Job Portal Web Application

1. Abstract

This project presents the design and development of a **job portal web application** that simulates job listings and allows users to search for relevant opportunities. The portal is implemented using the **Flask web framework in Python**, with data stored in a **JSON file**.

Instead of live scraping (restricted by platforms like Indeed), the system uses a **pre-collected JSON dataset** to represent job postings. The application provides users with the ability to filter jobs by title and location, thereby narrowing down the search space to find positions that match their interests.

The frontend interface is built using **HTML and CSS**, styled to resemble professional job portals. Unlike static listings, the application dynamically updates search results based on user queries, making it highly interactive.

This system demonstrates how simple but powerful technologies such as **Flask, JSON, and modern UI design** can be combined to create a fully functional prototype of a job search portal.

---

2. Introduction

The growth of online job platforms has made it challenging for seekers to filter opportunities efficiently. A dedicated job portal acts as a bridge between seekers and recruiters by consolidating listings into a single accessible interface.

This project was developed to showcase a minimal yet effective **job portal prototype** using Flask. The application reads job postings from a **JSON dataset** and allows users to search jobs using **keywords** (e.g., Python Developer, Backend Engineer) and **locations** (e.g., Remote, Bangalore).

The results are displayed in a **grid-based card layout**, which includes job title, company name, and location. The aim of this project is not only to demonstrate Flask in building lightweight web applications but also to highlight the importance of **structured data and user-friendly UI design**.

---

3. System Requirements

Hardware Requirements

- Minimum 4 GB RAM
- Intel i3 Processor or equivalent (i5 or above recommended)

- 500 MB storage for project files and dependencies

Software Requirements

- Operating System: Windows 10/Linux/Mac
- Programming Language: Python 3.8 or higher
- Framework: Flask
- Libraries: JSON (for handling job data), Jinja2 (template rendering)
- Tools: Any IDE (VS Code, PyCharm, Sublime Text), Browser (Chrome/Firefox)
- Package Manager: pip for installing Flask and dependencies

---

4. Project Architecture

The project follows a **client–server architecture** using Flask as the backend server.

- **Data Layer:** Jobs are stored in a `jobs.json` file, which acts as a local database.
- **Application Layer (Flask):** Flask handles user requests, loads data, filters results, and renders responses.
- **Presentation Layer (HTML + CSS):** Job listings and search results are displayed in an interactive interface styled with CSS grid layouts.

Flow of Execution:

1. User accesses the portal via the homepage (`/`).
2. A search query or location is entered in the search form.
3. Flask loads `jobs.json` and filters records based on query/location.
4. Filtered job listings are passed to the template engine (Jinja2).
5. The HTML page renders results in **job card format**.

---

5. Implementation

Folder Structure

```
job-portal/
│── appli.py
│── jobs.json
│── templates/
│     └── jobs-1.html
│── static/
│     ├── style.css
│     └── logo.png
```

Flask Backend (appli.py):

- Loads data from JSON file.
- Handles search queries.

- Filters results by keyword and location.
- Renders results in card format using Jinja.

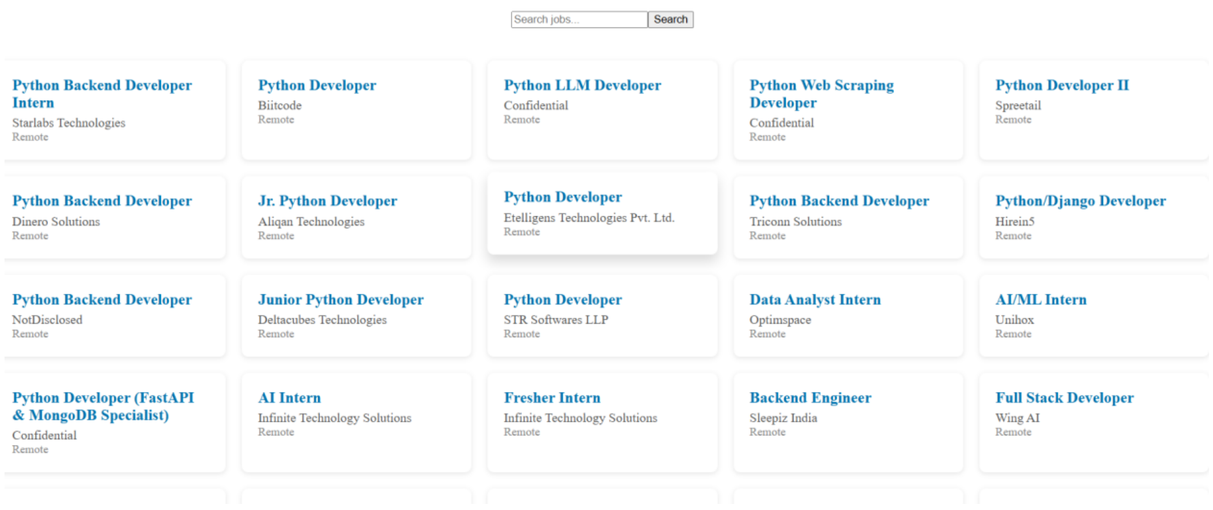
Frontend (HTML + CSS):

- A search form with fields for job title and location.
- Job cards with grid styling for responsiveness.
- Styling inspired by LinkedIn/Indeed color palettes for a professional look.

---

6. Output Description

1. **Homepage with Search Bar**
   - Displays site logo and search form.
   - Users can type job titles (e.g., Python Developer) and location (Remote).
2. **Search Results**
   - Displays filtered jobs as cards.
   - Each card shows job title, company name, and location.
3. **No Results Found**
   - If no job matches, the page shows empty results gracefully.



Python Backend Developer Intern
Starlabs Technologies
Remote

Python Developer
Biitcode
Remote

Python LLM Developer
Confidential
Remote

Python Web Scraping Developer
Confidential
Remote

Python Developer II
Spreetail
Remote

Python Backend Developer
Dinero Solutions
Remote

Jr. Python Developer
Aliqan Technologies
Remote

Python Developer
Etelligens Technologies Pvt. Ltd.
Remote

Python Backend Developer
Triconn Solutions
Remote

Python/Django Developer
Hirein5
Remote

Python Backend Developer
NotDisclosed
Remote

Junior Python Developer
Deltacubes Technologies
Remote

Python Developer
STR Softwares LLP
Remote

Data Analyst Intern
Optimspace
Remote

AI/ML Intern
Unihox
Remote

Python Developer (FastAPI & MongoDB Specialist)
Confidential
Remote

AI Intern
Infinite Technology Solutions
Remote

Fresher Intern
Infinite Technology Solutions
Remote

Backend Engineer
Sleepiz India
Remote

Full Stack Developer
Wing AI
Remote

---

7. Testing

- **Valid Query:** Searching “Python” returns all jobs with “Python” in title.
- **Location Test:** Searching “Remote” shows only remote jobs.
- **Combined Search:** Searching “Python” + “Remote” narrows results further.
- **Invalid Input:** Searching “Doctor” shows no results.
- **Case Sensitivity:** Searching “PYTHON” still returns results (case-insensitive).

---

8. Advantages and Limitations

Advantages

- Lightweight and runs locally with minimal setup.
- JSON-based storage is easy to update.
- Provides keyword and location search.
- Responsive design works on multiple devices.

Limitations

- Jobs data is **static** and must be manually updated.
- No real-time scraping or live integration.
- Lacks advanced filters (salary, job type, experience).
- No authentication or user profile features.

---

9. Future Enhancements

- Integrate real-time scraping using APIs or tools like BeautifulSoup.
- Add database support (SQLite, MySQL) for scalability.
- Implement user login to save jobs and track applications.
- Include advanced filters (salary, experience, job type).
- Deploy on cloud hosting (Heroku, Render, PythonAnywhere).

---

10. Conclusion

This project demonstrates a **functional job portal prototype** using Flask, JSON, and modern frontend design. It highlights how structured data and a clear UI can create an interactive experience for users.

Although currently limited to static job data, the system provides a strong foundation for future integration with **live data sources, databases, and advanced features**.