Low Level Design (HLD)

Job Recommender System

Revision Number: 1.0

Last date of revision: 19/12/2023

# Document Version Control

**Change Record**

| Date | Version | Author | Comments |
| --- | --- | --- | --- |
| 19/12/2023 | v1.0 | Abraham Audu | Introduction, Architecture and Architecture Description |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Reviews**

| Date | Version | Reviewer | Comments |
| --- | --- | --- | --- |
|  | v1.0 |  |  |
|  |  |  |  |

**Approval Status**

| Review Date | Version | Reviewed By | Approved By | Comments |
| --- | --- | --- | --- | --- |
|  | v1.0 |  |  |  |
|  |  |  |  |  |

**Table of Contents**

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

[**Abstract 4**](#_heading=)

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

[1.1 Why this Low-Level Design Document? 5](#_heading=)

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

[**2. General Description 5**](#_heading=h.ggkmsajlo83h)

[2.1 Product Perspective 5](#_heading=h.47rs17phkyud)

[2.2 Problem Statement 5](#_heading=h.2m4jh7xphb00)

[2.3 Proposed Solution 5](#_heading=h.a24luzxqjj4y)

[2.4 Further Improvements 6](#_heading=h.9tipoxtfyl22)

[2.5 Technical Requirement 6](#_heading=h.6e6nbndd0z0)

[2.6 Data Requirements 6](#_heading=h.u8e5flsgmxuv)

[2.7 Tools Used 6](#_heading=h.eeubfnt9ay1j)

[2.8 Constraints 6](#_heading=h.uq8kg0k4hxoi)

[2.9 Assumptions 6](#_heading=h.jy1d8hqituqs)

[**3. Design Details 7**](#_heading=h.4o154eianxnu)

[3.1a Development Process Flow 7](#_heading=h.2izcvvqfv1k2)

[3.1b Deployment Process Flow 7](#_heading=h.gegq5e7jn42o)

[3.2 Event Log 7](#_heading=h.7vujpp5x3o0f)

[3.3 Error Handling 8](#_heading=h.754hode35qq)

[**4. Performance 8**](#_heading=h.83j129w5xneh)

[4.1 Reusability 8](#_heading=h.k9trcoggc87)

[4.2 Application Compatibility 8](#_heading=h.va0xe28dsx4t)

[4.3 Resource Utilisation 8](#_heading=h.yci7e8mbhjs5)

[4.4 Deployment 8](#_heading=h.rwu1iljkevt8)

[**5. Conclusion 8**](#_heading=h.j1a2hkradlcn)

# Introduction

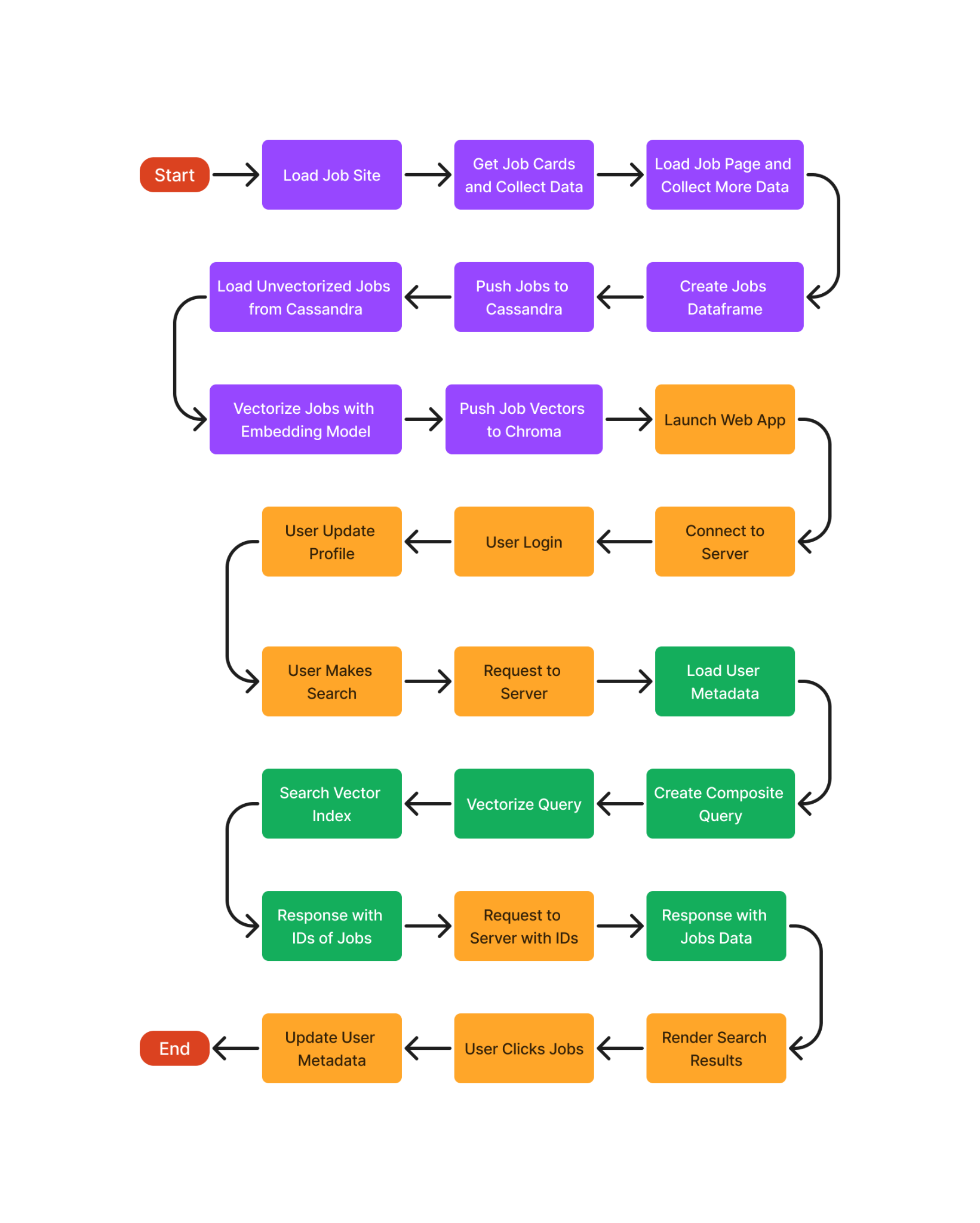
## Why this Low-Level Design Document?

The purpose of this Low-Level Design Document (LLD) is to give the internal logical design of the program code for the Job Recommender System Solution. This LLD describes the modules, functions and relationships between them. It describes the modules in a way that gives the programmer enough information to write said modules and functions only by reading this document.

## 1.2 Scope

This LLD is a component level design process that follows a step-by-step process to determine and refine what happens at each step of the entire process. This process factors the data structures used, software architecture, the flow of the source code and performance algorithms. Data organisation may be determined during requirement analysis,but there is still room for refinement during the data design work.

# 2. Architecture



# 3. Architecture Description

## 3.1 Data Description

All raw/human-readable data is stored on a Cassandra database. The jobs data is obtained from multiple websites, parsed and stored on the job listings table of the database.

User data is collected upon registration and updated as the user interacts with the app. User data is collected across three tables. The primary user data table consists of basic bio, work history and preferences as well as username and user ID. The other tables consist of search metadata and clicks metadata to track search history and click history based on the user’s unique ID.

Vectorised data is stored on the jobs collection of the Chroma database. Each job is represented by a sequence of floats generated by the embedding model.

## 3.2 Data Pre-Processing

HTML data fetched during scraping is parsed based on predefined functions designed for each site to collect specific data fields, and if the data is not present for the particular job, it is filled with an NA value to preserve the shape of the data collected .

## 3.3 Model Training and Evaluation

A pre-trained embedding model from sentence transformers was used for the creation of vector embeddings from text data across the project.

## 3.4 Data from User

The user is able to edit their data such as name, email, work experience and job preferences in the profile section of the website, which is updated on the database when the user saves said changes.

## 3.5 Data Validation

All data tables use data model classes which define the data types and content expected in each field, ensuring the integrity of the data across all tables

## 3.6 Inference Generation

When the user makes a search, the search query as well as the user’s data and metadata are combined into a composite query, vectorised and searched against the vectorised job index. Internally Chroma carries out an ANN search and returns the top ten job IDs based on similarity with the search vector. These IDs are then used to query the job listings table on the cassandra database.

## 3.7 Deployment

The Application will be deployed locally.

### 3.7.1 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.

### 3.7.2 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.

# 4. 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 |