

A Report on



Submitted to

at

IBM Hack Challenge 2020

by

Team Blueprint

Yashasvi Misra

Siddhant Thakur

Shushrut Sharma

Mihir Singh

CONTENTS

1. Introduction	
1.1. Overview.....	08
1.2. Purpose.....	08
2. Literature Survey	
2.1. Existing Problem.....	08
2.2. Proposed Solution.....	08
3. Theoretical Analysis	
3.1. Block Diagram.....	08
3.2. Software Design.....	08
4. Experimental Investigation	
5. Flowchart	
6. Result	
7. Advantages and Disadvantages	
8. Applications	
9. Conclusion	
10. Future Scope	
11. Bibliography	
12. Appendix	

1. Introduction

1.1 Overview

Recruitment refers to the overall process of attracting, shortlisting, selecting and appointing suitable candidates for jobs (either permanent or temporary) within an organization. Recruitment can also refer to processes involved in choosing individuals for unpaid roles. Managers, human resource generalists and recruitment specialists may be tasked with carrying out recruitment, but in some cases public-sector employment agencies, commercial recruitment agencies, or specialist search consultancies are used to undertake parts of the process. Internet-based technologies which support all aspects of recruitment have become widespread.

1.2 Purpose

The project aims to develop a prototype of a platform which can help streamline and scale up the hiring process by automating the steps of recruitment that involve accepting applications, gathering data and insights into the candidate pool, comparing candidates, ranking and shortlisting all applicants. Such a platform could prove to be useful in carrying out large scale recruitment campaigns by narrowing down the candidate pool for any specific job posting.

2. Literature Survey

2.1 Existing Problem

Problem Statement: AI Recruiter – Shortlist a Suitable candidate for a specific Job Role.

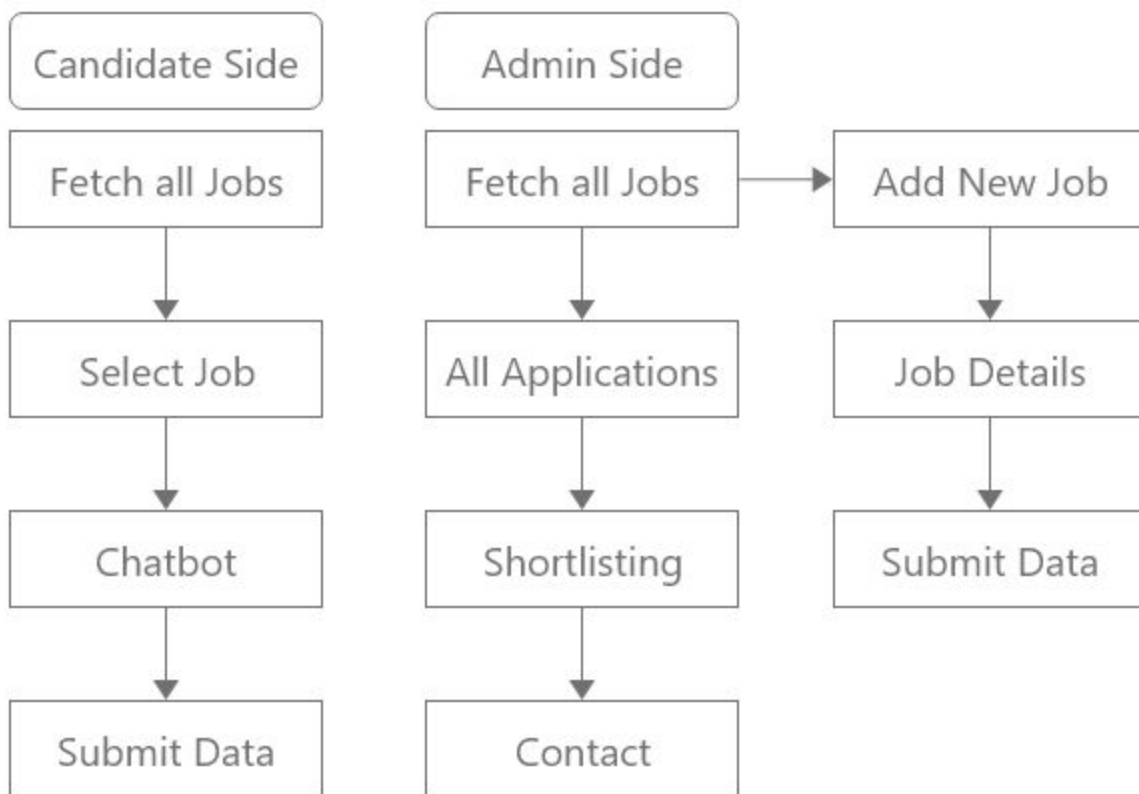
52% of talent acquisition leaders say that the most difficult part of their job is to shortlist the right candidate and 3% of candidates never hear back from a company after one touchpoint. On the flip side, it's a challenge for employers to communicate well with all their candidates. For high volume recruiting, this would require communicating with thousands of candidates, in addition to a recruiter's normal screening functions and other duties. Artificial Intelligence enabled software bots can definitely provide a solution for this problem.

2.2 Proposed Solution

Our proposed solution is a prototype of a platform which augments the hiring process by introducing a "filter" which acts, not to replace the traditional interview, but as a preliminary interview to gather data about each applicant in the candidate pool. It helps shortlisting the large number of applicants and narrow them down to a select few by ranking them against specified criterias for the job they are applying for. A chatbot conducts an interview wherein it gathers all data and responses to common interview questions along with an aptitude test to judge the candidate's logical, verbal and quantitative reasoning. This data is relayed to the hiring team for generating insights and shortlist the favourable candidates.

3. Theoretical Analysis

3.1 Block Diagram



1.2 Software Design

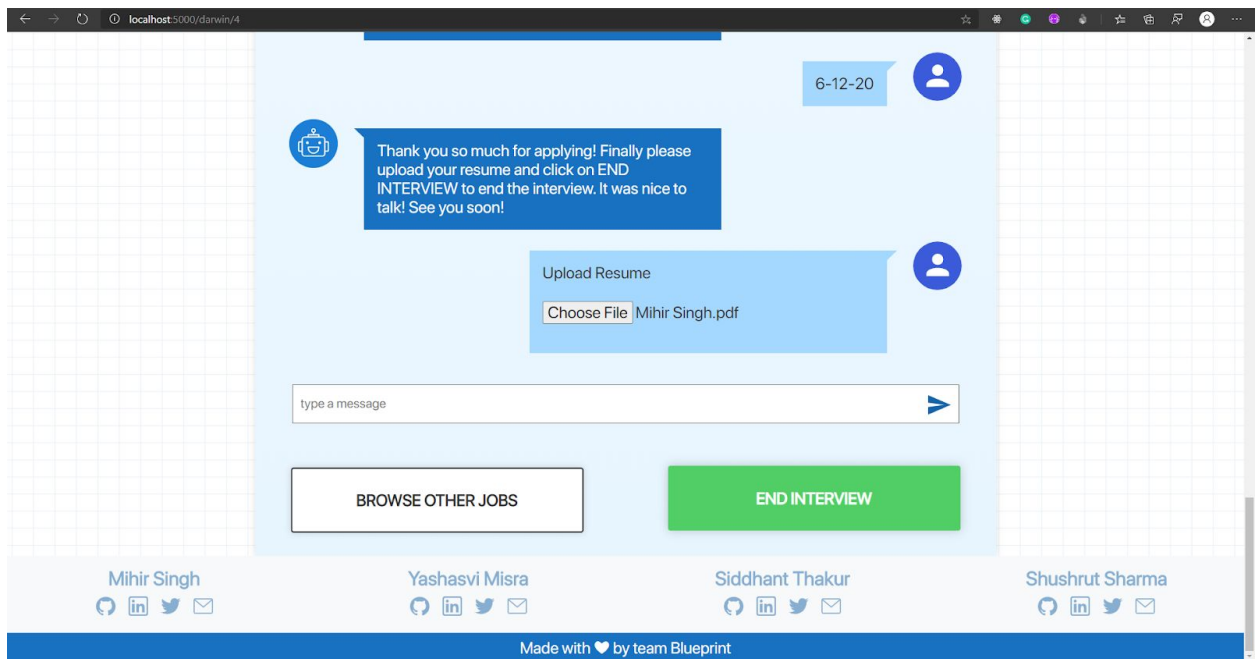
What we are proposing, is a family of systems under one roof working together, both linearly and isometrically, to handle a large scale of applications by narrowing the crowd through filters defined by the requirements requested by the organisation. An established pipeline for shortlisting candidates across popular platforms, engaging through automated responses and artificially intelligent conversational agents and selecting recruits based on analysis of data

points gathered for each candidate throughout the process. A tunable workflow that lets any organisation handle recruitment campaigns of vast scale but with fine quality.

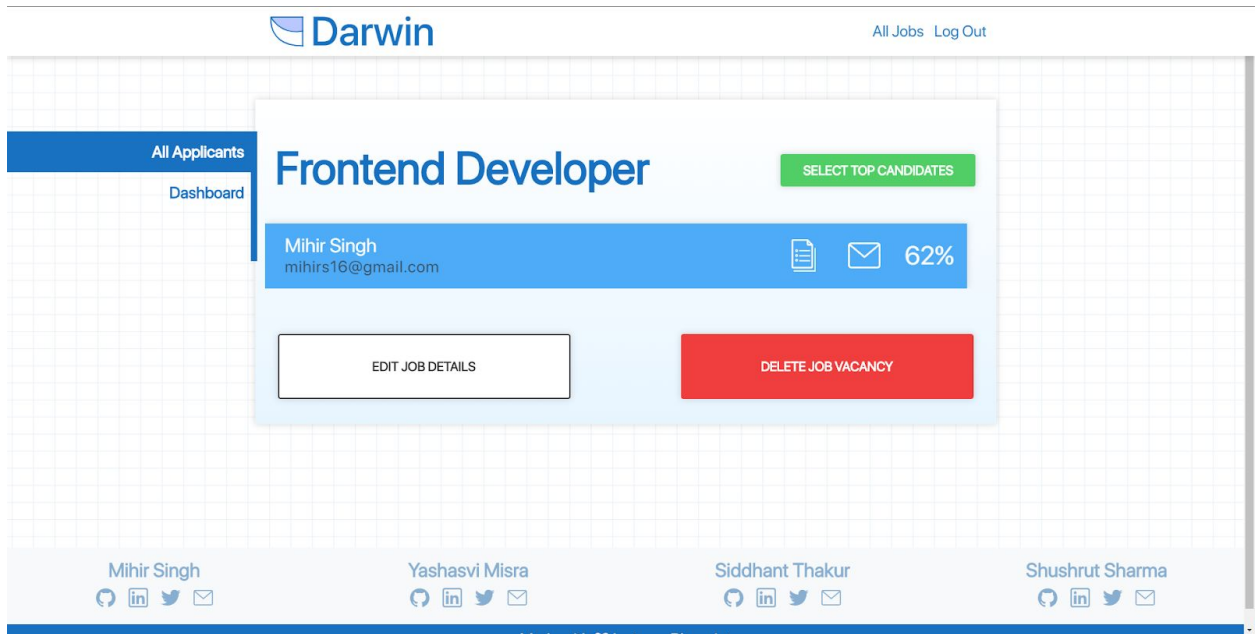
- Candidate Side
 - Powered by APIs from Twitter and GitHub to record and validate the work profiles of applicants.
 - IBM Watson's Discovery and/or Knowledge Studio to gain insights from resume's and submitted forms and documents.
 - Powered by IBM Watson, a conversational agent to handle all a preliminary communication with candidates to judge interest and personality.
 - IBM Watson's Personality Insights powers the insights into a candidates portfolio through the Big 5 and Moral qualities.
- Admin Side
 - Secure Login separate from the candidate side
 - Database of candidates and organisations' requirements as SQL Databases hosted on the IBM Cloud's DB2 and Cloud Object Storage services.
 - Analytics pipeline to gain valuable information from the gathered candidates' data and rank it w/ comparison to an ideal pseudo-profile built according to the job requirements.

4. Experimental Investigation

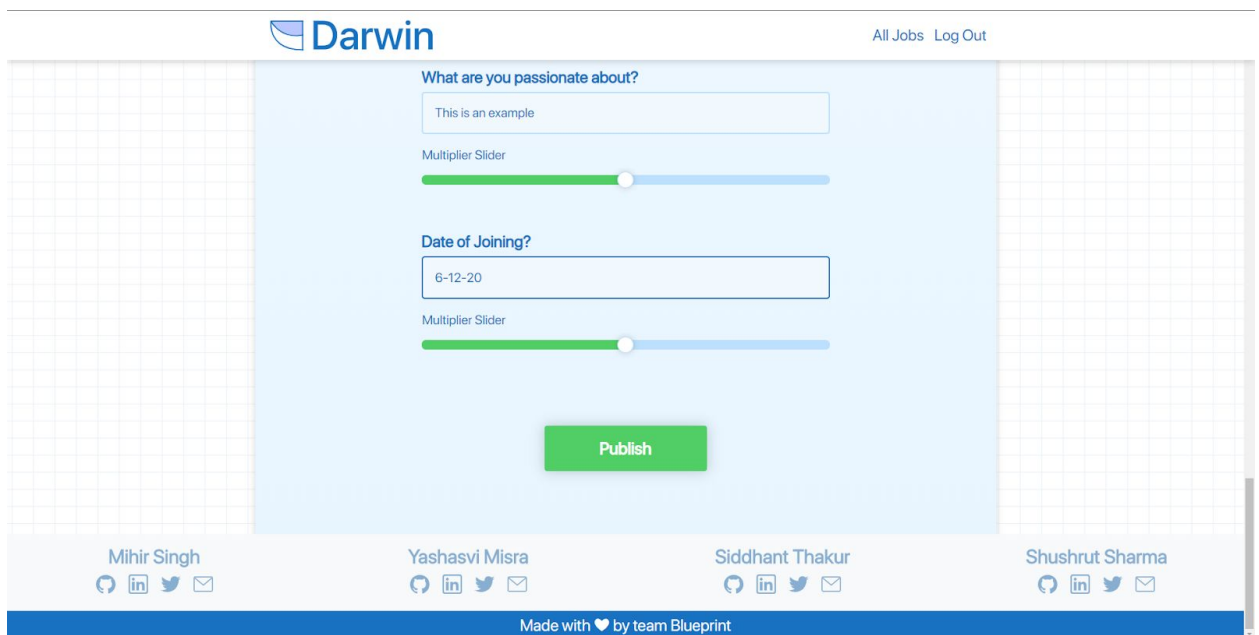
- Adding a new Candidate



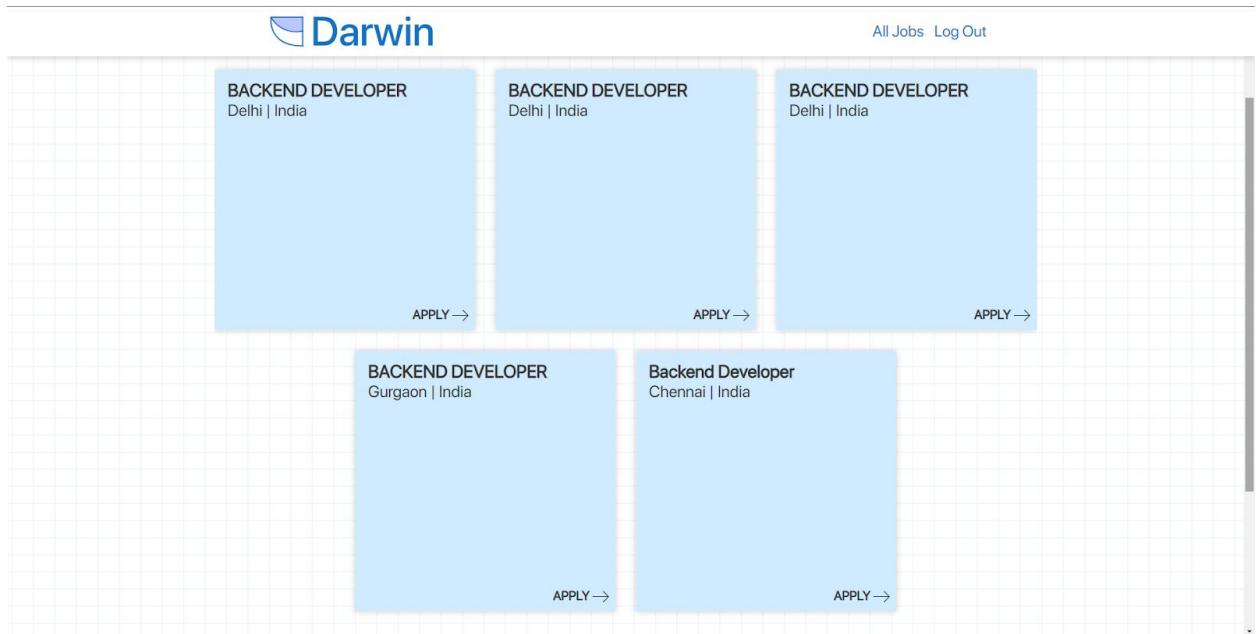
- Candidate Scored and confirmed



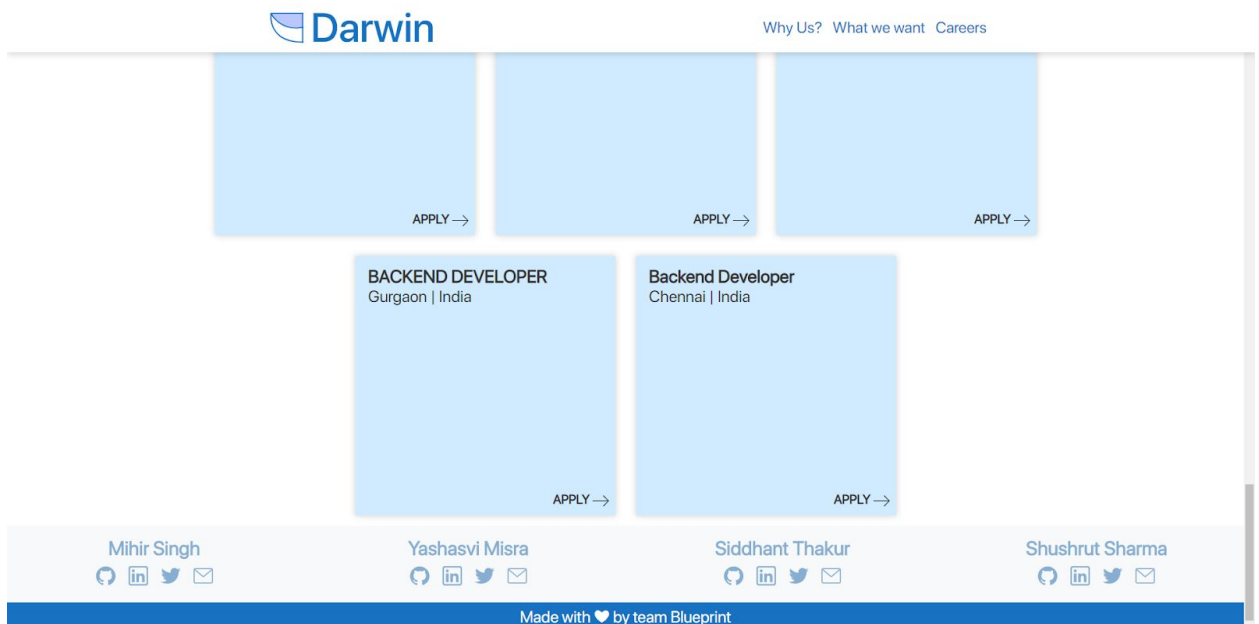
- Adding new Job Posting



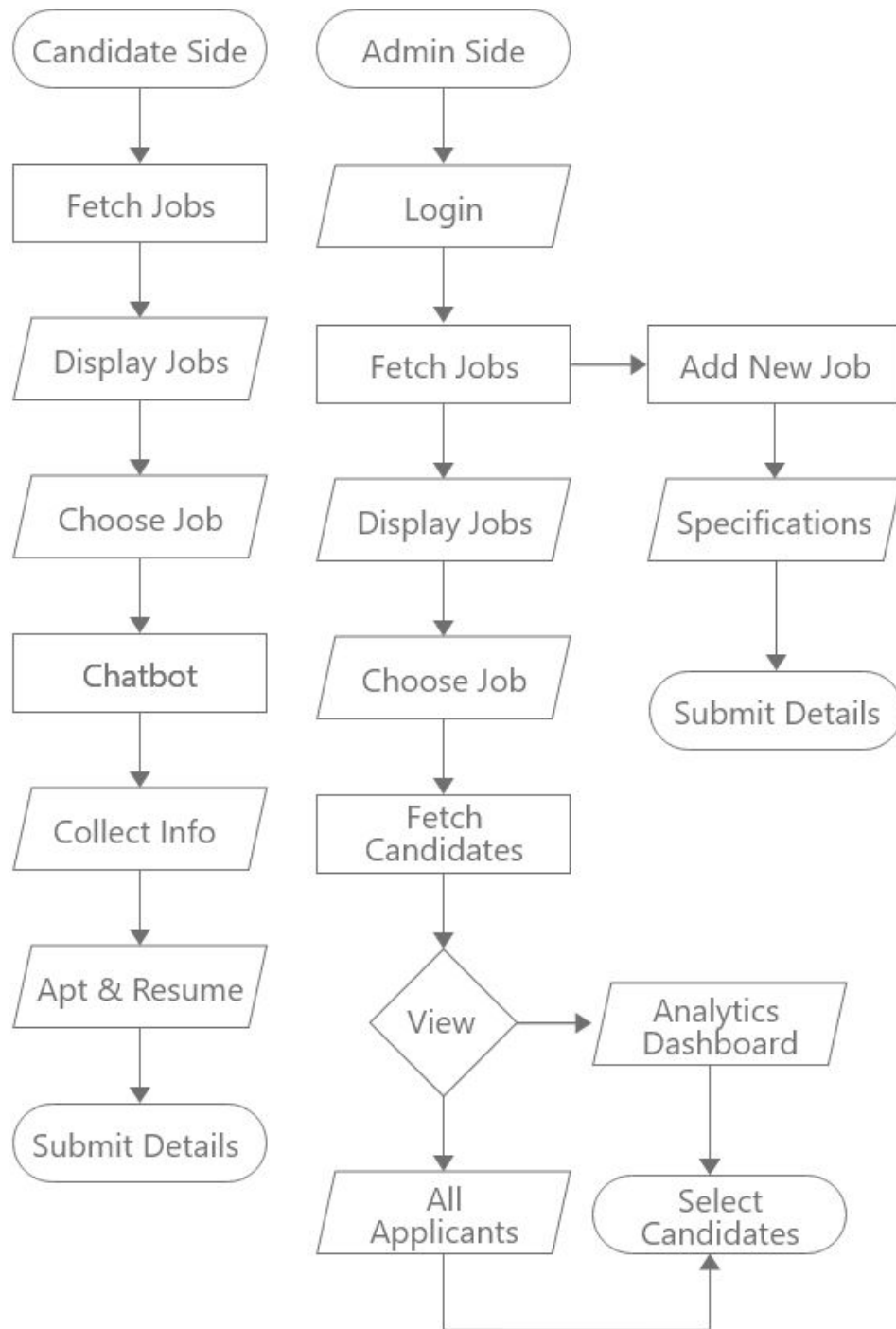
- Confirm Posting on Admin Side



- Confirm Posting on Candidate Side



5. Flowchart



6. RESULT

As a result of the efforts of the team members and the guidance of their mentors and advisors, they were able to develop a functioning prototype that demonstrates the filtering and ranking of candidates through the platform.

7. ADVANTAGES & DISADVANTAGES

7.1 Advantages

- More efficient hiring process.
- Less labor required to carry out recruitments.
- Recruitment campaigns of a larger scale with greater ease.
- Streamlined recruiting process with every action available through a one-stop platform.

7.2 Disadvantages

- Being judged by a machine might not be favorable to candidates.
- The platform has loopholes such as the event of absence of a social account on twitter.
- The platform cannot directly replace the interview process, but augment it as a preliminary filter.

8. APPLICATIONS

- Large scale recruitments augmented through filtration

- Smaller scale recruitments streamlined with increased efficiency
- A unique use-case can also be covered by minimal tuning to implement the same platform to judge current employees and rank them against each other

9. CONCLUSION

The project was concluded at a stage where it is a functioning prototype with the stages of application and shortlisting working fine using Web and Data technologies including but not limited to IBM Cloud Services. This project acts as a proof-of-concept for possible later development to a production level.

10. FUTURE SCOPE

- The chatbot could be enhanced to ask more questions with deeper insights
- The platform for shortlisting itself could be improved in order to cover candidates without an active GitHub and/or Twitter account
- The aptitude test can be made more comprehensive with a larger question bank
- The platform could be made more robust by allowing sign up and registration of candidates and multiple companies

11. REFERENCES

1. [Wikipedia.org](https://www.wikipedia.org)
2. [IBM Cloud Documentation](https://www.ibm.com/cloud/docs/)

3. [Natural Language Toolkit — NLTK 3.5b1 documentation](#)
4. [Pandas Documentation](#)
5. [Plotly for JavaScript Graphing Library](#)
6. [Flask - Microweb Framework Docs](#)
7. [Stack Overflow - Where Developers Learn, Share, & Build Careers](#)
8. [The world's leading software development platform · GitHub](#)
9. [Project Jupyter | Home](#)

12. Appendix

The complete codebase can be found at the [GitHub Repository](#).

```

import threading
import database
import json
import os
import time

import stats
import flask
import flask_login
import pandas as pd
from ast import literal_eval
from flask_cors import CORS
from werkzeug.utils import secure_filename
from k3y5 import ADMIN_USERNAME, ADMIN_PASSWORD, ADMIN_KEY

app = flask.Flask(__name__)
login_manager = flask_login.LoginManager()
login_manager.init_app(app)
CORS(app, support_credentials=True)
app.config['FILE_UPLOADS'] = 'resume_uploads'

# ----- flask login setup -----
app.secret_key = ADMIN_KEY
users = {ADMIN_USERNAME: {'password': ADMIN_PASSWORD}}

class User(flask_login.UserMixin):
    pass

@login_manager.user_loader
def user_loader(email):
    if email not in users:
        return
    user = User()
    user.id = email
    return user

@login_manager.request_loader
def request_loader(request):
    email = request.form.get('email')
    if email not in users:
        return
    user = User()
    user.id = email
    user.is_authenticated = request.form['password'] == users[email]['password']
    return user

# ----- public services -----
@app.route('/', methods=['GET'])
def homepage():
    return flask.render_template('homepage.html')

@app.route('/darwin/<jobid>', methods=['GET'])
def darwin(jobid):
    return flask.render_template('chatbot.html', jobid = jobid)

@app.route('/data/getAllJobs', methods=['GET'])
def getAllJobs():
    allJobs = database.getAllJobs()
    return flask.jsonify(allJobs)

@app.route('/data/getResume/<id>', methods=['GET'])
def getResume(id):
    database.resume_vault.download_item(id, os.path.join(os.path.join(app.root_path, app.config['FILE_UPLOADS']), id+str('.pdf')))
    del_thread = threading.Thread(target=delay_delete, args=(5, os.path.join(os.path.join(os.path.join(app.root_path, app.config['FILE_UPLOADS']), id+str('.pdf')))))
    del_thread.start()
    return flask.send_from_directory(directory=os.path.join(app.root_path, app.config['FILE_UPLOADS']), filename=id+str('.pdf'))

```

```

def delay_delete (t, path):
    print ("started")
    time.sleep(t)
    print ("trying to delete")
    os.remove(path)
    print ("done")
    return

@app.route('/data/newCandidate', methods=['POST'])
def newCandidate():
    resumeFile = flask.request.files['resumeFile']
    candyInfo = json.loads(flask.request.form['jsonInput'])
    path = os.path.join(app.config["FILE_UPLOADS"], resumeFile.filename)
    resumeFile.save(path)
    print (":adding new candidate")
    print (candyInfo)
    database.add_candidate({
        "jobId": str(candyInfo['jobid']),
        "cname": candyInfo['cname'],
        "email": candyInfo['email'],
        "gitId": candyInfo['gitId'],
        "tweetId": candyInfo['tweetId'],
        "yoe": int(candyInfo['yoe']),
        "jobskills": candyInfo['jobskills'],
        "self_desc": candyInfo['self_desc'],
        "job_want_why": candyInfo['job_want_why'],
        "job_req_what": candyInfo['job_req_what'],
        "passion": candyInfo['passion'],
        "date_join": candyInfo['date_join'],
        "apt": candyInfo['apt']
    }, path)
    print (":new candidate added")
    os.remove(path)
    return flask.redirect(flask.url_for('homepage'))

@app.route('/data/getQuestions', methods=['GET'])
def getQues():
    _qt = pd.read_csv('data_src/g4g/quants.csv', converters={"OPT": literal_eval}, dtype=str)
    _lg = pd.read_csv('data_src/g4g/logix.csv', converters={"OPT": literal_eval}, dtype=str)
    _vr = pd.read_csv('data_src/g4g/verbs.csv', converters={"OPT": literal_eval}, dtype=str)

    qt = _qt.sample(n=5)
    lg = _lg.sample(n=5)
    vr = _vr.sample(n=5)

    response = {
        "que": list(qt['QUE'].values) + list(lg['QUE'].values) + list(vr['QUE'].values),
        "opt": list(qt['OPT'].values) + list(lg['OPT'].values) + list(vr['OPT'].values),
        "ans": list(qt['ANS'].values) + list(lg['ANS'].values) + list(vr['ANS'].values)
    }
    print (response)
    return flask.jsonify(response)

# ----- admin services -----
@app.route('/admin', methods=['GET', 'POST'])
def admin():
    if flask.request.method == 'GET':
        return flask.render_template('admin_login.html')

    email = flask.request.form['email']
    if email in list(users.keys()):
        if flask.request.form['password'] == users[email]['password']:
            user = User()
            user.id = email
            flask_login.login_user(user)
            return flask.redirect(flask.url_for('billboard'))
    return flask.redirect(flask.url_for('admin'))

@app.route('/admin/billboard')
@flask_login.login_required
def billboard():
    return flask.render_template('job_billboard.html')

```



```

@app.route('/applicants/<jobid>')
@flask_login.login_required
def applicants(jobid):
    return flask.render_template('job_applicants.html', jobid=jobid)

@app.route('/admin/jobdetails/add', methods=['GET', 'POST'])
@flask_login.login_required
def jobdetails_add():
    if flask.request.method == 'POST':
        jobData = flask.request.json
        database.add_job(jobData)
        return flask.redirect(flask.url_for('billboard'))
    return flask.render_template('job_manage.html')

@app.route('/data/admin/getCandidates/<jobid>')
@flask_login.login_required
def getCandidates(jobid):
    allCandidates = database.getAllCandidates(jobid)
    return flask.jsonify(allCandidates)

@app.route('/data/admin/getJobStats/<jobid>')
# @flask_login.login_required
def getJobStats(jobid):
    allCandidates = database.getStats(jobid)
    doj = [can["Date_Of_Joining"] for can in allCandidates]
    ski = [can["Skill"] for can in allCandidates]
    yoe = [can["Year_of_Experience"] for can in allCandidates]
    ovr = [can["overall_score"] for can in allCandidates]
    dfAllCan = pd.DataFrame(data = {'Date_Of_Joining': doj, 'Skill': ski, 'Year_of_Experience': yoe, 'overall_score': ovr})
    ski_x, ski_y = stats.create_ski(dfAllCan)
    plotData = {
        "perc": {
            "x": list(stats.create_perc(dfAllCan).index),
            "y": [int(x) for x in list(stats.create_perc(dfAllCan).iloc[:, 0].values)]
        },
        "doj": {
            "x": list(stats.create_doj(dfAllCan).index),
            "y": [int(x) for x in list(stats.create_perc(dfAllCan).iloc[:, 0].values)]
        },
        "yoe": {
            "x": [int(x) for x in list(stats.create_yoe(dfAllCan).iloc[:, 0].values)],
            "y": [int(x) for x in list(stats.create_yoe(dfAllCan).iloc[:, 1].values)]
        },
        "ski": {
            "x": [int(x) for x in ski_x],
            "y": ski_y
        }
    }
    print (plotData)
    return json.dumps(plotData)

@app.route('/logout')
def logout():
    flask_login.logout_user()
    return flask.redirect(flask.url_for('admin'))

@login_manager.unauthorized_handler
def unauthorized_handler():
    return flask.redirect(flask.url_for('admin'))
#

```

```

if __name__ == "__main__":
    # for hot reload and tracking static files and templates
    from os import path, walk

    extra_dirs = ['templates/', 'static/']
    extra_files = extra_dirs[:]
    for extra_dir in extra_dirs:
        for dirname, dirs, files in walk(extra_dir):
            for filename in files:
                filename = path.join(dirname, filename)
                if path.isfile(filename):
                    extra_files.append(filename)

    # flask app run
    app.run(debug=True, extra_files=extra_files)

```