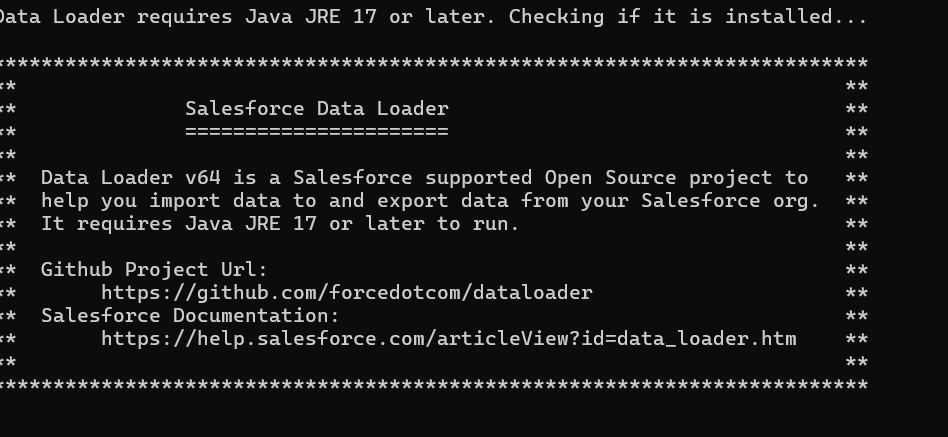
**JOB RECRUITMENT & HIRING MANAGEMENT SYSTEM**

**NAME: KOLLI JAGAN MOHAN RAO**

**Phase 8: Data Management & Deployment**

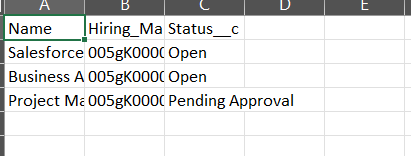
**Objective**: To master the tools and processes for managing data in bulk and deploying the application's components (metadata) from a development environment to another location.

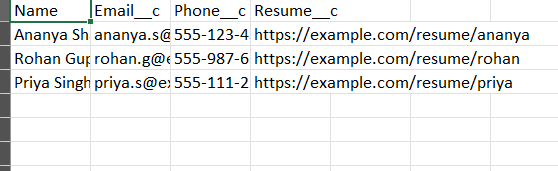
**Data Management Strategy**: The strategy was to populate the custom objects with realistic sample data to facilitate testing and prepare for the final demo. The chosen tool for this was **Salesforce Data Loader**, an external client application designed for high-volume data operations.



**Data Preparation**:

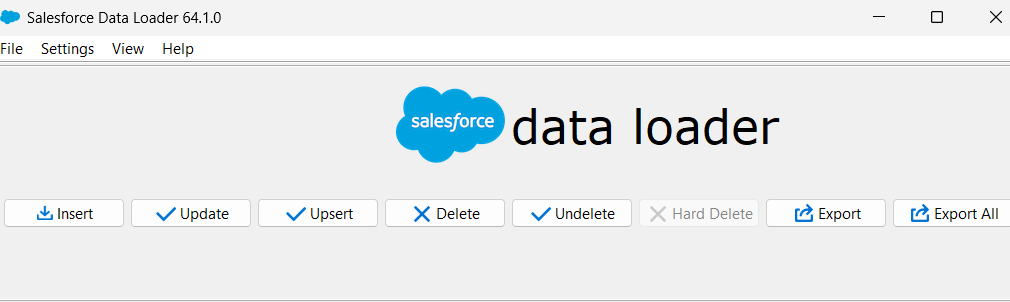
* **File Format**: Data was prepared in **CSV (Comma-Separated Values)** format using a standard spreadsheet application.
* **Candidate Data (candidates.csv)**: A CSV file was created with columns matching the API names of the Candidate\_\_c object's fields (Name, Email\_\_c, Phone\_\_c, Resume\_\_c).
* **Job Data (jobs.csv)**: A second CSV file was created for Job\_\_c records. This step introduced the critical concept of handling relationships in data loading. To populate the Hiring\_Manager\_\_c lookup field, the **18-digit Salesforce ID** of the target User record was required, not their name. This ID was obtained from the User record's URL.

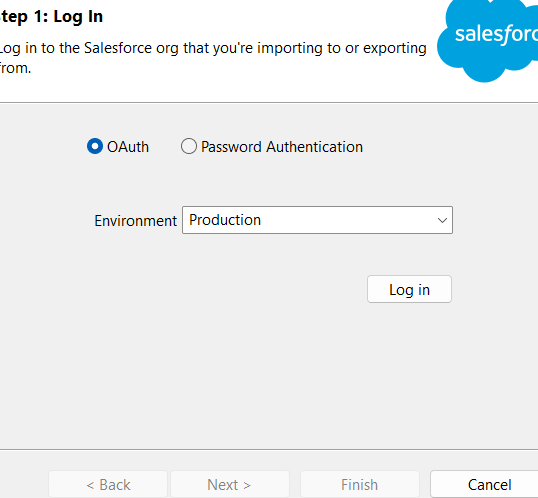


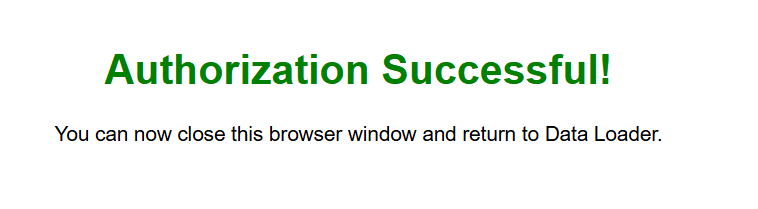


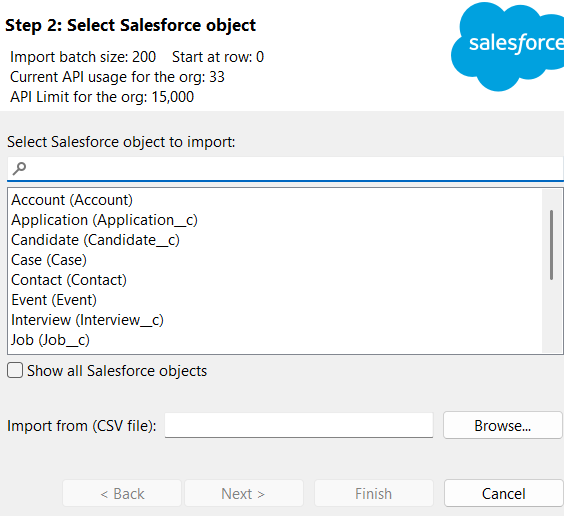
**Data Loading Process**:

* **Tool**: Salesforce **Data Loader** was installed and used.
* **Action**: The **Insert** operation was used for both uploads.
* **Authentication**: The tool was authenticated with the Developer Org using the standard OAuth login flow.
* **Object Mapping**: In the Data Loader wizard, the columns from the CSV files were mapped to the corresponding fields on the Candidate\_\_c and Job\_\_c objects. The **Auto-Match** feature was used to streamline this process.
* **Verification**: After each successful Insert operation, the newly created records were verified directly in the Salesforce UI by navigating to the "Candidates" and "Jobs" tabs.





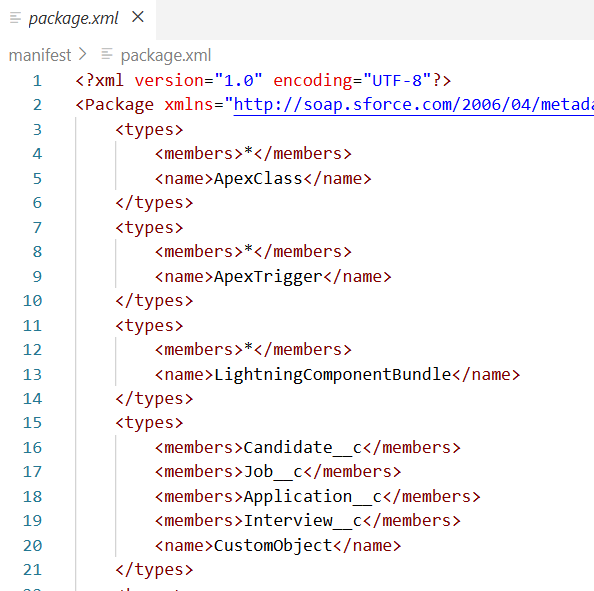




**Deployment Strategy**: The strategy was to use the modern, source-driven development methodology with **Salesforce DX (SFDX)** and **Visual Studio Code**. This approach treats the application's components as files on a local machine, which can be version-controlled and deployed systematically.

**Metadata Management**:

* **package.xml Manifest**: A manifest file, manifest/package.xml, was created to act as a "shopping list" of all the components that make up the application. This file was explicitly configured to include all custom metadata types created during the project, including ApexClass, CustomObject, LightningComponentBundle, Layout, FlexiPage, and CustomApplication.
* **Troubleshooting**: An important lesson was learned when an initial retrieve failed due to a warning. We identified a CustomTab entry for the Interview\_\_c object in the manifest that did not exist in the org. The manifest was corrected by removing the erroneous line, demonstrating the need for precision in deployment manifests.



**Deployment Process (Retrieve/Deploy Cycle)**:

1. **Retrieve**: The **SFDX: Retrieve Source in Manifest from Org** command was executed from within VS Code. This command used the package.xml file to pull down the latest version of all application components from the Salesforce org into the local project folder.
2. **Deploy**: The **SFDX: Deploy Source in Manifest to Org** command was then executed. This took the complete set of metadata from the local project folder and pushed it to the Salesforce org, simulating a deployment to a new environment.

