Internship Report

Hardik Soni 20CS30023 Gmail

Project Overview

During this internship, I developed a comprehensive, responsive **Angular-based application** designed to streamline data import processes by integrating external data sources directly into a *database*. The project aimed to create a **user-centric, highly functional data import system** that supports **CSV** and **text file** imports from diverse sources, including *local file systems, AWS S3 buckets, and Microsoft Azure Blob Storage*.

Drawing inspiration from the *Microsoft Access Data Import Wizard*, the application was designed to deliver a **familiar, intuitive user experience** with a focus on robust functionality. Through this project, I tackled both data management and UI challenges to develop a responsive, secure, and streamlined solution that aligns with professional standards. This approach ensures that users can effortlessly navigate and utilize the application, reducing learning curves while integrating seamlessly with widely-used cloud storage services. The result is a scalable, cloud-compatible data import system ready for deployment in modern, data-driven environments.

Objectives and Goals

The primary objectives of this project are as follows:

- **Seamless Data Importation**: Allow users to *import data from various sources*, including **local files, AWS S3 buckets, and Microsoft Azure**, ensuring broad compatibility and easy data integration.
- **User-Friendly Interface**: Design a **step-by-step import wizard** that closely mirrors the *Microsoft Access Data Import Wizard*, ensuring users can *quickly and accurately* **map, validate, and import** data into the database.
- **Robust Error Handling**: Implement features to detect and handle **data inconsistencies**, ensuring *clean and reliable data entry*.
- **Enhanced Security**: Provide **secure access** to *cloud data sources*, ensuring only *authorized users* can access, import, or modify data.

Features and Functionality

The application will include the following **key features**:

1. File Selection and Source Integration

- **Local Environment**: Users can directly *upload CSV or text files* from their **local device**.
- **AWS S3 Integration**: The application will support *direct imports from AWS S3 buckets*, providing options for **bucket selection**, **file preview**, and **secure authentication**.
- **Microsoft Azure Integration**: Users can connect to *Microsoft Azure* and select files stored in **Azure Blob Storage**, with the system managing all necessary **access credentials and permissions**.

2. Data Format and Configuration Options

Users will be able to configure **import settings** based on the specific format of the external file:

- **CSV (Comma-Separated Values)**:
  - The application will **parse CSV files**, allowing users to **preview data in tabular format** and **map columns** to corresponding database fields.
  - Users will have options for **specifying delimiters, managing headers, and defining text qualifiers**, ensuring flexible handling of *varying CSV formats*.
- **Fixed-Width Text Files**:
  - For *fixed-width files*, the application will provide a **configuration interface for defining column breaks**, ensuring accurate parsing of data that lacks standard delimiters.
  - The interface will allow users to **visually map each field** by setting *start and end positions for each column*.

3. Data Mapping and Transformation

In line with the *Microsoft Access Data Import Wizard*, this application will feature a comprehensive **data mapping process**:

- **Field Mapping**: Users can *map imported fields to target database columns*, with the application automatically **suggesting matches based on field names and data types**.
- **Data Type Validation**: The system will validate data against the **target field type**, identifying potential conflicts and providing options to address them.
- **Data Transformation Options**: Basic transformations, such as *date format adjustments or numeric precision adjustments*, will be available to ensure **data consistency**.

4. Data Validation and Error Handling

To ensure *data accuracy*, the application will include **extensive error-handling capabilities**:

- **Real-Time Validation**: The application will *validate data in real time*, highlighting issues like **missing values, type mismatches, or malformed entries**.
- **Error Reporting**: *Detailed error logs* will be generated, allowing users to **review and resolve issues** before finalizing the import.

5. Preview and Confirmation

Prior to completing the import process, users will have the option to *preview data mappings and validate the results*:

- **Data Preview**: A *full preview of the parsed data* will be available, allowing users to **inspect records** and ensure correct mappings.
- **Confirmation Step**: The final step will include a **confirmation dialog**, summarizing the *import settings, data sources, and estimated record counts*, ensuring users are confident in their selections.

Project Scope and Core Features

The project architecture drew inspiration from the *Microsoft Access Data Import Wizard*, focusing on delivering a **user-centric interface** that simplifies data importation with step-by-step guidance. My main tasks involved:

- **File Source Parsing and Integration**:
  - I explored and implemented various **Node.js packages**, such as *ngx-doc-viewer*, *parquets*, *csv-parser*, and *papaparse*, to handle formats like CSV, fixed-width, and delimited text files. This allowed the application to parse data accurately, maintaining data integrity and avoiding false positives.
  - Designed the file selection functionality to handle data incrementally using a **streaming approach**. This ensured efficient processing of large files by limiting resource usage and reducing bottlenecks, providing a more responsive user experience.
- **Backend Integration with Flask for Secure AWS S3 Access**:
  - Developed a robust **Flask server** to facilitate communication with the *Angular frontend*, managing the generation of **pre-signed URLs** for AWS S3 file access. This implementation provided secure, temporary file links, ensuring that sensitive AWS credentials remained protected and out of reach for frontend exposure.
  - Configured AWS S3 to support handling large files efficiently by leveraging the *Papaparse step-by-step parsing feature*, which kept browser memory usage low and ensured performance consistency across files of various sizes.
- **UI/UX Enhancements for File Mapping and Data Transformation**:
  - Created an intuitive **data mapping interface** that allowed users to map and validate imported fields against database columns, mirroring the *Microsoft Access Data Import Wizard*.
  - Enhanced delimiter detection for delimited files, using a combination of *radio buttons and checkboxes* for user-customizable parsing. Additionally, built-in type detection allowed users to adjust field types dynamically.
  - Implemented a fixed-width file parser with a graphical tool that enabled users to define column breaks within the UI, leveraging *fixed-width-parser* library functionality for accurate data handling.

Technical Implementation and Development Challenges

Key technical challenges involved parsing large files, handling secure authentication, and creating a responsive UI for smooth, multi-platform usage. I addressed these through:

- **Streamlined File Parsing**: For local file selection, developed an incremental reading strategy using `this.setLimit`, optimizing large-file performance. For AWS S3, integrated AWS SDK methods into Flask for pre-signed URL generation.
- **Responsive and Secure Design**: The application utilized a Flask backend to authenticate and manage cloud credentials, ensuring secure and restricted access to cloud files without compromising overall storage security.
- **Enhanced Frontend Usability with Bootstrap and Custom CSS**: Designed a professional, responsive UI, optimizing *processing time and system feedback* to ensure smooth user interactions during file loading and S3 access.