**Low Level Design (HLD)**

**Job Recommendation System**

| Written By | Rakshit Khajuria, Prikshit Sharma |
| --- | --- |
| Document Version | v3 |
| Last Revised Date | 11-March-2023 |

**Document Control**

### **Change Record:**

| **Version** | **Date** | **Author** |
| --- | --- | --- |
| V1 | 19 - Feb - 2023 | Rakshit Khajuria,  Prikshit Sharma |
| V2 | 26 - Feb - 2023 | Rakshit Khajuria,  Prikshit Sharma |
| V3 | 11 - Mar - 2023 | Rakshit Khajuria,  Prikshit Sharma |

**Contents**

1. Introduction . 3

1.1 What is a Low-Level design document? 3

1.2 Scope 3

2. Architecture 4

3. Architecture Description 5

3.1 Data collection 5

3.2 Data Pre-processing 5

3.3 Data Insertion into Database 5

3.4 Export Data from Database 5

3.5 Model Building 6

3.6 Data from User 6

3.7 Data Validation 6

3.8 Text Preprocessing 6

3.9 User Data Inserting into Database 7

3.10 Model Call 7

3.11 Job Recommendation 7

3.12 Deployment 7

1. **Introduction**

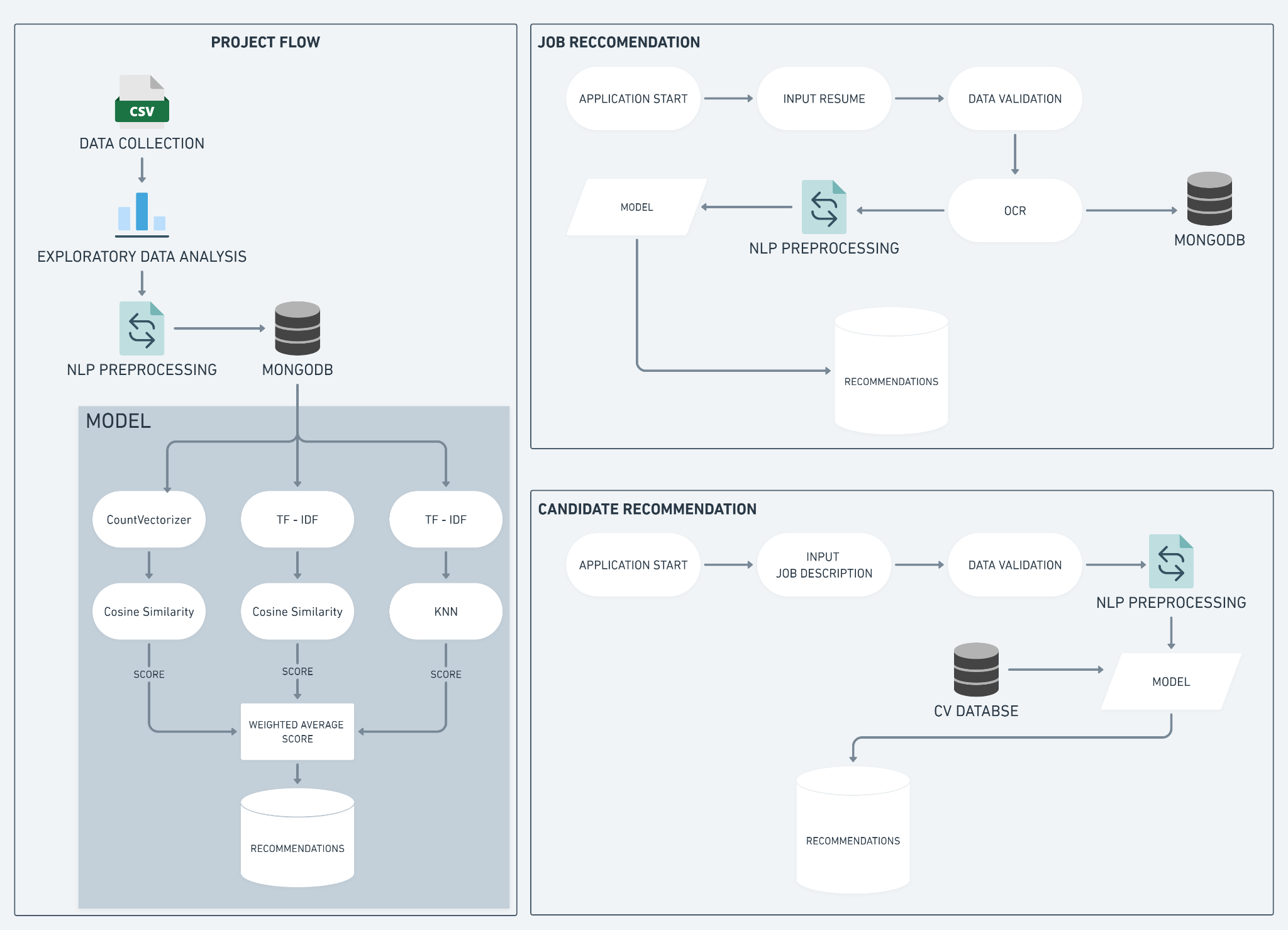
**1.1 What is a Low-Level design document?**

The goal of LLD or a low-level design document (LLDD) is to give the internal logical design of the actual program code for the Job Recommendation System. LLD describes the class diagrams with the methods and relations between classes and program specs. It describes the modules so that then programmer can directly code the program from the document.

**1.2 Scope**

The HLD documentation presents the structure of the system, such as the database architecture, application architecture (layers), application flow (Navigation), and technology architecture. The HLD uses non-technical to mildly-technical terms which should be understandable to the administrators of the system.

1. **Architecture**

****

**3 Architecture Description**

**3.1 Data collection**

With web scraping restricted on job search engines like Indeed, we turned to Apify to collect relevant job-related data. Utilizing Apify's advanced tools and automation capabilities, we were able to efficiently and accurately extract large volumes of data. This enabled us to provide up-to-date and precise job postings and candidate recommendations to job seekers and recruiters.

**3.2 Data Pre-Processing**

Data preprocessing is an essential step in building a Machine Learning model and depending on how well the data has been preprocessed; the results are seen.

In NLP, text preprocessing is the first step in the process of building a model.

* Word tokenization:
* Stop words removal
* Lemmatization
* Bigram Collection Finder

**3.3 Data Insertion into Database**

We are inserting the data in MongoDB database job recommendation, and in that Database we are storing the the data in three distinct collections: Resume\_Data, all\_locations\_Data, and preprocessed\_jobs\_Data.

Where as there are two more collections inside the same database which are Resume\_from\_CANDIDATE and Resume\_from\_RESUME\_ANALYZER in order to store resumes entered by users on the job recommendation and resume analyzer pages. These collections will serve as a repository for user-provided resumes, allowing for more personalized and accurate job recommendations and analysis.

**3.4 Export Data from Database**

Data Export from Database - The data in a stored database is exported as a CSV file to be used for Model Training.

**3.5 Model Building**

| **Model 1** | **Model 2** | **Model 3** |
| --- | --- | --- |
| count vectorizer | TF - IDF | TF - IDF |
| cosine similarity | cosine similarity | KNN |
| SCORE - 1 | SCORE - 2 | SCORE - 3 |

We will scale and add all the three scores to make recommendations

**3.6 Data from User**

Here we will collecting the data from user :-

* **JOB SEEKERS** - The user inputs their resume, selects desired location and number of recommendations.
* **RECRUITERS** - The user inputs the Job Description and the number of candidates he wants to shortlist.
* **RESUME ANALYZER** - The user inputs their resume

**3.7 Data Validation**

To ensure the accuracy and validity of user-entered data, we check whether the input data is valid or not.

**3.8 Text Pre-processing**

We used OCR to extract text from the PDF, and then apply NLP preprocessing techniques such as stop word removal, tokenization, lemmatization, and bigram collection. These techniques help to clean and prepare the text data for further analysis.

**3.9 User Data Inserting into Database**

We insert user data and resume to augment our dataset, improving our model's performance. This approach expands our data, allowing for more accurate predictions.

**3.10 Model Call**

Based on the user input, our model will be loaded and will be used to predict/recommend.

**3.11 Job Recommendation**

The model generates Jobs/Output recommendations based on input data. Recommendations are presented in a list or ranking and may be refined based on user preferences or constraints. The process facilitates efficient decision-making.

**3.12 Deployment**

We used Streamlit, an open-source framework for creating data apps, to deploy our system. Streamlit was chosen for its ease of use, flexibility with different programming languages, and ability to handle various data visualizations. We set up a Streamlit account, developed the app code, tested it locally, and then deployed it for public access..