Software Requirements Specification

for

Sales Forecasting Dashboard

Version 1.0 approved

Prepared by

Parkhi Joshi 2023352078
Vidushi Verma 2023480364

Sharda University

Table of Contents

Table of Contents ..... **ii**
Revision History ..... **ii**
1. Introduction **1**
1.1 Purpose 1
1.2 Document Conventions 1
1.3 Intended Audience and Reading Suggestions 1
1.4 Product Scope 1
1.5 References 1
2. Overall Description **2**
2.1 Product Perspective 2
2.2 Product Functions 2
2.3 User Classes and Characteristics 2
2.4 Operating Environment 2
2.5 Design and Implementation Constraints 2
2.6 User Documentation 2
2.7 Assumptions and Dependencies 3
3. External Interface Requirements **4**
3.1 User Interfaces 4
3.2 Hardware Interfaces 4
3.3 Software Interfaces 4
3.4 Communications Interfaces 4
4. System Features **5**
4.1 System Feature 1 5
4.2 System Feature 2 (and so on) 5-6
5. Other Nonfunctional Requirements **7**
5.1 Performance Requirements 7
5.2 Safety Requirements 7
5.3 Security Requirements 7
5.4 Software Quality Attributes 7
5.5 Business Rules 7
6. Other Requirements **7**
Appendix A: Glossary **8**
Appendix B: Analysis Models **8**
Appendix C: To Be Determined List **8**

Revision History

Name	Date	Reason For Changes	Version
Vidushi Verma	08/08/2025	Initial draft	1.0
Parkhi Joshi	08/08/2025	Completed the changes as suggested	1.0

1. Introduction

1.1 Purpose

The purpose of this SRS is to define the functional and non-functional requirements of the Sales Forecasting Dashboard. This system is designed to forecast sales trends for businesses using historical sales data. The document outlines the scope, interfaces, performance requirements, and design constraints to guide development, testing, and deployment.

1.2 Document Conventions

Requirements are labeled as REQ-x for easy tracking.

Technical terms and acronyms are defined in the Glossary (Appendix A).

High-priority requirements are explicitly marked as [High].

1.3 Intended Audience and Reading Suggestions

Developers: For implementation details (Sections 3, 4, and 5).

Project Managers: For project scope and constraints (Sections 1 and 2).

Testers: For functional requirements and use cases (Section 4).

End Users/Stakeholders: For understanding system capabilities (Sections 2 and 4).

1.4 Product Scope

The Sales Forecasting Dashboard enables businesses to:

Upload historical sales data (CSV format).

Automatically preprocess and aggregate data.

Forecast sales trends for up to 24 months using Facebook Prophet.

Visualize sales patterns through interactive dashboards.

Export forecast reports for decision-making.

It supports data-driven decision-making by helping organizations plan inventory, allocate resources, and estimate revenue.

1.5 References

Taylor, S. J., & Letham, B. (2018). Forecasting at Scale.

Prophet Documentation: https://facebook.github.io/prophet/

Streamlit Documentation: https://docs.streamlit.io/

Plotly Python Graphing Library: https://plotly.com/python/

Pandas Library: McKinney, W. (2010). Data Structures for Statistical Computing in Python.

2. Overall Description

2.1 Product Perspective

The dashboard is a standalone software tool, not dependent on existing systems. It reads CSV sales data and applies forecasting models, then presents results visually.

2.2 Product Functions

- Upload raw CSV files.
- Select date and sales columns.
- Preprocess and clean data automatically.
- Generate sales forecasts using Prophet.
- Display results with confidence intervals.
- Export forecast data.

2.3 User Classes and Characteristics

- Business Analysts: Moderate technical knowledge, frequent users.
- Managers: Non-technical, need intuitive visual insights.
- Developers/Testers: Technical users validating performance.

2.4 Operating Environment

- Hardware: Intel i5+, 8GB RAM, 256GB SSD.
- OS: Windows 10/Linux/macOS.
- Software: Python 3.x, Streamlit, Prophet, Pandas, Plotly.
- Network: Stable internet connection for deployment.

2.5 Design and Implementation Constraints

- Forecasting limited to Prophet (future extension may include ARIMA/LSTM).
- Input restricted to CSV format.
- Limited to batch data uploads (no real-time database integration).

2.6 User Documentation

The following user documentation will be provided:

- **User Manual (PDF/Word):** Step-by-step guide on uploading data, selecting columns, interpreting charts, and downloading forecasts.
- **Online Help / Tutorials:** Simple walkthroughs integrated into the dashboard UI for first-time users.
- **Technical Documentation (for analysts):** Explanation of data preprocessing steps, forecasting model assumptions, and limitations.

2.7 Assumptions and Dependencies

- Assumes input data is clean, consistent, and representative.
- Depends on third-party libraries: Prophet, Streamlit, Pandas, Plotly.
- Accuracy depends on quality and quantity of historical sales data.

3. External Interface Requirements

3.1 User Interfaces

Web-based dashboard built with Streamlit.

Features: File upload button, dropdowns for column selection, interactive charts, CSV download option.

3.2 Hardware Interfaces

Standard computing hardware; no special devices required.

3.3 Software Interfaces

Python libraries (Prophet, Pandas, Plotly).

Optional integration with databases/APIs (future).

3.4 Communications Interfaces

Runs on localhost or cloud (Heroku/Streamlit Cloud).

Supports standard HTTP/HTTPS connections.

4. System Features

4.1 Data Upload and column section

4.1.1 Description and Priority

This feature allows users to upload raw sales data in **CSV format** and select the relevant columns (date and sales amount) required for forecasting. It is a **High Priority** feature since it is the entry point for all system functionality.

4.1.2 Stimulus/Response Sequences

- **Stimulus:** User clicks the "Upload CSV" button and selects a file.
- **Response:** System reads the file and displays available column names.
- **Stimulus:** User selects the date column and the sales/amount column.
- **Response:** System validates input and prepares the data for preprocessing.
- **Error Case:** If the file is not CSV or columns are missing, system displays an error message prompting re-upload.

4.1.3 Functional Requirements

- **REQ-1:** The system shall allow users to upload files in CSV format only.
- **REQ-2:** The system shall display all detected column names for user selection.
- **REQ-3:** The system shall validate that the selected date column can be parsed into valid datetime format.
- **REQ-4:** The system shall validate that the selected sales column contains numeric values.
- **REQ-5:** The system shall display an error message if invalid file format or incorrect column types are selected.

4.2 Data Preprocessing and Aggregation

4.2.1 Description and Priority

This feature automatically preprocesses the uploaded data, converts date columns into a consistent format, removes invalid entries, and aggregates sales data into **monthly granularity**. It is a **High Priority** feature since clean data is essential for accurate forecasting.

4.2.2 Stimulus/Response Sequences

- **Stimulus:** User uploads valid CSV and selects required columns.
- **Response:** System automatically cleans invalid dates, drops empty rows, and resamples data to monthly totals.
- **Error Case:** If no valid rows remain after cleaning, system prompts user to upload a new dataset.

4.2.3 Functional Requirements

- **REQ-6:** The system shall convert all selected date values into datetime format.
- **REQ-7:** The system shall remove rows with invalid or missing date values.
- **REQ-8:** The system shall resample daily data into monthly totals.
- **REQ-9:** The system shall notify users if insufficient data is available for forecasting.

4.3 Forecasting Using Prophet

4.3.1 Description and Priority

This feature applies the **Prophet time-series forecasting model** to the preprocessed dataset to generate predictions up to **24 months ahead**. It is a **High Priority** feature since forecasting is the core functionality of the system.

4.3.2 Stimulus/Response Sequences

- **Stimulus:** User confirms data selection and requests forecast.
- **Response:** System trains Prophet model on historical sales data and generates forecasts with upper and lower confidence intervals.
- **Error Case:** If insufficient historical data is provided (e.g., <12 months), the system displays a warning about reduced accuracy.

4.3.3 Functional Requirements

- **REQ-10:** The system shall apply Prophet forecasting on the preprocessed dataset.
- **REQ-11:** The system shall provide forecasted values for up to 24 months into the future.
- **REQ-12:** The system shall compute upper and lower confidence bounds.
- **REQ-13:** The system shall notify users if forecast reliability is low due to short history.

4.4 Visualization and Interactive Dashboard

4.4.1 Description and Priority

This feature enables visualization of historical and forecasted sales trends through interactive charts and KPIs. It is a **Medium Priority** feature, but highly valuable for user experience and decision-making.

4.4.2 Stimulus/Response Sequences

- **Stimulus:** User selects date range or filtering options.
- **Response:** System updates forecast plots, KPIs, and confidence intervals dynamically.
- **Stimulus:** User hovers/clicks on chart data points.

- **Response:** System displays exact values and forecast intervals.

4.4.3 Functional Requirements

- **REQ-14:** The system shall display historical sales and forecast values on an interactive line chart.
- **REQ-15:** The system shall allow date range selection through sliders.
- **REQ-16:** The system shall display KPI metrics (total sales, forecasted growth/decline).
- **REQ-17:** The system shall highlight predicted values with confidence bounds (yhat_upper, yhat_lower).

5. Other Nonfunctional Requirements

5.1 Performance Requirements

Forecast generation must complete within 10 seconds for datasets <100k rows.

5.2 Safety Requirements

System must prevent crashes due to invalid inputs by rejecting them gracefully.

5.3 Security Requirements

Support user authentication in future.

Ensure uploaded files are not stored permanently.

5.4 Software Quality Attributes

Usability: Intuitive UI for non-technical users.

Reliability: Accurate results within Prophet's confidence bounds.

Scalability: Extendable to other forecasting models.

5.5 Business Rules

Forecasting limited to provided CSV data.

No forecasts generated for incomplete/missing sales columns.

6. Other Requirements

Future integration with ERP systems and real-time APIs.

Role-based user management for multi-user environments.

Appendix A: Glossary

CSV: Comma-Separated Values.

Prophet: Open-source time series forecasting library by Facebook.

Streamlit: Python framework for creating web apps.

KPI: Key Performance Indicator.

Appendix B: Analysis Models

Data flow diagram (input → preprocessing → forecasting → visualization → output).

Appendix C: To Be Determined List

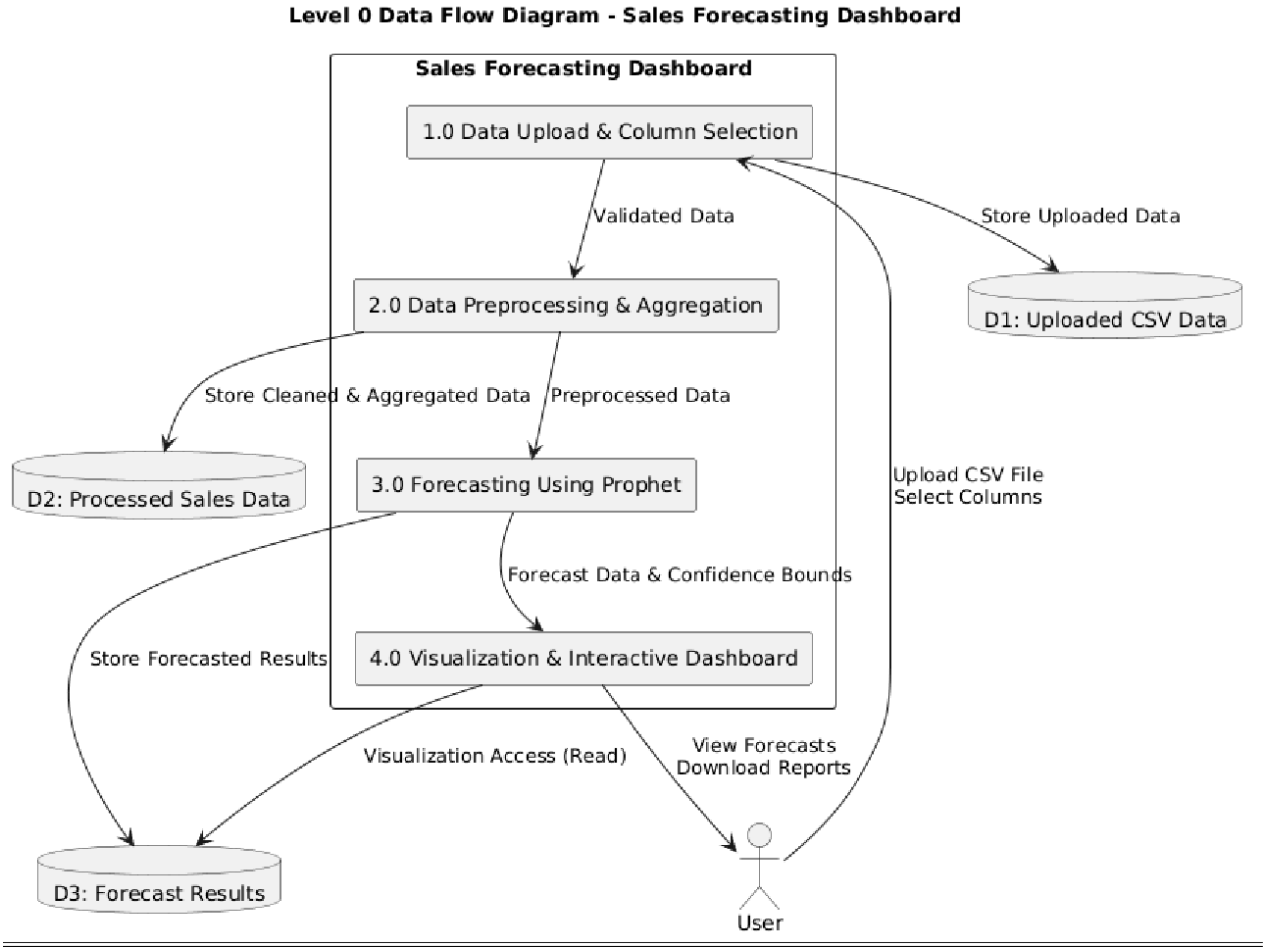
TBD-1: Authentication mechanism.

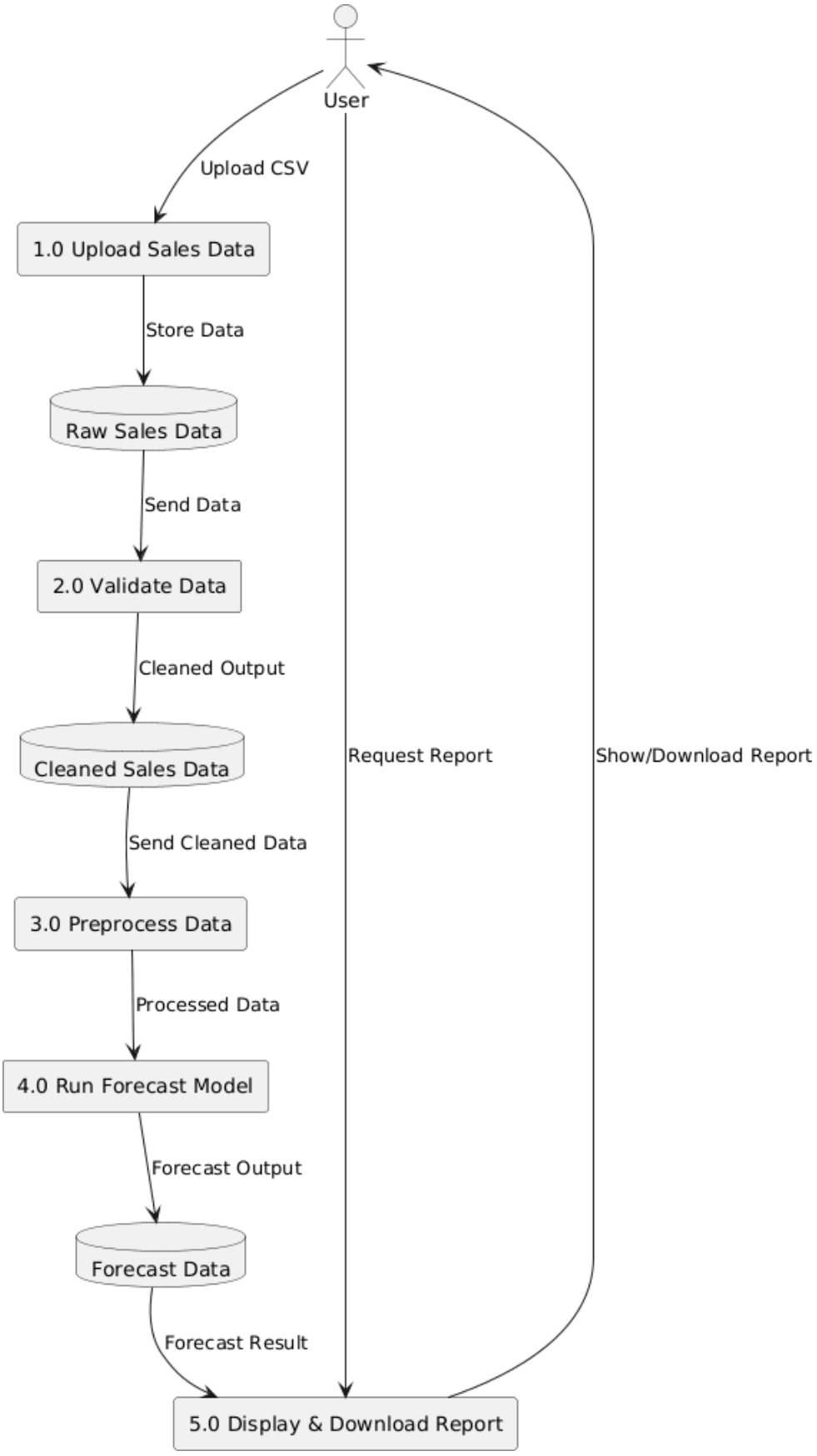
TBD-2: Support for multiple file formats (Excel, JSON).

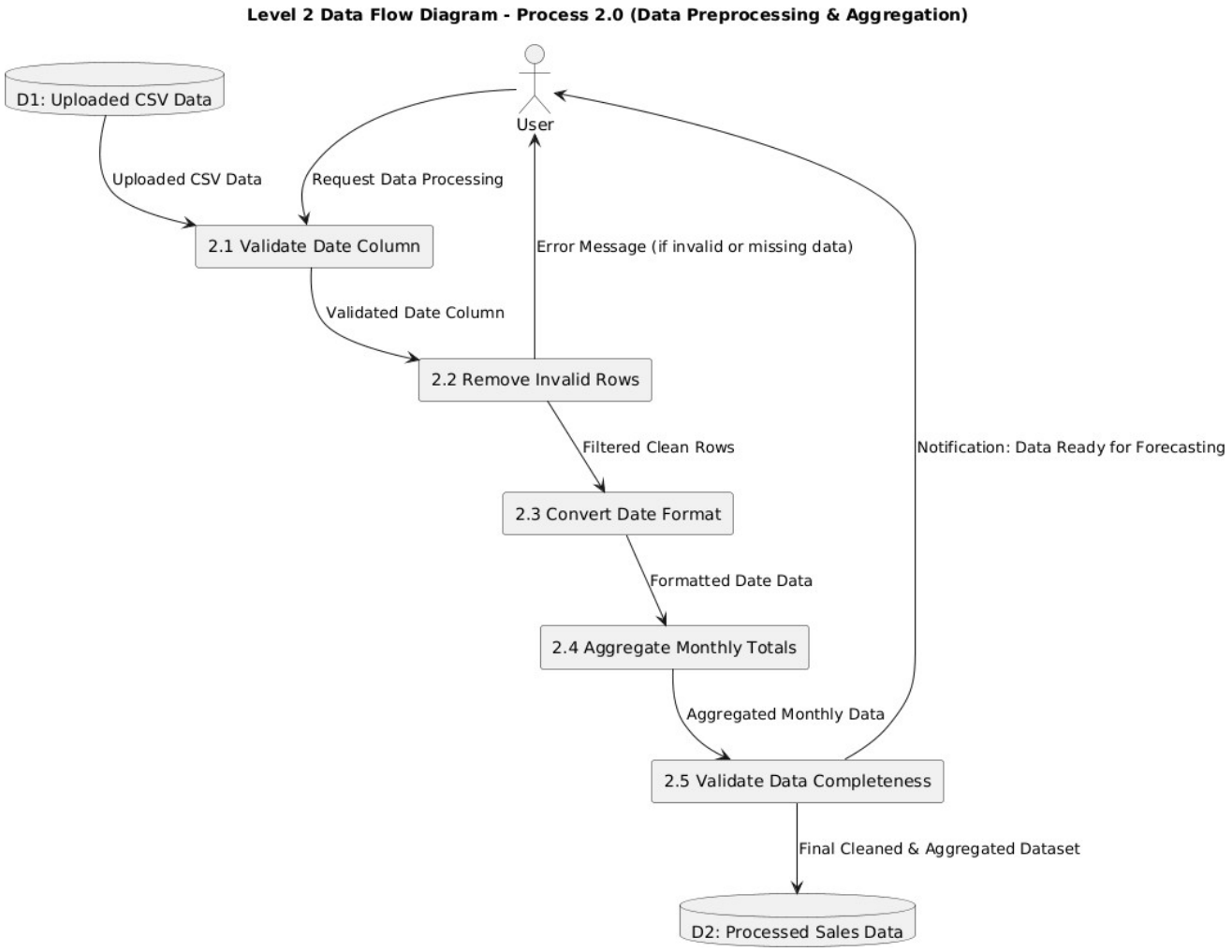
TBD-3: Integration with live databases.

EXPERIMENT – 02

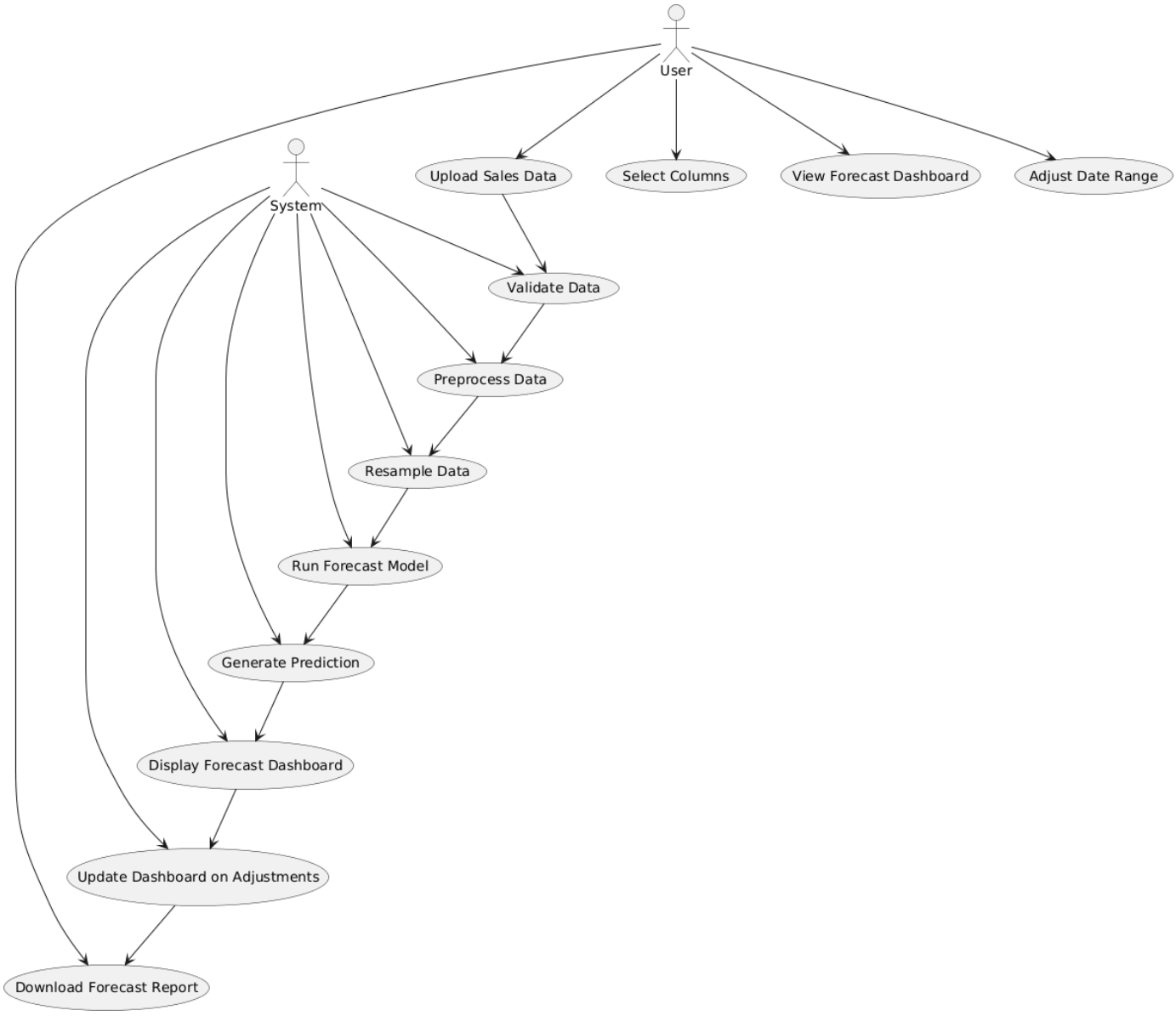
Aim: Draw the user's view analysis for the suggested system: Data flow diagram, Use case diagram and Sequence diagram.



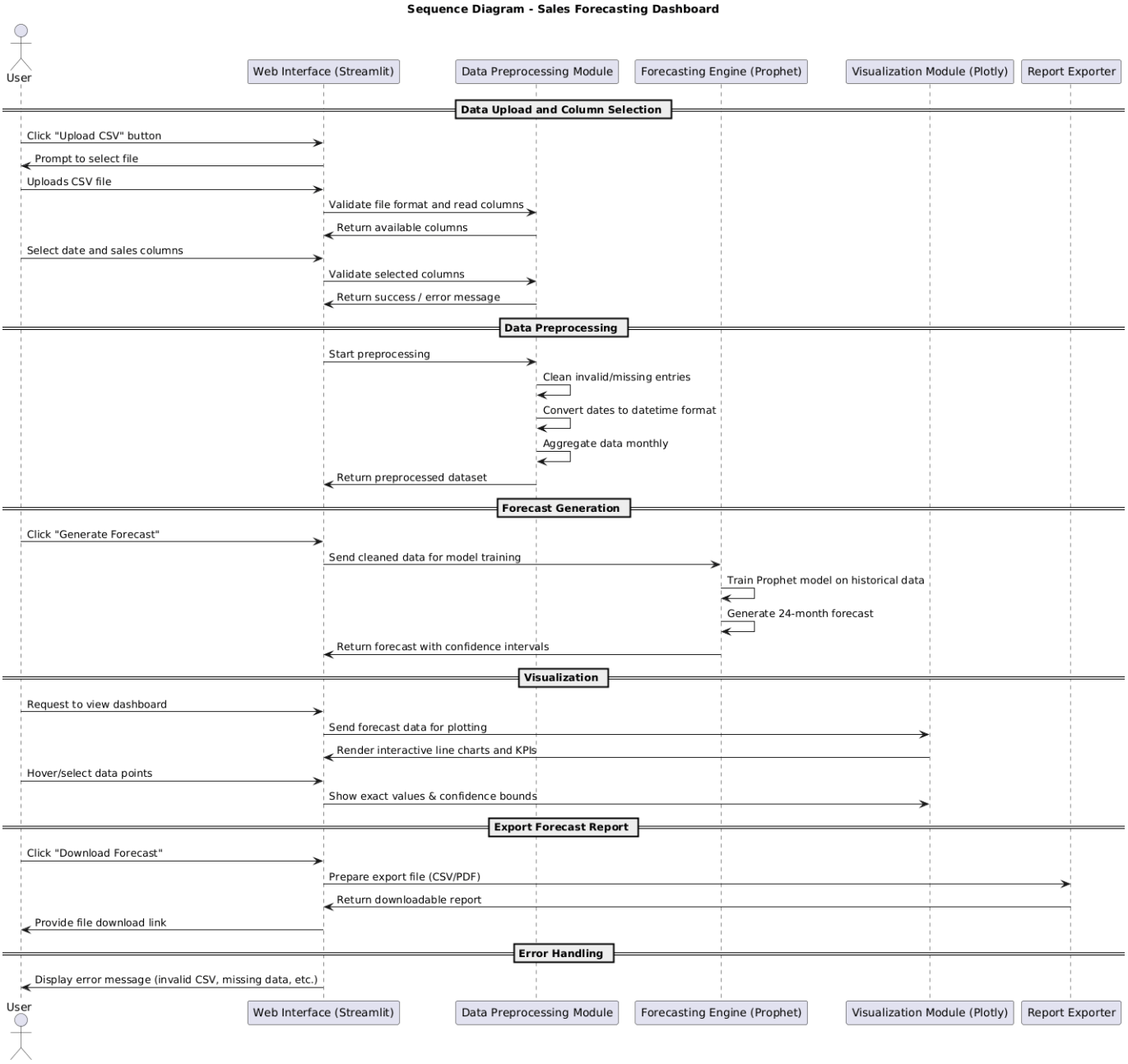

User
Upload CSV
1.0 Upload Sales Data
Store Data
Raw Sales Data
Send Data
2.0 Validate Data
Cleaned Output
Cleaned Sales Data
Send Cleaned Data
3.0 Preprocess Data
Processed Data
4.0 Run Forecast Model
Forecast Output
Forecast Data
Forecast Result
5.0 Display & Download Report
Request Report
Show/Download Report


Level 2 Data Flow Diagram - Process 2.0 (Data Preprocessing & Aggregation)
D1: Uploaded CSV Data
User
Uploaded CSV Data
Request Data Processing
2.1 Validate Date Column
Error Message (if invalid or missing data)
Validated Date Column
2.2 Remove Invalid Rows
Filtered Clean Rows
Notification: Data Ready for Forecasting
2.3 Convert Date Format
Formatted Date Data
2.4 Aggregate Monthly Totals
Aggregated Monthly Data
2.5 Validate Data Completeness
Final Cleaned & Aggregated Dataset
D2: Processed Sales Data

USE CASE DIAGRAM:



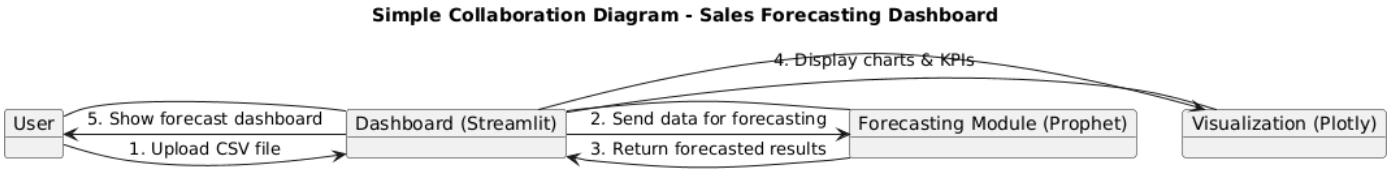
SEQUENCE DIAGRAM:



EXPERIMENT – 03

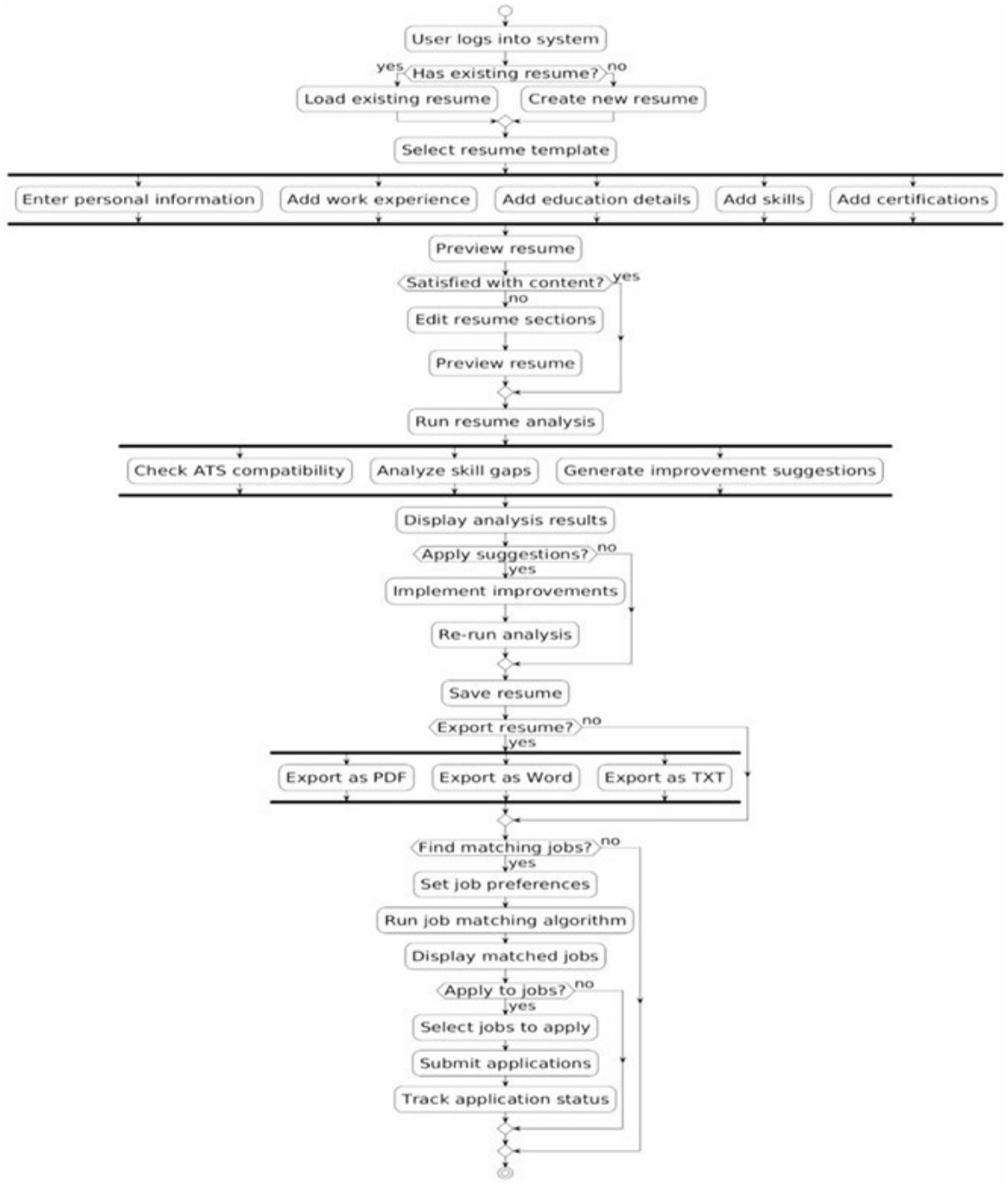
Aim: Draw the structural view diagram for the system: Collaboration diagram and Activity diagram

COLLABORATION DIAGRAM:



.

ACTIVITY DIAGRAM:



IMPLEMENTATION:

```
import streamlit as st
import pandas as pd
from prophet import Prophet
import plotly.graph_objects as go

st.set_page_config(page_title="📈 Sales Forecast Dashboard", layout="wide")

# --- Sidebar ---
st.sidebar.title("💲 Upload & Settings")

uploaded_file = st.sidebar.file_uploader("Upload Raw Sales CSV (with 'date' & 'money' columns)", type=["csv"])
show_bounds = st.sidebar.checkbox("Show Confidence Intervals", value=True)
show_table = st.sidebar.checkbox("Show Forecast Table", value=False)

@st.cache_data
def load_and_forecast(file):
    df = pd.read_csv(file)

    # Ensure columns exist
    if 'date' not in df.columns or 'money' not in df.columns:
        raise ValueError("CSV must contain 'date' and 'money' columns.")

    df['date'] = pd.to_datetime(df['date'])
    df = df[['date', 'money']]
    df = df.rename(columns={'date': 'ds', 'money': 'y'})

    # Resample to monthly totals
    df = df.set_index('ds').resample('M').sum().reset_index()

    model = Prophet()
    model.fit(df)

    future = model.make_future_dataframe(periods=24, freq='M')  # 2 years
    forecast = model.predict(future)
    return forecast, df
```

```
try:
    if uploaded_file:
        forecast_df, original_df = load_and_forecast(uploaded_file)

        last_actual_date = original_df['ds'].max()
        forecast_df['type'] = ['Historical' if d <= last_actual_date else 'Forecast' for d in forecast_df['ds']]

        # Date slider
        min_date, max_date = forecast_df['ds'].min(), forecast_df['ds'].max()
        selected_range = st.slider(
            Select forecast date range:",
            min_value=min_date.to_pydatetime(),
            max_value=max_date.to_pydatetime(),
            value=(min_date.to_pydatetime(), max_date.to_pydatetime())
        )

        df_range = forecast_df[(forecast_df['ds'] >= selected_range[0]) & (forecast_df['ds'] <=
selected_range[1])]

        # --- Forecast Plot ---
        st.subheader(" Forecasted Sales")

        fig = go.Figure()

        hist = df_range[df_range['type'] == 'Historical']
        future = df_range[df_range['type'] == 'Forecast']

        fig.add_trace(go.Scatter(x=hist['ds'], y=hist['yhat'], name='Historical', line=dict(color='gray',
dash='dot')))
        fig.add_trace(go.Scatter(x=future['ds'], y=future['yhat'], name='Forecast',
line=dict(color='royalblue')))

        if show_bounds:
            fig.add_trace(go.Scatter(x=df_range['ds'],  y=df_range['yhat_lower'],
                                     name='Lower Bound', line=dict(color='lightblue', dash='dot'), showlegend=False))
            fig.add_trace(go.Scatter(x=df_range['ds'], y=df_range['yhat_upper'],
                                     name='Upper Bound', line=dict(color='lightblue', dash='dot'), fill='tonexty',
showlegend=False))
```

```
        fig.update_layout(xaxis_title="Date", yaxis_title="Sales", template="plotly_white")
        st.plotly_chart(fig, use_container_width=True)

        # --- KPIs ---
        st.markdown("### Forecast Summary")
        latest = df_range.iloc[-1]
        prev = df_range.iloc[-2] if len(df_range) > 1 else latest
        delta = latest['yhat'] - prev['yhat']

        col1, col2 = st.columns(2)
        col1.metric("Latest Prediction", f"{latest['yhat']:,.2f}", f"{delta:,.2f}")
        col2.metric("Forecast Date", latest['ds'].strftime('%b %Y'))

        # --- Table ---
        if show_table:
            st.markdown("### Forecast Table")
            st.dataframe(df_range[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].rename(columns={
                'ds': 'Date', 'yhat': 'Prediction', 'yhat_lower': 'Lower Bound', 'yhat_upper': 'Upper Bound'
            }), use_container_width=True)

        # --- Download Forecast ---
        def convert_df_to_csv(df):
            return  df.to_csv(index=False).encode('utf-8')

        st.download_button(
            label="t Download Forecast CSV",
            data=convert_df_to_csv(forecast_df),
            file_name="sales_forecast_2yr.csv",
            mime='text/csv',
        )
    else:
        st.info(" Please upload a raw CSV with columns: date and money.")
except Exception as e:
    st.error("⚠ Error processing file. Make sure it has columns named 'date' and 'money'.")
    st.exception(e)
```