**Tech Stack Document (Backend)**

**1. Overview**

* **Project Name**: Trace Direct Scam Recovery
* **Description**: Django REST Framework-based application for scam detection, investigation, and reporting.
* **Primary Framework**: Django 4.x + DRF

**2. Backend**

* **Language**: Python 3.10+
* **Framework**: Django 4.x
* **API Framework**: Django REST Framework (DRF)
* **Authentication**: Token Authentication (DRF AuthToken)
* **Schema/Docs**: DRF Spectacular (OpenAPI/Swagger)

**3. Database**

* **Primary Database**: SQLite3 (development default)
* **ORM**: Django ORM
* **Migrations**: Django makemigrations + migrate

**4. Third-Party APIs & Services**

| **Service** | **Description** | **Behaviour** |
| --- | --- | --- |
| **Shodan** | Search engine for internet-connected devices. Returns open ports, services, and vulnerabilities for IP/domain. | Input: IP/Domain Output: JSON with ports, services, vulnerabilities |
| **LeakCheck** | Checks if a user’s email has been exposed in a data breach. | Input: Email Output: Breach list (site, date, compromised data) |
| **Reverse Image Search** | Detects if a scammer’s photo is reused across multiple websites/accounts. | Input: Uploaded Image.  Output: Screenshot grid + URLs |
| **Telegram Profile Lookup (Telethon + Apify)** | Scrapes Telegram profiles linked to usernames or phone numbers. | Input: Telegram username/phone Output: Name, bio, profile pic, last seen |
| **PDF Report Generation** | Generates a professional PDF report from investigation results. | Input: Case data Output: Styled PDF report |
| **Suspicious URL Check (Apivoid)** | Checks suspected URLs for scam details. | Input: URL Output: Scam details (name, email, metadata) |
| **OpenAI**  **Upstage** | Summarizes investigation results for final report generation.  Extract image or pdf text | Input: Investigation JSON  Output: Concise natural language summary  Input: image/pdf Output: text |

**5. Infrastructure**

* **Web Server**: Django’s ASGI (with Channels for async support)
* **Task Queue**: Celery + Redis (planned for async jobs like API calls, PDF generation)
* **Environment Management**: .env files (dotenv)
* **CORS & CSRF**: django-cors-headers configured for frontend integration

**6. DevOps**

* **Containerization**: Docker (planned)
* **CI/CD**: GitHub Actions (planned)
* **Email Service**: Gmail SMTP (development)

**7. Testing**

* **Framework**: Django Test Framework
* **Focus**: Unit tests for API endpoints (Shodan, Leakcheck, Apivoid, OpenAI integrations)

**8. Deployment**

* **Default Setup**: Local with SQLite3
* **Planned Cloud Options**: AWS/GCP/Azure
* **Static & Media Files**: Local storage (dev), S3 integration (prod-ready config in settings.py)

**9. Security**

* HTTPS with SSL (enforced in production)
* CSRF + CORS protections enabled
* Environment variables for all sensitive API keys (Shodan, Apivoid, OpenAI, etc.)
* Role-based access via DRF permissions

**10. Workflow Example**

1. User submits IP, email, image, or Telegram username.
2. DRF backend calls external API (Shodan, LeakCheck, Apivoid, etc.).
3. Results stored in SQLite3.
4. OpenAI generates a human-readable summary.
5. PDF report is created and returned to the user.

**11. Architecture**

