Documentation

Resume Ranker

Web site created using create-react-app

https://ccibt9.me

The Job Finder is a web app that allows recruiters to post multiple job openings and correspondingly rank resumes of the applicants. The applicants can apply on the job portal and get further insights through the dynamic job cards available therein.

The Recruiter has a general as well as a job-specific dashboard, through which they can get deeper insights into the real-time statistical data of the applicants.

▼ Features

- Rank based on cosine similarity
- Ability to rank based on the following criteria:

  Domain experience, Number of job switches, Organizations worked for, Keyword matches, Going by academic document profiles
- The ability to generate reports with custom filters
- Search engine to display details based on criteria provided

Resources

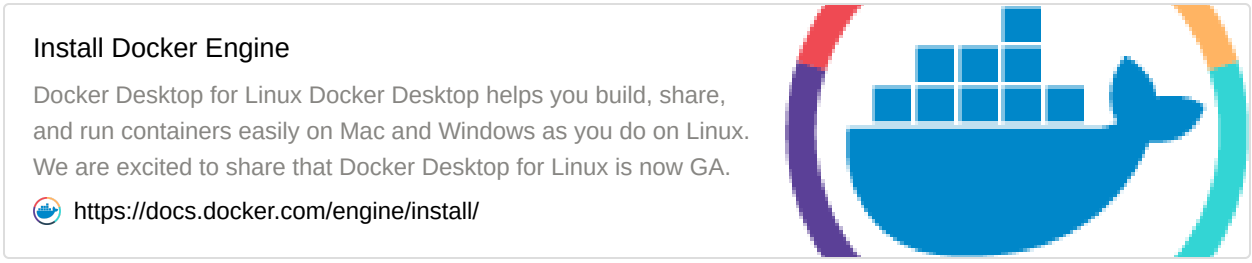
Aa Name
🚀 Quick Start
📦 Technical Specs
🎖️ Metrics and Code Quality
💻 API Documentation
👨🏻‍💻 Client Documentation
🤖 Ranking Algorithm
🔖 Operational Handbook

Quick Start

Using Docker

For convenience, the project is shipped with a `docker-compose` file which is used by Docker's compose tool for container orchestration. This file contains a specification to get all components of the project up and running using a single command.

- Make sure you have Docker installed. If not, install Docker using the instructions for your platform found here.



- Clone the repository.

```
$ git clone https://github.com/RameshSankarS/TEAM-B4
$ cd TEAM-B4
```

- Start the containers. (The `-d` flag is to run the containers in the background.)

```
$ docker compose up --build -d
```

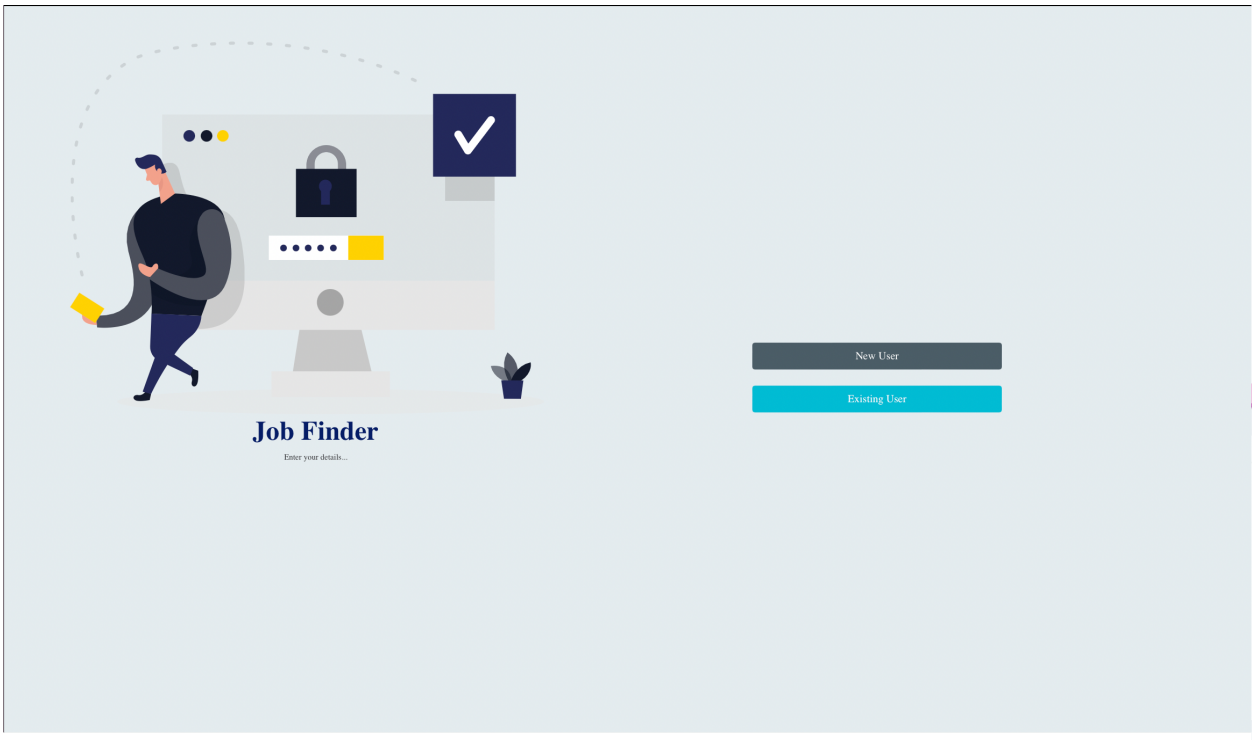
- Wait for the client to build.

```
$ docker compose logs client -f
```

You should get logs similar to this:

```
...
ranking_algo_client | File sizes after gzip:
ranking_algo_client |
ranking_algo_client |   397.95 kB  build/static/js/main.3154da2d.js
ranking_algo_client |   19.86 kB   build/static/css/main.f0be168c.css
ranking_algo_client |
ranking_algo_client | The project was built assuming it is hosted at /.
ranking_algo_client | You can control this with the homepage field in your package.json.
ranking_algo_client |
ranking_algo_client | The build folder is ready to be deployed.
ranking_algo_client | You may serve it with a static server:
ranking_algo_client |
ranking_algo_client |   npm install -g serve
ranking_algo_client |   serve -s build
ranking_algo_client |
ranking_algo_client | Find out more about deployment here:
ranking_algo_client |
ranking_algo_client |   https://cra.link/deployment
ranking_algo_client |
```

- Once the client is built, you are good to go. Open `https://localhost:1337/` in a browser. You should be greeted with a page that looks like this



You’re good to go. Play around with the interface and try to break things :)

Technical Specs

The project is divided into three main components.

- The Server
- The Client
- The ML models

The Server

The server mainly depends on:

- NodeJS (18.0.0)
- ExpressJS (express@4.18.1)

The Client

The client is a web app written using:

- NodeJS (v18.0.0)
- ReactJS (react@18.2.0)

The ML models

The ML models are written in Python using PyTorch and exposed by Celery workers.

- Python (3.10)
- celery (5.2.7)
- numpy (1.22.3)
- pandas (1.4.2)

- PyTorch (1.11.0)
- HuggingFace Transformers (4.21.)
- nltk (3.7)
- spacy (3.3)

Other Services

The project uses MongoDB as it’s primary data source. RabbitMQ is used to enable communication between the server and the celery workers.

- MongoDB (5.0)
- RabbitMQ (3.7.28)

Metrics and Code Quality

Code Formatting and Linting

Code quality is important. To ensure standardisation of the code and meeting quality metrics throughout the project we've used linters and formatters.

Language	Tools
Python	- flake8 (linter) - black (code formatter)
JavaScript	- JSLint (linter) - prettier (code formatter)

These tools are registered as `pre-commit` hooks to the git repository. Everytime a patch is added to the code, the hooks are fired and the code is formatted and linted by appropriate tools.

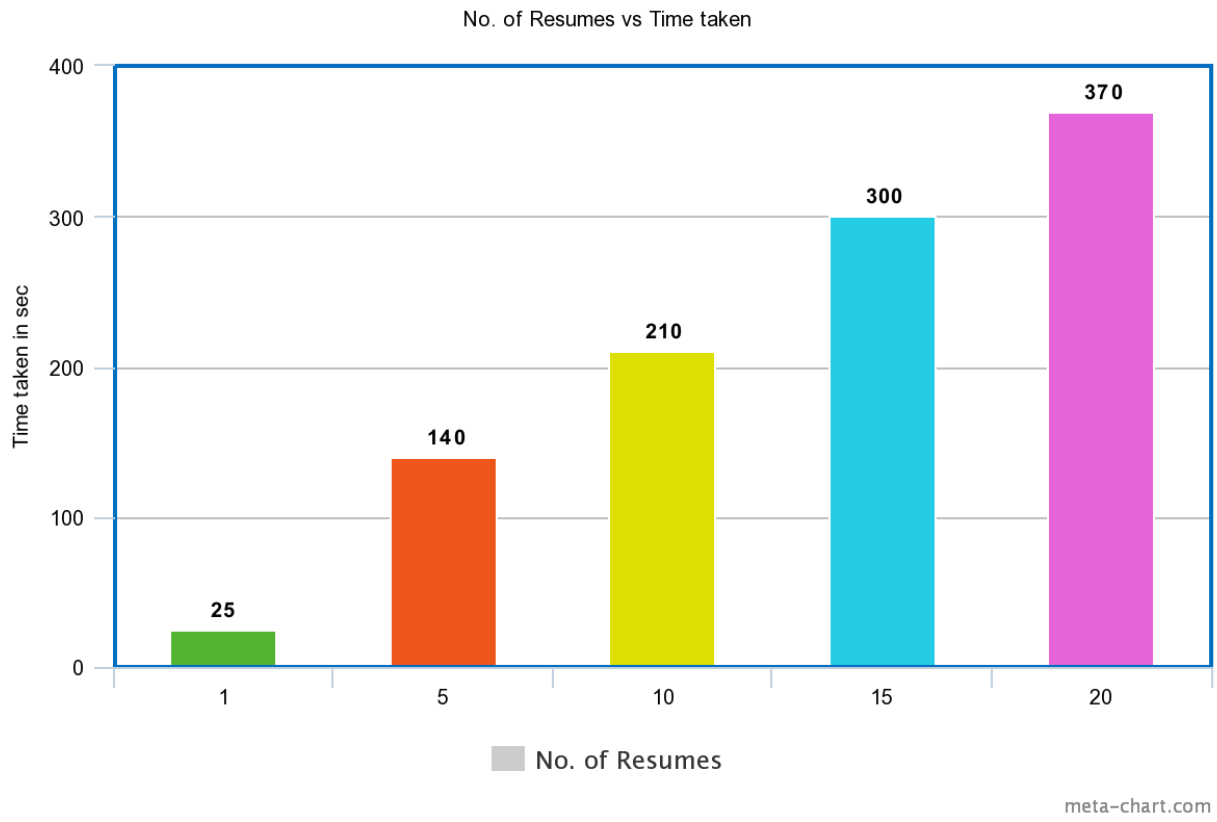
Code Testing

To ensure the code works under all* conditions, we've written an exhaustive test suite consisting of unit tests for all components.

Language	Tools
Python	- unittest
JavaScript	- jest

Load Metrics

No. of Resumes Vs Time taken to rank in sec



Results

Accuracy of the Spacy Named Entity Recognition model fine-tuned with Roberta Transformers is 96%

All the testcases were passed in the ML side and WebDev side too

API Documentation

The server is documented using a collection on Postman. The collection is live and gets updated with every API change.

https://www.getpostman.com/collections/70d28fddf08813da291c

Client Documentation

Description

A web app that allows recruiters to post multiple job openings and correspondingly rank resumes of the applicants. The applicants can apply on the job portal and get further insights through the dynamic job cards available therein.

The Recruiter has a General as well as a job-specific dashboard, through which they can get deeper insights into the real-time statistical data of the applicants.

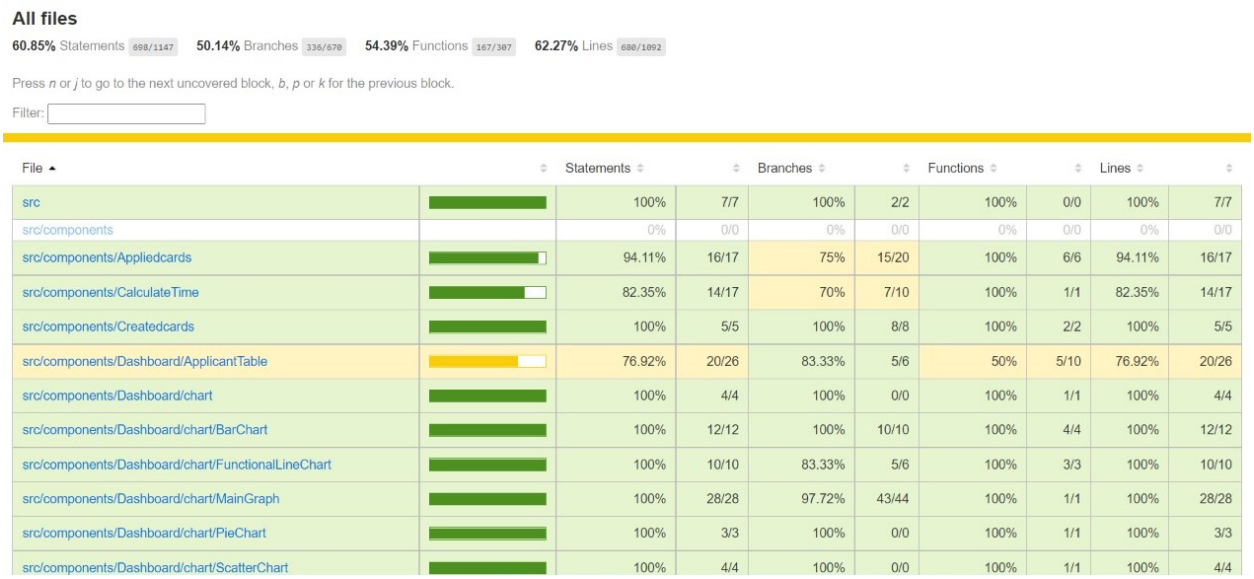
Code Overview

Code structure and the overview of the client can be identified at

https://github.com/RameshSankarS/TEAM-B4/tree/main/client

Testing Results

Client testing coverage



All files

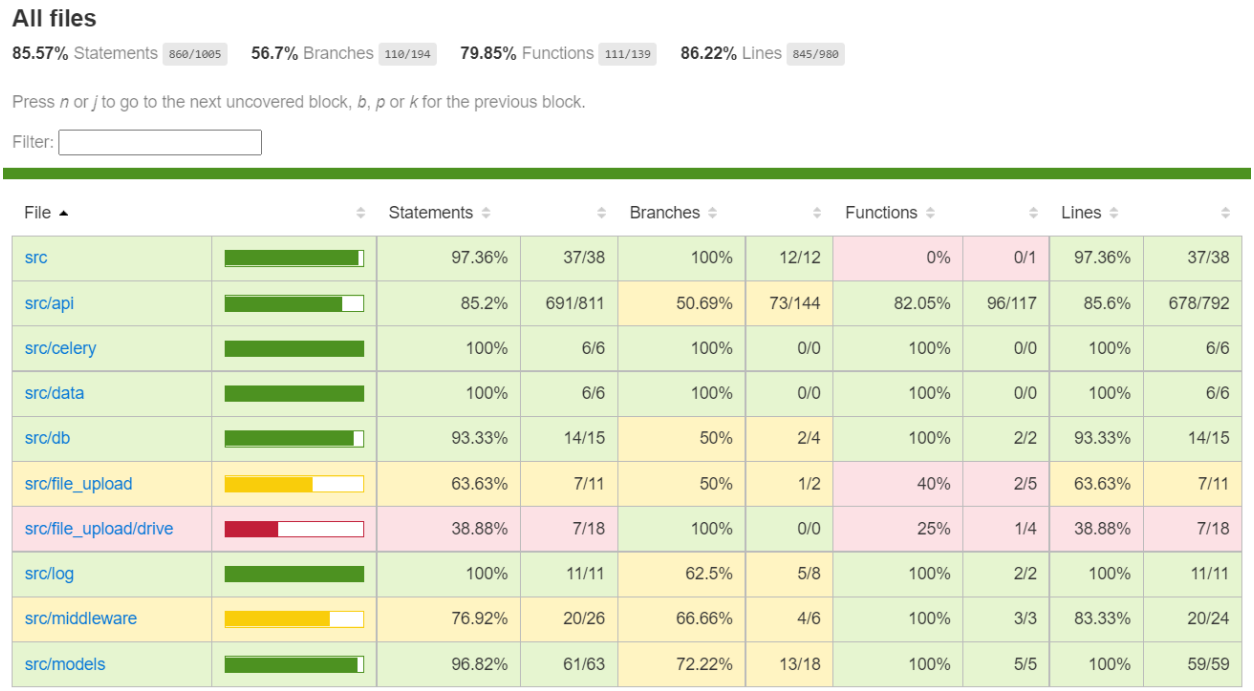
60.85% Statements 698/1147 **50.14%** Branches 336/670 **54.39%** Functions 167/307 **62.27%** Lines 680/1092

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File		Statements		Branches		Functions		Lines	
src		100%	7/7	100%	2/2	100%	0/0	100%	7/7
src/components		0%	0/0	0%	0/0	0%	0/0	0%	0/0
src/components/Appliedcards		94.11%	16/17	75%	15/20	100%	6/6	94.11%	16/17
src/components/CalculateTime		82.35%	14/17	70%	7/10	100%	1/1	82.35%	14/17
src/components/Createdcards		100%	5/5	100%	8/8	100%	2/2	100%	5/5
src/components/Dashboard/ApplicantTable		76.92%	20/26	83.33%	5/6	50%	5/10	76.92%	20/26
src/components/Dashboard/chart		100%	4/4	100%	0/0	100%	1/1	100%	4/4
src/components/Dashboard/chart/BarChart		100%	12/12	100%	10/10	100%	4/4	100%	12/12
src/components/Dashboard/chart/FunctionalLineChart		100%	10/10	83.33%	5/6	100%	3/3	100%	10/10
src/components/Dashboard/chart/MainGraph		100%	28/28	97.72%	43/44	100%	1/1	100%	28/28
src/components/Dashboard/chart/PieChart		100%	3/3	100%	0/0	100%	1/1	100%	3/3
src/components/Dashboard/chart/ScatterChart		100%	4/4	100%	0/0	100%	1/1	100%	4/4

Server testing coverage



All files

85.57% Statements 860/1005 **56.7%** Branches 110/194 **79.85%** Functions 111/139 **86.22%** Lines 845/980

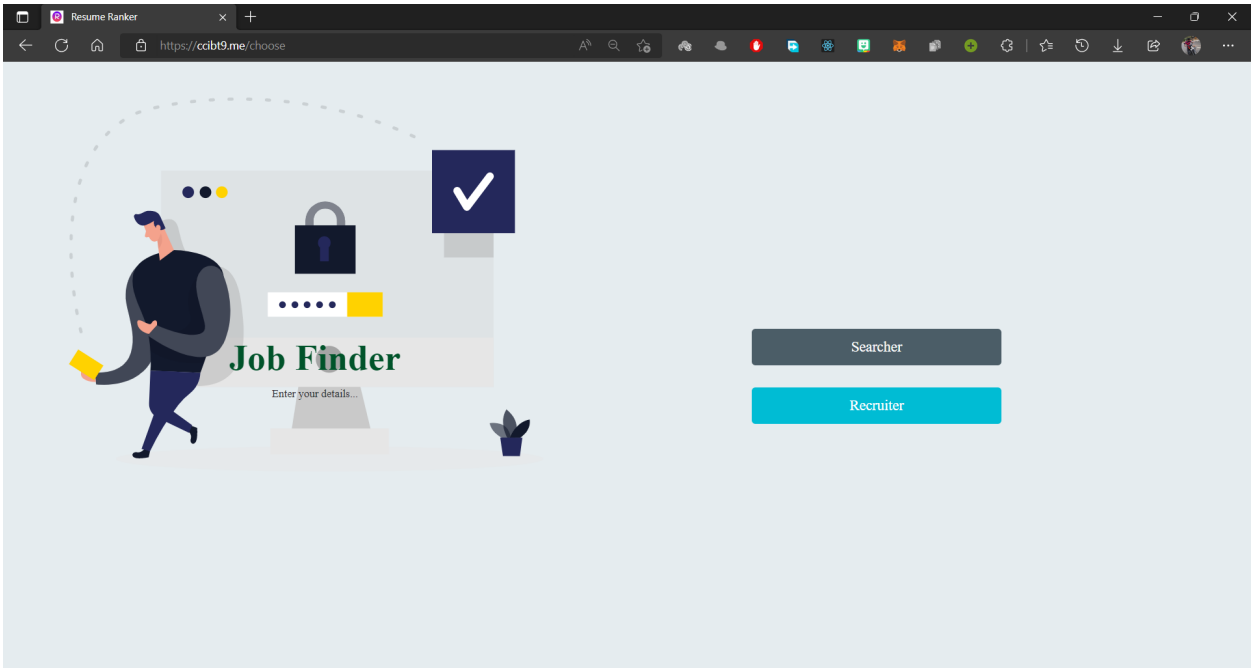
Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

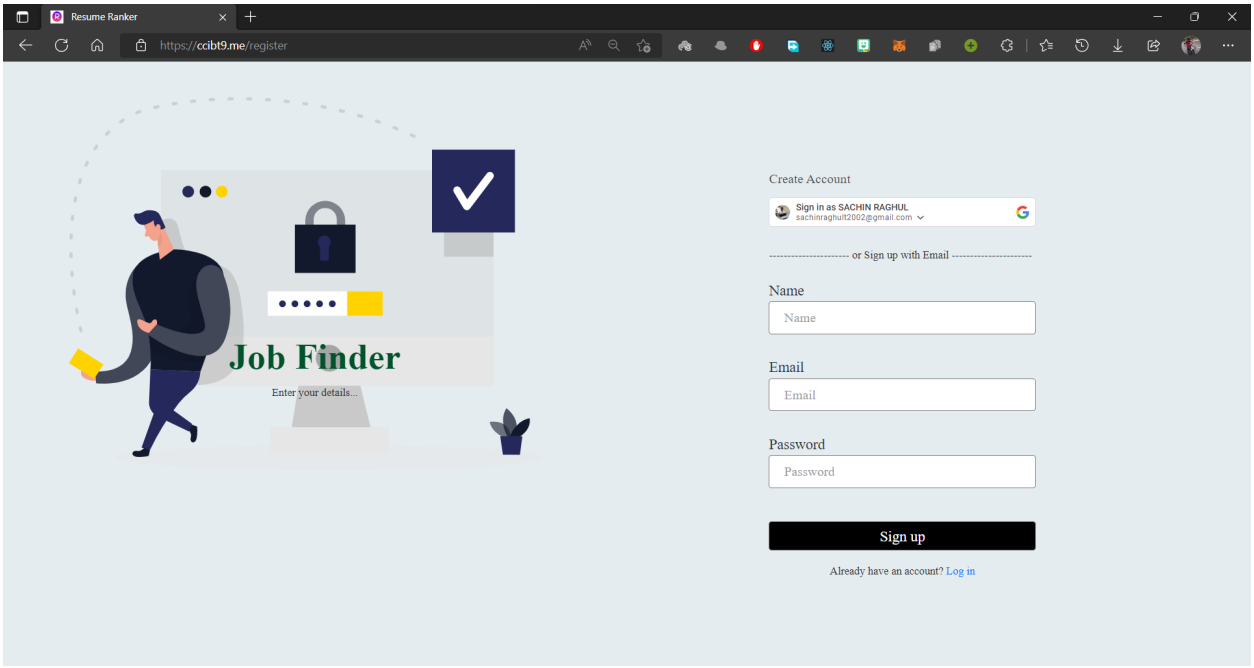
File		Statements		Branches		Functions		Lines	
src		97.36%	37/38	100%	12/12	0%	0/1	97.36%	37/38
src/api		85.2%	691/811	50.69%	73/144	82.05%	96/117	85.6%	678/792
src/celery		100%	6/6	100%	0/0	100%	0/0	100%	6/6
src/data		100%	6/6	100%	0/0	100%	0/0	100%	6/6
src/db		93.33%	14/15	50%	2/4	100%	2/2	93.33%	14/15
src/file_upload		63.63%	7/11	50%	1/2	40%	2/5	63.63%	7/11
src/file_upload/drive		38.88%	7/18	100%	0/0	25%	1/4	38.88%	7/18
src/log		100%	11/11	62.5%	5/8	100%	2/2	100%	11/11
src/middleware		76.92%	20/26	66.66%	4/6	100%	3/3	83.33%	20/24
src/models		96.82%	61/63	72.22%	13/18	100%	5/5	100%	59/59

HOME

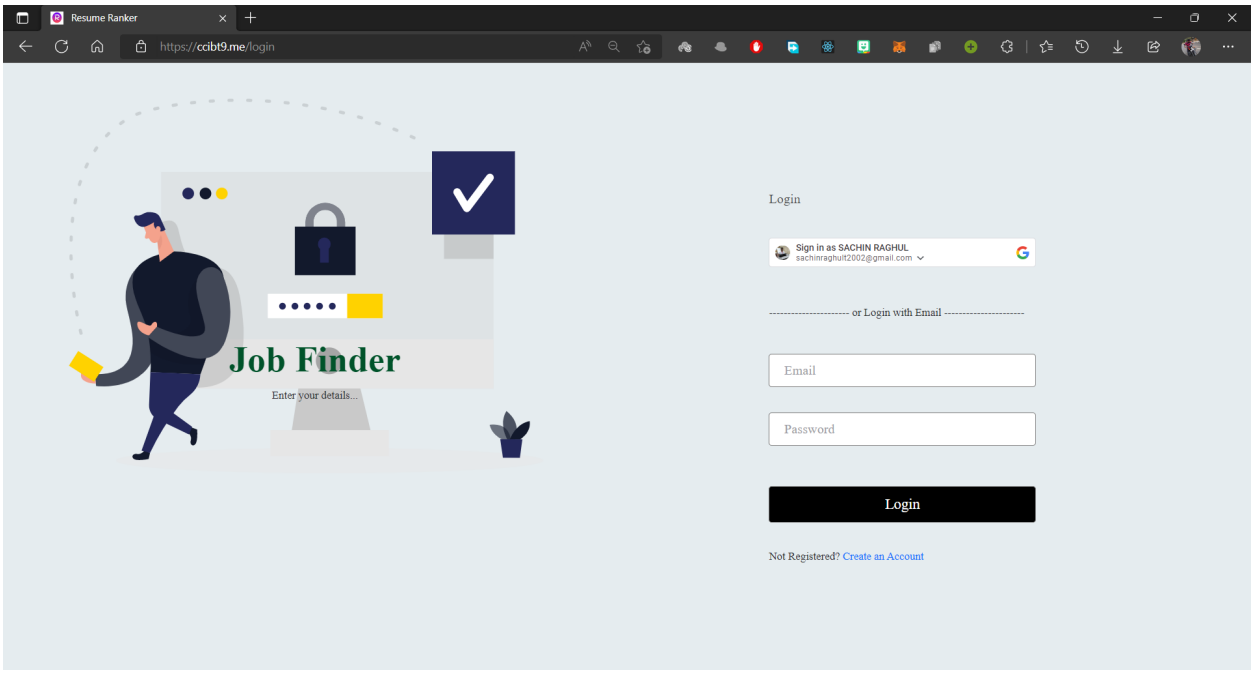
Welcome



Recruiter

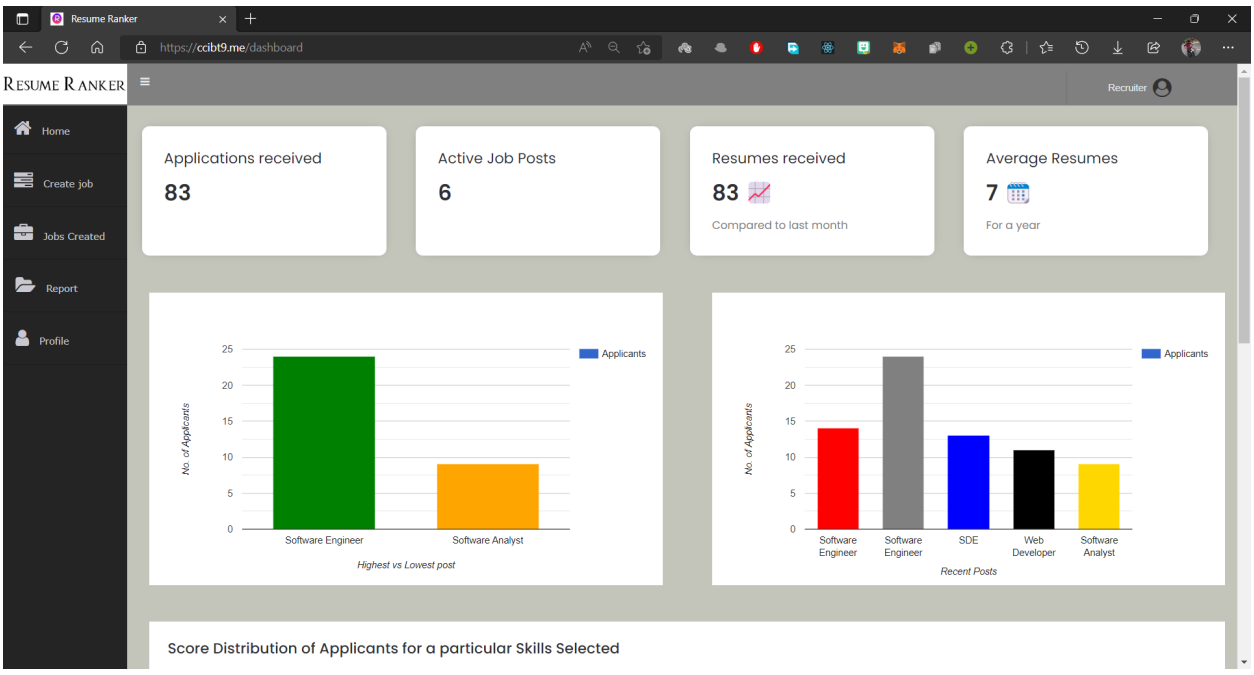


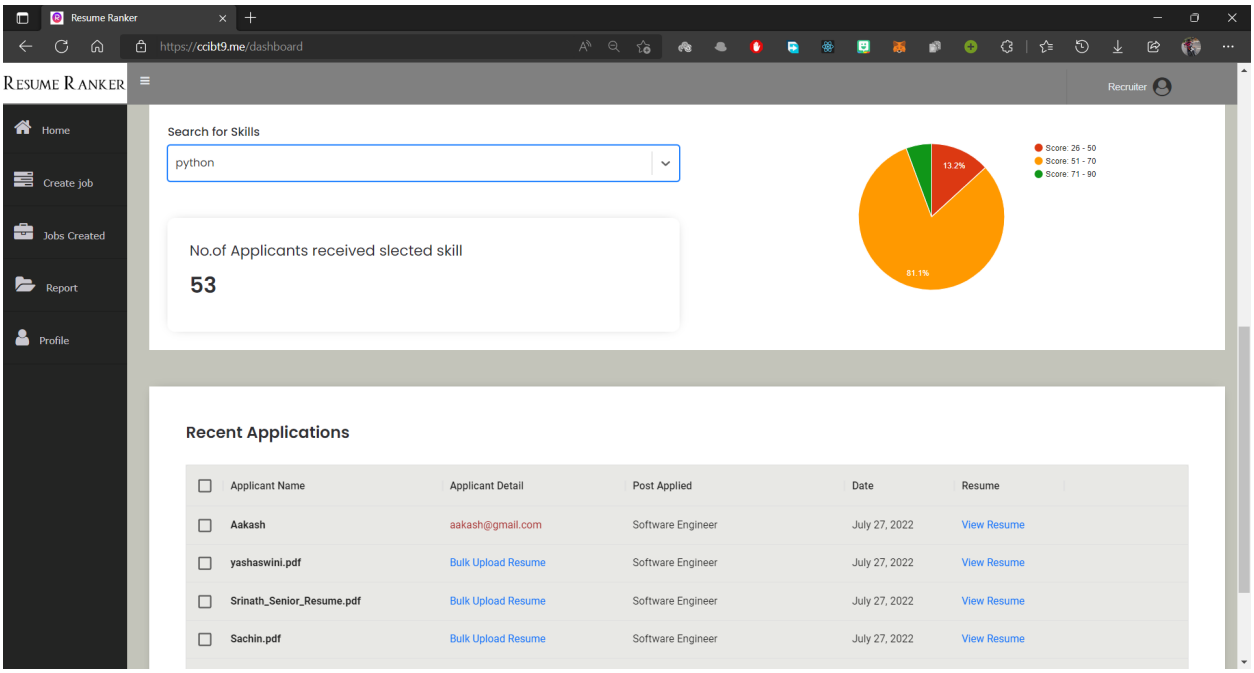
Login



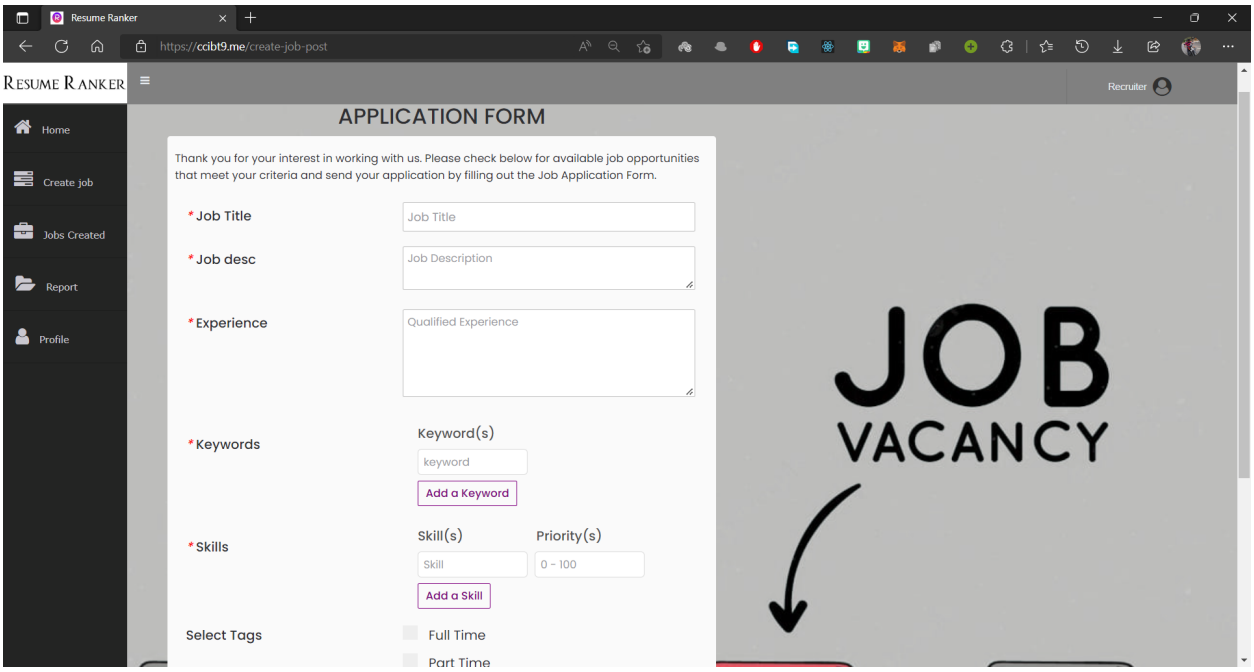
RECRUITER

Landing Dashboard

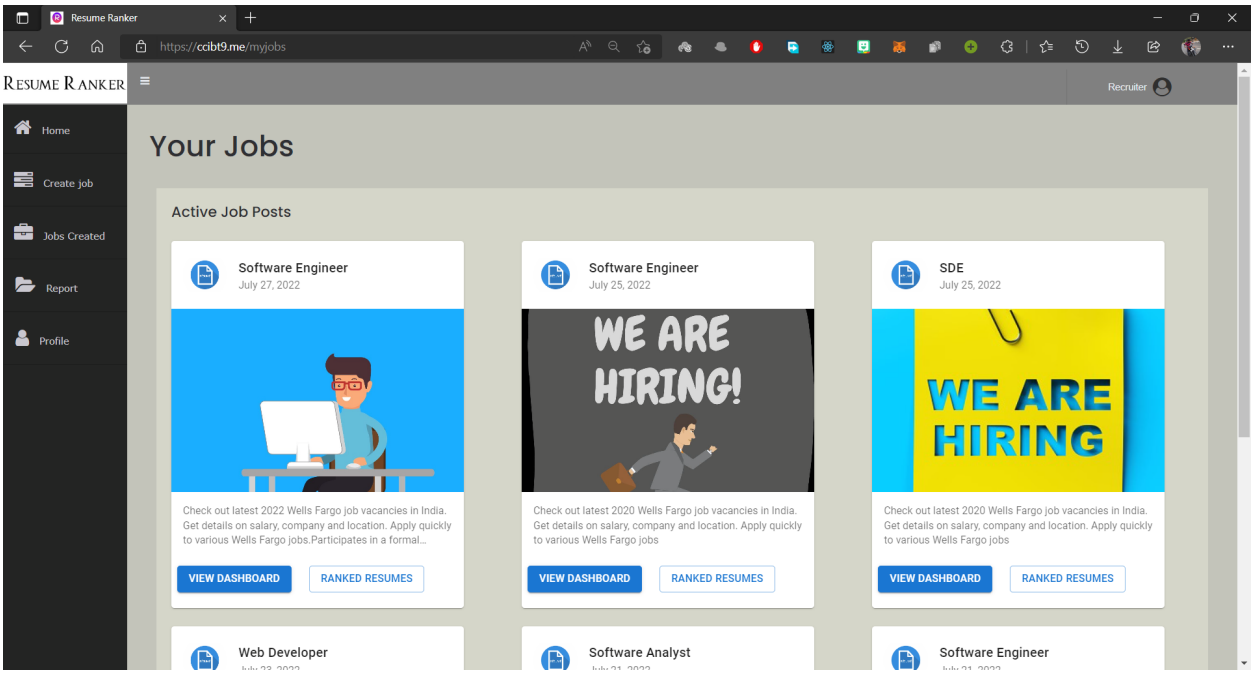




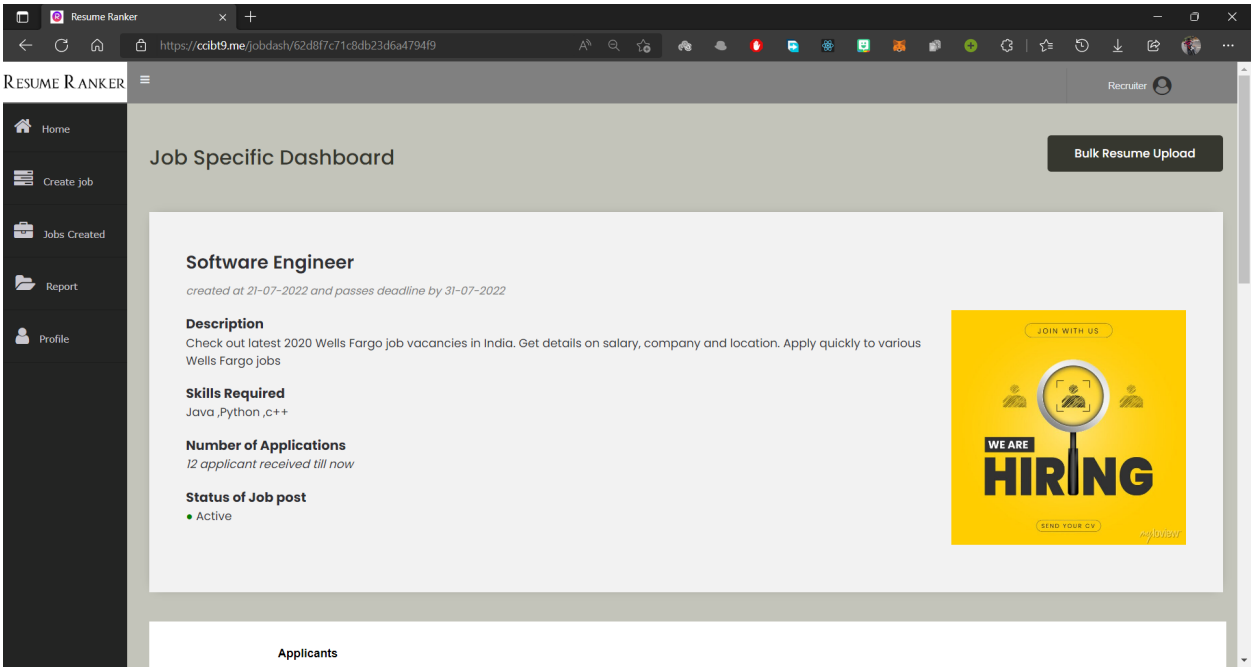
Job Creation Form

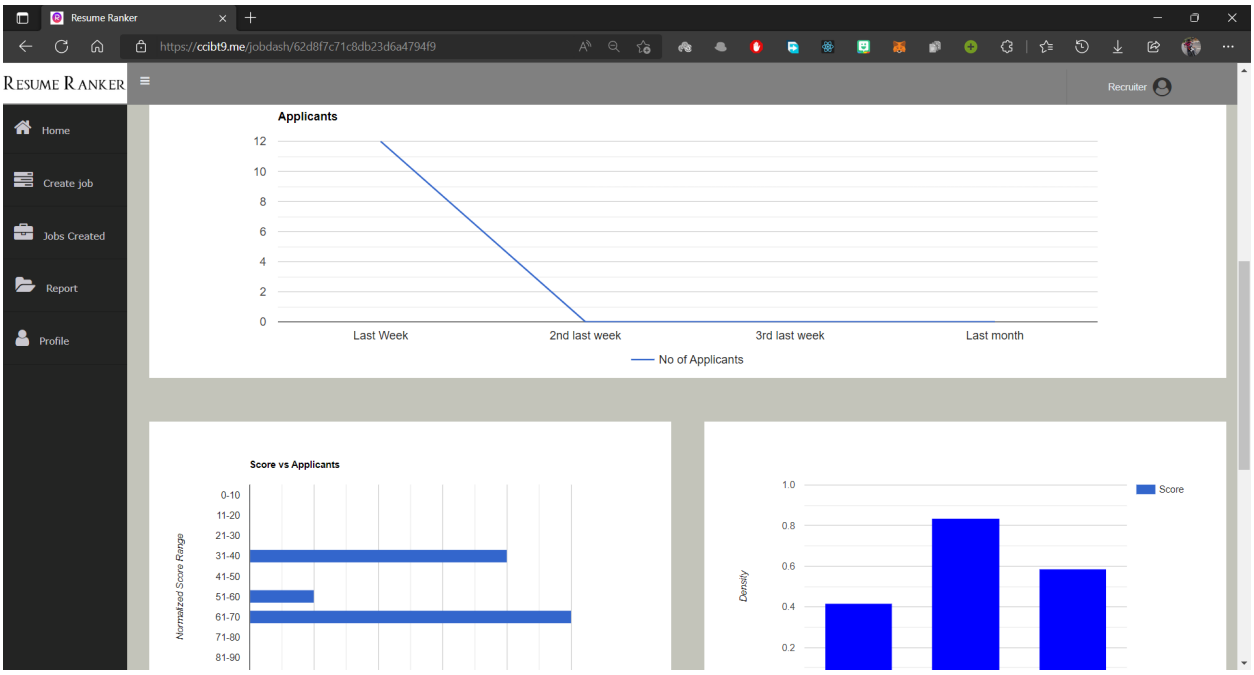


Jobs Created

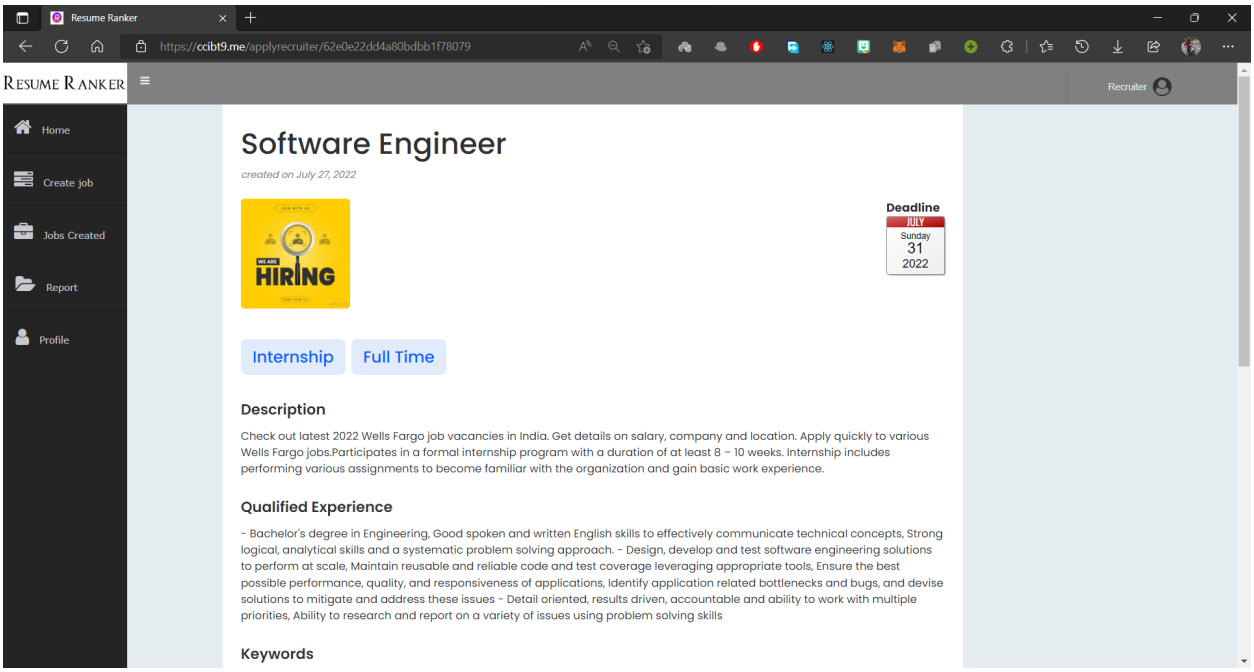


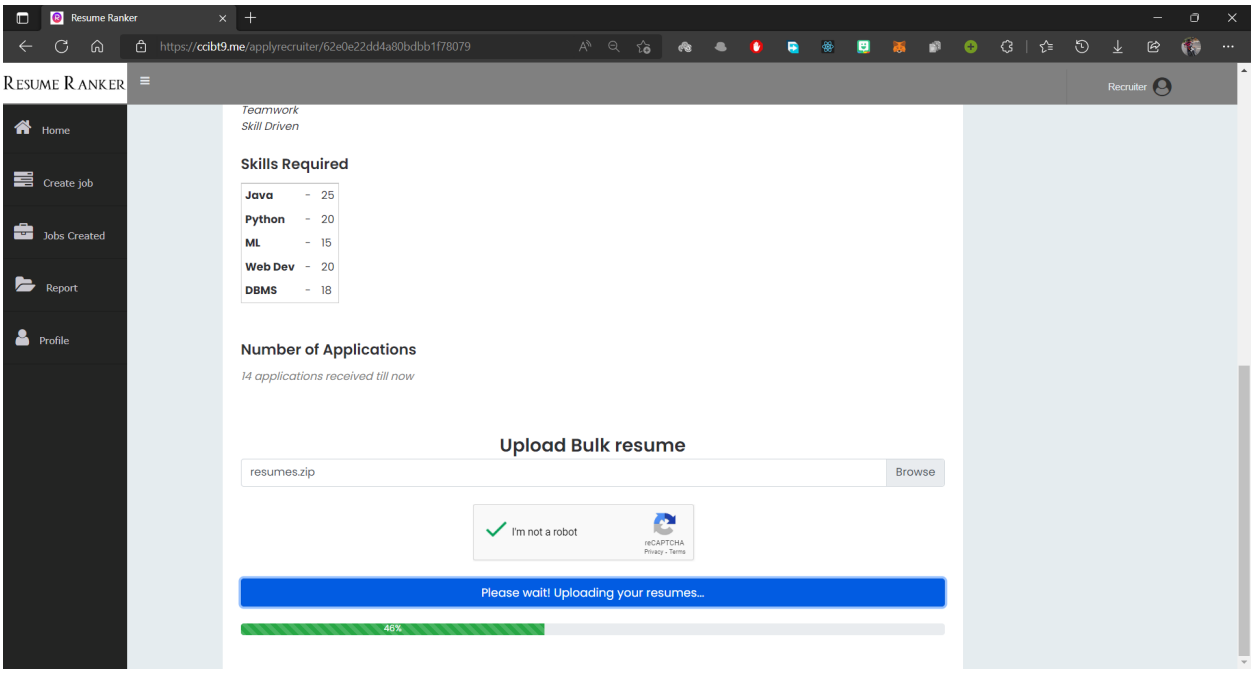
Job Specific Dashboard



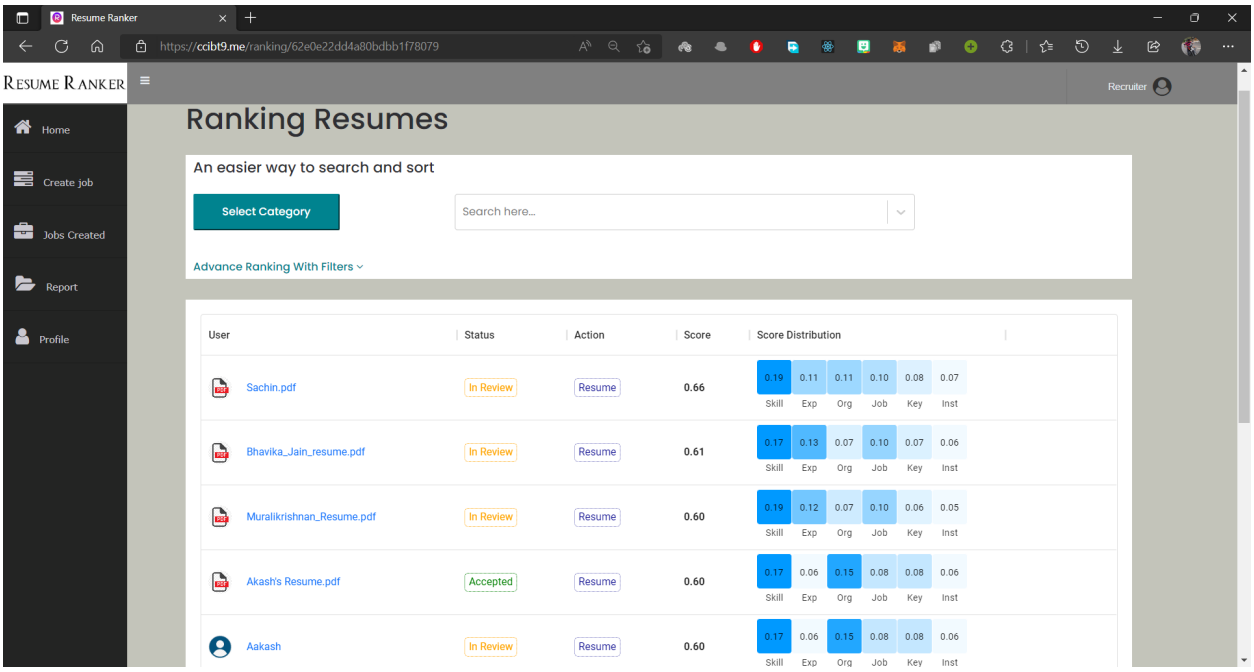


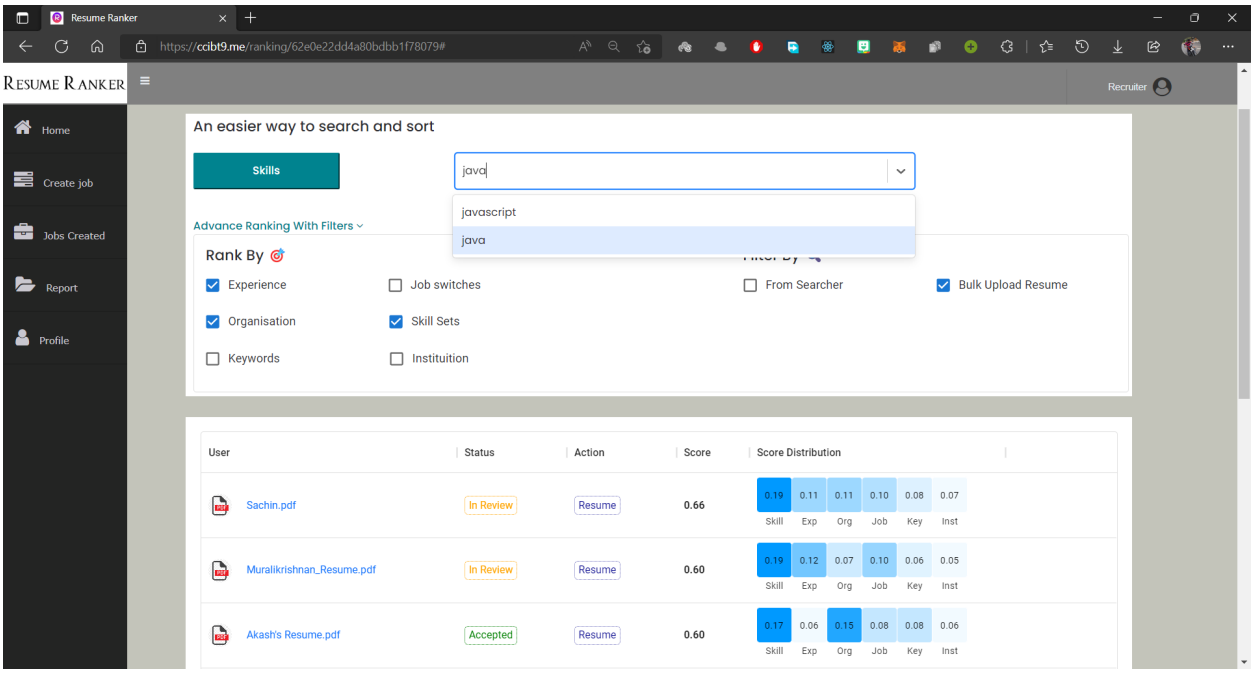
Bulk Upload Resume



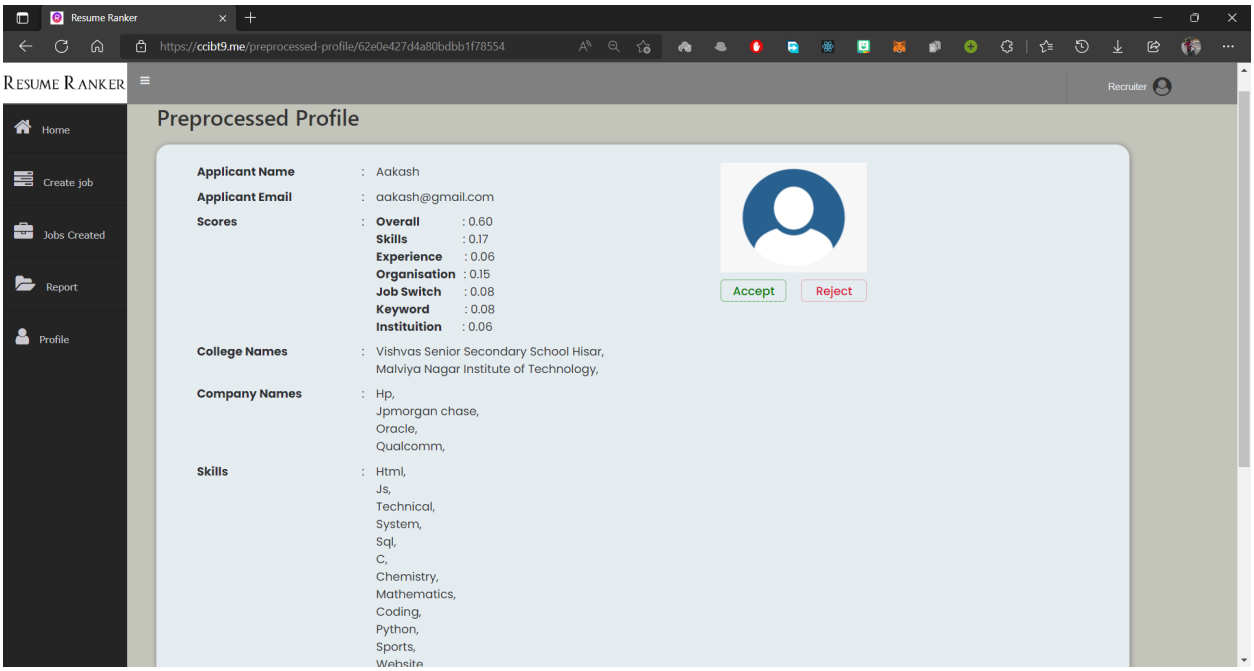


Ranking Table

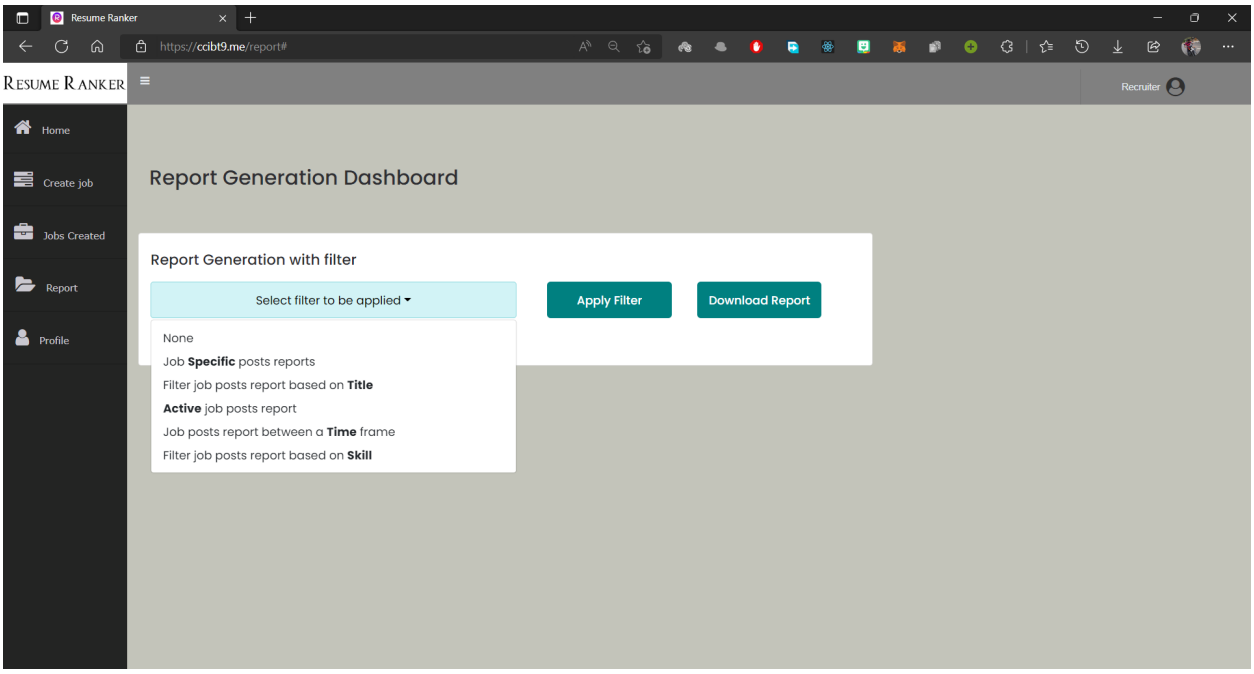


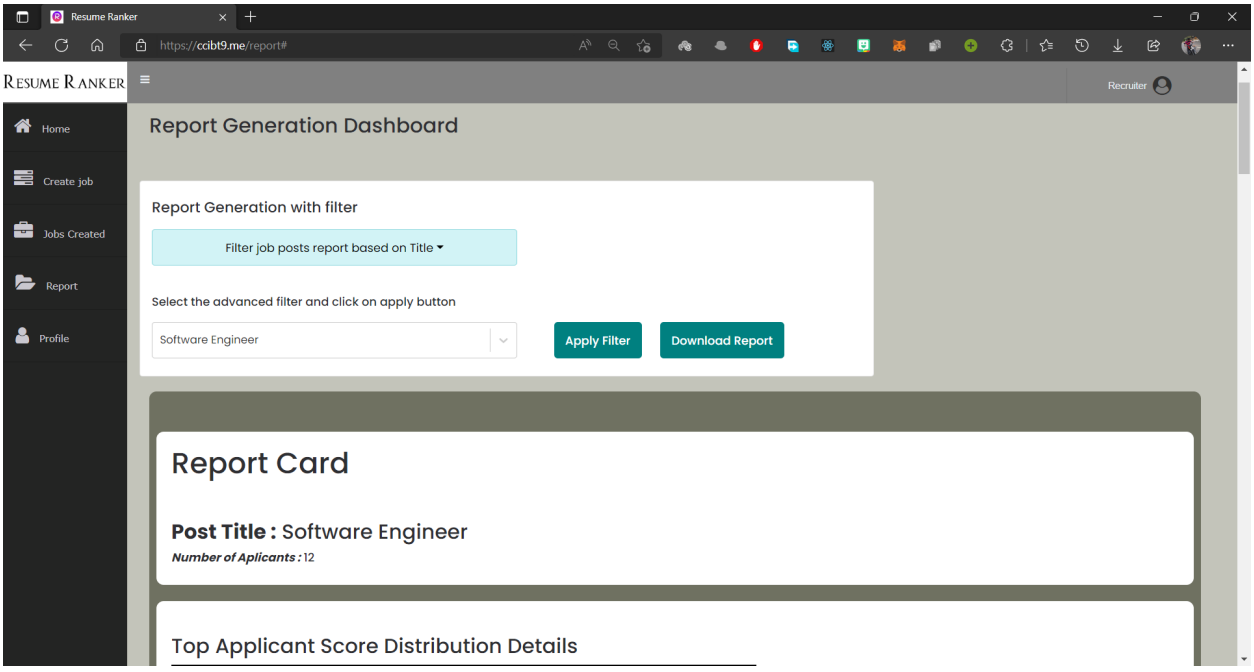


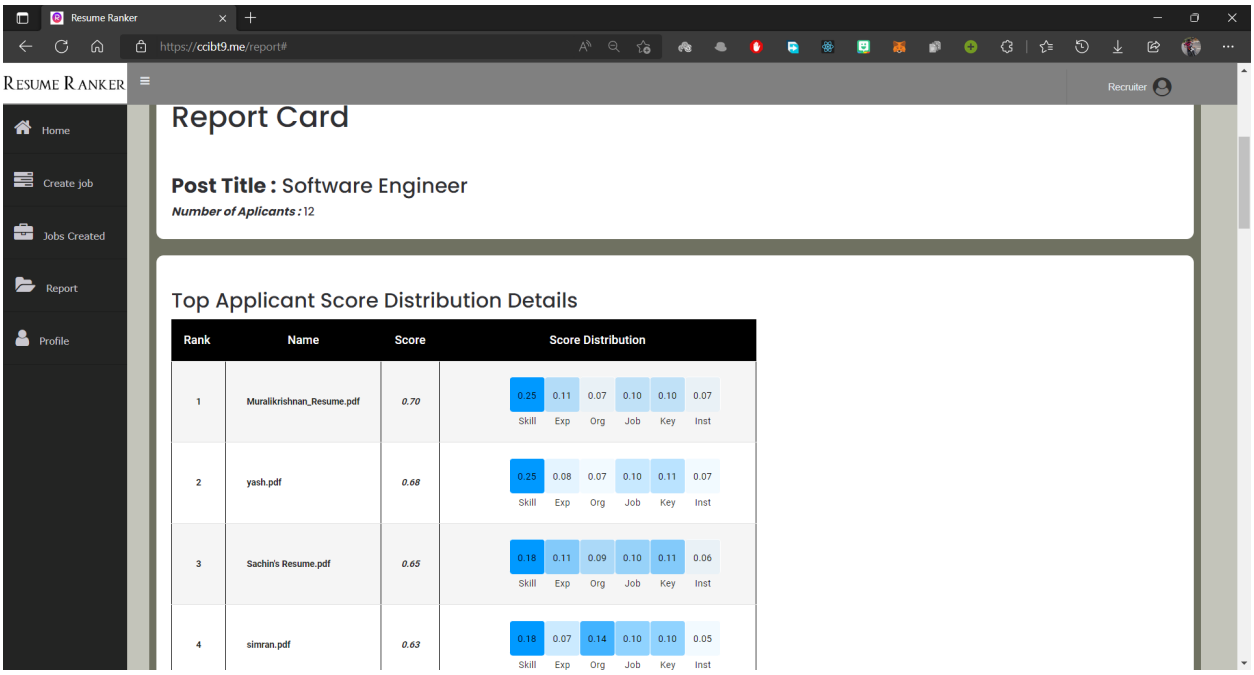
Preprocessed Profile

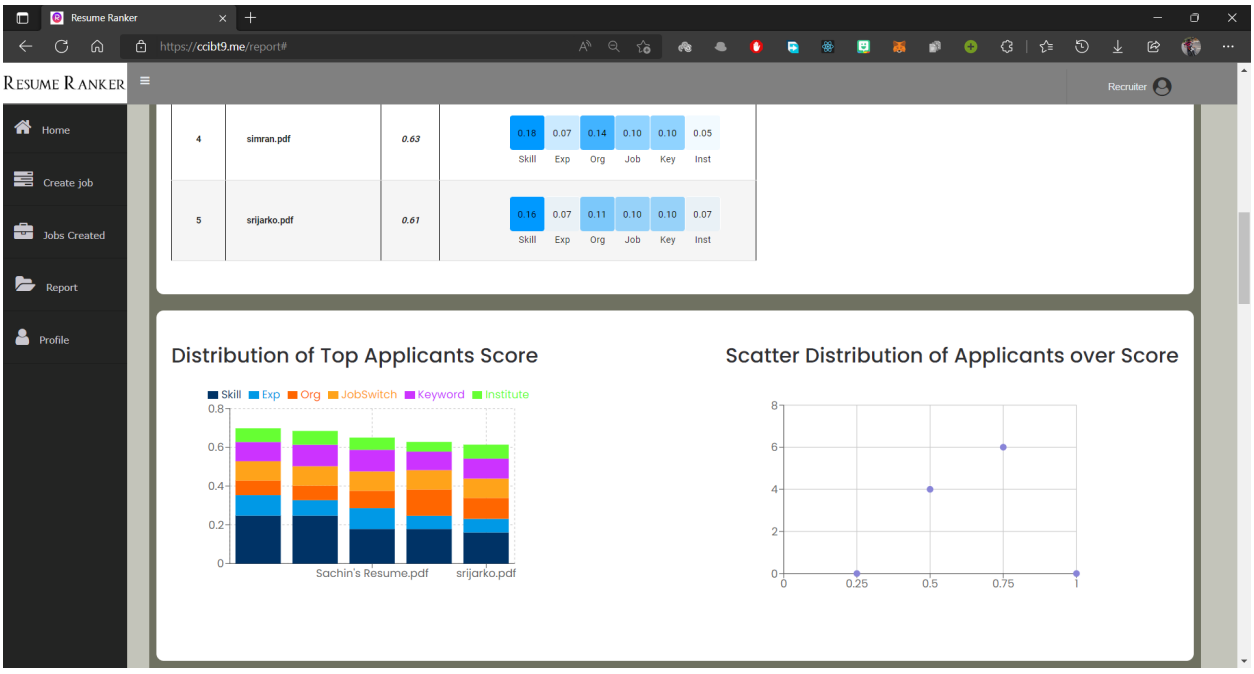


Report Generation

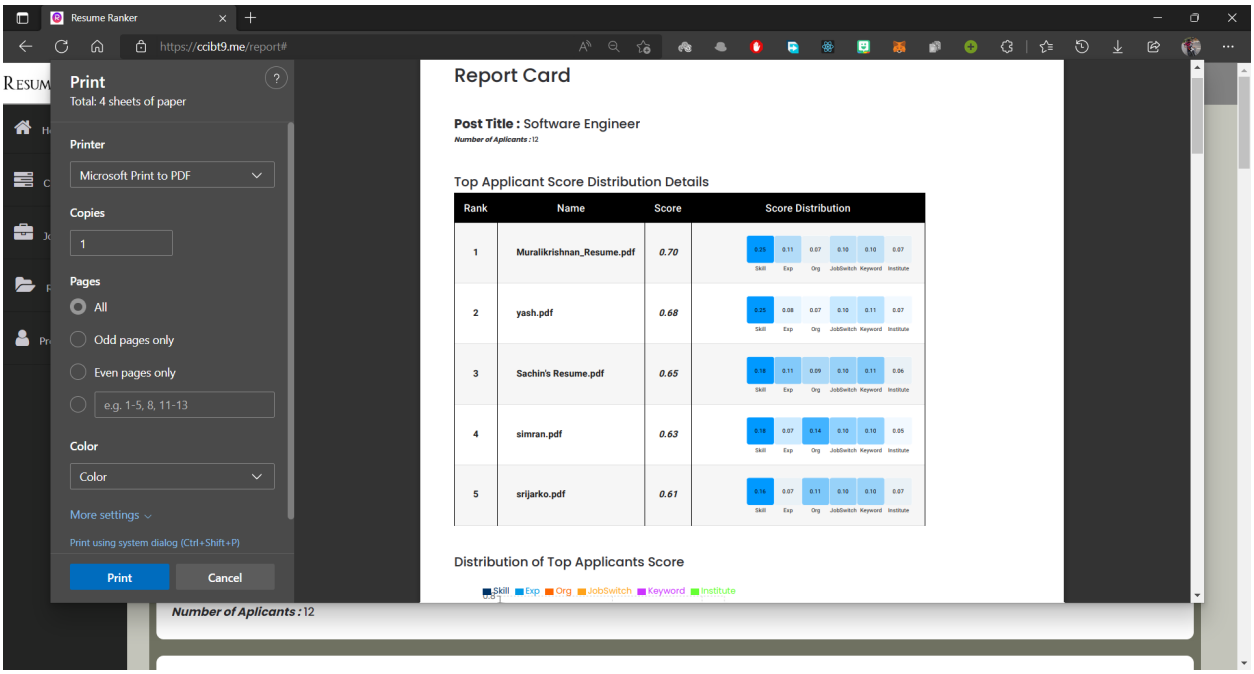




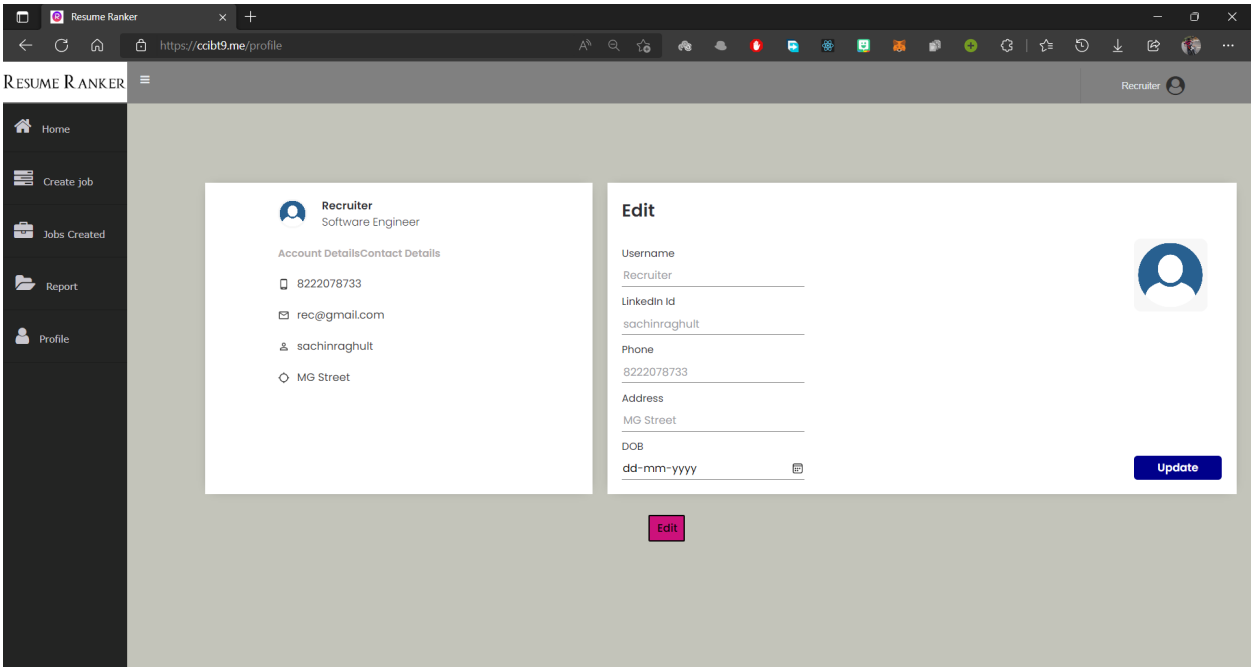

Resume Ranker
https://ccibt9.me/report#
RESUME RANKER
Recruiter
Home
Create job
Jobs Created
Report
Profile
Report Card
Post Title : Software Engineer
Number of Aplicants : 12
Top Applicant Score Distribution Details
Rank
Name
Score
Score Distribution
1
Muralikrishnan_Resume.pdf
0.70
0.25 0.11 0.07 0.10 0.10 0.07
Skill Exp Org Job Key Inst
2
yash.pdf
0.68
0.25 0.08 0.07 0.10 0.11 0.07
Skill Exp Org Job Key Inst
3
Sachin's Resume.pdf
0.65
0.18 0.11 0.09 0.10 0.11 0.06
Skill Exp Org Job Key Inst
4
simran.pdf
0.63
0.18 0.07 0.14 0.10 0.10 0.05
Skill Exp Org Job Key Inst


Resume Ranker
https://ccibt9.me/report#
RESUME RANKER
Recruiter
Home
Create job
Jobs Created
Report
Profile
4
simran.pdf
0.63
0.18 0.07 0.14 0.10 0.10 0.05
Skill Exp Org Job Key Inst
5
srijarko.pdf
0.61
0.16 0.07 0.11 0.10 0.10 0.07
Skill Exp Org Job Key Inst
Distribution of Top Applicants Score
Skill Exp Org JobSwitch Keyword Institute
0.8
0.6
0.4
0.2
0
Sachin's Resume.pdf
srijarko.pdf
Scatter Distribution of Applicants over Score
8
6
4
2
0
0
0.25
0.5
0.75
1

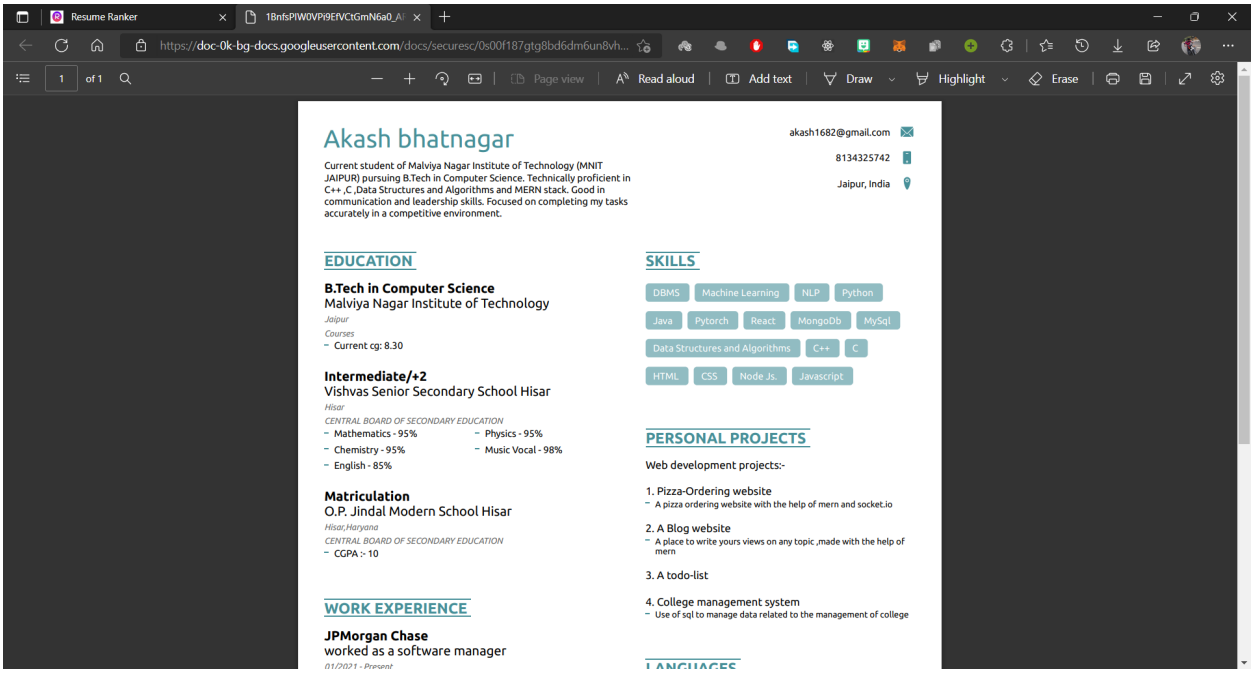
Downloadable Report



Profile Settings

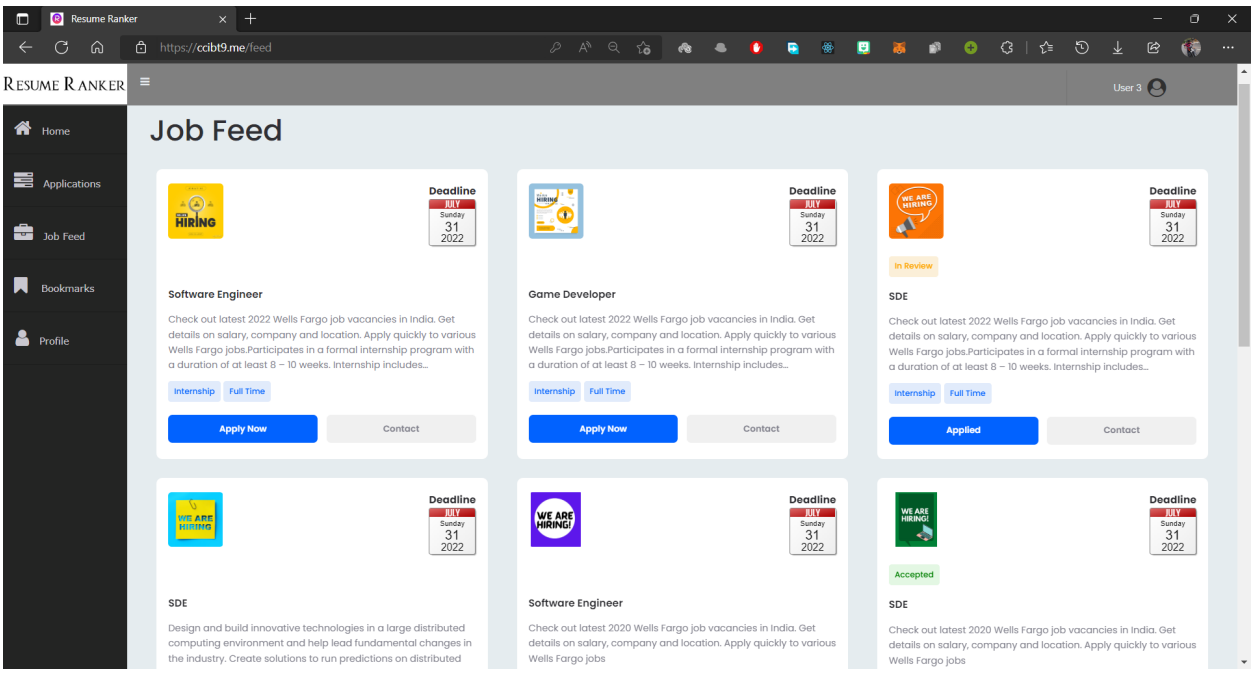


View Resume

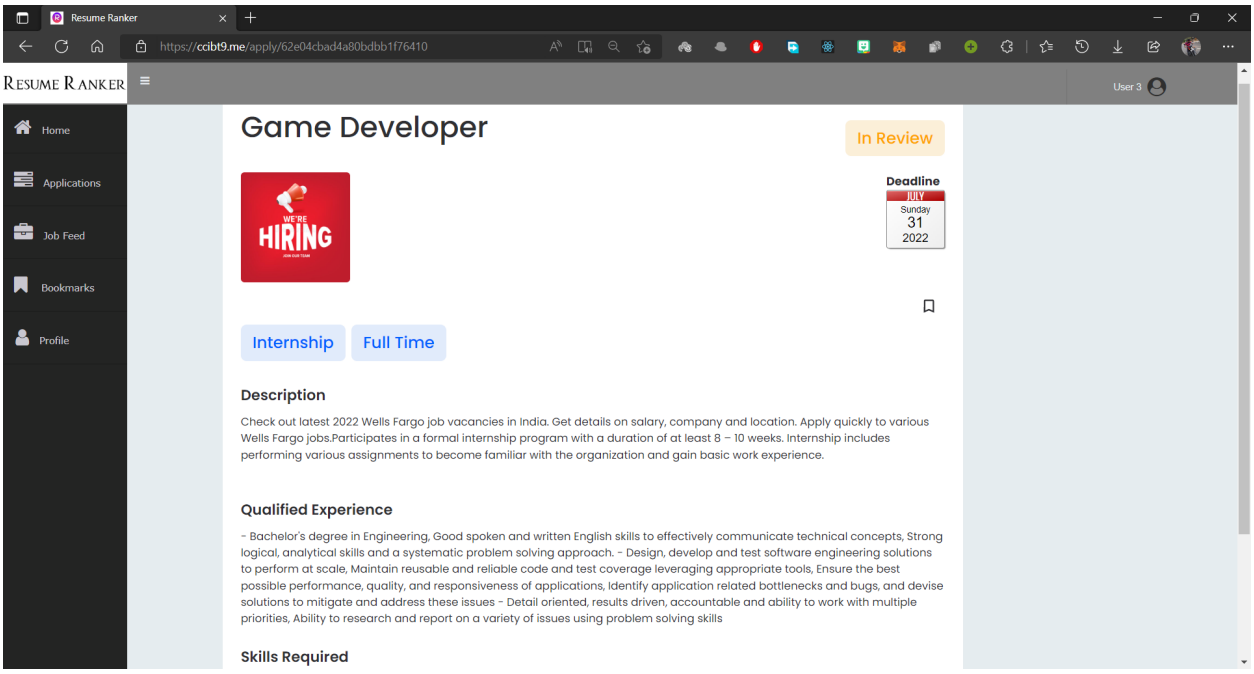


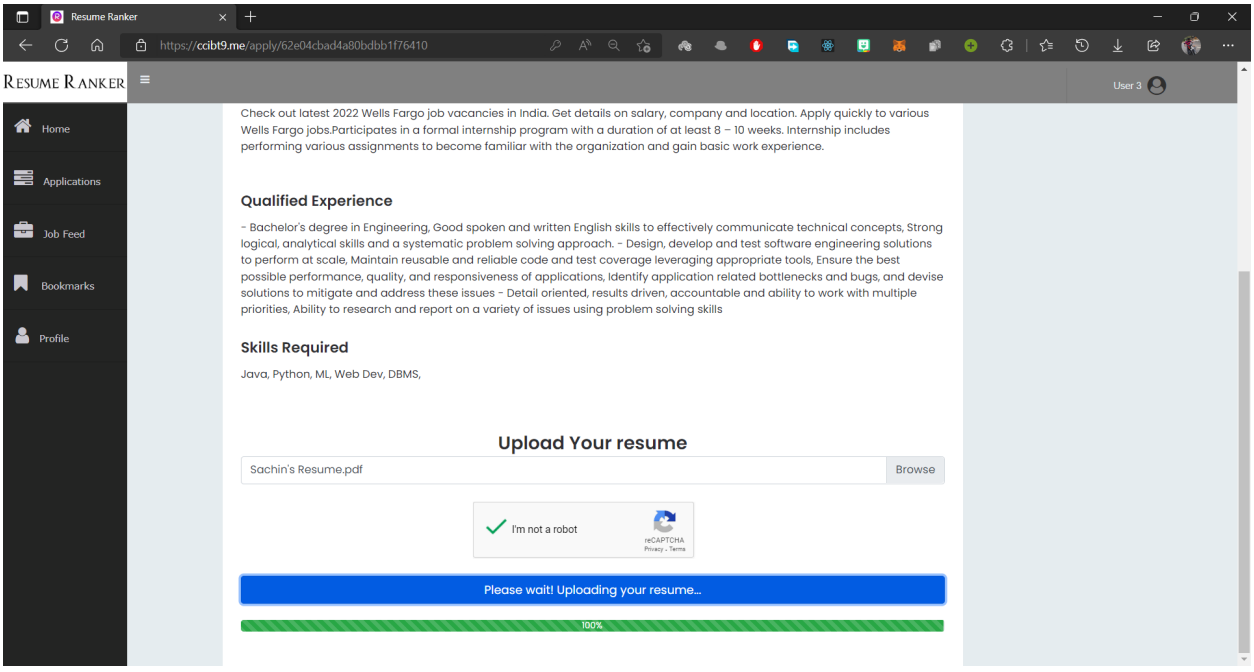
SEARCHER

Job Feed

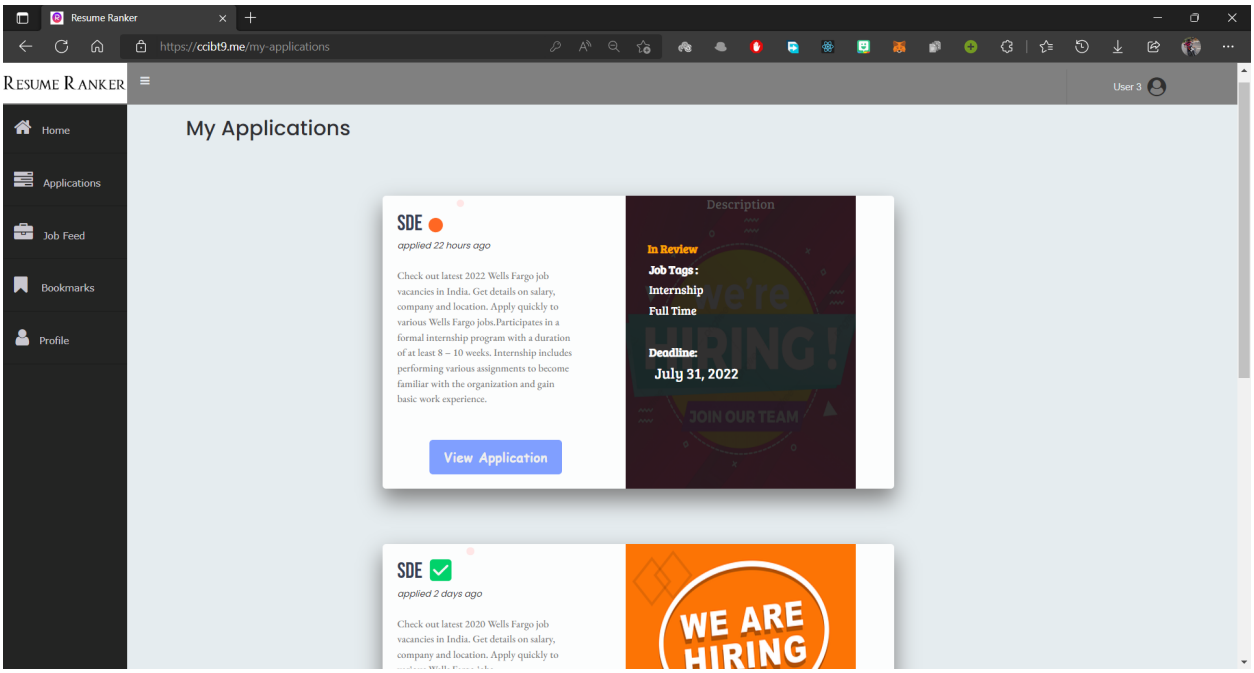


Job Application

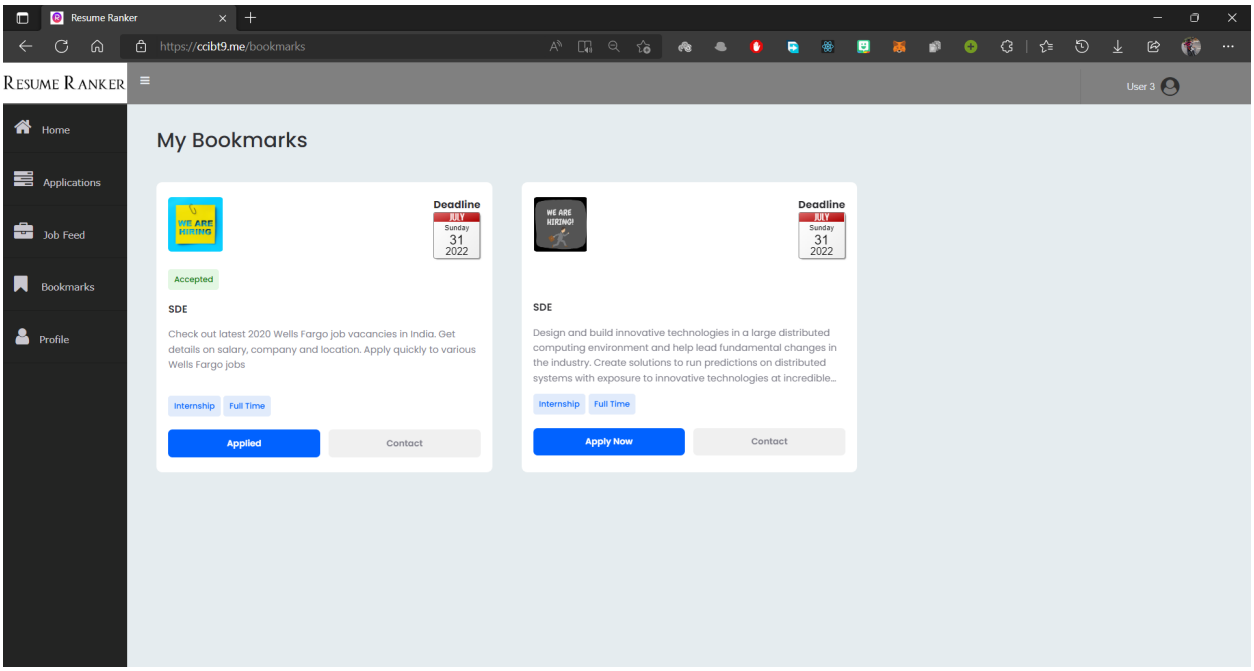




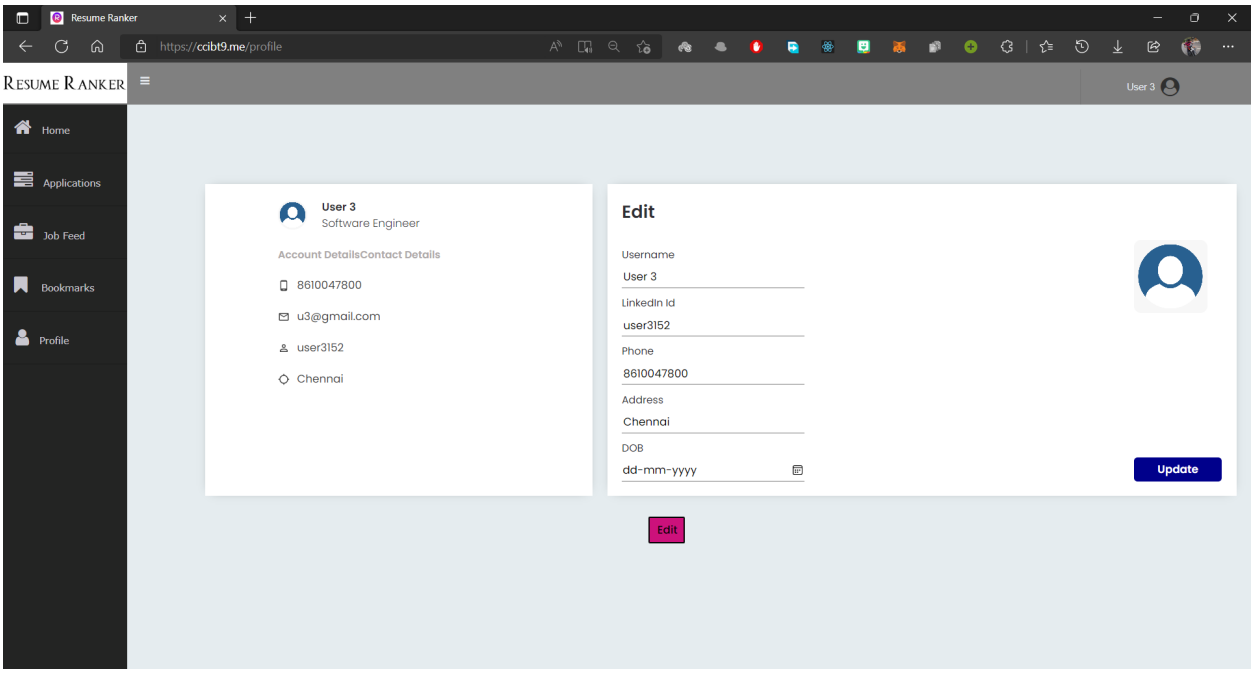
Jobs Applied



Bookmarks



Profile Settings



References

- https://reactjs.org/docs/faq-state.html
- https://www.npmjs.com/
- https://reactjs.org/docs/hooks-reference.html
- https://mui.com/material-ui/getting-started/overview/
- https://jestjs.io/docs/getting-started
- https://www.react-google-charts.com/

Ranking Algorithm

File Structure

The main ML setup is made up by `ranker/` directory, including:

- Main Ranking Algorithm module `algo/`
- ML-Backend Integration modules:
  - `db_utils/` for utility functions and
  - `celery_tasks/` for running tasks in the background
- Training notebooks directory `notebooks/`
- Test files for each criterion `tests/`
- Download script for weights and nltk data `weights/`
- Main ranking file on ML side `test.py`
- Dockerfile to start the containers including **celery**
- ML side Unit Tests `unittests.py`

Running the `main.sh` file via CLI will start the ranking algorithm and return a table of ranked resumes

Quick Start

- Create a directory inside `ranker/` that will contain all resume files that you want to be ranked, move all resumes inside the directory, and check if the files have been properly moved

```
$ mkdir sample/
$ cd sample
$ mkdir resumes/
$ cd resumes/
$ ls
```

- Install all dependencies via

```
$ pip install -r requirements.txt
```

- Run the shell script to download the weights, run unit tests and the main ranking file via

```
$ ./main.sh
```

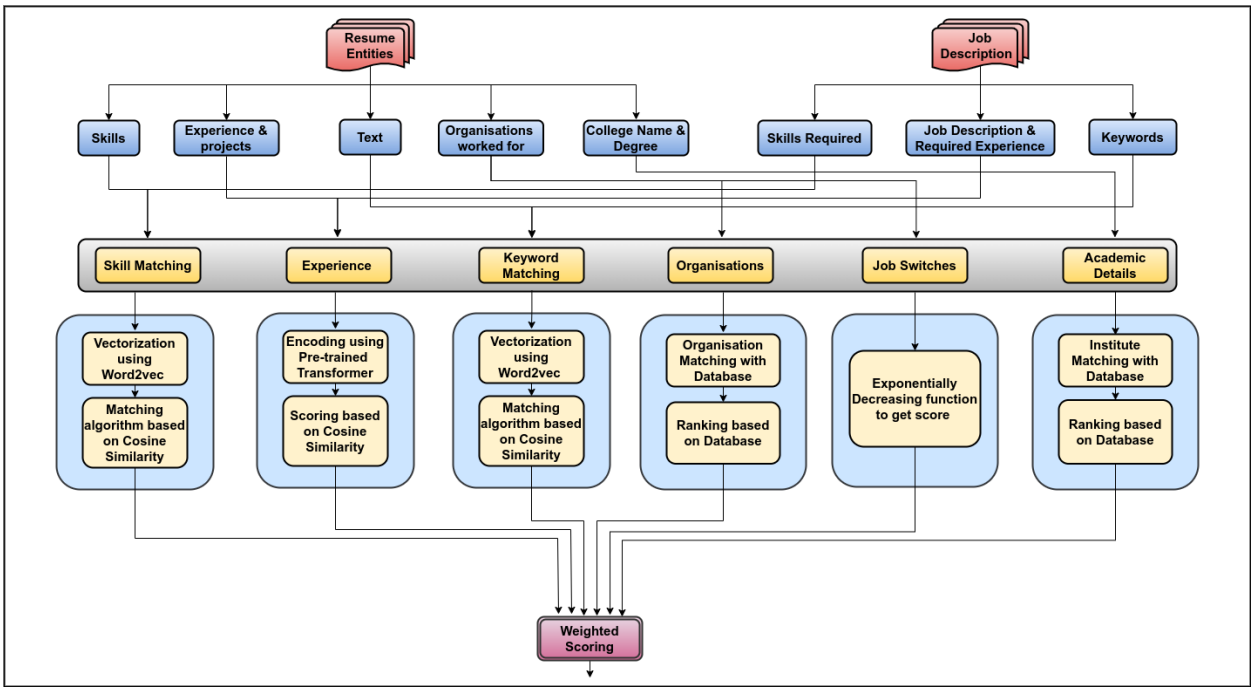
Note : In case you are not using the path mentioned in the guidelines above, please consider changing the path to the resume directory locally, on the `test.py`
file as

```
resume_dir = "<path/to/resume/directory>"
resumes = os.listdir(resume_dir)
```

Check out the Training Notebooks for training details of Fuzzyset, NER and Word2Vec.

Ranking Algorithm

Workflow Diagram



Arguments passed to the Ranking Algorithm

- **Resume Entities**: Extracted from Resumes uploaded by Job Searchers

```
{
    "skills": [],
    "college_name": [],
    "experience": [],
    "company_names": [],
    "total_experience": [],
    "jd_key_words":[],
    "degree":None,
    "text": None,
}
```

- **Job Description**: Fetched from UI, to be uploaded by Job Recruiters

```
{
    "title":
        "SDE",
    "desc":
        "Individual Contributor Role focused on developing RIA application using ML",
    "exp":
        "Bachelor's degree in Engineering. \nGood spoken and written English skills to effectively communicate technical concepts. \nStrong
    "skills":
        [{ "skill":"Python", "value":75},{ "skill":"machine-learning", "value":25}],
    "keywords":
        [{"keyword":"Python"}, {"keyword":"ML"}],
    "tags":
        "Internship"
}
```

Code Structure and Main Files

- Main Modules

```
|---- algo
        |---- parser (module for parsing the resumes)
                |---- constants.py (lists of subsections/names/stopwords/etc.)
                |---- utils.py (utility functions called by parser)
                |---- parser.py (main class for parsing)
        |---- criteria (module for assigning scores based on each criteria)
                |---- academicCriteria.py (scoring based on academic institutes)
                |---- domainCriteria.py (scoring based on domain experience)
                |---- jobSwitchCriteria.py (scoring based on no. of job switches)
                |---- keywordCriteria.py (scoring based on keyword matching bw resume and jd)
                |---- organizationCriteria.py (scoring based on organizations worked in)
                |---- skillRankCriteria.py (scoring based on skill match with jd)
        |---- database (folder storing data regarding orgs, skills, and academic institutions)
                |---- organizations.txt (file storing organization rankings)
                |---- Engineering.csv (file storing engg clg rankings)
                |---- Management.csv (file storing mgmt clg rankings)
                |---- skills.csv (fiel storing skillsets)
```

- Helper Modules and Files

```
|---- algo
        |---- ner (called in parser)
                |---- spacyNer (uses spacy ner fine-tuned with roberta transformers for extraction of college names)
                          |---- config.py
                          |---- ner.py
        |---- utils.py (file containing utility functions)

|---- config (module containing file ids for pre-trained weights)
        |---- weights_download.json
|---- weights (module for downloading trained ner, word2vec, and other helper models)
        |---- download.py (download script for weights download from gdrive via gdown)
```

- Main Scoring Files

```
|---- algo
        |---- ranker.py (file returning weighted average of scores based on individual criterion, makes function calls to all main modules
|---- test.py (file calling the ranker module and returing ranks of a bunch of resumes in a table by a simple sorting mechanism of scores a
```

Working

High Level

- Arguments (resume_path, jobdescription) are passed within the `../test.py` file to the function called using `Scorer()` object from `ranker.py`
- All objects from `parser` and `criteria` module are initialized

- All entitities are extracted from the resumes using the `ResumeParser()` object
- Scores for each criterion are calculated and stored using objects created from `criteria` module, passing corresponding entities keys and jobdescription keys as arguments
- Each score is multiplied by their weights and stored in a list
- Overall score is calculated by taking the mean of the weighted scores and appended to the beginning of the list
- This list is returned to the `../test.py` file
- The returned list is sorted in descending order based on the overall scores for each resume and displayed in a table

Details of Each Criterion

- **academicCriteria**:

```
- College name and Degree are passed as arguments
- Fuzzysets is applied to extract a generalized name for the institutions
- Institute matching with database depending upon the Degree (Engineering/Management)
- Database has the names of the institutes ordered by rank, hence (index of institute)+1 gives the rank
- Scoring is based on this index passed in a mathematical function: 1000 * math.exp(-(ranks) / 400)
- Normalized score (using sigmoid function) is returned
```

- **domainCriteria**:

```
- Experience and Job Description is passed as arguments
- Encoding of both arguments is performed using pre-trained transformers
- Scoring is based on "cosine similarity"
- Mean score is returned
```

- **jobSwitchCriteria**:

```
- List of organizations is passed as argument
- Length of the list of organizations is calculated to find the number of job switched
- Scoring is based on a hand coded exponentially decreasing function
- Normalized score is returned
```

- **keywordCriteria**:

```
- Text extracted from the resume and Keywords given in Job Description are passed as arguments
- More keywords are extracted from the job description and stored
- Vectorization is performed using "word2vec"
- In case word2vec results in KeyError, "tfidf" is used
- Scoring is based on maching using "cosine similarity"
- Normalized score is returned
```

- **organizationCriteria**:

```
- List of organizations is passed as argument
- Fuzzysets is applied to extract a generalized name for the organization
- Organization matching with database
- Database has the names of the organizations ordered by rank, hence (index of institute)+1 gives the rank
- Scoring is based on this index passed in a mathematical function: 1000 * math.exp(-(ranks) / 400)
- Normalized score (using sigmoid function) is returned
```

- **skillRankCriteria**:

```
- Skills extarcted from resume and weighted skills mentioned in the jobdescription are passed as arguments
- Vectorization is performed using "word2vec"
- In case word2vec results in KeyError, "tfidf" is used
- Scoring is based on maching using "cosine similarity"
- Score calculated for each is multiplied with its given weight and added
- Normalized score is returned
```

Import

The `ranker.py` file is called in the main test file to get the final scoring of the resumes based on which ranks are calculated.The class from `ranker.py` can be imported into `../test.py` and initialized by:

```
from algo import Scorer
scorer = Scorer()
```

References

References

Aa Reference Type	Reference Link
Datasets	• Resume Dataset • Resume Corpus • Online Job Posting
Blogs	• NLP made easy using SpaCy • Keyword Extraction with NLP • Keyword Extraction Process with NLP • tf-idf vs word2vec vs BERT • Information Extraction from CV
Libraries	• spacy • pyresparser • resume-parser • transformer • fuzzyset
Research Papers	• Duplicate Resume Checker (Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks) • Resume Ranker IOSRJEN
Notebooks & Pre-trained weights	• Guide to NLP in SpaCy • Google word2vec • Google Word2Vec pretrained weights

- Check out the documentation for parser initialization here.
- Check out the documentation for fuzzyset file generation here.
- Check out the documentation for each criteria initialization here.