Architecture

Job Recommender System

Revision Number: 1.0

Last date of revision: 20/12/2023

# 

# 

# 

# 

# 

# 

# 

# 

# Document Version Control

**Change Record**

| Date Issued | Version | Description | Author |
| --- | --- | --- | --- |
| 20/12/2023 | v1.0 | First Draft | Abraham Audu |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Table of Contents**

[Document Version Control 2](#_heading=h.gjdgxs)

[Abstract 4](#_heading=)

[1. **Introduction**](#_heading=) 5

[1.1 Why this Architecture Document? 5](#_heading=)

[1.2 Scope 5](#_heading=h.d1qeboog8101)

[1.3 Constraints 5](#_heading=h.25had5vi2514)

[1.4 Risks 5](#_heading=h.ko1w5tjrieq1)

[1.5 Out of Scope 5](#_heading=h.3mgtvva5v9t0)

[**2. Technical Specifications 6**](#_heading=h.ggkmsajlo83h)

[2.1 Dataset Overview 6](#_heading=h.ik6kx6qkz252)

[2.2 Predicting Monthly Payment Default 6](#_heading=h.3a339y863l5j)

[2.3 Logging 6](#_heading=h.7nfstg7w5zjh)

[2.4 Deployment 6](#_heading=h.u0x9ys6hgr05)

[**3. Technology Stack 7**](#_heading=h.6vm6ejoevwnp)

[**4. Proposed Solution 7**](#_heading=h.anf9p88rjr2c)

[**5. Model Training and Validation Workflow 7**](#_heading=h.ut8anbobgfdx)

[**6. User I/O Workflow 8**](#_heading=h.1l7gex6ibgcx)

[**7. Exceptional Scenarios 8**](#_heading=h.l3ugmcy4seyh)

[**8. Unit Test Cases 9**](#_heading=h.m04g3yo1j0pj)

[**9. Key Performance Indicators (KPIs) 9**](#_heading=h.36tw07kh7b0k)

# Abstract

Banking institutions offer credit to customers as one of their major businesses and this comes with some associated credit risk. WIth millions of customers, it is important to have an automated system which can filter through customer information and financial history to determine the probability that a card owner will default in order to limit their access to current or future credit services and minimise risk to the issuing organisation. This project covers the implementation of a machine learning based platform which is able to take a credit card owner’s information and financial history and predict the probability that they will be credit defaulters.

# Introduction

## Why this Architecture Document?

The purpose of this architecture document is to present a detailed description of the Job recommender System solution. It will explain the purpose and features of the system, the interfaces of the system, how the system will operate and the reaction of the system to specific action by the end user. This document is intended for both stakeholders and the developers of the system, and will be proposed to higher management for its approval.

The main objective of this project is to recommend the best suited jobs to users based on their professional profiles and personal preferences. The user profiles consist of information such as skills, previous work experience and job preferences. Other user data includes the users previous searches and interactions on the platform. All these data points will be considered by the recommender system when comparing jobs to user data in order to shortlist the most relevant jobs to the user when they make searches.

This project shall be delivered in two phases:

Phase 1: Recommender development

Phase 2: API and UI development

## 1.2 Scope

This software system will be a web application designed to collect jobs from job websites and return job search results which are relevant to the user by comparing the vectorised user data with job data vectors in order to find the best matches.

## 1.3 Constraints

The system will only work with the job websites the scrapers have been configured with, and the recommendations will be document based, as there is no pre-existing user interaction data to train a collaborative filtering model.

## 1.4 Risks

As with all technological implementations, there is bound to be inaccuracies and glitches in the system. Seeing as this system is designed for searching for jobs, it is important to still read through the recommended jobs to ensure they are actually a good fit.

## 1.5 Out of Scope

This project is designed to refer users back to the original job sites and will not claim ownership of job listings.

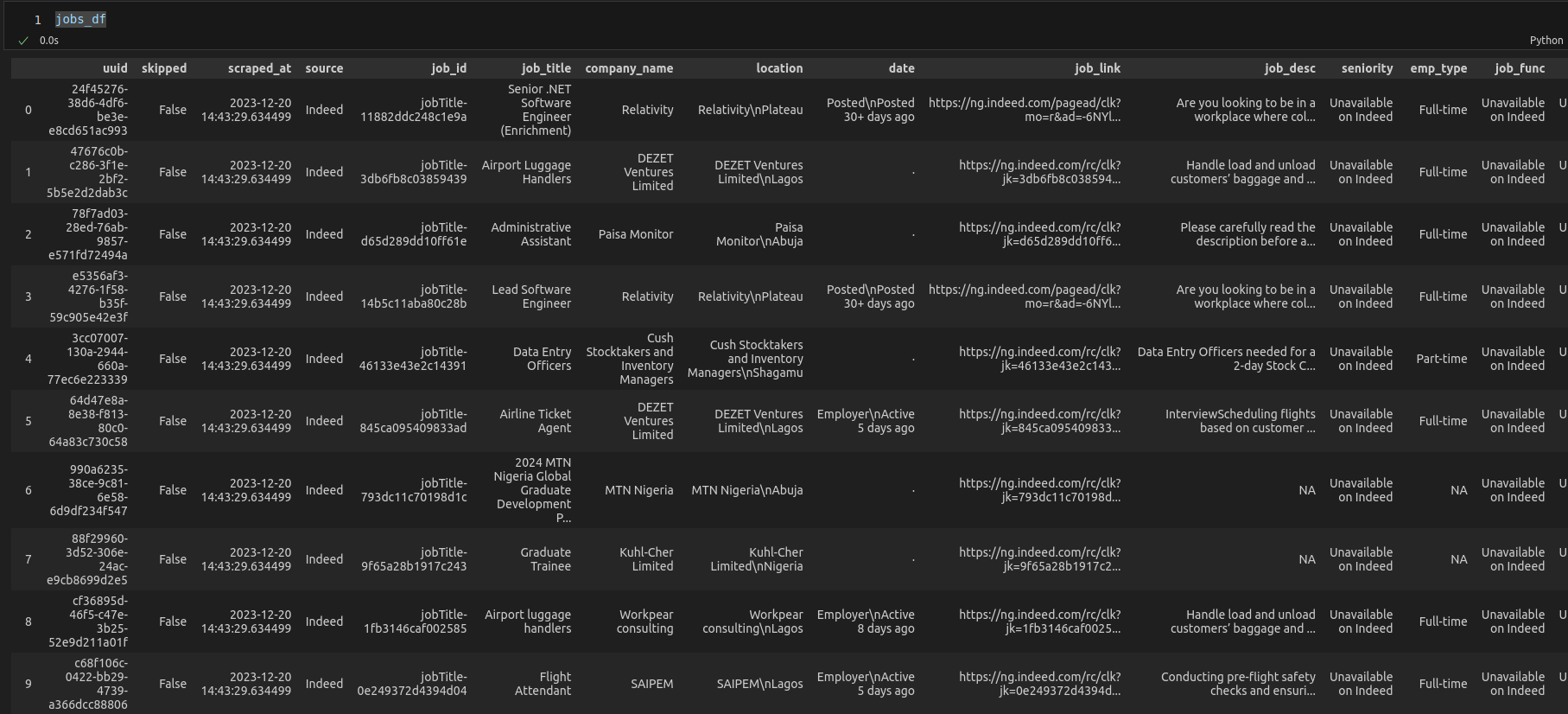
# 2. Technical Specifications

## 

## 2.1 Dataset Overview

The jobs data scraped from job listing websites contains the following fields:

* uuid (UUID): Primary key representing the unique identifier for each job listing.
* skipped (Boolean, optional): Indicates whether the job was skipped during processing.
* scraped\_at (DateTime): Timestamp indicating when the job listing was scraped.
* source (Text, optional): Source of the job listing.
* job\_id (Text, optional): ID associated with the job.
* job\_title (Text): Title of the job.
* company\_name (Text, optional): Name of the company offering the job.
* location (Text, optional): Location of the job.
* date (Text, optional): Date of the job posting.
* job\_link (Text): Link to the job listing.
* job\_desc (Text): Description of the job.
* seniority (Text, optional): Level of seniority associated with the job.
* emp\_type (Text, optional): Type of employment for the job.
* job\_func (Text, optional): Function or role of the job.
* ind (Text, optional): Industry associated with the job.



*Sample Dataframe Before Upload to Database*

## 2.2 Retrieving Job Recommendations

* The user is prompted to log in
* The user is prompted to update their profile information if not already updated
* The user is prompted to search for a job by job title
* The system will then use the user’s profile information as well as usage history to fetch the most relevant jobs to the user based on their details and search query.

## 2.3 Logging

The system will log activities on three layers:

* The ETL pipeline for scraping jobs
* The routes and server side
* User activity on the interface

Logs will be saved to dedicated files for the different layers of the application

## 2.4 Deployment

The Application will be deployed locally.

### 2.4.1 Local Deployment Details

Code Repository: <https://github.com/theabrahamaudu/Multi-Platfrom_Job_Recommender>

Refer to *README.md* on the code base for *Quick Start* steps.

### 2.4.2 Cloud Deployment

Due to the app requiring RAM >=7.5GB, no free hosting service can support it. Running a paid instance will cost around ~$32 to run on AWS EC2 t2.large for two weeks, around which this project might be evaluated, and I’m not able to fund it at this time.

# 3. Technology Stack

| **Database** | Cassandra |
| --- | --- |
| **Vector Database** | Chroma |
| **Web Scraper** | Python Selenium, Firefox, Geckodriver, Cron |
| **Frontend** | Python Streamlit |
| **Backend Server** | Python FastAPI |
| **Deployment** | Docker, Local machine |

# 4. Proposed Solution

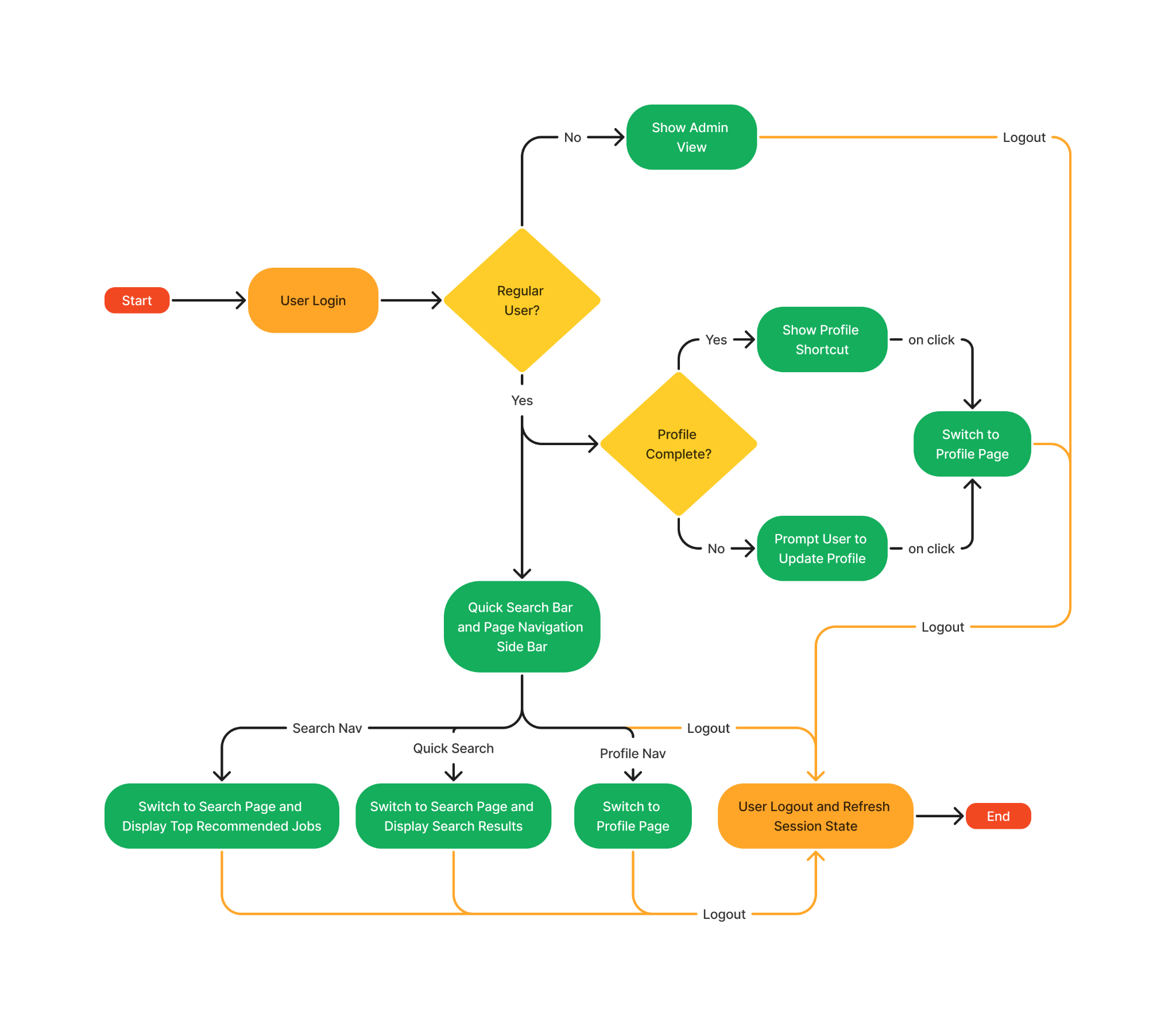
The problem is fundamentally a recommendation problem requiring a dynamic method to return contextually relevant search results on a per-user basis. It also requires being smart enough to return matches that go beyond keyword match, but a more nuanced match based on the relevance of the job to who the user is professionally.

To solve this, a document-based recommender system which will compare the contents of user profiles and their recent activity metadata to the contents of job listings will be implemented. This will be achieved by vectorizing all job listings and storing them on a vector database which can then be queried with a query vector constructed from user’s profile and the specific search query they enter via ANN search to find the top k relevant jobs for the user.

# 5. Model Training and Validation Workflow

For the initial version, a pre-trained Sentence Transformer model (all-mpnet-base-v2) is used as the vector embedding model.

# 6. User I/O Workflow



# 

# 7. Exceptional Scenarios

| **Step** | **Exception** | **Mitigation** | **Module** |
| --- | --- | --- | --- |
| Fetch user data for login | Error retrieving user details | Log the error to the backend log and return response with 404 not found error code | user.py |
| Create new user | User with same ID already exists | Log the error to the backend log and return response with 409 conflict error code | user.py |
| Load config YAML file | Error reading file | Log error message | cassandra\_conn.py |
| Connect to Cassandra cluster | Error connecting to cluster | Log error message | cassandra\_conn.py |
| Create keyspace | Error creating keyspace | Log error message | setup\_db.py (cassandra) |
| Create tables | Error creating tables | Log error message | setup\_db.py (cassandra) |
| Drop tables | Error dropping tables | Log error message | setup\_db.py (cassandra) |
| Load job vectors table | Error loading table | Log error message, set table variable to None | job\_index.py |
| Search vector table | Error searching table | Log warning message, attempt to reload vector table | job\_index.py |
| Load config YAML file | Error reading file | Log error message | chroma\_conn.py |
| Connect to Chroma client | Error connecting to client | Log error message | chroma\_conn.py |
| Create new Chroma collection | Error creating collection | Log error message | setup\_db.py (chroma) |
| Delete Chroma collection | Error deleting collection | Log error message | setup\_db.py (chroma) |
| Load config YAML file | Error reading file | Log error message | admin.py |
| Check that Chroma DB is setup | No jobs collection on Chroma DB | Log warning message, setup and create collection, return 200 code if successful or 500 code if internal error | admin.py |
| Start job scraping pipeline in new process | Error creating new process | Log error, return 500 code | admin.py |
| Get count of users | Error executing query | Log error, return 500 error code | admin.py |
| Get count of jobs | Error executing query | Log error, return error code | admin.py |

# 8. Unit Test Cases

| **Test Case Description** | **Prerequisite** | **Expected Result** |
| --- | --- | --- |
| Verify that new firefox profile is created with the set naming convention | Site scraper initialization process | Default profile name followed by ten random characters |
| Verify that user searches retrieved from database are parsed as expected | response from DB is a list of dictionaries | Single string with all values from key-value pairs |
| Verify that empty searches response is handled properly | response from DB is empty list | empty string |
| Verify that user clicks retrieved from database are parsed as expected | response from DB is a list of dictionaries | list of job UUIDs |
| Verify that empty clicks response is handled properly | response from DB is empty list | empty list |
| Verify that job descriptions fetched are parsed as expected | response from DB is list of dictionaries | Single string with all descriptions concatenated |
| Verify that optional description truncation works as expected | response from DB is list of dictionaries | All descriptions in single string are truncated to the number of characters as specified in the ‘trunc’ parameter |
| Verify that user data fields are parsed as expected | response from DB is list of dictionaries | Single string with all user data values |
| Verify that user data without any of the fields is parsed without error | response from DB is list of dictionaries with some keys having ‘None’ as value | Single string with all available user data values |
| Verify that strings as single string, dictionary of strings and list of dictionary of string all return a valid embedding | each data format contains valid strings | return value as list of sequence of floats |
| Verify that the configuration class methods return the right settings with the corresponding flag | configuration yaml file with all the expected fields specified | calls to the config methods return local configs when deployment is false, and docker configs when true. |
| Verify that the hashing function works as expected | string for hashing is passed to the hashing function | the same hash value can be reproduced by directly calling the hashing library function on the string. |
| Verify that string formatting functions work as expected | inputs are valid strings | same outcomes can be achieved by manually adding the markdown formatting to the strings |

# 9. Key Performance Indicators (KPIs)

* Search time
* Overall search result relevance