A Software Design Specification for ClinicTrends AI Project

Design and Creative Technologies

Torrens University, Australia

Group: 1

Students:

Luis Guilherme de Barros Andrade Faria - **A00187785**

Luong Hai Chau - **A00117495**

Subject Code:

SEP 401

Subject Name:

Software Engineering Principles

Assessment No.: 2

Title of Assessment:

Proposal

Lecturer: Dr. Ranju Mandal

Date: Aug 2025

Table of Contents

1. Introduction

1.1 Purpose of this document

This Software Design Specification (SDS) outlines the architectural blueprint and detailed design of the **ClinicTrendsAI** system. It serves as a critical bridge between the high-level functional and non-functional requirements defined in the Software Requirements Specification (SRS) and the implementation phase. This document provides a comprehensive and structured reference to guide the development, testing, and maintenance of the system. Specifically, this SDS aims to:

- Define the overall system architecture and component-level structure.
- Specify interfaces and interactions between subsystems and modules.
- Document key design patterns, architectural decisions, and the rationale behind them.
- Offer detailed implementation guidance, including algorithms, data structures, and processing workflows.
- Foster a shared understanding of the system design among all stakeholders, ensuring alignment throughout the development lifecycle.

This document is intended for a technical audience, including software developers, QA engineers, project managers, and system architects. It will be actively maintained and iteratively refined as the design evolves during the development process.

1.2 Scope of the development project

ClinicTrendsAI is a standalone, Python-based web application designed to assist businesses, particularly aesthetic clinics, in analyzing and forecasting customer satisfaction trends using historical Net Promoter Score (NPS) data and textual feedback. The solution leverages modern machine learning (ML) techniques to provide predictive insights and actionable visual analytics.

The scope of this project includes the development of key functional modules and supporting infrastructure as detailed below:

1.2.1 In-Scope Elements

Core Functionalities:

- CSV file upload with schema validation and data preprocessing.
- Interactive dashboards for visualizing NPS trends with filtering by date, clinic, or other metadata.
- Four machine learning pipelines for forecasting future satisfaction scores using various algorithms.
- Sentiment analysis powered by multiple NLP techniques, including TextBlob, scikit-learn, and Hugging Face Transformers.
- Confidence interval–based alerts for identifying potential drops in customer satisfaction.
- Export functionality for reports and visualizations in both PDF and CSV formats.
- Real-time translation of customer feedback and file content using the Deep Translator API.

Technical Implementation:

- Python is mainly used for building the application
- User interface development using **Streamlit** (Streamlit n.d.) for rapid prototyping and interaction.
- Data processing pipelines implemented using **pandas** for ETL tasks.
- Machine learning modules using **scikit-learn** and **Transformers** for modeling and prediction.
- Visualization components built with **Altair** and **Plotly** for interactivity and responsiveness.
- Deployment setup to support both **local execution** and **cloud-based hosting** (Streamlit Cloud).

1.2.2 Out-of-Scope Elements

The following features are considered out-of-scope for the current MVP but may be considered in future iterations as we evolve ClinicTrendsAI as a software:

- User authentication and role-based access control.
- Integration with a persistent database for long-term data storage.
- CRM or third-party feedback collection system integration.
- Mobile-native versions of the application.
- Real-time data ingestion or streaming from external sources.
- Automated model retraining and CI/CD deployment pipelines.

1.3 System Overview

ClinicTrendsAI transforms raw customer feedback into actionable business insights through a modern, modular architecture. The system follows a modular monolithic architecture pattern where distinct components handle specific functional areas while maintaining clear interfaces between them. This promotes separation of concerns, easier integration, and future extensibility.

The system is engineered to support non-technical business users, particularly in aesthetic clinics, providing intuitive interfaces and real-time insights derived from structured CSV survey data. Below is a breakdown of the core architectural layers and their responsibilities:

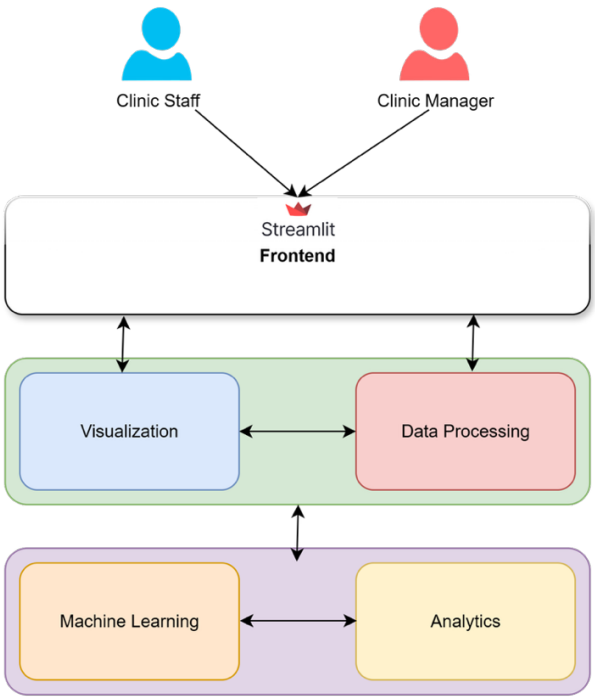

***Figure 1.0**: High level system architecture*

- **Frontend**: Streamlit-based web application providing an interactive user interface with navigation, visualizations, and configurable parameters
    - Multi-page navigation system
    - Responsive design adapting to different screen sizes
    - Interactive widgets for data filtering and model selection
- **Data Processing Layer**: Python modules for data ingestion, validation, cleaning, and transformation
    - CSV parser and validator
    - Data preprocessing pipelines
    - Feature engineering components
- **Analytics Layer**: Machine learning and statistical analysis components
    - Multiple ML model implementations (TF-IDF + Logistic Regression, Transformer-based)
    - Feature importance calculator
    - Statistical analysis engines
- **Visualization Layer**: Interactive charting and reporting components
    - Altair-based interactive visualizations
    - Word cloud generators for text analysis
    - Matplotlib/Plotly components for specific chart types
- **Machine Learning Layer**:
    - Hugging Face Transformers integration for sentiment analysis (Hugging Face. n.d.)
    - TextBlob for simple sentiment scoring
    - Deep translator for multilingual support

The application is optimized for CSV files up to 200MB, ensuring quick processing, intuitive analytics, and real-time feedback. Its architecture ensures modularity, supports scalability, and prioritizes user accessibility, making it ideal for small to medium-sized businesses seeking AI-powered customer insights without heavy infrastructure.

2. Design Considerations

This section outlines the key factors, assumptions, constraints, goals, and methods that shaped the overall design of **ClinicTrendsAI**. These considerations ensure that the system is both feasible and aligned with stakeholder needs while maintaining scalability and maintainability

2.1 Assumptions and Dependencies

- **End-user environment:** Users will access the system through modern browsers (e.g., Chrome, Edge, Firefox) with stable internet connectivity.
- **Data source:** Input will be supplied as structured CSV files containing at least the fields *Date*, *Store*, *Score*, and *Comment*
- **User profile:** Target users are non-technical managers and analysts requiring actionable insights without deep technical knowledge
- **External services:** NLP and translation rely on external APIs (e.g., Hugging Face, Deep Translator), assuming their availability and acceptable latency
- **Model lifecycle:** ML models will be periodically updated but not retrained on the fly within the MVP.
- **Data privacy:** Users are responsible for data privacy compliance.

2.2 General Constraints

- **Performance constraint:** System must maintain responsiveness, with processing and visualization completed within **10 seconds** for files up to **200 MB**
- **Storage constraint:** No persistent database is implemented in the MVP; all data resides in memory during a session
- **Security constraint:** HTTPS is required for transport security, but no user authentication is included in the initial release
- **Third-party limits:** External APIs may impose rate limits or quotas affecting NLP features.
- **Accessibility constraint**: Application designed for single-user sessions due to Streamlit's architecture.

2.3 Goals and Guidelines

- **Modularity:** Each subsystem (data processing, analytics, visualization, NLP) is isolated, enabling independent updates and future enhancements.
- **Simplicity (KISS principle):** UI and workflows remain intuitive and minimal for non-technical users.
- **Explainability:** Every prediction should include a rationale (e.g., feature importance scores) to support business decision-making
- **Reusability and scalability:** Components are designed to support future additions, such as persistent storage or additional ML models.
- **Open-source preference:** Tools and frameworks chosen (Streamlit, scikit-learn, Altair) favor cost-effectiveness and community support.

2.4 Development Methods

- **Agile Scrum methodology**: ClinicTrendsAI adopts an Agile Scrum methodology with six two-week sprints over a 12-week development timeline. This approach enables iterative development with continuous feedback integration and adaptation to changing requirements.
    - Early prototypes and stakeholder feedback
    - Continuous integration and regular sprint reviews ensure quality and progress
- **Tools:**
    - **Git/GitHub** for version control and collaboration.
    - **Streamlit** for rapid UI development and iteration.
    - **pandas/scikit-learn/Transformers** for data processing and analytics.
    - **Altair/Plotly** for visualization.

3. User-Interface Design

3.1 Personas

***Figures 1.1** and **1.2:** Business Manager and Operations Analyst representation images generated using Grok™ Artificial Intelligence.*

- Business Manager (Figure 1.1): A time-pressed executive focus on outcomes. Seeks quick, actionable insights without technical complexity. This persona values clarity, minimalism and immediate decision-making support (ex: alerts, trends, sentiment summaries).
- Operations Analyst (Figure 1.2): A data-savvy staff member requiring deep drill-down capabilities, transparency into ML processes and confidence in model accuracy. Prioritizes data filtering, interpretability (feature importance) and exportability of detailed reports.

3.2 Scenarios

- **Scenario 1:** The Business Manager uploads a CSV and views a summary dashboard showing overall customer sentiment, key alerts, and one-click report to export for a board meeting.

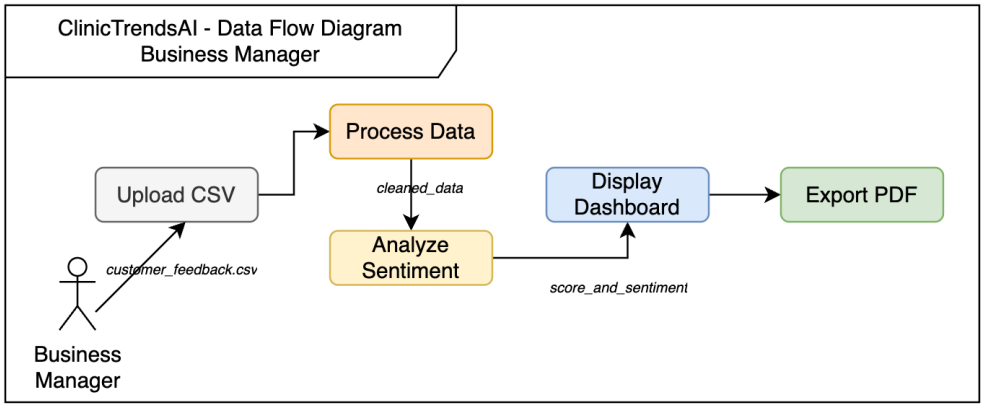

Figure 1.3: *Data Flow Diagram for the Business Manager Experience Flow.*

- **Scenario 2:** The Operations Analyst filters results by clinic and month, toggles between ML models, and inspects sentiment breakdowns and feature influence before refining campaign strategy.

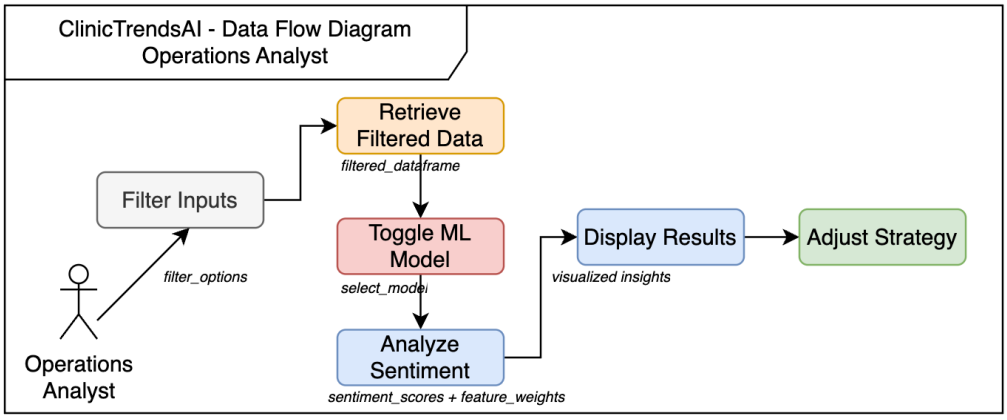

Figure 1.4: *Data Flow Diagram for the Operations Analyst Experience Flow.*

3.3 Task Flows

This is an example of User Journey on the app

Step	Business Manager	Operations Analyst
1	Open Dashboard	Open dashboard
2	Upload csv file	Upload csv file
3	View summary metrics and alerts	Inspect model predictions and confidence interval
4	Export PDF	Export CSV
5	Schedule actions	Adjust campaigns based on insights

3.4 Swimlane Diagrams

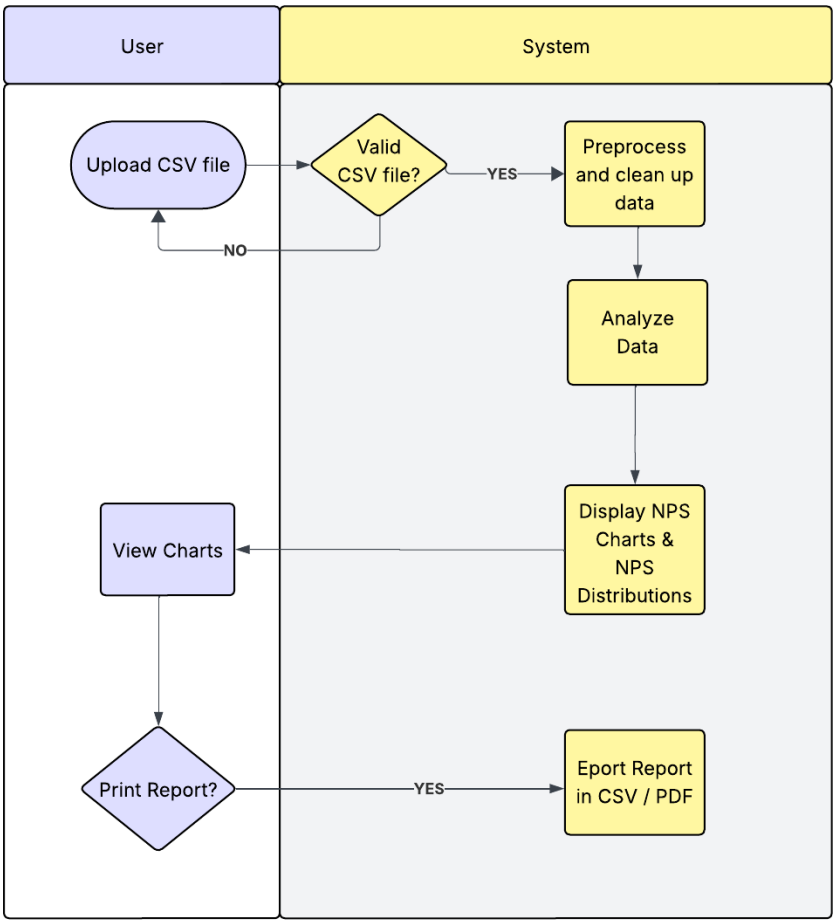

Figure 1.5: *A simple horizontal Swimlane diagram can illustrate responsibilities across UI, Data Processor, and ML Engine as each user interacts with the system from upload to output generation.*

3.5 Storyboard

One of the possible storyboards can be described below:

- 1st: User launches ClinicTrendsAI web app;
- 2nd: User is greeted with a clean homepage with a call-to-action to start navigation;
- 3rd: Upon upload, the dashboard expands with trend lines, pie charts KPIs;
- 4th: Alerts for decision trends appear;
- 5th: User clicks “Export” to save a summarized PDF report.

3.6 Mock-ups

This section presents the **visual prototypes** of the ClinicTrendsAI user interface. The mock-ups serve as a blueprint for the layout, navigation flow, and interaction design before full implementation. They illustrate how end-users will upload data, configure analysis options, and view results through dashboards and reports. The purpose of these mock-ups is to:

- Provide a clear visualization of the key screens and components described in the User Interface Design.
- Validate usability and workflow with stakeholders prior to development.
- Ensure consistency in design elements, such as color schemes, fonts, and widget placement.
- Highlight how system features—such as NPS trend charts, sentiment summaries, and export buttons—are integrated into a cohesive user experience.

From Figure 1.7 to Figure 1. 11, providing major screens of the Clinic Trends AI application designed for desktop and mobile devices.

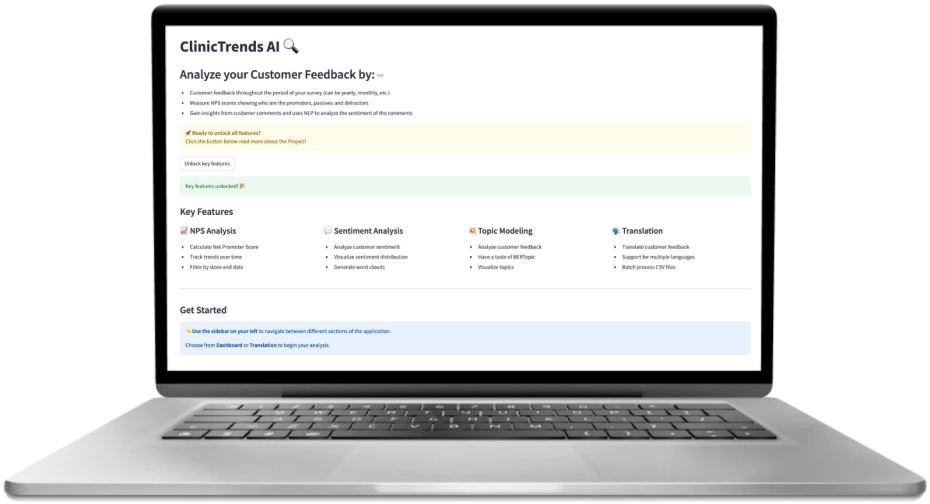

Figure 1.6: *Welcome screen with call to Action and Clear explanation of usage for enhanced User Experience on a big screen (laptop).*

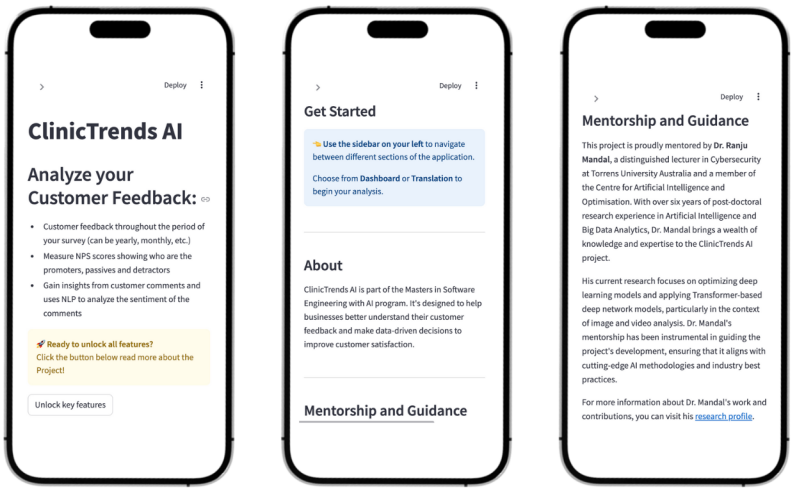

Figure 1.7: *Welcome screen with call to Action and Clear explanation of usage for enhanced User Experience in a small screen (mobile).*

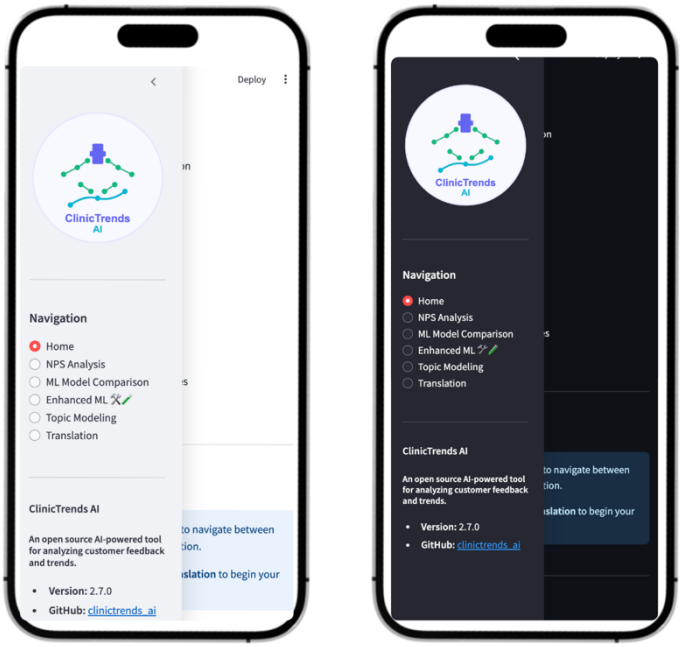

Figure 1.8: *Navigation tab for accessing different pages and functionalities in light/dark theme mode.*

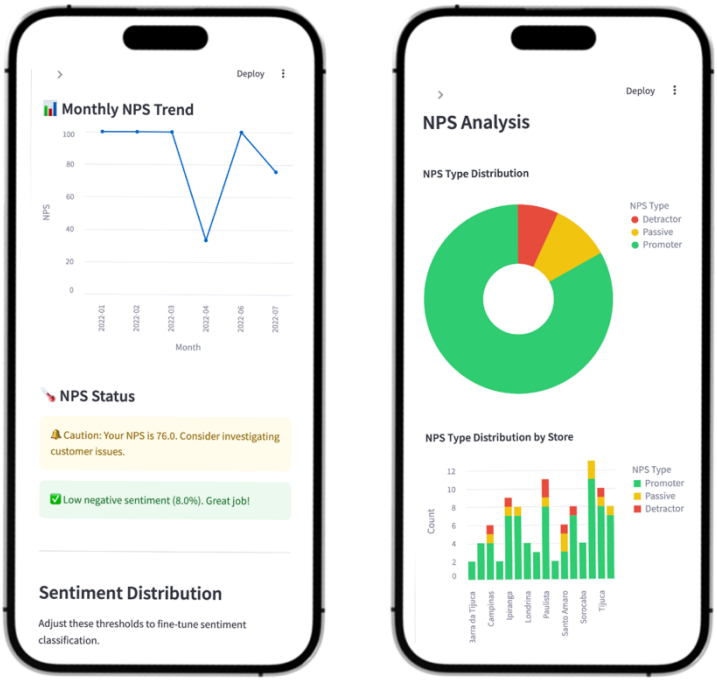

Figure 1.9: *NPS Trend, Status and Alerts using standard colors.*

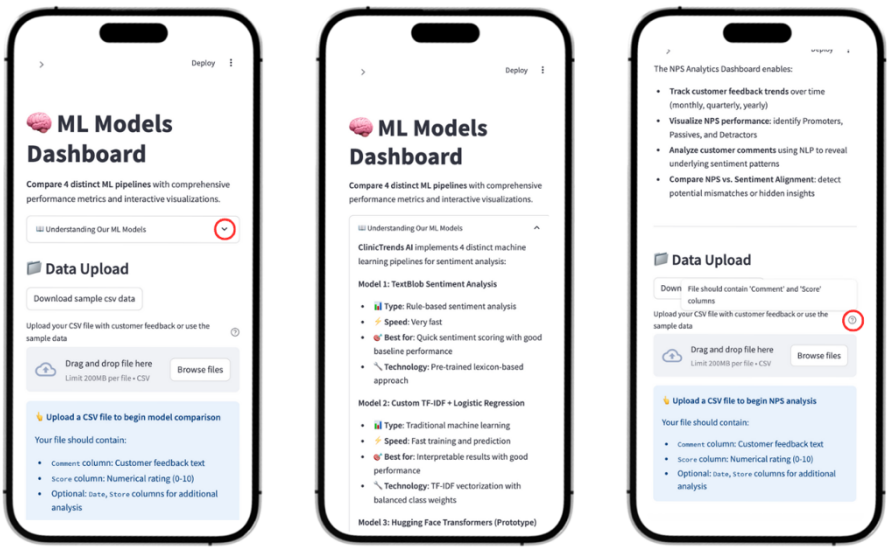

Figure 1.10: *Examples of Tooltip and Dropdown menus present on the application to guide users in a smooth experience.*

4. Architectural Strategies

The **ClinicTrendsAI** system adopts a modular, microservices-inspired architecture designed to satisfy the functional and non-functional requirements outlined in the SRS. The following strategies were selected to ensure modularity, scalability, maintainability, and usability.

- **Separation of Concerns**
    - **Strategy:** Each major function (UI, data processing, analytics, visualization, NLP) is implemented as an independent module with well-defined interfaces.
    - **Rationale:** Reduces coupling, makes the system easier to test and maintain, and allows future enhancements without impacting unrelated modules.
- **Streamlit-based Presentation Layer**:
    - **Strategy**: Adopt Streamlit as the front-end framework for rapid development of interactive dashboards.
    - **Rationale**: Streamlit offers a fast development cycle and built-in responsiveness for non-technical users, even though it trades off multi-user concurrency
- **Component Reusability**:
    - **Strategy:** Core functions such as data validation, preprocessing, and sentiment analysis are encapsulated as reusable services.

    - **Rationale:** Enables re-use across different pipelines and simplifies integration of new features (e.g., additional ML models)
- **Asynchronous Processing**:
    - **Strategy:** Long-running tasks such as model inference run asynchronously with progress indicators in the UI.
    - **Rationale:** Maintains responsiveness and meets the latency requirement for typical data sizes
- **Stateless Session Management**:
    - **Strategy:** The MVP maintains all session data in memory without persistent storage.
    - **Rationale:** Accelerates development and leverages state-of-the-art NLP models, with the trade-off external dependency and rate limits
- **API-Driven NLP Services**:
    - **Strategy:** Integrate external APIs (Hugging Face, Deep Translator) through clear service interfaces.
    - **Rationale:** Accelerates development and leverages state-of-the-art NLP models, with the trade-off external dependency and rate limits.
- **Deployment Flexibility**
    - **Strategy:** Design for both local execution and cloud deployment (Streamlit Cloud)
    - **Rationale:** Supports development testing and real-world accessibility for stakeholders without infrastructure overhead

5. System Architecture

ClinicTrendsAI is an intelligent analytics platform for customer feedback, combining advanced NLP, machine learning, and interactive visualization. The system is designed for extensibility, modularity, and real-time insights, supporting both classical and state-of-the-art AI techniques.

5.1 High Level Architecture

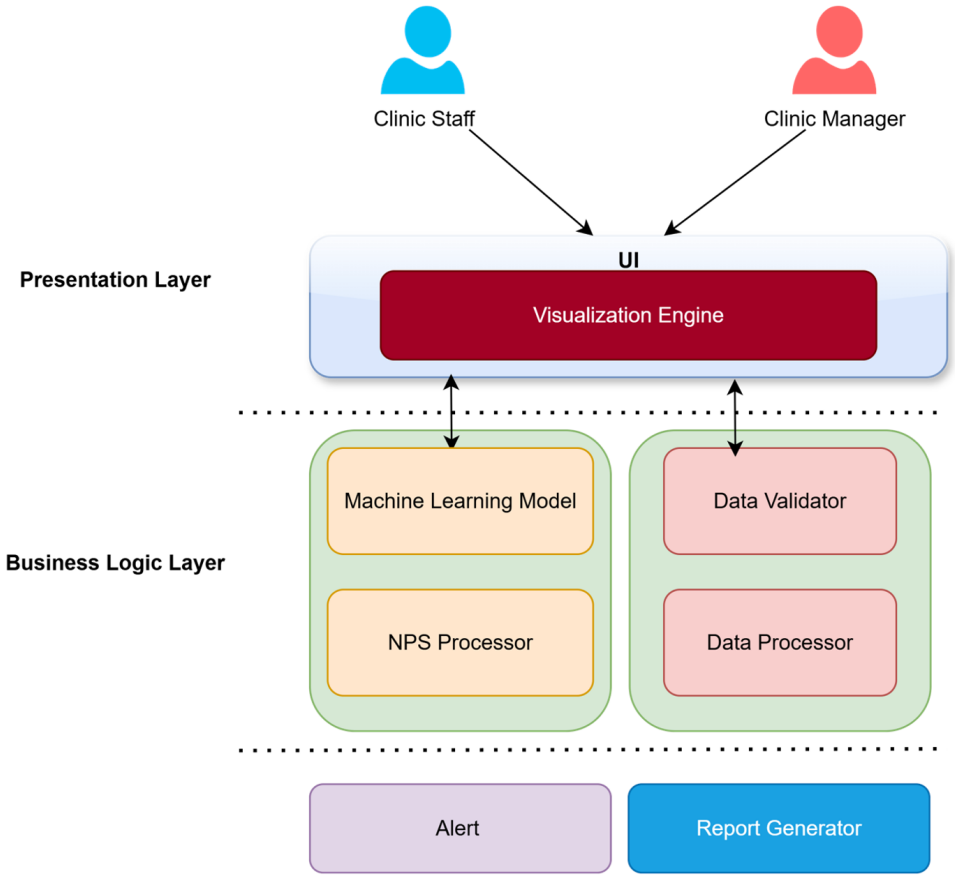

Figure 1.11: *High-level architecture*

The high-level architecture of ClinicTrendsAI in Figure 1.12 is designed in terms of essential layers:

- **Prestation Layer:**
  - **Responsibility**: UI logic, user navigation, and orchestration of business logic.
  - **Interfaces**:
    - Calls business logic via function imports.
    - Receives processed dataframes, model results, and visualizations.
    - Emits UI events (file upload, navigation, parameter selection).
- **Business Logic Layer**:

    - **Responsibility**: Data preprocessing, NLP, visualization, translation, and model training.
    - **Interfaces**:
        - Exposes stateless functions.
        - Accepts pandas DataFrames, returns processed DataFrames or visualization objects.
        - Supports both classical (scikit-learn) and transformer-based (HuggingFace) models
- Foundation Layer
    - **Responsibility**: Facilitating users by alerting, storing data, and generating reports.
    - **Interfaces**:
        - Accesses by UI to upload data and display reports

ClinicTrendsAI is composed of these primary components:

- **Frontend UI (Streamlit)**
    - Provides the main interface for end-users.
    - Handles file upload, navigation across views, and displaying interactive visualizations.

- **Data Processor**
    - Validates the schema of uploaded CSV files.
    - Cleans, transforms, and preprocesses data for downstream analytics.
- **ML Model**
    - Implements machine learning pipelines using scikit-learn and other libraries.
    - Predicts NPS scores and provides feature importance metrics.
- **NLP Processor**
    - Performs sentiment analysis on textual feedback.
    - Supports translation of text and files via Hugging Face Transformers and Deep Translator.
- **Alert System**

- Monitors predictions for low NPS scores.
- Triggers visual or exportable alerts when thresholds are crossed.

- **Visualization Engine**
  - Generates interactive charts, trend lines, and word clouds for insights.
  - Uses libraries such as Altair, Plotly, and Matplotlib.
- **Report Generator**
  - Creates downloadable reports in PDF and CSV formats.
  - Packages both raw results and visualizations for offline use.

5.2 Subsystem Architecture

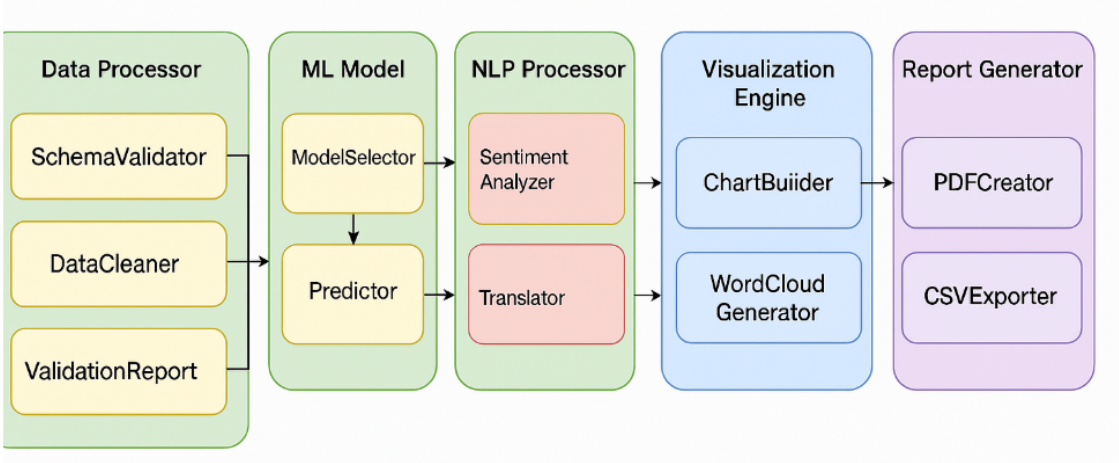

Figure 1.12: *Subsystem Architecture in Detail*

ClinicTrendsAI is an intelligent analytics platform for customer feedback, combining advanced NLP, machine learning, and interactive visualization. The system is designed for extensibility, modularity, and real-time insights, supporting both classical and state-of-the-art AI techniques. Figure 1.13 presents the key components of the system, including

- **Data Processor**
  - SchemaValidator
  - DataCleaner
  - ValidationReport
- **ML Model**
  - ModelSelector
  - Predictor

    - FeatureImportanceCalculator
- **NLP Processor**
    - SentimentAnalyzer
    - Translator
- **Visualization Engine**
    - ChartBuilder
    - WordCloudGenerator
- **Report Generator**
    - PDFCreator
    - CSVExporter

5.3 Technique & Trade-offs

- **Multi-Model ML Architecture:** Four distinct ML pipelines (TF-IDF, hybrid, transformer, and fusion) allow for A/B testing and benchmarking.
    - **Trade-off**: Increased complexity and resource usage but enables rapid experimentation and best-model selection.
- **Dynamic Data Preprocessing**: Auto-detects NPS scale (1–5 or 1–10), robustly handles missing/dirty data
    - **Trade-off:** Slightly higher code complexity for greater user flexibility.
- **Real-Time Alerts & Visualization**: Immediate feedback on NPS and sentiment
    - **Trade-off:** Requires careful UI/UX design to avoid alert fatigue

6. Detailed System Design

This section provides a comprehensive breakdown of the internal structure and behavior of the **ClinicTrendsAI** system. While the System Architecture section outlined the high-level components and their interactions, this section drills into the design of individual modules and subcomponents that reflect in the detail design architecture (Figure 1.14), detailing how each one fulfills its responsibilities. By presenting these details, this section ensures that:

- Developers can confidently implement and integrate each component.
- Testers can trace functional requirements back to specific design elements.

- Stakeholders have a clear understanding of how the system meets both functional and non-functional requirements.

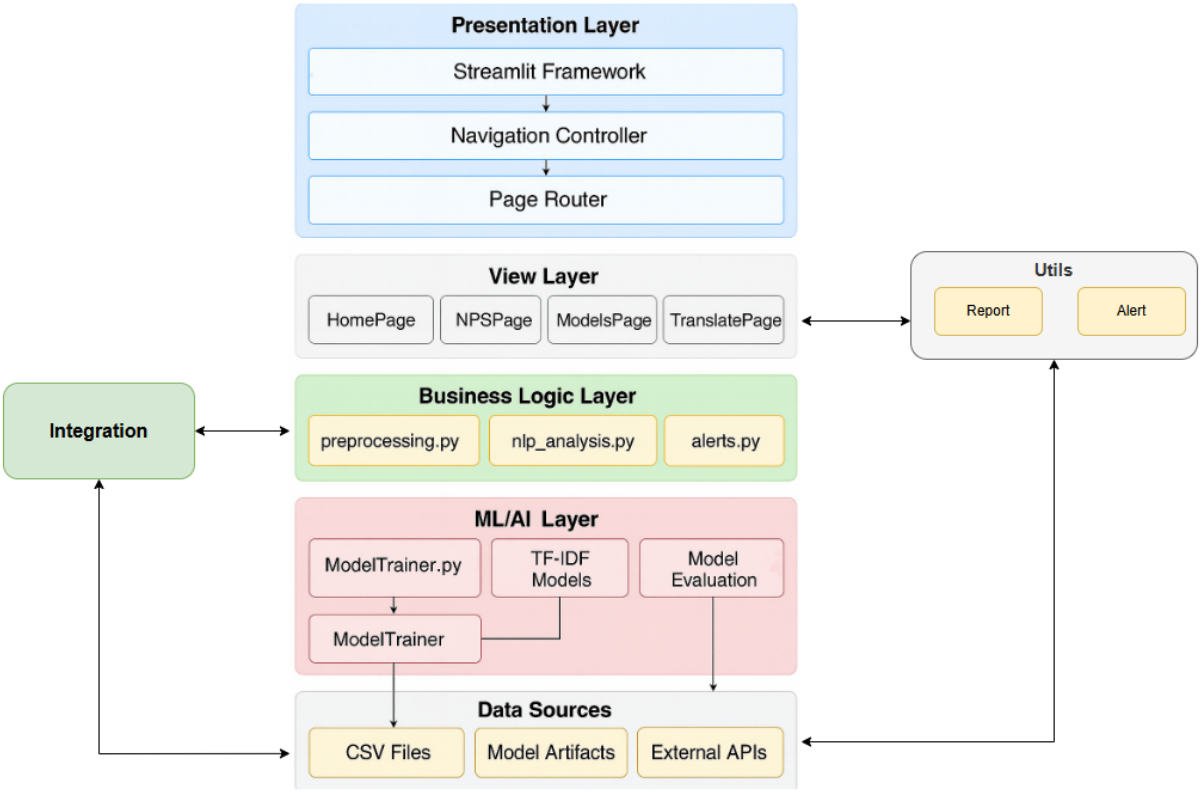

Figure 1.13: *Detail Design Architecture*

6.1 Classification

Each major element of **ClinicTrendsAI** is classified according to its role within the overall system and the level of abstraction at which it operates. The classification helps readers quickly understand what type of software entity they are dealing with and how it fits into the modular design.

Components	Type	Description
Frontend (Streamlit UI)	Presentation Layer	A top-level module that handles user interaction. Responsible for file upload, navigation, parameter selection, displaying dashboards and outputs
Data Processor	Business Logic Layer	Manages ingestion of CSV files, schema validation, data cleaning,

		NPS type extraction and prepares dataset for downstream analysis.
Analytics	ML/AI Layer	Core processing logic that runs ML models, computes NPS-sentiment agreement, sentiment distribution and other advanced metrics from the dataset.
Rendering Processor	View Layer	Generates charts, plots and dashboards (e.g., sentiment bar charts, word clouds, time series). Presents analytical results to the user in intuitive ways.
External Integration	Integration Layer	Modules responsible for interacting with external libraries (e.g., HuggingFace, DeepTranslator) for NLP, translation and sentiment inference tasks.
Report Generator	Utility Layer	Assembles user-friendly outputs, including PDF reports and downloadable CSVs.

6.2 Definition

- **Frontend (Streamlit UI)**: Handles user interaction: file uploads, parameter inputs, visualizations, and navigation.
- **Data Processor:** Cleans raw CSV input, standardizes column names, maps NPS scores, and prepares sentiment columns.
- **Analytics:** Performs sentiment classification, topic modeling (e.g., BERTopic), and calculates metrics like agreement rate and NPS score.
- **Rendering Processor:** Generates bar charts, word clouds, and time-series graphs using matplotlib, seaborn, or plotly.
- **External Integration Modules**: Optional components to connect to APIs (e.g. email triggers, GPT-4).

- **Report Generator:** Builds downloadable PDF or CSV reports, aggregating all dashboard insights.

6.3 Responsibilities

Frontend

- Receive user CSV upload
- Route inputs to backend components
- Display insights and summaries

Data Processor

- Validate file format
- Handle missing values
- Derive NPS sentiment from score

Analytics

- Compare model sentiment vs NPS
- Extract topics
- Flag anomalies

Rendering Processor

- Translate analysis into charts
- Make results interpretable for business and analyst personas

External Integration

- Handle API requests/responses
- Format external data inputs into system-compatible formats

Report Generator

- Summarize findings

- Allow one-click export for presentations and decision-making

6.4 Constraints

- **Technical Constraints**
  - **Technology Stack:**
    - The system is built on **Python 3.9+**; compatibility issues may arise with older runtimes.
    - Reliance on the **Streamlit framework** introduces limitations in advanced UI customization and multi-user concurrency.
    - Library and package versions must be carefully managed to avoid compatibility issues.
  - **Performance:**
    - Response times are expected to remain under **10 seconds** for files up to 200 MB. Large data volumes may degrade performance.
    - Scalability is constrained by the Streamlit runtime and available computer resources.
  - **Security:**
    - Limited authentication and authorization features in the MVP.
    - Data privacy requirements must align with **GDPR** and similar regulations, with minimal personally identifiable information (PII) retained to prevent disclosure the identity of individuals (Zhang, Y., Fan, et al. 2025)
- **Business Constraints**
  - **Functional Scope:**
    - Core functionality is restricted to NPS analysis and sentiment insights; no extended analytics or CRM integration in MVP.
    - User Interface (UI) design is simplified to meet rapid development timelines.
  - **Operational Scope:**
    - Deployment targets are constrained to Streamlit Cloud or local environments.

        - Maintenance and dependency management require minimal overhead but limit advanced scaling options.

- **Architectural Constraints**
    - **Design Patterns:**
        - The system follows a simplified layer architecture that defines the application to be multiple horizontal layers, each layer is specificized responsibility (Richards, M. 2022)
        - Modularity is prioritized, but some tightly coupled components may restrict future refactoring.
    - **Integration:**
        - Dependent on external APIs (e.g., Hugging Face, Deep Translator), which introduces reliability risks and rate limits.
        - Only CSV-based data sources are supported in the current release.
- **Operational Constraints**
    - **Deployment:**
        - Streamlit Cloud imposes constraints on horizontal scaling and high-concurrency usage.
        - Limited customization of deployment environments may affect performance tuning.
    - **Monitoring:**
        - Observability is basic, with **minimal logging and alerting**; advanced monitoring tools are not integrated in the MVP.
- **Compliance & Regulatory Constraints**
    - **Data Protection:**
        - The system must comply with **GDPR** principles such as data minimization and right-to-deletion, which play a vital role in the technology platform and data architecture (Li, H., et al. 2019)
        - No sensitive healthcare or financial data should be processed unless enhanced compliance measures, such as The Privacy Act 1988 (Privacy Act 1988**.** n.d.) are implemented.
    - **Industry Standards:**

        - Must adhere to applicable international privacy regulations depending on deployment of regions.
- **Resource Constraints**
    - **Development Resources:**
        - Small team with mixed skill levels and limited development time (12-week timeline).
        - Expertise needed in Python, ML, and Streamlit may slow onboarding of new contributors.
    - **Infrastructure:**
        - Limited hardware resources and budget constrain the size of datasets and training tasks.
        - The MVP environment is designed to operate without persistent storage or high-availability clusters.

6.5 Composition

- **Frontend** includes upload form, page router, chart renderer
- **Data Processor** uses Pandas functions for transform
- **Analytics Layer** calls models stored locally or loaded via huggingface
- **Rendering** includes matplotlib or plotly as subcomponents
- **Reports** composed of pandas summary + FPDF (or similar)

6.6 Uses/Interactions

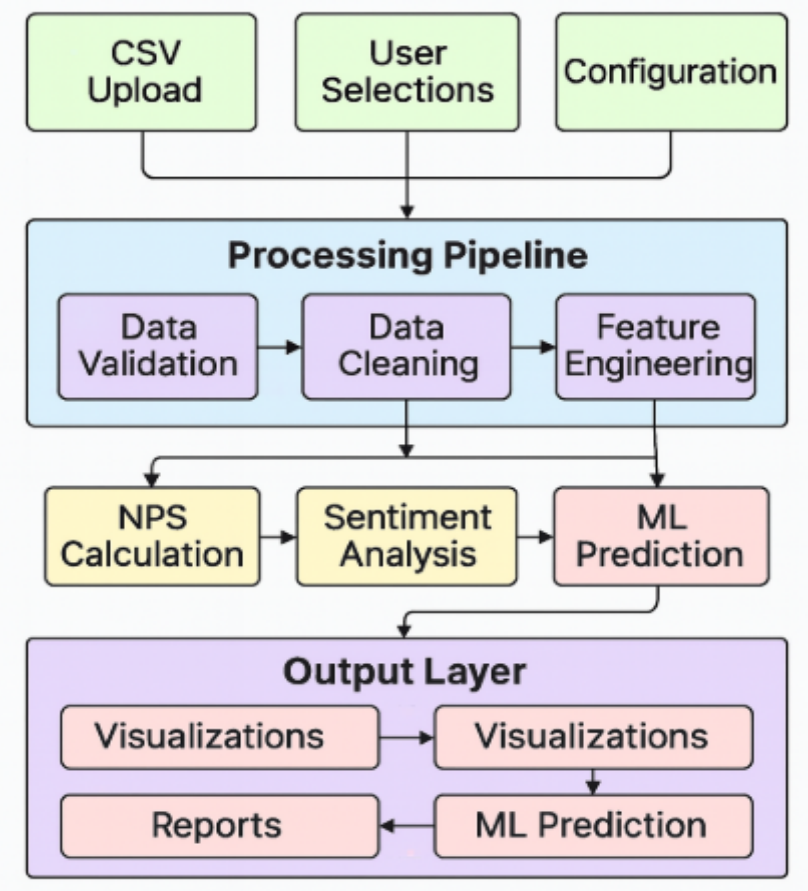

Figure 1.14: *Data flow diagram*

- **Input Layer**
  - **CSV Upload**, **User Selections**, and **Configuration** act as entry points.
  - Business users upload raw CSV files containing NPS and feedback data.
  - User selections (e.g., filters, time ranges) and configuration parameters (e.g., thresholds, model settings) are captured to guide the processing pipeline.
  - These inputs converge and flow into the **Processing Pipeline** for validation and transformation.

- **Processing Pipeline**
  - **Data Validation** checks the structure and schema of the uploaded CSV files, ensuring required fields and formats are present.

    - Once validated, data flows into **Data Cleaning** where missing values are handled, duplicates are removed, and data types are standardized.
    - Cleaned data is then passed to **Feature Engineering** to derive additional attributes, such as rolling averages or sentiment-ready tokens (Sadik, M. R et al, 2025).
    - The output of this pipeline is a curated dataset ready for analysis.
- **Analysis Engine**
    - From Feature Engineering, the data branches into three analytic processes:
        - **NPS Calculation** computes Net Promoter Scores from the validated feedback.
        - **Sentiment Analysis** interprets text feedback for positive, neutral, or negative sentiments.
        - **ML Prediction** uses trained models to forecast future satisfaction trends or classifies feedback.
    - These components may run in parallel, producing complementary insights that converge in the output layer.
- **Output Layer**
    - Results from the analysis engine flow into **Visualizations**, where charts, dashboards, and trend lines are generated.
    - **Reports** are then compiled from these visualizations and analytic outputs.
    - Final deliverables such as **Reports**, **Visualizations**, and any notifications can be exported or shared with stakeholders.

6.7 Resources

- CPU for local processing
- Memory for large CSVs
- Internet
- Streamlit session state
- Temporary storage for PDFs

6.8 Processing

- **Application Orchestrator**
  - **Page Routing Algorithm:** Implements constant-time O(1) routing using Streamlit's session state and radio/page selector logic.
  - **State Management:** Manages session-scoped variables, per-component states, and globally shared configurations.
  - **Exception Handling:** Includes graceful fallbacks for import or initialization failures (e.g., missing models, broken imports), ensuring continued UI availability.
- **Data Preprocessing Component**
  - **CSV Loading Algorithm:** Uses pandas for linear-time O(n) file parsing and validation, where n is the number of rows.
  - **NPS Classification Algorithm:** Extracts and classifies scores using regular expressions and conditionals in O(n) time.
  - **NPS Calculation Algorithm:** Applies the standard NPS formula via vectorized filtering and counting.
  - **Exception Handling:**
    Captures errors in file format, schema mismatches, and missing fields, with user-friendly feedback messages.

- **NLP Analysis Component**
  - **Sentiment Analysis Algorithm:** Applies Text Blob's rule-based sentiment scoring per comment in O(n) time.
  - **Word Cloud Generation:** Batch processing of tokens with memory-aware filtering for display optimization.
  - **Sentiment Distribution:** Aggregates and visualizes sentiment categories via statistical grouping and rendering.
  - **Exception Handling:** Handles tokenization failures, Unicode issues, and external API instability gracefully.
- **ML Model Trainer Component**

    - **TF-IDF Model Training:** Performs training with the average document length, and the number of features.
    - **Optimal Feature Selection:** Uses cross-validation to identify optimal feature dimensionality.
    - **Transformer Model Training:** Supports batch-based training with memory-efficient tokenization and encoding.
    - **Model Evaluation:** Produces evaluation metrics such as accuracy, F1-score, and confusion matrix for model comparison.
- **Visualization Component**
    - **NPS Donut Chart:** Generates interactive Altair-based donut charts in O(n) time.
    - **Time Series Trend Chart:** Plots monthly NPS trends with dynamic axes and tooltips.
    - **Exception Handling:** Captures rendering failures (e.g., malformed data) with default chart fallback or skip logic.
- **Alert System Component:**
    - **Discord Integration:** Sends formatted notifications via webhook with timeout and HTTP status validation.
    - **Exception Handling:** Employs silent failure with internal logging and retry logic for transient network errors.

6.9 Interface/Exports

- Frontend exposes buttons and file upload
- Backend exposes internal functions:
    - process_data(df)
    - calculate_nps_sentiment(df)
    - generate_wordcloud(text_list)
    - export_report(df)

6.10 Detailed Subsystem Design

Not applicable. Detailed design elements are already included through visual architecture diagrams (Section 5) and component breakdowns (Sections 6.1 to 6.9). No additional code-level pseudocode or class diagrams are required.

7. Glossary

Term	Definition
SEP401	Software Engineering Principles subject we are studying
NPS	Net Promoter Score, a metric to gauge customer loyalty and satisfaction.
ML	Machine Learning: computer algorithms that learn patterns from data.
MVP	Minimum Viable Product: the simplest version of a product delivering usable value.
API	Application Programming Interface: a set of functions enabling software to communicate.
Altair	A Python library for declarative statistical visualization.
CSV	Comma-Separated Values: a text format for tabular data storage.
Dashboard	The main visual interface in Streamlit where charts, tables, and results are displayed.
Data Frame	A two-dimensional labeled data structure used in pandas for data manipulation.
ETL	Extract, Transform, Load: the process of moving and transforming data.
Regression Model	A type of ML model predicting numeric values based on input features.
Scikit-learn	A Python library for machine learning tasks.
Sentiment Analysis	A process of determining emotional tone from text data.
Streamlit	A Python framework for creating data-driven web applications.
UI	User interface
TF-IDF	Statistical method for text feature extraction.

8. Bibliography

Hugging Face. (n.d.). *Transformers documentation*. Retrieved July 28, 2025, from https://huggingface.co/docs/transformers/en/index

Li, H., Yu, L., & He, W. (2019). *The impact of GDPR on global technology development. Journal of Global Information Technology Management*, 22(1), 1-6.

Privacy Act 1988**.** (n.d.), Federal Register of Legislation. Australian Government. Retrieved July 28, 2025, from https://www.legislation.gov.au/C2004A03712/2019-08-13/text

Richards, M. (2022). Software Architecture Patterns (2nd ed.).

Sadik, M. R., Himu, U. H., Uddin, I. I., Abubakkar, M., Karim, F., & Borna, Y. A. (2025, April). *Aspect-Based Sentiment Analysis of Amazon Product Reviews Using Machine Learning Models and Hybrid Feature Engineering*. In 2025 International Conference on New Trends in Computing Sciences (ICTCS) (pp. 251-256). IEEE.

Streamlit. (n.d.). *Streamlit documentation*. Retrieved July 28, 2025, from https://docs.streamlit.io/

Zhang, Y., Fan, S., Hui, H., Zhang, N., Li, J., Liao, L., ... & Wu, Y. (2025). *Privacy protection for open sharing of psychiatric and behavioral research data: ethical considerations and recommendations*. Alpha Psychiatry, 26(1), 38759.