**Functional Requirements Specification (FRS)**

**Project: Interactive Data Management Tool**

**Version:** 1.0

**Date:** 10 September 2025

**Developer:** Jefri Thimoathi Johnson

**1. Introduction**

**1.1 Purpose**

This document outlines the functional and non-functional requirements for an interactive data management web application. The purpose of this application is to serve as a technical demonstration of full-stack development capabilities. It will allow users to dynamically add data to a table, visualize that data in a chart, process an uploaded file using Optical Character Recognition (OCR), and generate summary reports.

**1.2 Scope**

The scope of this project is to develop a single-page web application that fulfills all the requirements detailed in this document. The application will be a self-contained, client-side solution. For functionalities that would typically require a paid or server-side service (such as advanced OCR or backend data storage), the implementation will demonstrate the front-end capability, supplemented with documentation and sample code explaining the architecture for a full-scale implementation.

**1.3 Audience**

This document is intended for the hiring committee and technical reviewers to evaluate the development skills of the candidate. It will serve as the basis for the development work and the benchmark against which the final product is assessed.

**2. Overall Description**

**2.1 Product Perspective**

The application will be a standalone, client-side web tool. It will not require a backend database or server-side authentication for its core functionality. It is designed to showcase proficiency in front-end technologies, state management, data manipulation, and the integration of third-party JavaScript libraries.

**2.2 User Characteristics**

The intended user is a technical evaluator who will be interacting with the application's features to assess their implementation, functionality, and the quality of the underlying code.

**2.3 Assumptions and Dependencies**

* The application must run in modern web browsers (e.g., Chrome, Firefox, Edge).
* All development will be done using HTML, CSS, and JavaScript, potentially with a front-end framework (e.g., React, Vue, Svelte).
* For OCR and PDF generation, client-side JavaScript libraries (e.g., Tesseract.js, jsPDF) will be utilized to meet the requirements without a backend.

**3. Specific Requirements**

**3.1 Functional Requirements**

**FR-001: Data Table Display** The system shall display a data table as the central component of the user interface.

**FR-002: Add Table Row** The system shall provide a user-clickable button labelled "Add New Line" or similar. Upon clicking this button, a new, empty, editable row shall be appended to the data table.

**FR-003: In-Table Data Entry** Each row in the data table shall contain multiple input fields for data entry. At least one of these fields must be configured to accept numerical cost figures (e.g., type="number").

**FR-004: Data Visualization Chart** The system shall display a chart or graph (e.g., bar chart, pie chart) that visually represents a subset of the data from the table (e.g., the cost figures).

**FR-005: Dynamic Chart Updates** The chart shall be linked to the data in the table and must update in real-time to reflect any changes, including the addition of new rows or modification of existing data.

**FR-006: File Upload Functionality** The system shall provide a file upload control (e.g., a button) that allows the user to select a file from their local machine.

**FR-007: OCR Data Extraction** The system shall be capable of processing an uploaded PDF file using an Optical Character Recognition (OCR) engine to extract textual data from it.

**FR-008: Populate Table from OCR** The system shall use the data extracted via the OCR process to automatically populate the input fields of a new row in the data table.

**FR-009: Audit Summary Generation** The system shall provide a user-clickable button that, when activated, calculates and displays an audit summary. The summary must include key calculated metrics from the table's data, such as total cost, average cost, or number of entries.

**FR-010: PDF Report Generation** The system shall provide a user-clickable button that generates a PDF document containing the current data from the table. The generated PDF should be made available to the user as a download.

**3.2 Non-Functional Requirements**

**NFR-001: UI Design Flexibility** The specific visual design, including colour palettes, typography, and layout, is not prescribed and is left to the developer's discretion. The interface should, however, be clean, intuitive, and user-friendly.

**NFR-002: Client-Side Operation** The application must function entirely on the client-side, without requiring a connection to a backend server for its core features.

**3.3 External Interface Requirements**

**EIR-001: Test PDF Document** A simple, machine-readable PDF document shall be created and provided alongside the project submission. This document will serve as the test file for the OCR functionality (FR-007, FR-008).

**4. Appendix**

**4.1 Approach for Server-Dependent Features**

For any requirement that cannot be fully implemented on the client side or would necessitate a paid service, the following approach will be taken:

1. The user interface element (e.g., button, upload form) will be fully implemented.
2. The client-side JavaScript code to handle the user interaction and data preparation will be written.
3. A placeholder function or API call will be created to represent the server-side interaction.
4. A separate document or a well-commented code block will be provided to explain the required backend architecture, including server-side code samples (e.g., a Node.js endpoint) and the steps needed to make the feature fully operational.

**4.2 Fault Finding / Debugging Methodology**

**Scenario:** The data visualization chart updates based on all lines of the table apart from one specific line.

**Debugging Steps:**

1. **Replication:** Consistently reproduce the bug by performing the exact steps that lead to the error. Confirm that the issue is tied to a specific row or the data entered into it.
2. **Data Inspection:** Examine the data within the problematic row directly. Check for data type mismatches (e.g., a string in a number field like "100a" instead of 100), special characters, empty or null values, or formatting issues that might cause a parsing error.
3. **State Inspection:** Use browser developer tools (e.g., React DevTools, Vue DevTools, or standard console logging) to inspect the application's state management. Verify that the data for the problematic row is correctly stored in the state array/object that feeds the chart component.
4. **Code Tracing:** Place breakpoints or console.log() statements in the data processing logic that prepares the dataset for the charting library. Trace the data flow from the table input to the function that renders the chart, observing how the problematic row's data is handled at each step.
5. **Hypothesis and Verification:** Form a hypothesis based on the observations (e.g., "The cost parsing function fails if the input contains a comma"). Test this hypothesis by manually correcting the data to see if the chart renders correctly.
6. **Resolution:** Implement a robust fix. This could involve adding input validation to sanitize user input, improving the data parsing logic to handle edge cases, or ensuring proper data type casting before passing the data to the chart library.
7. **Verification:** After applying the fix, re-test the original scenario and other edge cases to ensure the bug is resolved and no new issues have been introduced.