Stock Market Prediction Project Documentation

Developer: **Leonce Oswald Jesugnon C. AGBODJALOU**

Summary

1. Introduction

This project is a stock market price prediction application that uses a combination of Django (Backend) and React (Frontend) and Machine Learning (Neural LSTM). Users can enter a ticker, a time period, and get market predictions based on indicators such as moving averages and the neural LSTM model.

The app retrieves market data via a quotation API and performs a series of calculations to generate forecasts and charts, displaying everything dynamically in the user interface.

2. **Project Architecture**

Backend : Django

- Django handles REST APIs, authentication, prediction logic, and user management.

- Prediction data is stored in a SQLite PostgreSQL database, while the generated images and graphs are saved in a specific directory.

Frontend : React

- React allows you to manage user input forms and display predictions.

- React components are dynamic and interact with the endpoints APIs provided by Django to retrieve and display information.

Prediction: Algorithm and Machine Learning

- Collection, extraction, processing and structuring of monetary data.

3. **Main Components**

Backend (Django)

- **Django REST Framework** : Provides endpoints for CRUD operations of predictions.

- **Django ORM** : Used to interact with PostgreSQL for users and prediction data.

- **Prediction Algorithm** : Implemented in Python, this module collects, extracts, processes, structures monetary data, calculates moving averages and other indicators.

Frontend (React)

- **Forms** : Ticker and period entry form, with validation to force uppercase and allow only valid formats.

- **Prediction Display** : React components for the dynamic display of charts and moving average values.

Prediction (Machine Learning)

Via a robust and independent algorithm, we retrieve data from the stock markets via an **API**. This allows us to filter this data and submit it to the LSTM model previously implemented thanks to

Jupiter, **Pandas**, **Numpy**. This prediction is finally made possible with Tensorflow, which accurately generates expected prices with data it did not know.

4. **Model Management and Database**

User Model

- **Custom Model** : Email and password-based authentication, managed by 'UserManager'.

- **Authentication endpoints** : Registration, login, and session management are accessible via the API.

Model for Prediction Data

- Stores ticker information, period start and end dates, and prediction results.

- Links to generated images are also stored, with a unique numbering system for each image.

5**. APIs and External Services**

The API uses a quotation service to retrieve market data via an independent algorithm. The 'APIService' class handles external calls with error handling for invalid status codes.

Core REST Endpoints

- **'POST /api/v1/activitie/':** Choice of user profile

- '**POST /api/ v1/predict/':** Takes the ticker, dates and makes the prediction.

- **'GET /api/ v1/user/':** Retrieves the authenticated user's information.

- **'POST /api/ v1/register/'** and **'POST /api/ v1/login/':** User authentication management.

6. **Key Features**

Market Prediction

The prediction algorithm is based on moving averages (MAs) with configurable intervals and the LSTM model.

Image Management

Graphics are saved with a unique name, such as 'EURUSD1.png', and dynamic URLs are sent to the frontend for display.

Dynamic Validation

The React form forces the uppercase character to ticker it while allowing the '/' characters.

7. **Implementation of the Prediction Algorithm**

The algorithm for calculating moving averages uses a 'MovingAverageCalculator' class:

- **Attributes** : List of data ('rates') and calculation interval.

- **Main method** : 'compute_moving_average()' calculates the moving averages for each data point.

- **Output** : Returns a dictionary with dates and average values.

- **Jupyter**: Conda's environment for preparing the prediction model

- **Pandas**: recovery of the failures processed by filtering by DataFrame

- **Numpy**: Converts rates to digitized values (0, 1)

- **Tensorflow**: Final prediction.

8. **Image and Graphics Management**

The images of the predictions are generated with 'matplotlib' and saved in a media folder. They are uniquely numbered.

'save_plot' function

- This function receives the file name and saves the graphic.

- Image URLs are dynamically generated and stored in the database.

9. **Validation and Errors**

- **Ticker validation** : If an incorrect ticker is entered, the API returns an error message.

- **API Calls limit** : Management is in place to limit the number of calls to the external API and avoid quota errors.

Error Handling in 'StockPredictionAPIView'

API errors are intercepted, and clear messages are sent to the frontend. **Example** :

'''Python

return Response({"error": "Ticker invalid", "status": status. HTTP_404_NOT_FOUND})

```

10. **Deployment**

To deploy the app:

1. **Configurations** :
   - Configure environment variables for API keys and database information.
   - Configure 'settings.py' for production mode with 'DEBUG = False'.
2. **Web server** : Use 'gunicorn' to serve Django, and set up an NGINX server as a front-end.
3. **File Server** : Set up a file store (such as AWS S3) to host the generated graphics.

11. **Best Practices and Security**

- Secure Password Storage: Passwords are hashed.
- Error Handling: All API errors are intercepted and logged.
- Access Controls: Only authenticated users can access predictions.

This project is now fully functional and production-ready, offering a smooth user experience, secure data management, and reliable stock market price prediction.
```