Simple Currency Exchange Service (CES) - Python Django & Celery

1. Project Overview and Value Proposition

The **Simple Currency Exchange Service (CES)** is a high-performance, financially compliant backend application built using **Python, Django, and Celery**. It serves as the single source of truth for foreign exchange (FX) rates, prioritizing auditability and read performance.
This project is a high-impact portfolio piece, demonstrating expertise in:

- **Asynchronous Resilience:** Using **Celery Beat and Workers** for guaranteed, scheduled data ingestion and retries.
- **Performance Optimization:** Implementing a full **Redis caching backend** for sub-10ms latency on core rate lookups.
- **Financial Integrity:** Designing an **immutable data model** for rates and a strict ConversionAudit log, critical for financial reconciliation.
- **Rapid Administration:** Leveraging the **Django Admin** for immediate visibility and management of all FX data.

2. Core Architectural Components

Component	Technology	Role
Web Framework	Python / **Django**	Handles synchronous API endpoints, ORM, and database transactions.
Scheduled Tasks	**Celery Beat**	The scheduler that triggers rate ingestion tasks (e.g., hourly).
Task Runner	**Celery Worker**	Executes the heavy-lifting, external API calls, and handles retries.
Database	PostgreSQL	Persistent storage for historical rates and immutable audit logs.
Caching/Broker	**Redis**	Used both as the **Cache Backend** (low-latency read) and the **Celery Broker** (task queue management).
API	Django Rest Framework (DRF)	Used for building clean, professional RESTful endpoints.

3. Financial Integrity & Performance Principles

3.1 Immutable Rate Management

The ExchangeRate model is designed to be history-preserving. Instead of updating a rate, the scheduled task always **inserts a new record**. This historical record is then linked via a **Foreign Key** to every conversion logged in the ConversionAudit table, providing a non-repudiable audit trail.

3.2 High-Availability Ingestion

The ingestion process is decoupled from the web application via Celery. If the external FX API endpoint returns an error:

1. The Celery task fails and automatically **retries** with an exponential backoff.
2. The application **continues serving the last known good rate** from Redis. The API never suffers downtime due to external provider failure.

3.3 Low-Latency Read

All rate lookups by the Conversion API are directed to the **Redis Cache** using Django's built-in cache.get() API, ensuring maximum read speed and minimizing load on the PostgreSQL database.

4. Local Development and Setup

4.1 Prerequisites

- Python 3.10+
- Docker (Recommended for PostgreSQL and Redis)

4.2 Infrastructure Setup (Docker)

You must run a PostgreSQL database for persistence and a Redis instance to serve as both the Celery Broker and the Caching Backend.

```
# Start PostgreSQL (for DB persistence)
docker run --name ces-postgres -e POSTGRES_USER=cesuser -e
POSTGRES_PASSWORD=cespass -e POSTGRES_DB=cesdb -p 5432:5432 -d postgres

# Start Redis (for Cache & Celery Broker)
docker run --name ces-redis -p 6379:6379 -d redis redis-server --appendonly yes
```

4.3 Environment Configuration

Critical settings are loaded via environment variables (e.g., using django-environ).

Variable Name	Description	Rationale
DATABASE_URL	PostgreSQL connection string.	Standard practice for portability.
REDIS_URL	Redis connection string (for broker/cache).	Required for Celery/Caching.
FX_API_KEY	API key for the external FX provider.	Security: Must be kept out of code.
CONVERSION_MARGIN	Decimal margin applied to the rate (e.g., 0.005).	Business logic parameter.

4.4 Running the Application

1. **Clone and Setup:**
   git clone [your-repository-url]
   cd currency-exchange-service
   pip install -r requirements.txt
   python manage.py migrate
   python manage.py createsuperuser # For accessing Django Admin

2. **Start Django Server:**
   python manage.py runserver
   # Server running on http://localhost:8000
   # Django Admin available at http://localhost:8000/admin/

3. **Start Celery Worker (Task Execution):**
   celery -A your_project_name worker -l info

4. **Start Celery Beat (Scheduler):**
   celery -A your_project_name beat -l info --scheduler
   django_celery_beat.schedulers:DatabaseScheduler

   (Note: Using django-celery-beat allows you to manage schedules directly via the Django Admin.)

5. Key API Endpoints (DRF)

5.1 GET /api/v1/rates

Purpose: Retrieves the current exchange rate, optimized for speed by querying Redis first.
Example: GET /api/v1/rates?from=USD&to=NGN

5.2 POST /api/v1/convert

Purpose: Executes conversion, applies margin, and creates an immutable audit record.

Request Body (JSON):

```
{
  "amount": 500.00,
  "from_currency": "USD",
  "to_currency": "NGN"
}
```