**Functional Design Specification (FDS)**

**Project: Interactive Data Management Tool**

**Version:** 1.0

**Date:** 10 September 2025

**Developer:** Jefri Thimoathi Johnson

**1. Introduction**

**1.1 Purpose**

This document provides a detailed functional design for the Interactive Data Management Tool. It elaborates on the requirements outlined in the Functional Requirements Specification (FRS) by defining the system's architecture, component breakdown, data structures, and user interface design. The primary objective is to serve as a technical blueprint for the development of the application.

**1.2 Scope**

This design specification covers all functionalities of the application, which will be developed as a client-side, Single Page Application (SPA). It will detail the implementation strategy for dynamic data tables, real-time charting, file handling, client-side Optical Character Recognition (OCR), and report generation. The design will focus on creating a modular, maintainable, and efficient codebase.

**2. System Overview and Architecture**

**2.1 Architecture**

The application will be architected as a modern, client-side Single Page Application (SPA). This approach ensures a fluid and responsive user experience by dynamically updating the view in the browser without requiring full page reloads. All data processing and state management will be handled within the browser.

**2.2 Technology Stack**

* **Front-End Framework:** The application will be built using **React**, a declarative and component-based JavaScript library, to facilitate the creation of a dynamic and stateful user interface.
* **State Management:** React's built-in Hooks (useState, useReducer) will be used to manage the application's state, including the data within the table. This ensures that changes in one component are efficiently propagated to others.
* **Styling:** A utility-first CSS framework like **Tailwind CSS** will be used to create a clean, modern, and responsive design efficiently.
* **Data Visualization:** The **Chart.js** library will be integrated to render the dynamic data chart, chosen for its ease of use, flexibility, and excellent performance.
* **OCR Engine:** Client-side OCR will be implemented using **Tesseract.js**, which allows for text recognition from images and PDFs directly in the browser, eliminating the need for a server-side component.
* **PDF Generation:** The **jsPDF** library will be used to generate PDF reports on the client side, providing a straightforward API to create and save documents from the table data.

**3. Detailed Component Design**

The application will be broken down into a series of reusable React components, each with a distinct responsibility.

**3.1 App Component (Main Container)**

This will be the root component that orchestrates the entire application. It will be responsible for managing the primary application state (the array of table data) and passing data and callback functions down to its child components as props.

**3.2 DataTable Component**

This component will render the main data table. It will receive the table data as a prop from the App component and map over the array to render a TableRow for each entry. It will also contain the "Add New Line" button, which will trigger a state update in the App component to add a new object to the data array.

**3.3 TableRow Component**

A single row within the table. This component will contain the input fields for data entry (description, cost, etc.) and the file upload button. It will manage its own local state for the input values and will call an update function (passed from App) whenever the data is changed, ensuring the central state is always synchronized.

**3.4 ChartComponent Component**

This component will receive the table data as a prop. It will be responsible for processing this data into the format required by Chart.js (e.g., labels and datasets). It will use a useEffect hook to detect any changes to the incoming data and will trigger a re-render of the chart to reflect the latest state.

**3.5 ActionControls Component**

This component will house the primary action buttons: "Produce Audit Summary" and "Generate PDF Report." Each button will have an onClick handler that executes the relevant data processing and reporting logic.

**3.6 AuditModal Component**

A modal dialog component that will be displayed when the audit summary is requested. It will receive the calculated summary data as props and present it in a clean, readable format.

**4. Data Design**

**4.1 State Management**

A central state object, managed within the App component, will be the single source of truth for the table data.

**4.2 Data Structure**

The core data will be an array of JavaScript objects, where each object represents a single row in the table. The structure of each object will be as follows:

{

id: string, // A unique identifier (e.g., UUID)

description: string, // Textual description for the item

cost: number, // The numerical cost figure

uploadedFile: File | null // The file object from the upload, if any

}

**5. Functional Implementation Details**

**5.1 OCR Workflow**

1. The user clicks the file upload button within a TableRow.
2. An <input type="file" accept=".pdf"> element will capture the file selection.
3. The onChange event handler will receive the File object.
4. This file object will be passed to a Tesseract.js worker.
5. The system will display a loading indicator while OCR is in progress.
6. Upon successful recognition, the Tesseract.recognize() method will return a result object containing the extracted text.
7. A parsing function will process this text to identify relevant data (e.g., using regular expressions to find cost figures).
8. The extracted data will be used to update the state for the corresponding row, populating the input fields.

**5.2 PDF Report Generation**

1. The user clicks the "Generate PDF Report" button.
2. The onClick handler initializes a new jsPDF instance.
3. The function retrieves the current table data from the application's state.
4. It iterates through the data array, using jsPDF methods (e.g., doc.text(), doc.autoTable()) to add headers and row content to the document.
5. Finally, the doc.save('report.pdf') method is called, which prompts the user to download the generated PDF file.

**6. User Interface (UI) and User Experience (UX)**

The UI will be designed with a clean, minimalist aesthetic, focusing on clarity and ease of use. The layout will be a single-view interface with the main components arranged logically: action buttons at the top, followed by the chart, and then the main data table. The design will be fully responsive to ensure usability across different screen sizes, from mobile devices to desktops. Interactive elements will have clear visual feedback (e.g., hover states, loading indicators) to create an intuitive user experience.