Commodity Price Prediction

Software Design Specifications

By

Owais Aijaz (oa07610@st.habib.edu.pk)
Sidra Aamir (sa07316@st.habib.edu.pk)
Aiza Imran (ai07597@st.habib.edu.pk)
Syed Hamza (hs07141@st.habib.edu.pk)

December 19, 2024

In partial fulfillment of the requirement for

Bachelor of Science
Computer Science

Dhanani School of Science and Engineering

Habib University

Fall 2024

1 Introduction

This document outlines the Software design based on the current baseline implementation and goals. This document details the architecture and design elements that form our software solution. It comprises very detailed sections on software design, including UML class diagrams, data design, data sources, preprocessing, and pipelines. This further elaborates more technological aspects like tech stack, feature selection, and model details, thereby setting a strong and scalable base for our project. The system diagram and user interface details elaborate on how all these parts come together integrally. SDS is used as a guiding framework throughout the development and implementation phases to keep clear communication among all members of the team, thereby resulting in successful collaboration.

2 Software Design

This section presents the UML class diagram and provides a brief description of each class in our system.

- **Data_Ingestion**: This class gathers raw weather, market, and economic data using methods `Fetch_Weather_Data()`, `Fetch_Market_Data()`, and `Fetch_Economic_Data()` from APIs or databases essential for system operations.
- **Data_Preprocessing**: This class prepares raw data for machine learning by cleaning errors with `Clean_Data()`, handling missing values with `Handle_Missing_Values()`, and standardizing data using `Normalize_Data()` for consistent scaling.
- **Feature_Engineering**: This class extracts and ranks key features using `Select_Features()`, `Extract_Time_Series_Features()`, and `Calculate_Feature_Importance()` to optimize data for predictive modeling.
- **Prediction_Model**: This class manages the Machine Learning lifecycle with `Train_Model()`, `Evaluate_Model()`, and `Save_Model()` methods.
- **Forecasting**: This class projects future trends using `Create_Forecast()` and `Calculate_Confidence_Interval()` to provide predictions with confidence levels.
- **Data_Visualization**: This class displays data through graphs using `Plot_Trends()`, `Show_Forecast()`, and `Visualize_Feature_Importance()`.

- **User_Interface**: This class facilitates user interaction with methods like `Get_User_Input()`, `Display_Forecast()`, `Generate_Report()`, `Display_Market_Trends()`, and `Apply_Search_Filters()` for a user-friendly experience.
- **Data_Storage**: This class handles data persistence by saving with `Save_Data()` and retrieving data using `Load_Data()`, ensuring availability throughout the system's workflow.
- **External_API_Handler**: This class interacts with third-party APIs using `Fetch_Data()` and `Validate_API_Response()` to maintain reliable system data.
- **Authentication**: This class secures user access with `Login()`, `Logout()`, and `Validate_Token()` methods, ensuring authenticated and secure system access.
- **User**: This `User` class represents individuals with attributes like `Username`, `Password`, and `Email`, collaborating with `Authentication` to manage accounts and support specialized roles.
- **Login**: This class manages authentication through the `Authenticate()` method, ensuring only valid users access the system.
- **Register**: This class handles user registration with the `Register_User()` method, verifying and creating new users in collaboration with `Authentication`, ensuring data integrity.

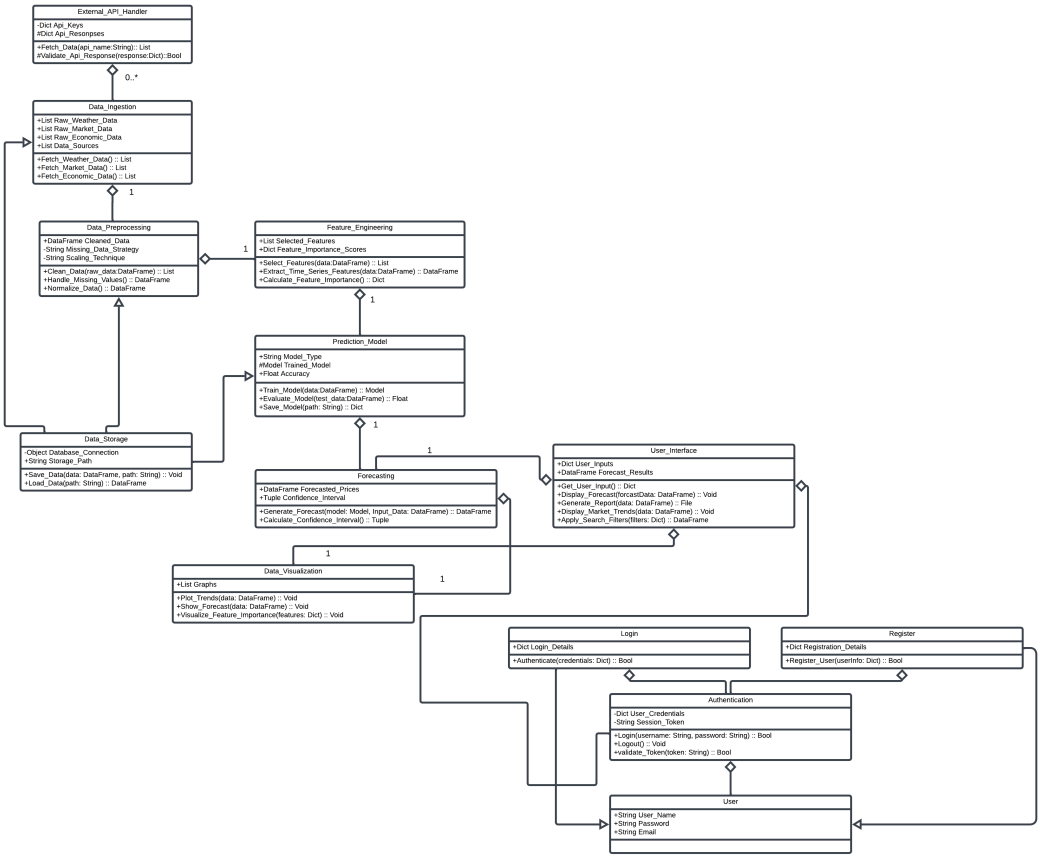

Figure 1: UML Class Diagram of the System

For better resolution,**click here**

3 Data Design

3.1 Data Sources

To develop a commodity price prediction model, the following data sources will be used:

- **Commodity Price Data:** Historical datasets from Zarai Mandi for cotton, maize, wheat, and sugar (2004–2024), with region-wise and station-wise details.
- **Weather Data:** Real-time weather data (temperature, rainfall, humidity) from APIs like OpenWeatherMap to capture current climatic conditions.
- **Supply and Demand Metrics:** Seasonal trends in crop yield, production, and demand extracted from reports and publications using NLP.
- **Commodity Market Updates:** Current prices and trends from Zarai Mandi and news reports processed with NLP for insights.
- **Exchange Rates:** Real-time currency exchange data from APIs like ExchangeRate, reflecting trade impacts on pricing.
- **Economic Indicators:** Macroeconomic factors like inflation rates, crude oil prices, and GDP to account for broader economic influences.

As the project evolves, additional data sources or external factors, such as government policies, international trade dynamics, or unforeseen market conditions, may be incorporated to further improve the accuracy and robustness of the predictions.

3.2 Data Preprocessing:

Data preprocessing ensures the collected data is clean, consistent, and suitable for real-time predictions of 6 months to 1 year ahead. Missing values in commodity datasets will be handled using interpolation techniques, while external factors such as weather and exchange rate data will be normalized to maintain uniform scales. Duplicates are removed, and outliers (e.g., anomalies in prices or external factors) are detected and corrected. The data will be aggregated into day-wise and week-wise formats to support province-wise and station-wise predictions. Text data from reports and news sources, extracted through NLP, will be cleaned and transformed into structured features for integration into the models. Some features will be time-lagged (e.g., past weather or market data) to capture temporal dependencies, while others might be binary indicators (e.g., festival seasons or policy changes) to represent specific events or conditions. Feature selection techniques,

such as correlation analysis, mutual information, and recursive feature elimination (RFE), will identify the most influential features. Continuous features like prices, weather, and exchange rates are standardized using Min-Max scaling or Z-score normalization. The data is split into training (2004–2020), validation (2021–2023) for hyperparameter tuning, and test (2024) sets. Finally, all preprocessed datasets are merged into a unified structure ready for the prediction model. These tasks will be carried out using tools like Pandas, NumPy, and NLP libraries such as NLTK and SpaCy.

3.3 Entity-Relationship Diagram (ERD)

The ERD is a high-level conceptual diagram explaining the working of our project. This diagram shows the entities and the relationships between them stored in the database. It helps explain the logical structure of the database. For better resolution, **click here**.

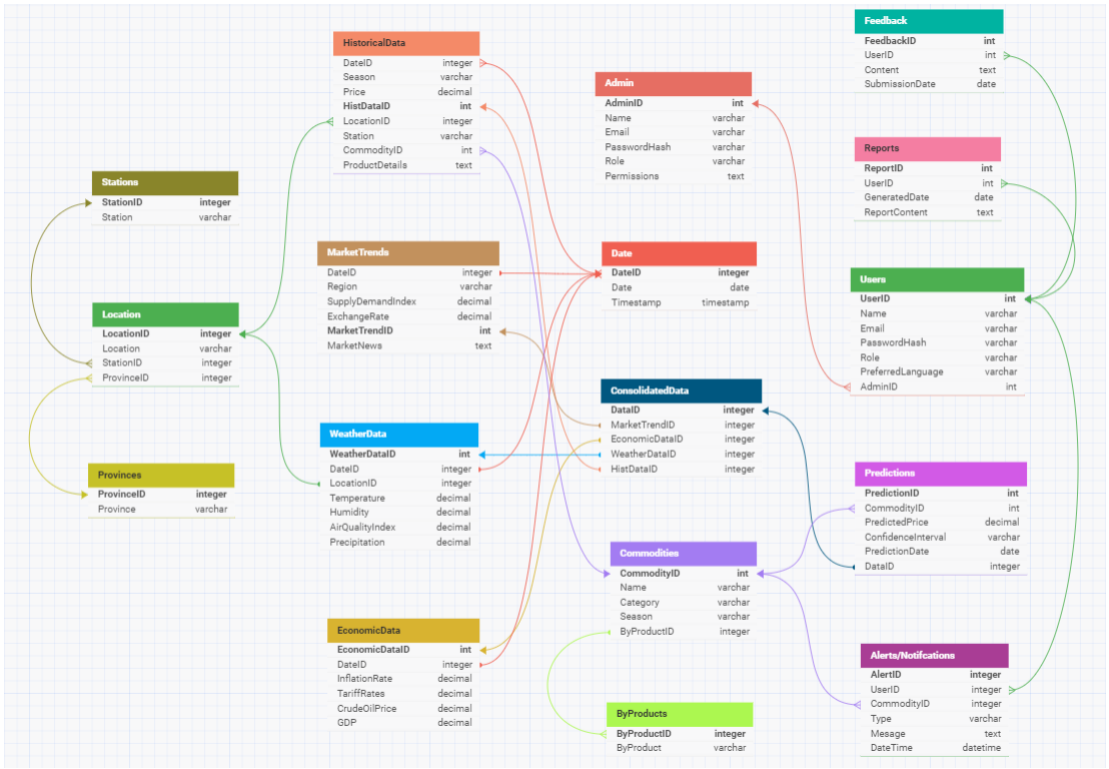

Figure 2: Entity Relationship Diagram

3.4 Data Pipeline

The data pipeline diagram showcases the flow from data ingestion to visualization, ensuring seamless processing and delivery of insights.

- **Data Sources:** Historical data (2004–2024), real-time APIs (weather, exchange rates), external factors, and reports/news for NLP.
- **Data Collection and Storage:** Raw data is preserved for traceability, while processed data is prepared for analysis and modeling.
- **Preprocessing & Feature Engineering:** Data is cleaned, normalized, and enriched with features like seasonal tags, market trends, and sentiment analysis.
- **Model Integration:** ML models are trained on processed data to forecast commodity prices for the next 6 months to 1 year.
- **Output & Visualization:** Predictions from the model are displayed on the dashboard alongside previous values. Alerts, market trends, and insights are shown using precomputed trend metrics from the preprocessing stage.

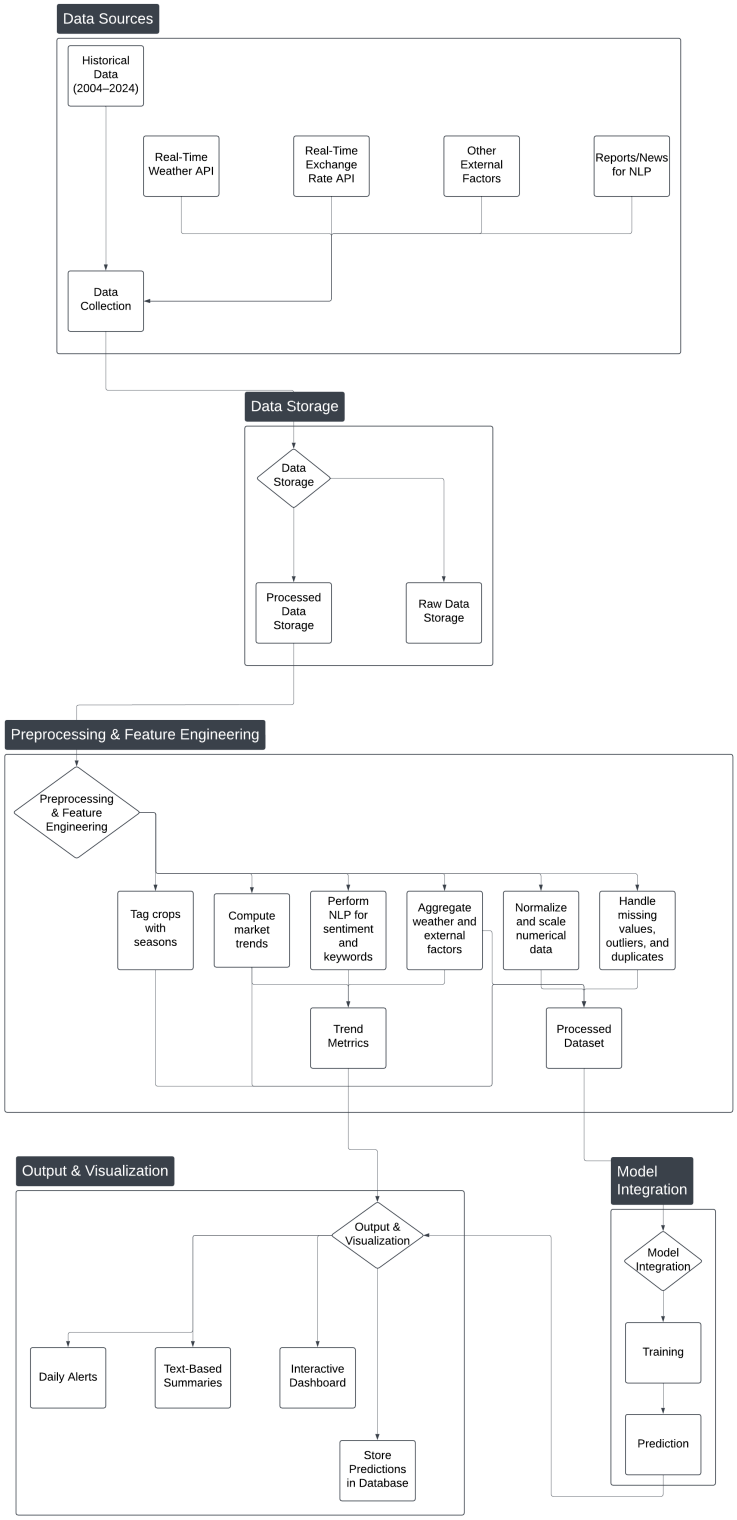

Figure 3: Data Pipeline Diagram

4 Technical Details

4.1 Technology Stack

At its core, the backend is powered by Flask, a lightweight and versatile web framework. Flask enables rapid development and provides robust support for creating RESTful APIs, ensuring smooth handling of API calls for model predictions and user interactions. Key components of the stack include:

- Backend Framework: Flask for building a scalable backend, efficiently managing API endpoints, and integrating machine learning models for real-time predictions. Python for data preprocessing, feature engineering, model training, and API development.
- Database: SQL-based relational databases for storing structured data, with support for historical prices, weather data, and market trends.
- Machine Learning Frameworks: TensorFlow, PyTorch, and scikit-learn for developing and deploying LSTM, XGBoost, Transformers, and ensemble models.
- Frontend Frameworks: React for creating an intuitive and user-friendly web interface.
- Data Handling and Visualization Tools: Pandas, NumPy, Matplotlib, and Seaborn for preprocessing, analysis, and visualization of model inputs and results.
- Deployment Tools: Gunicorn and Docker for deploying the Flask application and ensuring scalability and portability.

Flask's simplicity and flexibility make it ideal for this project, allowing efficient management of data flow between the machine learning models, the database, and the user interface.

4.2 Model Details

This subsection provides details about the machine learning models used for commodity price prediction, including algorithms, training processes, and performance metrics.

4.2.1 LSTM

Long Short-Term Memory (LSTM) is a Recurrent Neural Network (RNN) designed to address vanishing gradient issues using forget, input, and output gates. Its ability to capture long-term dependencies makes it ideal for time-series forecasting of non-linear trends and seasonal patterns. In this project, historical data (2004–2024) with features like weather and economic indicators was used. Implemented with TensorFlow/Keras, the training process optimized layers, hidden units, and dropout rates, with feature scaling and sliding window data preparation ensuring effective learning. LSTM was used to uncover temporal dependencies and benchmark performance against other models.

4.2.2 ARIMA

AutoRegressive Integrated Moving Average (ARIMA) is a statistical model that decomposes time-series data into autoregressive (AR), differencing (I), and moving average (MA) components. Effective for modeling linear and stationary data, ARIMA served as a benchmark in this project. Using Python's statsmodels library, its parameters (p, d, q) were tuned through autocorrelation analysis to capture seasonal trends. While limited in handling complex relationships, ARIMA provided interpretable results for comparison with advanced models.

4.2.3 XGBoost

XGBoost is a gradient-boosting algorithm that excels at modeling structured data by iteratively minimizing residual errors. For this project, it captured feature interactions from lagged prices, weather data, and economic trends. Implemented with the xgboost library, hyperparameters like max depth, learning rate, and subsampling were tuned. Feature engineering, such as moving averages and seasonality flags, enhanced predictive accuracy. XGBoost's speed and interpretability complemented sequential models like LSTM.

4.2.4 Transformers

Transformers, leveraging self-attention mechanisms, model long-range dependencies and global patterns in sequences. Unlike RNNs, they process entire sequences in parallel, improving efficiency. A simplified transformer architecture, implemented with the Hugging Face library, included positional encodings and external features like weather and economic data. Tuning parameters like attention heads and encoder-decoder layers allowed evaluation of their ability to capture multi-seasonal and global dependencies, making them a complementary tool alongside LSTM.

4.3 System Diagram

- **Front-end UI:** Built with React JS, styled using Tailwind CSS, and managed with Redux. Handles user interactions and sends API requests to the backend.
- **Interface Layer:** Manages API requests, user authentication, and data handling (GET and POST requests), ensuring smooth communication between the front end and backend services.
- **Prediction Layer:** Validates and processes data through real-time pipelines (AWS Kinesis, Google Pub/Sub), running machine learning models (LSTMs, Classical ML, RNNs) for predictions.
- **Data Layer:** Gathers real-time and historical data (exchange rates, weather, market data) and preprocesses it using NLP modules. Integrates historical crop price datasets.
- **Database:** Stores unstructured data in MongoDB, structured data in PostgreSQL, and trained models in a model registry.
- **Operating Environment:** Hosted on cloud platforms (AWS/Google Cloud) with dedicated server options and third-party API integration.

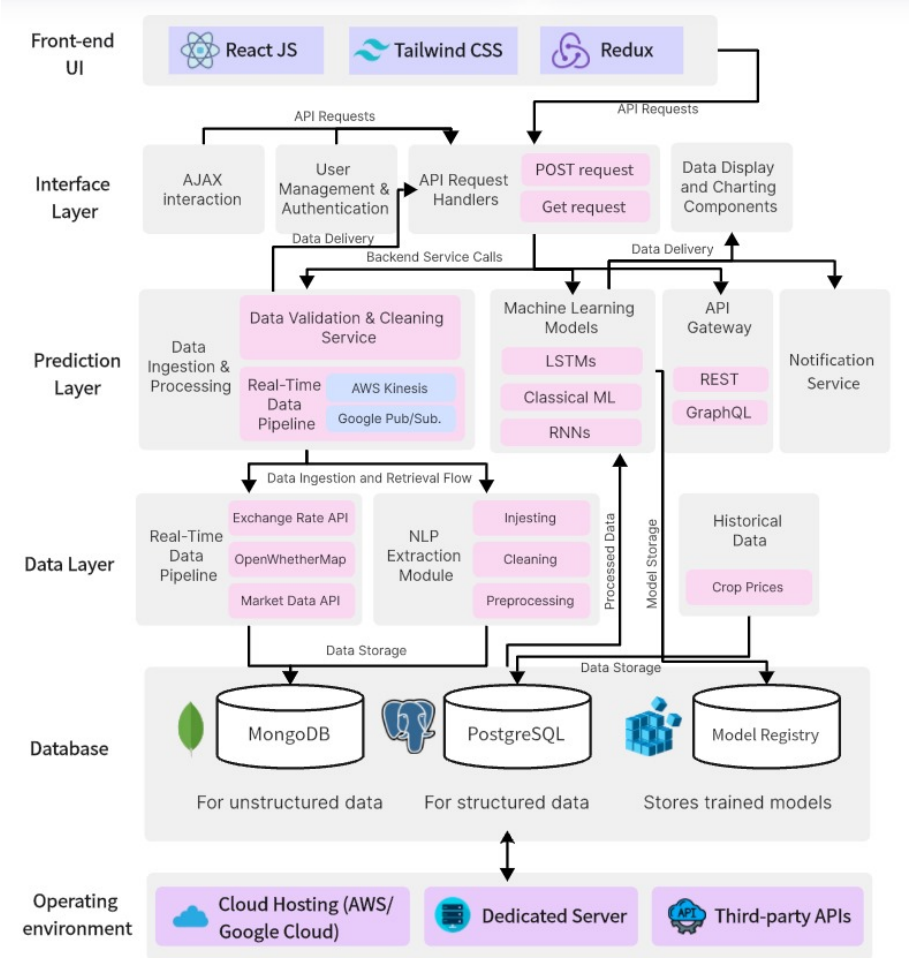

Figure 4: System Block Diagram

5 Conclusion

This Software Design Specification (SDS) document provides a structured blueprint for the development of the Commodity Price Prediction Application. By outlining the software design, data models, data pipelines, and technical details, it ensures that all team members and stakeholders have a clear understanding of the system's architecture and implementation strategy. This foundation facilitates the efficient development and deployment of an accurate, reliable, and user-friendly commodity price prediction system tailored to Pakistan's agricultural sector.