Currency Converter API

A FastAPI-based currency conversion microservice built following the 12-factor app methodology.

Github Repo: https://github.com/srmaarnav/12-factor-app

Features

- Real-time currency conversion with caching
- RESTful API endpoints
- Redis caching for improved performance
- Docker containerization
- Comprehensive documentation
- Testing support

Quick Start

Using Docker

1. Clone the repository:

   ```
   git clone https://github.com/srmaarnav/12-factor-app.git
   cd 12-factor-app
   ```

2. Set up environment variables:

   ```
   cp .env.example .env
   # Edit .env with your configuration
   ```

3. Build and run with Docker Compose:

   ```
   docker compose up --build
   ```

The services will be available at:

- API: http://localhost:8000
- Documentation: http://localhost:8001
- API Swagger UI: http://localhost:8000/docs
- API ReDoc: http://localhost:8000/redoc

Local Development

1. Create and activate virtual environment:

```
python3 -m venv .venv
source .venv/bin/activate  # On Windows: .venv\Scripts\activate
```

2. Install dependencies:

```
pip install -r requirements.txt
```

3. Run the application:

```
uvicorn app.main:app --reload
```

API Documentation

Once running, access the API documentation at:

- Swagger UI: http://localhost:8000/docs
- ReDoc: http://localhost:8000/redoc

Main Endpoints

- GET /convert: Convert between currencies
  - Query parameters:
    - from_currency: Source currency code (e.g., USD)

- to_currency: Target currency code (e.g., EUR)

- amount: Amount to convert

Project Structure

12-factor-app/

--- app/

--- __init__.py

--- api.py # API routes

--- config.py # Configuration management

--- main.py # Application entry point

--- services.py # Business logic

--- docs/ # Documentation

--- tests/ # Test suite

--- conftest.py

--- pytest.ini

--- test_api.py

--- .env.example # Example environment variables

--- .gitignore

--- docker-compose.yml

--- Dockerfile

--- mkdocs.yml # Documentation config

--- requirements.txt

12 Factor Implementation

1. Codebase: One codebase tracked in Git

2. Dependencies: Explicitly declared in requirements.txt

3. Config: Stored in environment variables

4. Backing Services: Redis treated as attached resource

5. Build, Release, Run: Clearly separated stages in deployment

6. Processes: Stateless application with Redis for state

7. Port Binding: Self-contained with port configuration

8. Concurrency: Horizontally scalable

9. Disposability: Fast startup/shutdown

10. Dev/Prod Parity: Docker ensures environment consistency

11. Logs: Treated as event streams

12. Admin Processes: One-off admin tasks as scripts

Development

Prerequisites

- Python 3.10+

- Docker and Docker Compose

- Redis

Testing

Run tests with:

```
pytest tests/
```

Documentation

Documentation is available in two ways:

1. Live Documentation (when running with Docker):

   Visit http://localhost:8001

2. Local Development:

```
mkdocs build

mkdocs serve  # Serves at http://localhost:8000
```