Text Analyzer System All-In-One Project Documentation

This document outlines the comprehensive design, architecture, features, and testing strategy for the Text Analyzer System, a robust Windows desktop application.

1. Project Brief

Overview

A Windows desktop application that analyzes text documents and produces detailed statistics about word usage, sentence structure, paragraph organization, and readability.

Purpose

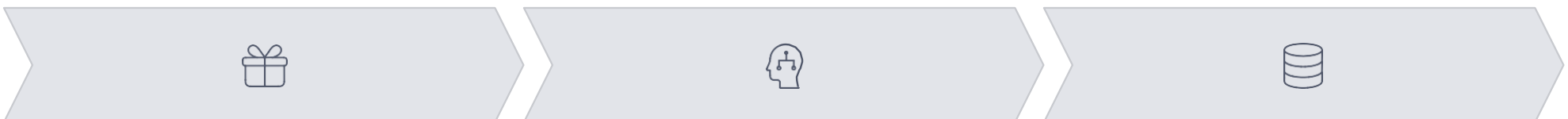
To help students, writers, teachers, and professionals understand text complexity and improve readability through instant, automated analysis.

In One Sentence

A Windows desktop application that analyzes text and provides detailed statistics about word usage, sentence structure, and readability level with multiple export options.

2. System Architecture

The system follows a three-layer architecture designed for clarity, scalability, and testability.



Presentation Layer (C# WinForms)

Handles user interaction and result visualization.

Business Logic Layer (F#)

Performs validation, tokenization, analysis, and statistics calculation.

Data Access Layer

Responsible for file loading and export operations.

3. Key Features

Multiple input options

Typing, file loading, clipboard paste.

Comprehensive text analysis

Includes readability scoring.

Export results

To JSON, TXT, CSV, or clipboard.

Offline desktop application

No internet dependency.

4. Testing Strategy (Detailed)

Testing is a core component of the Text Analyzer System. The application uses a multi-level testing strategy that validates individual functions, module interaction, edge cases, and real user behavior. All tests are automated using the xUnit framework.

4.1 Input Handler Tests

These tests validate the first line of defense in the system: text input validation. Their purpose is to ensure that the analysis pipeline never receives invalid or meaningless input.

- Reject empty text: Ensures an empty string returns an Error result.
- Reject whitespace-only text: Ensures strings containing only spaces, tabs, or newlines are rejected.
- Accept valid text: Confirms that normal textual input is accepted and wrapped in an Ok result.
- Handle surrounding whitespace: Ensures that leading and trailing whitespace does not cause validation failure.

4.2 Tokenizer Tests

Tokenizer tests verify that raw text is correctly decomposed into structured linguistic units. Correct tokenization is essential because all subsequent analysis depends on it.

- Paragraph splitting based on multiple line breaks.
- Sentence splitting using punctuation such as '.', ',', and '?'.
- Word extraction accuracy.
- Normalization through lowercase conversion.
- Removal of punctuation from words to avoid false tokens.

4. Testing Strategy (Detailed) - Continued

4.3 Metrics Calculator Tests

These tests validate the numerical correctness of linguistic metrics produced by the system.

- Average word length calculation using word character counts.
- Average sentence length based on word distribution.
- Correct identification of the longest word.
- Correct identification of the shortest word.
- Validation that readability score always falls between 0 and 100.

4.4 Frequency Analyzer Tests

Frequency analysis tests ensure that word usage statistics are accurate and consistent.

- Counting unique words using set/dictionary logic.
- Extraction of the top N most frequent words.
- Ensuring frequency results are correctly ordered.

4.5 Statistics Builder Tests

Statistics Builder tests validate the aggregation of all metrics into a single coherent statistics model.

- Verification that all statistical fields are populated.
- Validation of numeric values being greater than zero where applicable.
- Correct handling of minimal input such as a single-word text.

4.6 Edge Case Tests

Edge case tests focus on unusual but realistic inputs that could break poorly designed systems.

- Multiple consecutive line breaks.
- Excessive punctuation.
- Presence of numbers within text.
- Extremely short input text.

4. Testing Strategy (Detailed) - Integration & UI

4.7 Integration Tests

Integration tests verify that all system modules work together correctly as a complete pipeline.

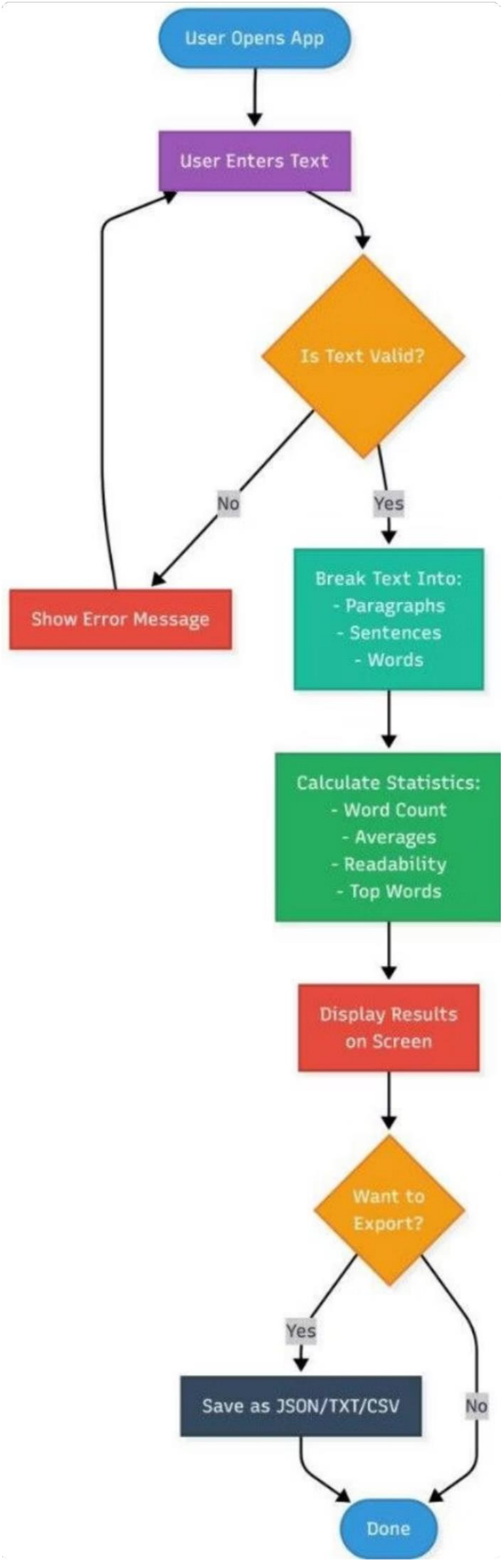
- Validation → Tokenization → Statistics generation.
- Correct paragraph, sentence, and word counts.
- Ensuring no module breaks downstream processing.

4.8 UI Automation Tests

UI automation tests simulate real user interaction with the Windows Forms interface. All UI tests are executed on a Single Threaded Apartment (STA) thread, which is required by WinForms.

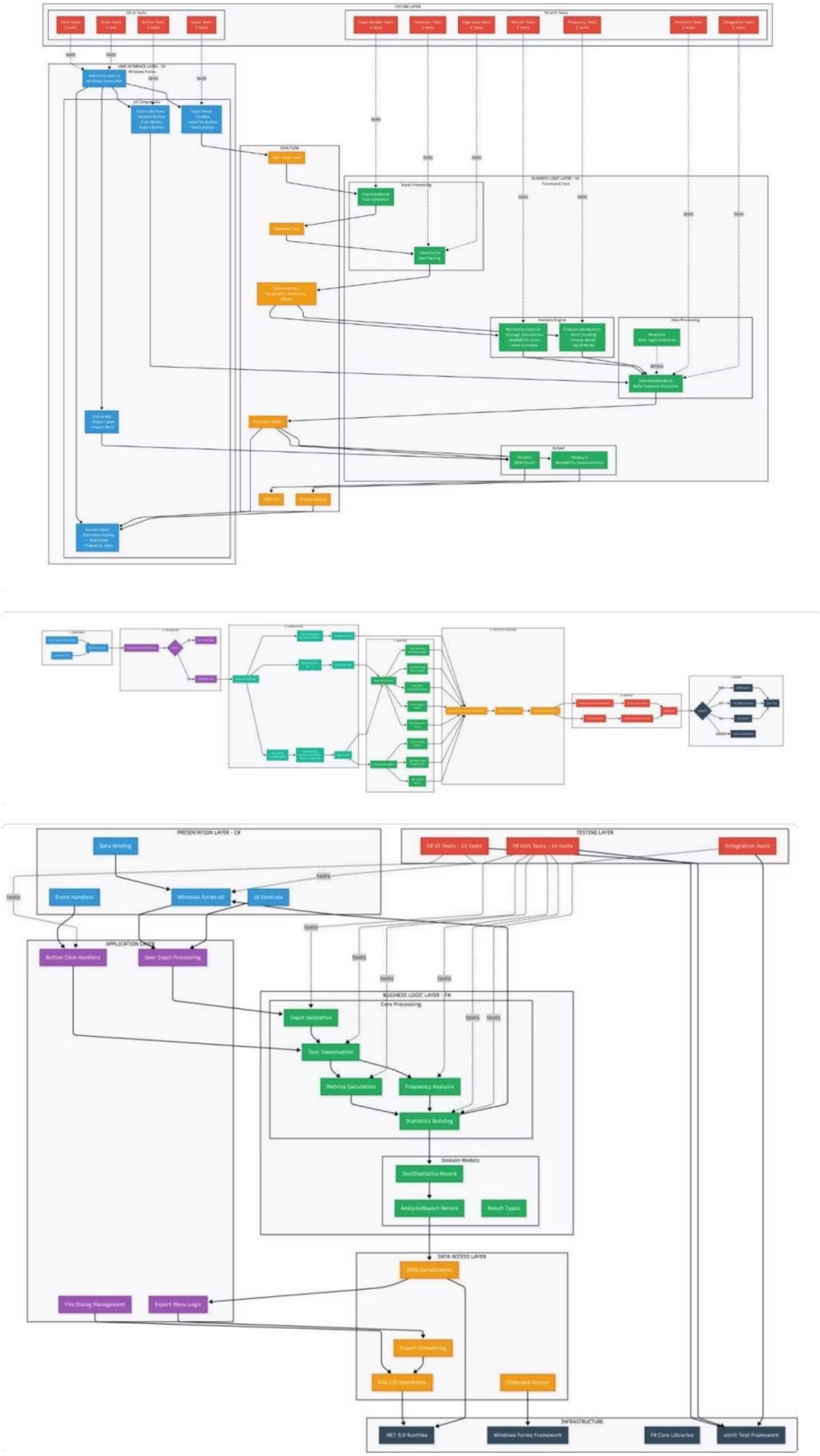
- Form initialization and window properties.
- Button state validation (enabled/disabled).
- Text input and live word count updates.
- Analyze workflow and result display.
- Clipboard integration and file loading.
- Reset and clear operations.

6. Diagrams: User Workflow

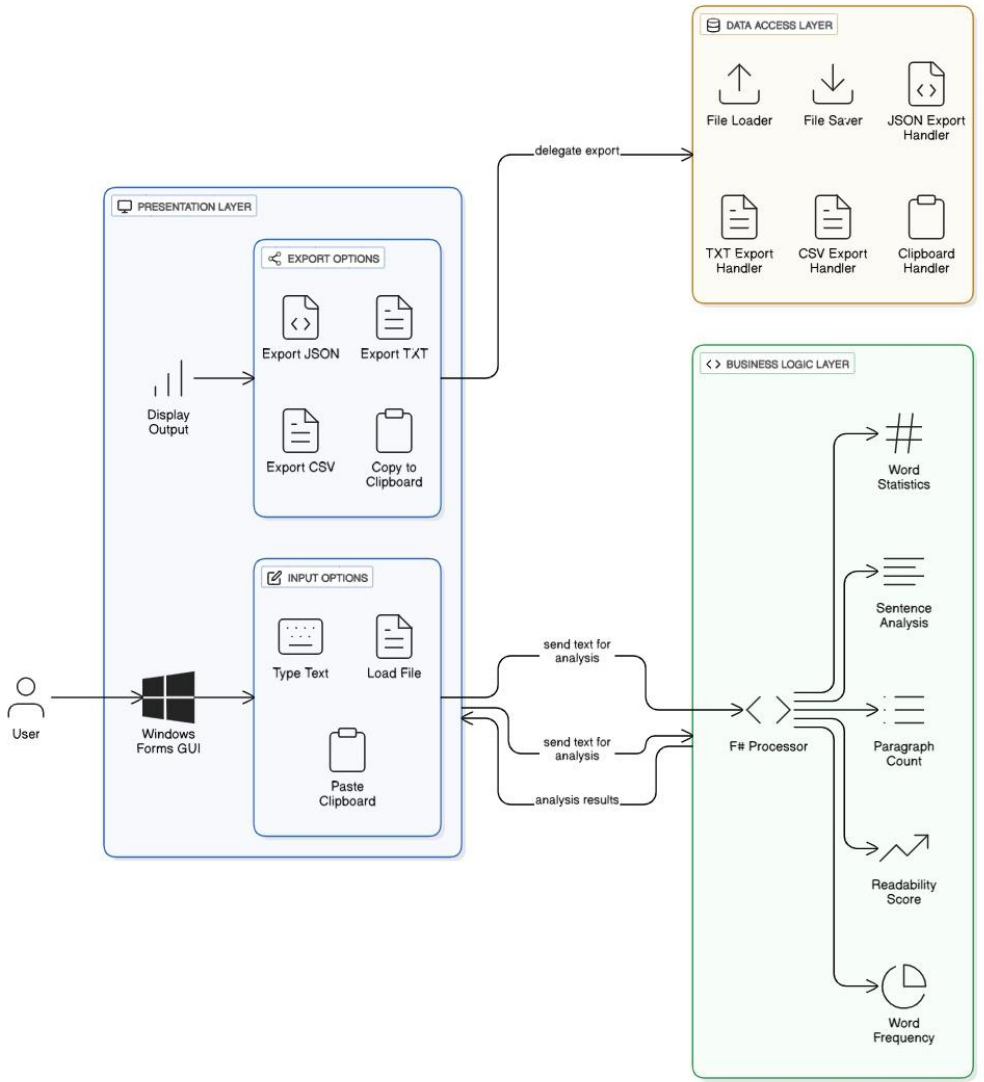


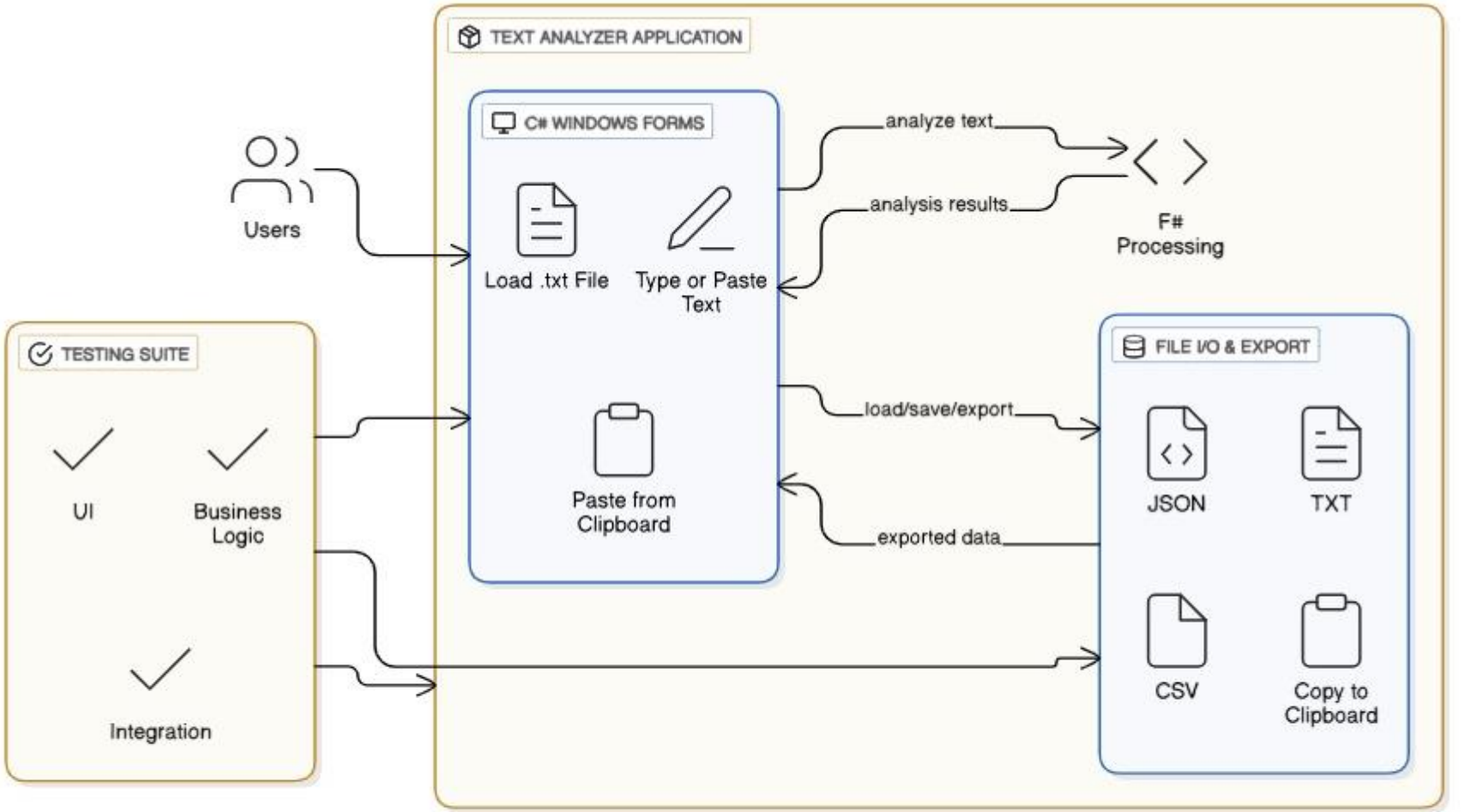
This flowchart illustrates the typical user journey through the Text Analyzer System, from opening the application to exporting results.

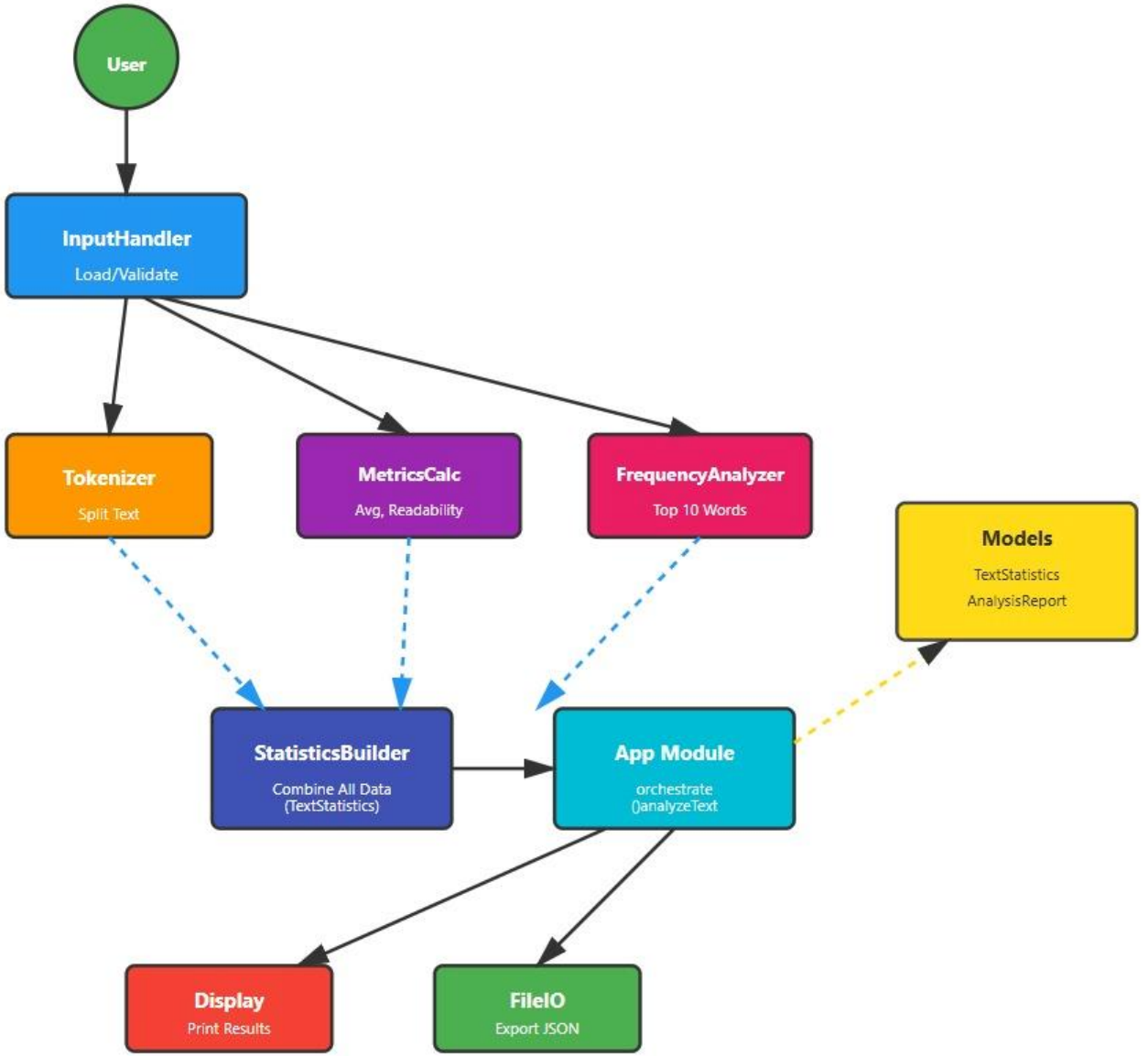
6. Diagrams: System Architecture & Testing Layers

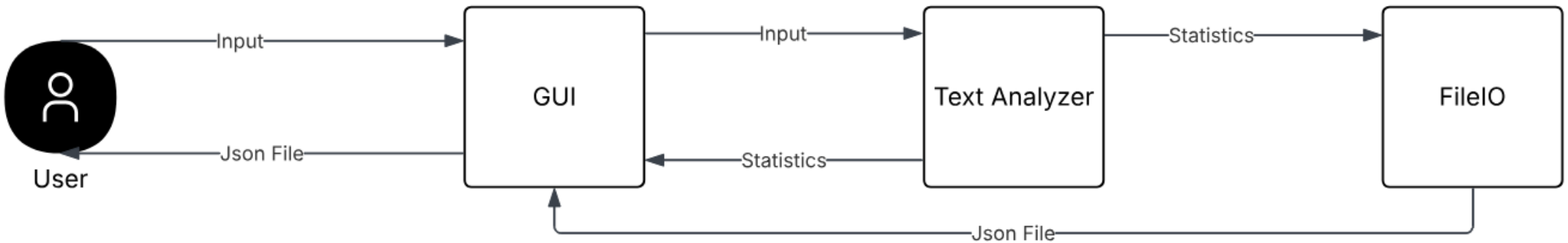


This set of diagrams provides a comprehensive overview of the system's layered architecture and the corresponding testing layers, highlighting the separation of concerns and dependencies.


DATA ACCESS LAYER
File Loader
File Saver
JSON Export Handler
TXT Export Handler
CSV Export Handler
Clipboard Handler
delegate export
PRESENTATION LAYER
EXPORT OPTIONS
Export JSON
Export TXT
Export CSV
Copy to Clipboard
Display Output
INPUT OPTIONS
Type Text
Load File
Paste Clipboard
User
Windows Forms GUI
send text for analysis
send text for analysis
analysis results
BUSINESS LOGIC LAYER
F# Processor
Word Statistics
Sentence Analysis
Paragraph Count
Readability Score
Word Frequency


TEXT ANALYZER APPLICATION
C# WINDOWS FORMS
Users
Load .txt File
Type or Paste Text
Paste from Clipboard
analyze text
analysis results
F# Processing
load/save/export
exported data
FILE I/O & EXPORT
JSON
TXT
CSV
Copy to Clipboard
TESTING SUITE
UI
Business Logic
Integration


User
InputHandler
Load/Validate
Tokenizer
Split Text
MetricsCalc
Avg, Readability
FrequencyAnalyzer
Top 10 Words
Models
TextStatistics
AnalysisReport
StatisticsBuilder
Combine All Data
(TextStatistics)
App Module
orchestrate
()analyzeText
Display
Print Results
FileIO
Export JSON



5. Conclusion

The Text Analyzer System employs a robust and professional testing strategy that ensures correctness, stability, and usability. By combining functional unit tests, edge case validation, integration testing, and UI automation, the system meets high academic and professional software engineering standards.

The multi-level testing approach guarantees a reliable and high-quality application for all users.