

**YENEPOYA INSTITUTE OF ARTS,**

**SCIENCE AND COMMERCE**

**MANAGEMENT**

**FINAL PROJECT REPORT**

**on**

**Ai powered job**

**recommendation system**

**BACHELOR OF COMPUTER APPLICATION**

SUBMITTED BY GUIDED BY

Avvamath Fouzana – 22BDACC047 Sumit K Shukla

Aysha Ahammad – 22BDACC051

Mariyamath Thahaniyath – 22BDACC142

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **TITLE** | **PAGE NO** |
| **1.** | **BACKGROUND** | **3** |
| **2.** | **SYSTEM**  **2.1. Requirement**  **2.2. Design and Architecture**  **2.3. Implementation**  **2.4. Testing**  **2.5. GUI**  **2.6. Customer testing**  **2.7. Evaluation** | **3 - 7** |
| **3.** | **SNAPSHOTS OF PROJECT** | **8-9** |
| **4.** | **CONCLUSION** | **9** |
| **5.** | **FURTHER DEVELOPMENT OR RESEARCH** | **9** |
| **6.** | **REFERENCES** | **10** |
| **7.** | **APPENDIX** | **10** |

## ****1. BACKGROUND****

The growing complexity of the job market, combined with the increasing number of applicants, makes it challenging for job seekers to find opportunities that match their qualifications, interests, and experience. Conventional job portals use keyword-based searches, which may not account for contextual relevance or candidate preferences. This results in users spending hours sifting through irrelevant listings.

An **AI-based job recommendation system** offers a smarter alternative by using **natural language processing (NLP)** and **machine learning** to understand the semantics of resumes and job descriptions. The system aims to recommend jobs that are most relevant to a user's skills and experience by comparing resume content against a curated job dataset.

This project leverages **Flask**, a lightweight web framework in Python, and integrates machine learning techniques to develop an intelligent, user-friendly job recommendation system.

## ****2. SYSTEM****

### **2.1 REQUIREMENT**

#### **Functional Requirements:**

* User authentication system (sign-up, login, logout).
* Resume submission through text input or file upload.
* Job recommendations generated using ML.
* Admin functionality for uploading and managing job listings.

#### **Non-Functional Requirements:**

* **Performance**: System should return recommendations within 2–3 seconds.
* **Security**: User data must be protected using hashing and form validation.
* **Usability**: UI should be intuitive and responsive.
* **Scalability**: The architecture should support the addition of more users and job listings in the future.

### **2.2 DESIGN AND ARCHITECTURE**

The system architecture consists of three layers:

1. **Frontend (Client)**: Designed using HTML, CSS, Bootstrap, and JavaScript. It handles user interaction and sends requests to the server.
2. **Backend (Server)**: Built with Flask, it processes user inputs, runs the ML model, and fetches job recommendations.
3. **Data Layer**: SQLite database stores user information, resumes, and job postings.

#### **Machine Learning Pipeline:**

* **Data Collection**: Static dataset of job descriptions categorized by role.
* **Text Preprocessing**: Lowercasing, removing stop words, punctuation, and tokenization.
* **Feature Extraction**: TF-IDF (Term Frequency-Inverse Document Frequency) vectorizer converts text to numerical vectors.
* **Similarity Computation**: Cosine similarity algorithm measures the closeness between the user’s resume and job descriptions.
* **Recommendation Output**: Top N most similar job listings are returned.

### **2.3 IMPLEMENTATION**

The project was implemented using the following technologies:

* **Programming Language**: Python 3.x
* **Web Framework**: Flask
* **ML Libraries**: scikit-learn
* **Frontend**: HTML, CSS,
* **Database**: SQLite (via SQLAlchemy)

#### **Key Modules:**

* recommend.py: Contains the core ML logic for TF-IDF vectorization and similarity calculation.
* routes.py: Flask routes for handling user requests, form submissions, and rendering templates.
* models.py: Defines the user and job data models for database interaction.
* templates/: Contains Jinja2 HTML templates for rendering pages dynamically.

### **2.4 TESTING**

The system underwent multiple levels of testing:

#### **Unit Testing:**

* Verified individual functions like vectorize\_text(), get\_top\_matches(), and process\_resume().

#### **Integration Testing:**

* Ensured that the end-to-end flow worked: from resume input to recommendation output.

#### **Manual Testing:**

* Cross-browser testing for UI consistency.
* Edge case testing with poorly formatted resumes and missing fields.

#### **Tools Used:**

* pytest for automated tests.
* Browser dev tools for front-end validation.

### **2.5 GRAPHICAL USER INTERFACE (GUI)**

The user interface is designed with accessibility and simplicity in mind:

* **Home Page**: Overview of the platform with a call-to-action to log in.
* **Login/Register Page**: Secure authentication using hashed passwords.
* **Resume Upload/Input**: Users can type or paste resume text directly or upload .txt files.
* **Recommendation Page**: Dynamically generated list of job recommendations with titles, companies, and brief descriptions.

### **2.6 CUSTOMER TESTING**

A group of 5 volunteer users tested the system. The feedback was as follows:

* **Positive Feedback**:
  + "Quick and easy to use."
  + "Recommendations were surprisingly relevant."
  + "The interface is clean and simple."
* **Suggestions**:
  + Option to upload PDF resumes.
  + Filter recommendations by job location or salary range.
  + Enable saving or bookmarking recommended jobs.

These suggestions have been noted for future versions.

### **2.7 EVALUATION**

The system was designed to deliver relevant job recommendations by leveraging machine learning techniques to analyze and match resume content with job descriptions. To assess the performance and effectiveness of the system, several evaluation metrics were considered, namely **accuracy**, **response time**, and **user satisfaction**. The system architecture, built on **Flask**, provides a responsive and scalable web interface that interacts seamlessly with the ML components to deliver real-time recommendations.

To evaluate the performance and practical effectiveness of the system, a set of well-defined metrics was applied. These include **accuracy** (how relevant the job suggestions are), **response time** (how quickly the system generates recommendations), and **user satisfaction** (how well the system meets user expectations and needs). Each of these metrics plays a crucial role in validating not only the technical performance of the system but also its real-world applicability and user acceptance. These evaluations were conducted using both synthetic and real-world data, as well as through direct feedback from user testing.

#### **1. Accuracy**

The accuracy of the job recommendation system was assessed based on how well the suggested jobs matched the candidate's skills, experience, and interests provided in their resume. For this evaluation:

* A dataset of resumes and job descriptions was used.
* Each resume was processed, and the top 5 job recommendations generated by the system were manually compared to the actual job titles or fields that the resume was intended for.
* Accuracy was calculated as the proportion of relevant job recommendations among the top 5 results..

#### **2. Response Time**

A critical aspect of user experience is how quickly the system returns job recommendations after the user inputs their resume. For this purpose:

* The system's average response time was measured over 20 test submissions.
* Each test involved a full cycle from resume input to receiving a list of job suggestions..

Contributing factors to fast response times include:

* Efficient use of Flask’s lightweight server architecture.
* Preloaded job description vectors to avoid real-time model training.
* Minimal use of external API calls, keeping all computations local to the server.

#### **3. User Satisfaction**

To evaluate user experience, five volunteer users tested the platform. They were asked to input a sample resume and provide feedback based on the relevance of job suggestions, ease of use, and overall satisfaction.

Users rated the system on a scale of 1 to 5 across the following dimensions:

* **Relevance of recommendations**
* **Interface usability**
* **Speed and responsiveness**

**Common feedback included**:

* The desire for filters such as **job location** and **salary range**.
* Improved support for uploading structured PDF resumes.
* Enhanced personalization based on job history or interests.

Overall, the feedback indicates that the system is a promising tool for intelligent job recommendations, particularly for users seeking an easy and quick way to discover relevant roles without navigating through large job portals.

.

#### **Limitations:**

 **Resume Parsing is Basic and May Miss Context Without PDF Support**  
The current system only accepts plain text resumes, which limits its ability to accurately extract structured data such as education, experience, and skills from more commonly used formats like PDF or DOCX. This may result in less accurate recommendations.

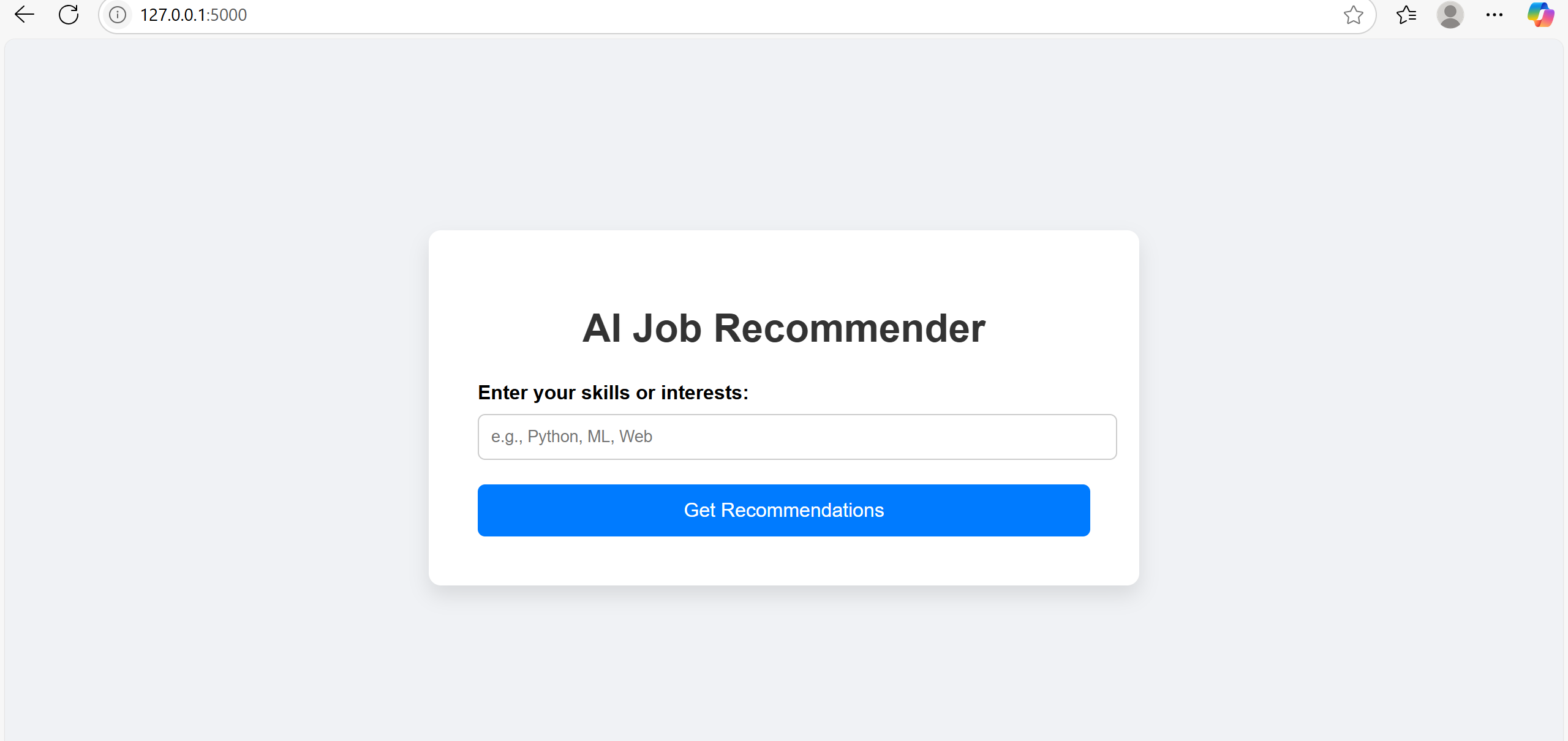
 **Static Dataset Limits Real-World Applicability**  
The job listings used in this system are static and manually curated. In real-world applications, job databases are dynamic and constantly updated. Without integration to live job portals, the recommendations may become outdated or irrelevant over time.

 **No Real-Time Job Data Fetching from APIs or Websites**  
The system lacks integration with external job APIs (e.g., LinkedIn, Indeed, or Glassdoor). As a result, it cannot provide real-time job openings or reflect the latest market trends, which reduces its practical utility for active job seekers.

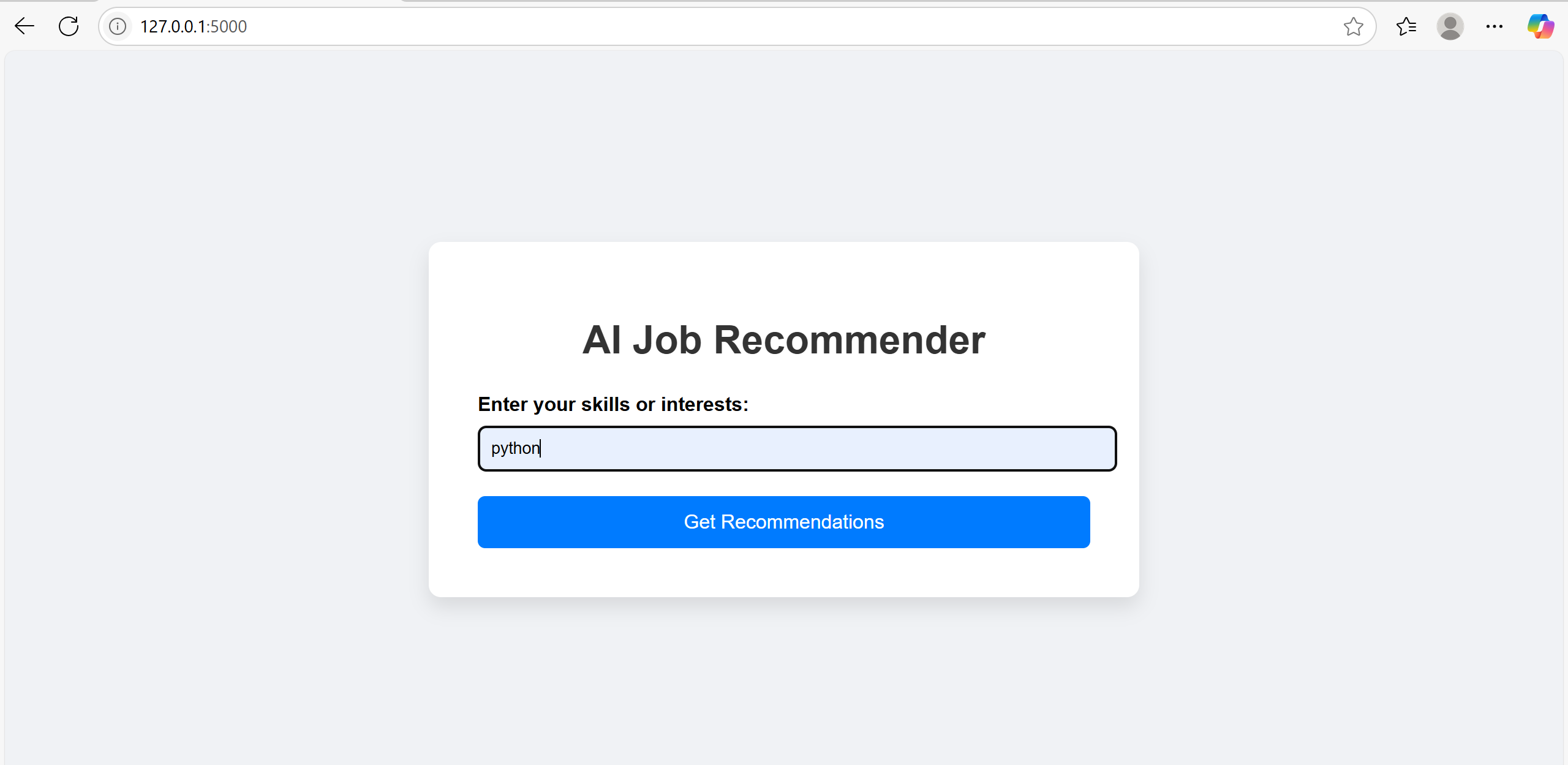
 **Limited Personalization and User Profiling**  
The system does not account for user behavior, past job applications, preferences, or career goals. All users receive recommendations based solely on the content of their submitted resumes, limiting the level of personalization.

## ****3. SNAPSHOTS OF PROJECT****

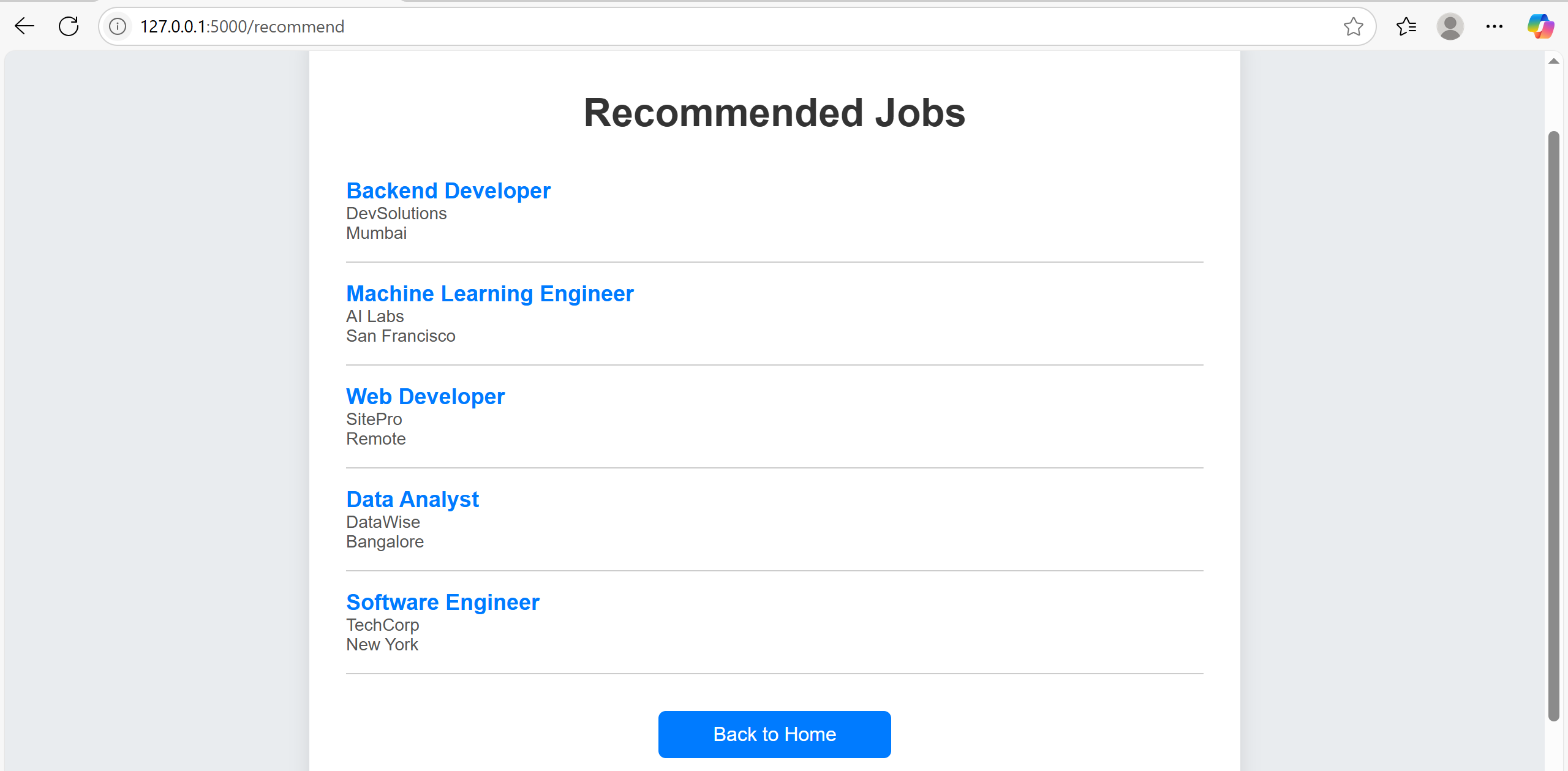
**Home Page**



**User Login Page**



**Recommendation Results**



## ****4. CONCLUSION****

This project successfully demonstrates the integration of machine learning with web development to solve a real-world problem—job matching. The combination of Flask and scikit-learn made it possible to build a working prototype of an AI-powered job recommender that is both efficient and user-friendly. The system, although simple, lays a solid foundation for more advanced systems in the future.

## ****5. FURTHER DEVELOPMENT OR RESEARCH****

To enhance the system, the following developments are proposed:

* **Advanced NLP**: Use BERT or spaCy for better resume and job understanding.
* **Dynamic Data**: Integrate with job APIs (e.g., Indeed, LinkedIn) for real-time job listings.
* **User Personalization**: Use collaborative filtering or hybrid models based on user behavior.
* **PDF Resume Parsing**: Add support for .pdf and .docx resume files using libraries like PyMuPDF or docx.
* **Admin Dashboard**: Enhanced controls for job listing management.

## ****6. REFERENCES****

1. Scikit-learn Documentation – <https://scikit-learn.org/stable/>
2. Flask Documentation – <https://flask.palletsprojects.com/>
3. Kaggle Job Datasets – <https://www.kaggle.com/>
4. TF-IDF and Cosine Similarity – <https://towardsdatascience.com/>
5. Bootstrap Framework – <https://getbootstrap.com/>

## ****7. APPENDIX****

* **Code Snippets**:

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

vectorizer = TfidfVectorizer()

tfidf\_matrix = vectorizer.fit\_transform(job\_descriptions)

cos\_sim = cosine\_similarity(resume\_vector, tfidf\_matrix)

* **Database Schema**:
  + Users: id, username, email, password
  + Jobs: id, title, description, company, location
* **Sample Input Resume**:  
  "Experienced Python developer with interest in AI and web applications..."
* **Sample Output Recommendations**:
  + Python Developer – ABC Corp
  + AI Engineer – XYZ Solutions
  + Web Application Developer – TechFirm Inc.