APPLICATION TRACKING SYSTEM

SOFTWARE DOCUMENTATION

INTRODUCTION

Applying for jobs and internships is not a simple procedure. Handling job applications takes a lot of time. The entire process might be difficult because of the deadlines and references. Without the need of laborious excel spreadsheets, our program enables you to monitor, control, and oversee the job application process.

Our project, Application Tracking System, is a software application designed to streamline and manage the entire recruitment process within an organization. Its primary purpose is to automate and centralize the various stages of hiring, from receiving resumes to the final selection of candidates.

The jobs you've added to your wish list are tracked by our application. It also maintains a record of the firms you have already applied to and any rejections you have received. With our application, an applicant may directly search for potential prospects using simple keywords instead of having to peruse each company's website. The candidate can then add any potential job offers to their wishlist.

POTENTIAL CONSUMERS

1. Students looking for full-time jobs and internships.
2. Individuals who are employed yet would like to start a new job.
3. Individuals who changed careers and are currently searching for chances in a different field.

TECH STACK

Front-End:

1. React
2. Node.js
3. HTML and CSS

Backend:

1. Python, Flask
2. Pytest

Database:

1. MongoDB

FUNCTIONALITIES:

app.py: This is the file that generates the Flask server - upon calling “flask run” within this directory, this is the file that will be called.

1. search(): HttpGet endpoint for getting search results on keyword e.g. job title, company name etc.
   a. Route: /search
   b. Input:
      i. keywords: A plain text input.
   c. Output:
      i. records: Returns a Json of searched results.

2. get_data(): HttpGet endpoint for fetching all the saved job data.
   a. Route: /application
   b. Output:
      i. apps_json: Return a Json of all the job information.

3. add_application(): HttpPost endpoint for saving new application.
   a. Route: /application
   b. Input:
      i. application: Job details
   c. Output:
      i. application: Return the Json of the newly saved application.

4. update_application(): HttpPut endpoint for updating an existing application.
   a. Route: /application
   b. Input:
      i. application: Updated job details
   c. Output:
      i. application: Return the Json of the updated application.

5. delete_application(): HttpDelete endpoint for deleting an existing application.
   a. Route: /application
   b. Input:
      i. application: Job details which will be deleted.
   c. Output:
      i. application: Return the Json of the deleted application.

TEST CASES:

For functional testing of the backend of our Flask application and database, we have a separate file named test_app.py in the backend directory and included pytest framework.

Test 1 (test_alive): In this test, we tested the GET method of our Flask client and tested whether the Flask application is live or not, by checking the string output of the client app.

Test 2 (test_search): In this test, we tested the search method of our application by asserting a mock test search and checking a fixed output.

Test 3 (test_get_data): In this test, we tested the GET method of the Flask client to get all the entries of the database and checked the database entries by mocking the output.

Test 4 (test_add_application): In this test, we tested the POST method of the Flask client to add a new entry to the database and checked the added data entry by mocking the output.

Test 5 (test_update_application): In this test, we tested the PUT method of the Flask client to update any entry of the database and checked the updated data entry by mocking the output.

Test 6 (test_delete_application): In this test, we tested the DELETE method of the Flask client to delete any entry from the database and checked the deleted data entry by mocking the output.

Test 7 (test_get_new_id): In this test, we tested the getting_new_id function and checked the output is returning the correct next new id for a new application added to the database by mocking the output of MongoEngine objects function.

Test 8 (test_alive_status_code): In this test, we tested the GET method of our Flask client and tested whether the Flask application is live or not, by checking the status code of the client app.