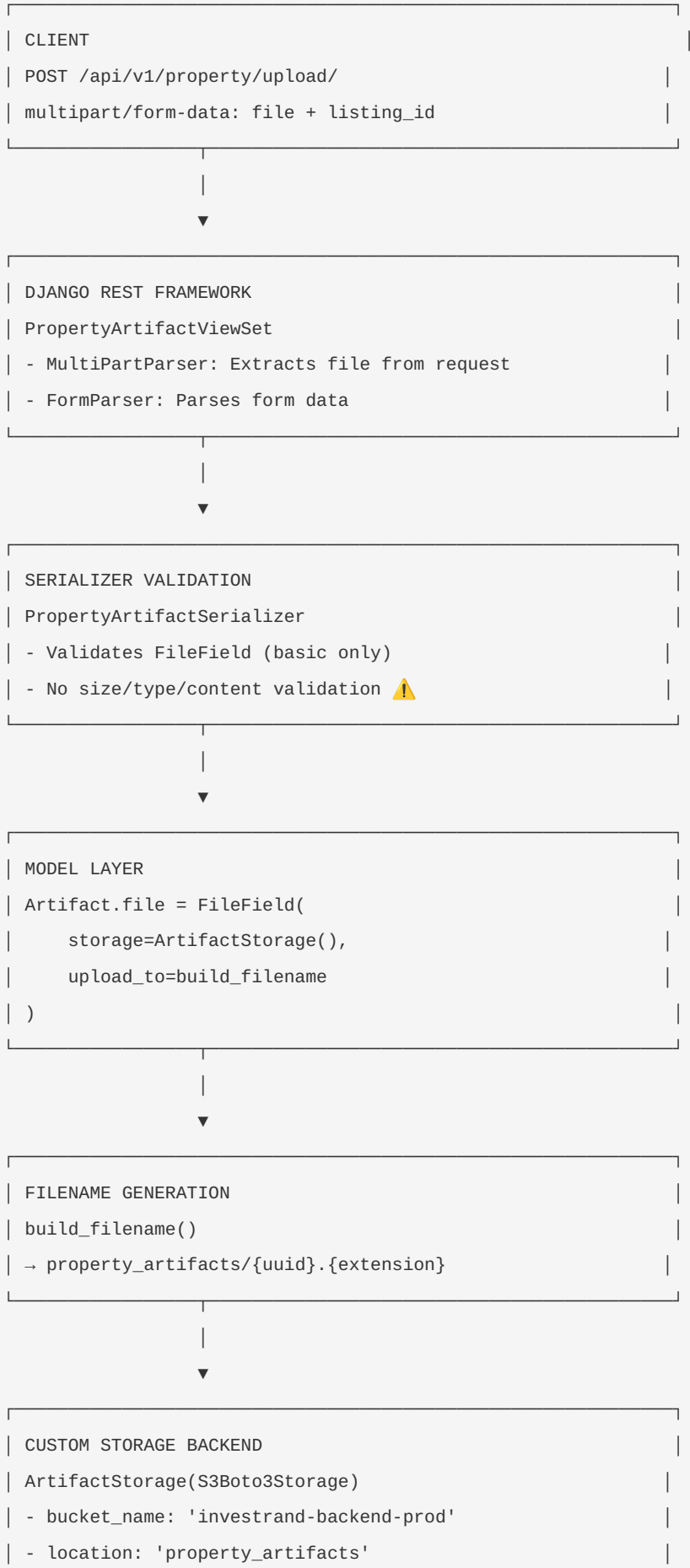
## 8. Database Record Creation

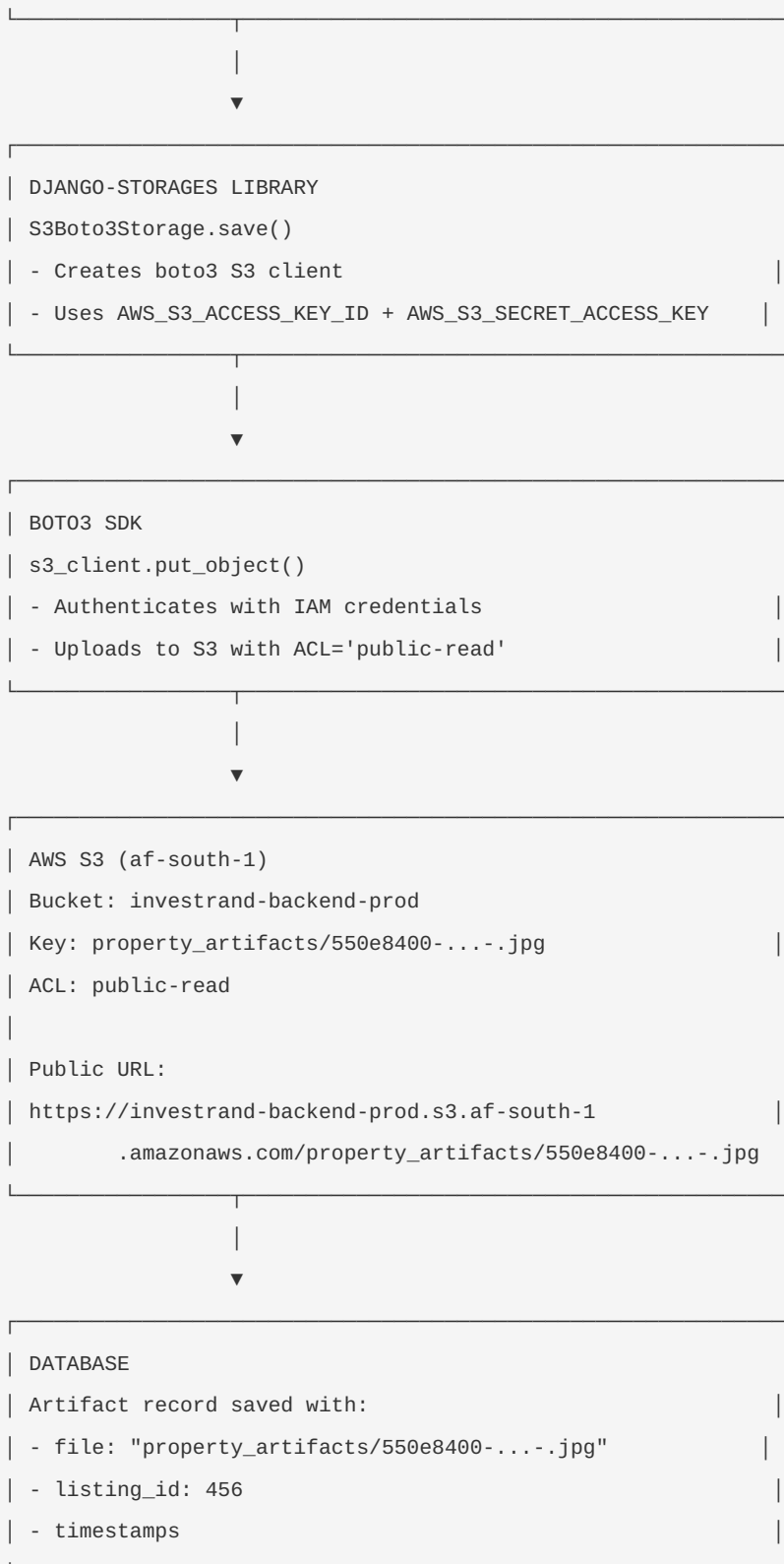
The `Artifact` model saves metadata to the database:

```
{  
  "id": 123,  
  "file": "property_artifacts/550e8400-e29b-41d4-a716-446655440000.jpg",  
  "listing_id": 456,  
  "created": "2025-11-16T10:30:00Z",  
  "modified": "2025-11-16T10:30:00Z"  
}
```

---

## Complete Upload Architecture Diagram





# Deployment Configuration

## Environments

1. **Local Development** ( `dev_local` ) - SQLite database
2. **Development** ( `dev` ) - AWS Lambda + PostgreSQL/MySQL
3. **UAT** - User Acceptance Testing
4. **Production** ( `prod` ) - AWS Lambda + PostgreSQL/MySQL

## Deployment Methods

- **Serverless:** AWS Lambda via Zappa (Python 3.9, 300s timeout)
- **Containerized:** Docker with Nginx + Gunicorn
- **Reverse Proxy:** Traefik with Let's Encrypt SSL

## Domains

- **Prod:** `api.platform.investrand.co.za`
- **UAT:** `uat-api-platform.investrand.co.za`
- **Dev:** `dev-api-platform.investrand.co.za`

## Database

- **Default:** SQLite (development)
- **Production:** PostgreSQL or MySQL
- **ORM:** Django ORM with migrations
- **Audit Logging:** `django-easy-audit` tracks all model changes
- **Multi-currency:** MoneyField support

## API Architecture

### REST API ( `/api/v1/` )

- DRF with JWT authentication
- OpenAPI/Swagger docs: `/api/schema/swagger-ui/`
- ReDoc: `/api/schema/redoc/`

### GraphQL API

- Authenticated: `/graphql`

- Public: `/u-gql`
- 10 GraphQL-enabled apps
- GraphiQL disabled in production

## Codebase Statistics

- **Total Python files:** 281
  - **Largest model file:** `property/models.py` (1,345 lines)
  - **Direct dependencies:** 99 packages in requirements.txt
- 

## Security Recommendations

### Immediate Actions Required

#### 1. Remove Sentry DSN from Source Code

**File:** `baseapp/settings/components/sentry_settings.py:7`

**Current:**

```
dsn="https://dc6338422fe34d9fa9857289eefe6e7a@o4505420590678016.ingest.sentry.io/4505420592644096"
```

**Fix:**

```
from baseapp._secrets import _secrets

dsn=_secrets.SENTRY_DSN
```

#### 2. Add Upload File Validation

**Add to serializers:**

```

from django.core.exceptions import ValidationError

ALLOWED_EXTENSIONS = ['.jpg', '.jpeg', '.png', '.pdf']
ALLOWED_MIME_TYPES = ['image/jpeg', 'image/png', 'application/pdf']
MAX_UPLOAD_SIZE = 5242880 # 5MB

class PropertyArtifactSerializer(serializers.ModelSerializer):
    file = serializers.FileField()

    def validate_file(self, value):
        # Check file size
        if value.size > MAX_UPLOAD_SIZE:
            raise ValidationError(f"File size must not exceed 5MB")

        # Check file extension
        ext = os.path.splitext(value.name)[1].lower()
        if ext not in ALLOWED_EXTENSIONS:
            raise ValidationError(f"File extension {ext} not allowed")

        # Check MIME type
        if value.content_type not in ALLOWED_MIME_TYPES:
            raise ValidationError(f"File type {value.content_type} not allowed")

        return value

    class Meta:
        model = Artifact
        fields = ['file', 'listing']

```

#### Add to settings:

```

DATA_UPLOAD_MAX_MEMORY_SIZE = 5242880 # 5MB
FILE_UPLOAD_MAX_MEMORY_SIZE = 5242880 # 5MB

```

### 3. Implement Rate Limiting

#### Install:

```

pip install django-ratelimit

```

#### Add to views:

```

from django_ratelimit.decorators import ratelimit
from django.utils.decorators import method_decorator

@method_decorator(ratelimit(key='user', rate='10/h', method='POST'), name='create')
class PropertyArtifactViewSet(ModelViewSet):
    serializer_class = PropertyArtifactSerializer
    parser_classes = (MultiPartParser, FormParser)

```

## 4. Review S3 ACL Settings

### Consider Private Buckets with Pre-Signed URLs:

Update `baseapp/settings/components/aws_settings.py` :

```

AWS_DEFAULT_ACL = None # Don't make files public
AWS_QUERYSTRING_AUTH = True # Use pre-signed URLs
AWS_QUERYSTRING_EXPIRE = 3600 # 1 hour expiry

```

This makes files private and generates temporary URLs for access.

## High Priority Actions

## 5. Use IAM Roles Instead of Access Keys (Lambda)

For Lambda deployments, use execution roles instead of storing credentials:

### Update `zappa_settings.json`:

```

{
  "prod": {
    "aws_region": "af-south-1",
    "role_name": "InvestRandLambdaExecutionRole",
    // Remove AWS credentials - use role instead
  }
}

```

### Create IAM Role with S3 permissions:

- AWSLambdaVPCLambdaAccessExecutionRole
- S3FullAccess (or custom policy with specific bucket access)

## 6. Externalize AWS Infrastructure IDs

Move from `zappa_settings.json` to environment-specific config:

**Create:** `baseapp/settings/environments/zappa_config.py`

```
from baseapp._secrets import _secrets

ZAPPA_VPC_CONFIG = {
    "SubnetIds": _secrets.AWS_SUBNET_IDS,
    "SecurityGroupIds": _secrets.AWS_SECURITY_GROUP_IDS
}

CERTIFICATE_ARN = _secrets.AWS_CERTIFICATE_ARN
```

## 7. Fix Test User Passwords

**Replace hardcoded passwords with Django's user creation:**

```
from django.contrib.auth.models import User

# Instead of:
self.user1 = User.objects.create(
    username="test_user@test.com",
    password="topsecret_password" # ❌ BAD
)

# Use:
self.user1 = User.objects.create_user(
    username="test_user@test.com",
    password=os.environ.get('TEST_USER_PASSWORD', 'test_password_123') # ✅ GOOD
)
```

---

## Medium Priority Actions

### 8. Add Virus Scanning

**Option 1: ClamAV Integration**

```
import pyclamd

def scan_file_for_viruses(file_obj):
    cd = pyclamd.ClamdUnixSocket()
    scan_result = cd.scan_stream(file_obj.read())
    file_obj.seek(0) # Reset file pointer

    if scan_result:
        raise ValidationError("File contains malware")
```

## Option 2: AWS GuardDuty or S3 Malware Scanning

## 9. Implement S3 Bucket Policy

Restrict bucket access by IP or VPC:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Principal": "*",
      "Action": "s3:*",
      "Resource": "arn:aws:s3:::investrand-backend-prod/*",
      "Condition": {
        "StringNotEquals": {
          "aws:SourceVpc": "vpc-xxxxxxx"
        }
      }
    }
  ]
}
```

## 10. Add User-Specific Upload Filtering

Update ViewSets to filter by user:

```
class PropertyArtifactViewSet(ModelViewSet):
    serializer_class = PropertyArtifactSerializer
    parser_classes = (MultiPartParser, FormParser)

    def get_queryset(self):
        # Only show user's own uploads
        return Artifact.objects.filter(listing__created_by=self.request.user)

    def perform_create(self, serializer):
        # Track who uploaded the file
        serializer.save(uploaded_by=self.request.user)
```

## 11. Complete or Remove GraphQL Upload Mutation

**File:** `property/schema.py:606-617`

**Current (Stub):**

```
def mutate(self, info, file, **kwargs):
    print(file) # ❌ Not production-ready
    return UploadMutation(success=True)
```

**Either:**

- Implement properly with file handling
- Remove if not needed

---

## Optional Enhancements

### 12. Add Image Processing

**For image uploads, add validation and processing:**

```
pip install Pillow
```

```

from PIL import Image
from io import BytesIO

def validate_and_process_image(file_obj):
    try:
        image = Image.open(file_obj)

        # Validate image
        if image.width > 4096 or image.height > 4096:
            raise ValidationError("Image dimensions too large")

        # Convert to RGB (remove alpha channel)
        if image.mode in ('RGBA', 'LA', 'P'):
            image = image.convert('RGB')

        # Resize if needed
        max_size = (1920, 1920)
        image.thumbnail(max_size, Image.Resampling.LANCZOS)

        # Save optimized version
        output = BytesIO()
        image.save(output, format='JPEG', quality=85, optimize=True)
        output.seek(0)

        return output
    except Exception as e:
        raise ValidationError(f"Invalid image: {str(e)}")

```

### 13. Add Secret Scanning to CI/CD

#### Install pre-commit hook:

```
pip install detect-secrets
```

#### Add to `.pre-commit-config.yaml` :

```

repos:
  - repo: https://github.com/Yelp/detect-secrets
    rev: v1.4.0
    hooks:
      - id: detect-secrets
        args: ['--baseline', '.secrets.baseline']

```

---

## Key Files Reference

### AWS & Upload Files

Component	File Path	Lines
AWS Settings	<code>baseapp/settings/components/aws_settings.py</code>	1-4
AWS Credentials (externalized)	<code>baseapp/_secrets/_aws_secrets.py</code>	N/A
Dev Local Config	<code>baseapp/settings/environments/dev_local.py</code>	122-124
Prod Config	<code>baseapp/settings/environments/prod.py</code>	58-62
ArtifactStorage	<code>property/storage.py</code>	23-25
PictureStorage	<code>service_providers/storage.py</code>	23-25
Artifact Model	<code>property/models.py</code>	1028-1031
PropertyArtifactViewSet	<code>property/views.py</code>	48-50
ServiceProviderPictureViewSet	<code>service_providers/views.py</code>	34-36

## Settings Files

File	Purpose
<code>baseapp/settings/components/base.py</code>	Core Django settings
<code>baseapp/settings/components/database.py</code>	Database configuration
<code>baseapp/settings/components/security_settings.py</code>	Auth, CORS, CSRF, passwords
<code>baseapp/settings/components/apis.py</code>	REST & GraphQL config
<code>baseapp/settings/components/middlewares.py</code>	Middleware stack
<code>baseapp/settings/components/email.py</code>	Email configuration
<code>baseapp/settings/components/sentry_settings.py</code>	Error tracking
<code>baseapp/settings/environments/dev_local.py</code>	Local development
<code>baseapp/settings/environments/dev.py</code>	Development environment
<code>baseapp/settings/environments/uat.py</code>	UAT environment
<code>baseapp/settings/environments/prod.py</code>	Production environment

## Summary

### Strengths

- ✓ Clean modular architecture with domain-specific apps
- ✓ Comprehensive security (JWT auth, CSRF, XSS, CORS, rate limiting)
- ✓ Secrets properly externalized to `_secrets/` module
- ✓ Multi-environment deployment strategy
- ✓ Full audit trail with django-easy-audit
- ✓ Both REST and GraphQL API support
- ✓ CI/CD pipeline configured
- ✓ Docker containerization
- ✓ Serverless deployment option

## Critical Issues

- ✗ Sentry DSN exposed in source code
- ✗ No file upload validation (type, size, content)
- ✗ Public S3 buckets with `public-read` ACL
- ✗ No virus scanning on uploads
- ✗ Test passwords hardcoded
- ✗ GraphQL upload mutation incomplete