Stock Chat Assistant Django App Documentation

1. Introduction

This document outlines the design and architecture of the Stock Chat Assistant built using Django. The project enables investors and analysts to interact through a stock-related chatbot, manage portfolios, and receive real-time or historical stock data.

2. Project Overview

This section provides a detailed explanation of the `core` Django app. The app facilitates a stock portfolio management system where users are either **Analysts** or **Investors**. Analysts can assign recommendations to investors, and investors can maintain their portfolios and receive suggestions.

3. Approach

My approach was systematic and thorough, focusing on:

- Reviewing the structure and contents of the Django project.
- Examining key files and templates related to authentication, dashboards, and chatbot functionality.
- Searching for the use of external APIs (such as yfinance) and understanding backend-to-frontend data flow.
- Explaining Django components like views.py, models.py, admin.py, and templates.
- Providing clear and tailored explanations based on the queries.

4. Methodology

- **Codebase Exploration:** Examined key Django files and templates.
- **Targeted Search:** Used keyword searches (e.g., yfinance, chat_api) to locate features or integrations.
- **Functional Mapping:** Mapped how user requests flow through URLs, views, templates, and models.
- **Explanation and Synthesis:** Explained how each file fits into Django's modular system.

5. Models

`Portfolio`

Tracks an investor's stock holdings.

```
class Portfolio(models.Model):
    investor = models.ForeignKey(User, on_delete=models.CASCADE,
related_name='portfolios')
    ticker = models.CharField(max_length=16)
    quantity = models.PositiveIntegerField()
    avg_buy_price = models.FloatField(default=0)
```

`Recommendation`

Represents an analyst's suggestion for an investor regarding a specific stock.

```
class Recommendation(models.Model):
    analyst = models.ForeignKey(User, on_delete=models.CASCADE,
related_name='recommendations')
    investor = models.ForeignKey(User, on_delete=models.CASCADE,
related_name='recommended_to')
    ticker = models.CharField(max_length=16)
    note = models.TextField(blank=True)
    created_at = models.DateTimeField(auto_now_add=True)
```

`AnalystInvestorAssignment`

Creates a link between analysts and their assigned investors.

```
class AnalystInvestorAssignment(models.Model):
    analyst = models.ForeignKey(User, on_delete=models.CASCADE,
related_name='assigned_investors')
    investor = models.ForeignKey(User, on_delete=models.CASCADE,
related_name='assigned_analyst')
```

6. Forms

`SignUpForm`

Custom user registration form with role selection.

```
class SignUpForm(UserCreationForm):
    role = forms.ChoiceField(choices=(('analyst', 'Analyst'), ('investor',
'Investor')), widget=forms.RadioSelect)
```

`RecommendationForm`

Used by analysts to provide stock suggestions.

```
class RecommendationForm(ModelForm):
    class Meta:
        model = Recommendation
        fields = ['ticker', 'note']
```

`BuyStockForm`

Used by investors to simulate stock purchases.

```
class BuyStockForm(forms.Form):
    ticker = forms.CharField(label='Stock Ticker', max_length=16)
    quantity = forms.IntegerField(label='Quantity', min_value=1)
    price = forms.FloatField(label='Buy Price', min_value=0)
```

7. Views

`signup(request)`

Handles new user registration with role-based group assignment.

`dashboard_redirect(request)`

Redirects user to the appropriate dashboard based on group.

`analyst_dashboard(request)`

Allows an analyst to:

- View assigned investors and their portfolios
- Submit stock recommendations
- View previously made recommendations

`investor_dashboard(request)`

Enables an investor to:

- View their portfolio
- Receive recommendations from their assigned analyst
- Buy new stocks using the `BuyStockForm`

`logout_view(request)`

Logs the user out and redirects to login page.

`chat_api(request)`

Enables chat functionality with AI-driven stock analysis:

- Detects ticker in user input

- Fetches stock data and graph
- Provides advice using a custom AI assistant

8. Templates

- `signup.html`: Form for user registration
- `analyst_dashboard.html`: Dashboard for analysts
- `investor_dashboard.html`: Dashboard for investors
- `chat_box.html` & `chat_history.html`: Interfaces for AI chat

9. Findings

- Project uses Django's standard MVC structure.
- User authentication implemented with role-based navigation (Investor/Analyst).
- Chatbot UI is built with HTMX for dynamic responses using chat_box.html.
- Backend logic in chat_api handles parsing, advice generation, and session chat storage.
- yfinance is installed but not yet integrated.
- Database schema managed via Django migrations.

10. Assumptions

- **App is designed for investor-analyst interactions and portfolio management.**
- **Two user roles (Investor and Analyst) are assumed.**
- **Chatbot intended to fetch stock data and answer queries.**

11. Tech Stack & Integration

- **Django** (core framework)
- **Django Auth Groups** (role management)
- **Django Forms** (user input)
- **Matplotlib/Pandas** (for `fetch_and_plot_stock`)
- **Custom AI integration** (`get_chat_advice`)
- **HTMX** (for dynamic chat UI)

12. Advantages

- Django: Fast development, admin interface, authentication, ORM.
- HTMX: Enables dynamic content without full page reload.
- yfinance: Free and easy access to market data.

13. Alternatives

- **Frameworks:** Flask (lightweight), FastAPI (async), Node.js (JavaScript).
- **Frontend:** React, Vue, or Django Channels for real-time updates.

- **Data Sources:** Alpha Vantage, IEX Cloud for stock data.
- **Chatbot:** Use GPT (OpenAI), Rasa, or Botpress for more intelligent responses.

  Although I don't have expertise in this + openAI API and key are not free that's why I went with this :}

14. Summary Table

Aspect	Current Approach	Advantages	Alternatives	Trade-offs
Framework	Django	Fast, secure, built-in features	Flask, FastAPI, Node.js	Simpler, but needs more config
Chat Frontend	HTMX + Django	Dynamic, easy setup	React, Vue, Channels	Better UX, but complex
Stock Data	Planned yfinance	Free, easy to use	Alpha Vantage, IEX	API limits, may need keys
Chatbot Logic	Custom Python	Simple, controllable	LLMs, Rasa, Botpress	More powerful, more setup

15. Recommendations

- Continue using Django unless real-time interaction is needed.
- Integrate yfinance for stock data access.
- Explore Alpha Vantage/IEX for more stable APIs.
- Consider Django Channels for real-time chat.
- Upgrade chatbot using GPT or similar LLMs.
- Analyst can also recommend new stocks and suggest selling existing ones.
- Analyst can advise investor based on current holdings.
- Investor is assumed to have an active stock portfolio.
