Job Auto-Filler Chrome Extension - Project Document

Project Title:

Job Auto-Filler Chrome Extension

Project Brief:

This project aims to develop a Chrome extension that automates the process of filling job application forms on job portals by storing and reusing key-value pairs like name, email, salary, etc. The extension will detect forms on websites and automatically input stored data, reducing the time required to apply for jobs. Additionally, if new information is entered by the user, the extension will store the data for future use.

Objective:

To automate the process of filling job application forms by creating a Chrome extension that:

- Autofills job application forms using stored key-value pairs.
- Stores new data entered by the user into a database for future use.
- Uses a Python Flask backend and MySQL for storage and processing.

Technologies Used:

Frontend (Chrome Extension):

- HTML: Structure the extension's interface (optional popup).
- CSS: Styling the user interface.
- JavaScript: Core logic to interact with web pages, detect forms, autofill values, and send/receive data from the backend.
- AJAX: For communication between the frontend (Chrome extension) and the Python backend.

Backend (Python Flask API):

- Python: Backend logic for handling requests and interacting with the database.
- Flask: Python micro-framework to handle HTTP requests from the Chrome extension.
- MySQL Connector: Python library for connecting and interacting with the MySQL database.

Database:

- MySQL: A relational database to store key-value pairs like Name, Salary, Email, etc.
- Pandas (optional): Python library to handle complex data manipulations (if needed).

Project Architecture:

1. Frontend (Chrome Extension):
   - Manifest File: Defines the extension settings, permissions, and background tasks.
   - Content Script: Injected into web pages to detect form fields and interact with them. It will retrieve data from the backend and autofill the fields.
   - Popup (Optional): A simple interface to allow users to view or edit stored key-value pairs.
   - AJAX Requests: Send and receive data from the backend using asynchronous JavaScript requests.

2. Backend (Flask API):
   - REST API Endpoints:
     - /get-data: Retrieves stored values from the database based on key.
     - /store-data: Stores new key-value pairs received from the Chrome extension.
   - MySQL: Used to store and manage the key-value pairs.
   - Flask: Handles requests and provides responses to the frontend (extension).

3. Database (MySQL):

- Table Structure: A simple key-value storage structure in MySQL.
- Schema Example:

```
CREATE TABLE key_value_store (
    `key` VARCHAR(255) PRIMARY KEY,
    `value` TEXT
);
```

- MySQL will handle queries for storing new data or fetching existing data.

Steps to Implement:

1. Set up the Chrome Extension:
   - Create the manifest.json file to define the extension's properties and permissions.
   - Write the content script to:
     - Detect form fields.
     - Autofill form data using values fetched from the backend.
     - Capture new input data and send it to the backend for storage.

2. Set up the Flask Backend:
   - Build the Flask API with two main endpoints:
     - /get-data: To fetch stored values based on the form field name.
     - /store-data: To save new key-value pairs entered by the user.
   - Use MySQL to store the key-value pairs and handle database queries via the MySQL connector.

3. Set up MySQL Database:
   - Create a MySQL database and a key_value_store table to hold key-value pairs.
   - Use the MySQL connector to interact with the database from the Flask API.

4. Integrate Frontend with Backend:

   - Use AJAX in the Chrome extension's content script to send requests to the Flask backend.
   - Autofill form fields with the data retrieved from the backend.
   - Store new data in the backend as the user enters it on the job portal.

5. Testing & Deployment:

   - Test the extension by loading it in Chrome (via chrome://extensions/).
   - Run the Flask API locally and ensure proper communication between the extension and the backend.
   - Deploy the Flask API and MySQL database (optionally) to a cloud service for real-world use.

Future Enhancements:

- Add more field type detection (e.g., checkboxes, radio buttons).
- Enable cross-device data synchronization by deploying the backend and database to a cloud server.
- Provide user authentication and secure data storage using encryption.

Tools & Libraries:

- Flask: To create the backend API.
- MySQL: Database for key-value storage.
- MySQL Connector: Python library to connect Flask to MySQL.
- JavaScript: For Chrome extension logic (content script).
- AJAX: For handling asynchronous requests between the frontend and backend.
- HTML/CSS: For creating the extension's interface.