

2. System Analysis

2.1 System Requirements Analysis

Before designing the forecasting application, it is essential to analyze requirements that reflect the needs of small retail operators. Understanding the requirements of the system is a foundational step in software engineering. This section outlines the functional, non-functional, and technical requirements for the mobile-based sales forecasting application. This section outlines the required functionalities and the quality constraints the system must adhere to.

2.1.1 Functional Requirements

The functional requirements describe the core features and capabilities that the system must provide for its users:

- **User Registration and Login**

The system must provide secure authentication using Firebase.

- **Sales Data Input**

Users should be able to manually enter or upload monthly sales data by product and year.

- **Data Storage**

The system must store sales history and predictions using Firebase Firestore with hierarchical structuring (product → year → month).

- **Data Editing**

Users must be able to update or delete existing data entries via the app interface.

- **Algorithm Selection**

The system allows users to select between ARIMA and RNN algorithms for forecasting.

- **Prediction Generation**

Upon clicking a prediction button, the app will generate forecasts for the next 1, 2, and 3 months based on the selected algorithm.

- **Prediction Display**

Predictions for 1, 2, and 3 months ahead must be displayed in a structured and readable format.

- **Offline Access**

Users should be able to access and input data without an internet connection; synchronization will occur once reconnected.

2.1.2 Non-Functional Requirements

The system should meet the following non-functional expectations to ensure usability, efficiency, and maintainability:

- **Usability**

The application must have a user-friendly interface suitable for users without technical backgrounds. It should use clear language, intuitive navigation, and minimal configuration steps to allow ease of use for small business operators.

- **Performance**

The application should deliver forecasting functionality with reasonable responsiveness. Although prediction time may vary depending on the device's hardware, performance has been observed to be satisfactory during testing across common Android devices.

- **Scalability**

The system should be designed to handle the addition of more products and future forecasting models without major restructuring. The modular architecture should support code reuse and easy integration of enhancements.

- **Maintainability**

The system should be built with clean, modular code, separating algorithm logic from UI components. This allows for easier debugging, updates, and expansion in future development cycles.

- **Compatibility**

The app must support a wide range of Android devices, ideally running Android 8.0 (Oreo) and above, to accommodate most of user demographics.

2.1.3 Technical Requirements

The development and deployment of a mobile-based forecasting system involve several technical considerations. These include hardware compatibility, platform support, and the underlying software architecture that enables seamless interaction between Android and Python layers. The following technical requirements reflect the system's architecture and technology stack:

- **Operating System Compatibility**

The application is designed for Android devices running Android 9 (Pie) or newer, which covers a majority of smartphones in current circulation. The application does not require root access or external APIs and functions independently of the device's manufacturer.

- **Hardware Architecture**

The Chaquopy plugin limits support to arm64-v8a and x86_64 architectures. These are the dominant CPU types across modern Android devices. Older 32-bit devices (armeabi-v7a) are not supported, which may slightly limit backward compatibility but enables better performance.

- **Local Python Environment Integration**

The application embeds Python code using Chaquopy. It is configured to use a local Python 3.8 interpreter, bundled with pip-installed packages including:

- `numpy` for numerical processing
- `pandas` for data structure support
- `statsmodels` for ARIMA forecasting
- `tensorflow==2.1.0` for RNN implementation

- **Database and Network**

Firebase Firestore is used as the backend database, offering scalable NoSQL storage. Although prediction models run locally, Firestore requires network access to sync data. In offline conditions, the application retains local writes and synchronizes automatically when the network is restored.

- **Model Execution**

Forecasting scripts are stored inside the Android project at `src/main/python`. The integration is seamless and enables on-device model computation with minimal delay. There is no need for REST APIs or server deployment.

- **Development Tools**

- Android Studio with Gradle
- Chaquopy plugin (no additional dependency blocks)
- XML-based layout design with `RecyclerView` and `FragmentManager`

This combination of technologies ensures that the application remains self-contained, responsive, and easily maintainable, even for users operating in resource-constrained retail environments.

2.2 Forecasting Algorithms Analysis

Table 2 – 1 show comparison between algorithms that have been thoroughly thought of for time series prediction.

Table 2 – 1. Comparison of Forecasting Algorithms

Algorithm	Category	Strengths	Weaknesses	Suitable For
ARIMA	Statistical	Easy to interpret, effective on stable trends	Limited with nonlinear data, requires stationary data	Small datasets, predictable patterns
RNN	Machine Learning	Captures nonlinear dependencies, adaptive	Needs more data and training time, less transparent	Small datasets, predictable patterns
LSTM	Deep Learning	Good for long-term dependencies	Complex, resource-intensive	Long sequential data
Prophet	Statistical	Handles holidays and seasonality, user-friendly interface	Not ideal for short or sparse datasets	E-commerce sales with recurring trends
Transformer	Deep Learning	Parallel processing, high accuracy on large data	High computation cost, complex to implement	E-commerce sales with recurring trends

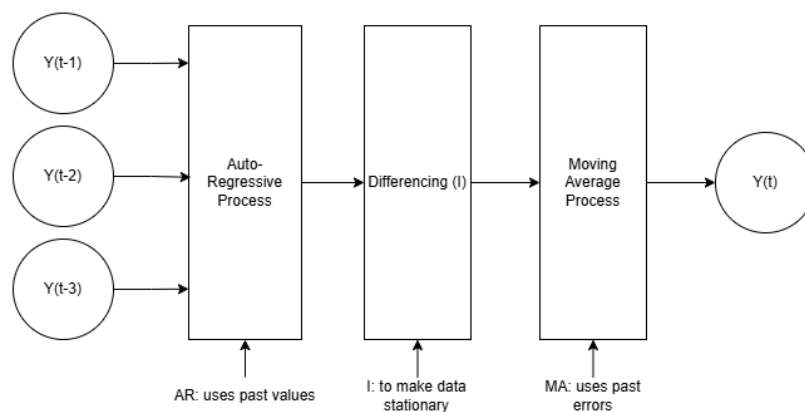
Two primary algorithmic approaches have emerged as suitable for mobile-based forecasting: statistical models like ARIMA and neural models such as Recurrent Neural Networks (RNNs).

- **Traditional Time Series Models (ARIMA)**

Among classical forecasting methods, the ARIMA (AutoRegressive Integrated Moving Average) model has been widely used for time series forecasting due to its interpretability and mathematical simplicity [14]. ARIMA is particularly effective for data that exhibits stable trends and no abrupt structural changes. It models time series data as a combination of autoregressive and moving average components, after applying differencing to ensure stationarity.

While ARIMA requires assumptions about linearity and stationarity, it has shown competitive performance in forecasting retail sales when applied to structured, clean datasets with consistent temporal patterns. Several studies have demonstrated that ARIMA performs well for retail settings with stable consumer demand and predictable seasonal patterns [4][18].

However, ARIMA's limitations become apparent when dealing with complex, nonlinear, or highly volatile datasets—common traits in small retail environments where external factors (such as local events or informal promotions) affect sales unpredictably. In such scenarios, alternative models like RNNs offer a more flexible approach.



Figures 2 – 1. ARIMA Structure Diagram

The ARIMA model begins by ingesting a sequence of monthly sales data as input $Y(t)$. This data first goes through preprocessing steps such as normalization and handling missing values. A stationarity check is then performed to determine whether differencing is needed to make the time series stationary. If the data is non-stationary, differencing is applied to remove trends or seasonality. Once the data is stationary, the appropriate ARIMA model order parameters (p , d , q) are identified. These represent:

- **p**: the number of autoregressive terms (**AR**),
- **d**: the number of differencing steps,
- **q**: the number of moving average terms (**MA**).

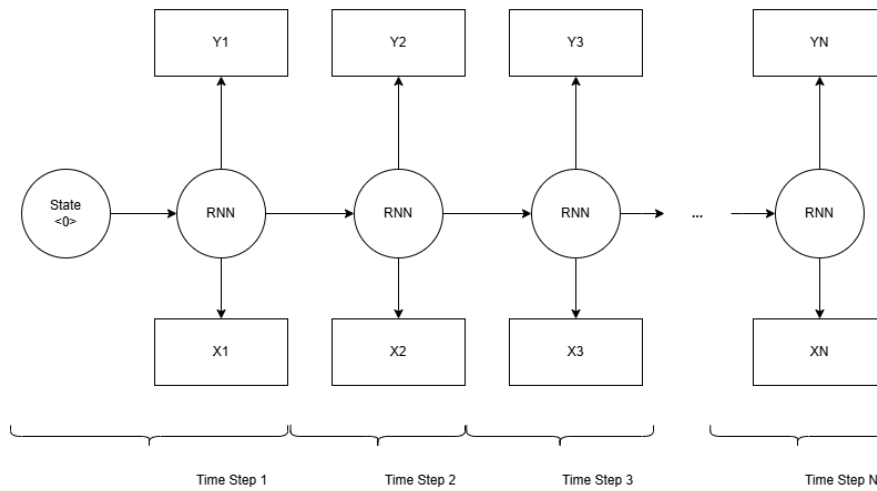
The model is then fitted using the identified parameters. It captures patterns in the stationary data through the AR and MA components and generates future forecasts. After forecasting, an inverse transform is applied (e.g., reversing differencing) to return the forecast to the original data scale. The final output is a forecast of future values based on patterns learned from the historical time series $Y(t)$.

- **Recurrent Neural Networks (RNN)**

RNNs are a class of artificial neural networks designed to process sequential data. Their architecture enables the model to maintain a form of memory by passing information from previous time steps to future ones. This makes RNNs particularly suited to time series forecasting, where the order and structure of the data are essential [2].

In retail forecasting, RNNs are valued for their capacity to model complex temporal dependencies and nonlinear trends that traditional models like ARIMA cannot adequately capture. When trained on sufficient data, RNNs can adapt to changes in patterns and provide more nuanced forecasts. Several comparative studies have found that RNNs outperform ARIMA in dynamic retail environments, especially where data exhibit non-stationary or erratic behavior [4][16].

Nevertheless, RNNs require significantly more computational resources and training data than ARIMA. For small-scale retailers who may have limited historical data, this poses a challenge. Moreover, interpreting the inner workings of RNNs is more complex, potentially limiting trust or transparency for end-users.



Figures 2 – 2. RNN Structure Diagram

The Recurrent Neural Network (RNN) model is designed to process sequential data by learning temporal dependencies across time steps. It begins with a sequence of historical monthly sales data, where each input value X_t represents the sales at time step t , and the corresponding output Y_t is the expected or predicted value. This data is first preprocessed through normalization and, if necessary, the handling of missing values.

A sliding window approach is then applied to convert the continuous time series into overlapping input-output pairs, enabling the model to learn patterns within a fixed-length context. These sequences serve as the training data, where each window of consecutive X values is used to predict the next Y .

The model architecture typically includes one or more recurrent layers, such as simple RNN or LSTM cells, followed by a dense output layer. As each input is processed, the RNN updates its internal hidden state and propagates it forward, allowing the model to retain information from previous steps and capture time-dependent patterns.

Once trained, the RNN forecasts future values one step at a time by feeding each predicted output back as the next input. After the final forecast is generated, an inverse transformation is applied to denormalize the results, producing sales predictions in their original scale. The result is a time-aware forecasting model that adapts to temporal patterns in the input data.

2.3 Summary

This chapter presented a detailed analysis of the system requirements and the forecasting algorithms employed in the development of the retail sales forecasting application. The requirements analysis outlined the functional, non-functional, and technical expectations of the system, ensuring it is tailored to the needs of small and medium-sized retail users operating in potentially constrained environments.

Furthermore, the chapter explored two forecasting algorithms ARIMA and RNN, selected for their complementary strengths in handling linear and non-linear time series data. Each model was described in terms of its operational flow and general structure, establishing a theoretical foundation for their integration into the system. These insights provide the analytical groundwork for the system design decisions detailed in the next chapter.

