Talent Acquisition Data Analytics

A Project Report
Presented to
The Faculty of the College of
Engineering

San Jose State University
In Partial Fulfillment
Of the Requirements for the Degree
Master of Science in Software Engineering

By
Kavitha Bangalore Nagaraju, Ashish Bende, Apurva Patel, Archana Taparia
Dec 2016

APPROVED

Prof. Ronald Mak, Project Advisor

Prof. Dan Harkey, Director, MS Software Engineering

Dr. Xiao Su, Department Chair

ABSTRACT

Talent Acquisition Data Analytics
By Kavitha bangalore Nagaraju, Ashish Bende, Apurva Patel, Archana Taparia

In today's fast-paced workplace, it is essential to have a good work culture that is not only conducive to getting things done but also a big motivating factor. Organizations invest a huge amount of time and money in hiring and retaining the right talent. However, a strong team impacts the business model more than any individual. Job boards like LinkedIn provide an individual candidate search, but there is no tool available to hire an entire effective team. Our aim is to address this problem with our Talent Acquisition System solution.

The built-in limitations of existing hiring tools can lead to biased decisions and can miss many underlying deciding factors like past team experiences, skill expertise, diversity and cultural suitability. With the proliferations of online user profile information, we can use it to provide more sustainable results. This report provides a complete documentation of the team recommendation system we have developed. We explain the current state of the project, and approaches used to recommend teams.

Our design is based on knowledge and collaboration competence, dynamic diversity factors and content-based recommendations. We generate a social network graph based on a comparative study that includes the candidate's expertise and connection strength with other candidates. We use a candidate rating to find a trust factor. Based on the social network graph and the trust factor, we recommend an optimal team for a given job.

Acknowledgments

The authors are deeply indebted to Professor Ronald Mak for his invaluable comments and assistance in the preparation of this study. The authors would also like to express gratitude to Mr. Chris Saccheri, one of the co-founders of LinkedIn, for his expert advice and encouragement during project development.

Table of Contents

Chapter 1. Project Overview 1
Introduction 1
Proposed Areas of Study and Academic Contribution 2
Current State of the Art 4

Chapter 2. Project Architecture 5
Introduction 5
Architecture Subsystems 7
1. Data aggregation (collection), transform and cleansing module 7
2. Recommendation Module 7
3. Web Application Module 16

Chapter 3. Technology Description 18
Client Technologies 18
Middle-Tier Technologies 18
Data-Tier Technologies 19
Additional Technologies 19
Development Tools 19
Deployment Tools 19

Chapter 4. Project Design 20
Client Design 20
1. Storyboard 20
2. Wireframes 23
Middle-Tier Design 28
1. Use case and Sequence Diagrams 28
Data-Tier Design 46
Data-Tier Design 49
1. ER Diagram 49

Chapter 5. Project Implementation 52
Client Implementation 52
1. Candidate Login Screen: 52
2. Candidate Profile Screen: 55
3. Posted Jobs Screen: 57
4. Job Profile Screen: 59
5. HR Profile Screen: 62
6. HR Post Job Screen: 63
7 View Candidate Profile Screen: 64
8 View Recommended Teams: 65
Middle-Tier Implementation 65
1. Backend API 65
2. Authentication API 66

3. Knowledge competence 67
4. Recommendation logic 68
5. Recommendation based on diversity in the team 70
Data-Tier Implementation 72
1. Database Collections 72

Chapter 6. Testing and Verification 77
1. Project Test Plan 77
1. Unit Testing 77
2. Integration Testing 80
3. System Testing 81
4. User Interface Testing 82

Chapter 7. Performance and Benchmarks 86
Performance 86
Benchmarks 86

Chapter 8. Deployment, Operations, Maintenance 88
Deployment 88
Operations and Maintenance 89

Chapter 9. Summary, Conclusions, and Recommendations 90
Summary 90
Conclusion 91
Interactive data-visualization 91
More data source input 91
Improving algorithms 91

Glossary 93

References 94

List of Figures

Figure 1 Architecture Diagram 5
Figure 2 Weighted graph architecture.....8
Figure 3 Weighted Social Network.....12
Figure 4 Recommendation architecture 14
Figure 5 Web Application Module 16
Figure 6 User Login page blueprint 24
Figure 7 Job Seeker Dashboard page blueprint 24
Figure 8 Job Seeker Job page blueprint-1 25
Figure 9 Log out warning blueprint 25
Figure 10 Job Seeker Job page blueprint-2 26
Figure 11 Employer Dashboard page blueprint 26
Figure 12 Employer Job page blueprint 27
Figure 13 Employer Candidate Profile page blueprint 27
Figure 14 Use case specification for User Authentication.....28
Figure 15 Sequence diagram for user authentication.....30
Figure 16 Use case diagram for Job Posting.....31
Figure 17 Sequence diagram for Posting a Job.....32
Figure 18 Use case diagram to create user profile 33
Figure 19 Sequence diagram for creating a profile.....34
Figure 20 :Use case diagram to update application status 35
Figure 21: Sequence diagram to update application status 36
Figure 22: Use case diagram to apply for a job 37
Figure 23:: Sequence diagram to apply for a Job 39
Figure 24: Use case diagram to generate recommendation based on GitHub and Location40
Figure 25: Sequence diagram to generate recommendation based on GitHub and Location41
Figure 26: Use case diagram to generate recommendation based on Diversity 42
Figure 27: Sequence diagram to generate recommendation based on cultural and gender diversity.....43
Figure 28: Use case diagram to visualize recommendation in graphical format.....44
Figure 29: Sequence diagram to visualize recommendation in graphical format.....45
Figure 30: Use case diagram for resume parsing.....46
Figure 31: Sequence diagram for resume parsing.....48
Figure 32: ER Diagram for Recruiter login 49
Figure 33:ER Diagram for candidate login.....50
Figure 34: Candidate Login Screen 52
Figure 35: Candidate Profile Screen 55
Figure 36:Posted Jobs Screen 57
Figure 37: Job Profile Screen.....59
Figure 38: HR Profile Screen.....62
Figure 39: HR Post Job Screen 63

Figure 40: View Candidate Profile Screen 64
Figure 41:Recommended Teams Screen 65

List of Tables

Table 1 Interaction log 9
Table 2 Skill Mapping 10
Table 3 Expert Score 11
Table 4:Example team weight calculation 14
Table 5 Client Technologies 18
Table 6 Middle-Tier Technologies 18
Table 7 Data-Tier Technologies 19
Table 8 Additional Technologies (Development Tools) 19
Table 9 Additional Technologies (Deployment Tools) 19

Chapter 1. Project Overview

Introduction

Subject: We are implementing Talent Acquisition System, and this report describes the application and its architecture in detail. Talent Acquisition System provides the facility to build a team of suitable candidates for the hiring company or recruiters. Current job recruiting portals do not have this feature, and this is driving factor for our newly developed system. In this report, we justify the requirements of our project in its predefined scope. We also emphasize our project schedule and progress as per the project plan. Following the conclusion, this report provides current state, literature research, and detailed explanations about project architecture, subcomponents, and other technical details.

Purpose: Team collaboration is equally important as an individual candidate's skill sets. A key asset for a successful business is a skilled team for any project implementation. Current job boards and social media do not provide such recommendations since they allow searches only of an individual candidate's background information. The lack of team recommendation system in such job portals do not allow them to predict team behaviors of candidates. [3] Companies are spending lots of resources and expenditures on team building activities for these hired candidates. The purpose of our Talent Acquisition System is to quantify different filters such as candidates' mutual collaboration and technical skills relevant for job and generate team recommendation for matching requirements. This recommendation can save organization's time and cost for team building activities.

Project Plan: We are following the agile process to build our system. We plan to build every component in parallel, and we have scheduled a project meeting every two weeks to check the status of our work. Once we finish our development, we plan to start testing and perform any required retrofitting. Our project plan and implementation design is ready and we are working on individual components now. We are following our project schedule to implement the Talent Acquisition System.

Scope: Talent Acquisition System is an application that is divided into two business processes: 1) Recruiters create a job pool and specify the requirements to build a team, and 2) Candidates create a profile and submit the information to the system. Our recommendation engine acts as an interface for both processes. It fetches a candidate's data from the dashboard (which is a web application interface for candidate interaction) and processes it to provide suggested teams to the recruiters based on their queries. This application provides team recommendations for recruiters or hiring companies on the basis of candidates' profiles and social networks.

Proposed Areas of Study and Academic Contribution

The goal of our application is to recommend teams as per the recruiters' requirements and to minimize organizations' costs to hire multiple candidates for teams. To implement this application, we have studied recommendation algorithms and graph techniques thoroughly. Therefore, we propose the following areas of study for our project:

- **Recommendation Algorithms** – A recommendation engine, also known as recommender system, uses recommendation algorithms based on the properties of related entities. [1] There are various approaches for these algorithms, out of which we have chosen a content-based filtering recommendation approach for our application [5]. The content-based filtering approach uses information retrieval and information filtering as integral parts of the algorithm; this is a perfect fit for our requirements. [2] We plan to study this approach more as we implement our solution.

- **User group** – Before starting application development, it is important to identify the target user group for the system. Since a recommendation engine works on the user's preference, getting to know the users of the system is a prime focus of our system. In our application, we have two kind of target audiences: 1) recruiters and hiring companies and 2) job seekers. Data collected from these two user groups is the primary data source for our recommender system.

- **Data Aggregation**: Data aggregation refers to strategies used to combine recommendation algorithms [5]. Aggregation can be performed for preference-based recommendation using arithmetic calculations such as the statistical mean [6]. Our application provides team recommendations on the basis of preferences; hence data aggregation methods play a key role in our system.

- **Collective Intelligence:** Collective intelligence is shared intelligence among group of individuals, or mutual collaboration. The basic objective of collective intelligence is to find out "how people are connected to each other through computer" [4]. We plan to study this field as part of the team recommendation system.

- **Social Network Analysis:** We fetch network connections of individuals from the social network. This requires us to research social network analysis. We are studying in detail how to utilize social network analysis to visualize and understand collaboration, communication, and cooperation among individuals [7].

- **Data Transformation and Cleaning:** Data cleaning or scrubbing is a mandatory process before analyzing data. We use the extract, transform and load (ETL) [9] approach for data cleansing.

- **Dynamic Diversity factor:** Final step in generating teams is dynamic diversity adjustments. Gender, ethnicity and locations are some of the criteria we have considered for diversity calculation.

Our application development process is completely research-based as no existing system provides team recommendations. Limited study material for this topic leads us to write a technical paper on the implementation process. We plan to provide full details in the paper such as technical architecture, algorithms, and the reason to choose those solutions. This paper will contain all the information of the development process that we will follow. Written documentation like this would be helpful in future to build this system. Along with

documentation, we plan to provide our code and APIs as open source for contribution to the community.

Current State of the Art

Current job portals, recruiting agencies, and social network media are helping organizations to get suitable candidates. However, this process does not guarantee team performance for the organization. Companies acknowledge the significance of team performance and some organizations are working towards this field. The work done related to Talent Acquisition System can be categorized in following fields:

- **Collaborative platform**: Some companies provide a collaborative platform where individual candidates can perform as a team for recruiting process. For example, SmartRecruiters provides such platform.
- **Automation of hiring process**: The hiring process can be automated through data analytics, candidate management, and follow-up emails. For example, companies like Glid use this approach.
- **Data Analytics**: Greenhouse is a platform that uses data analytics for effective recruiting.

At present there are some companies like Go Elevator that strongly believe in team building to ensure project success. According to the CEO of Go Elevator, employees are more productive if they are part of a previous team." [2] This statement is supported by various tech giants and the fact that a powerful team can deliver unbelievable results.

Although organizations recognize the importance of good teams, at present there is no platform available in the market that uses data analytics and prediction-based solutions to provide a dream team for the company.

Chapter 2. Project Architecture

Introduction

The architecture comprises four core components:

- User interface or web application module
- Data aggregator, transform and cleansing module
- Recommendation system module
- Data visualization module

The following diagram provides a high level system architecture.

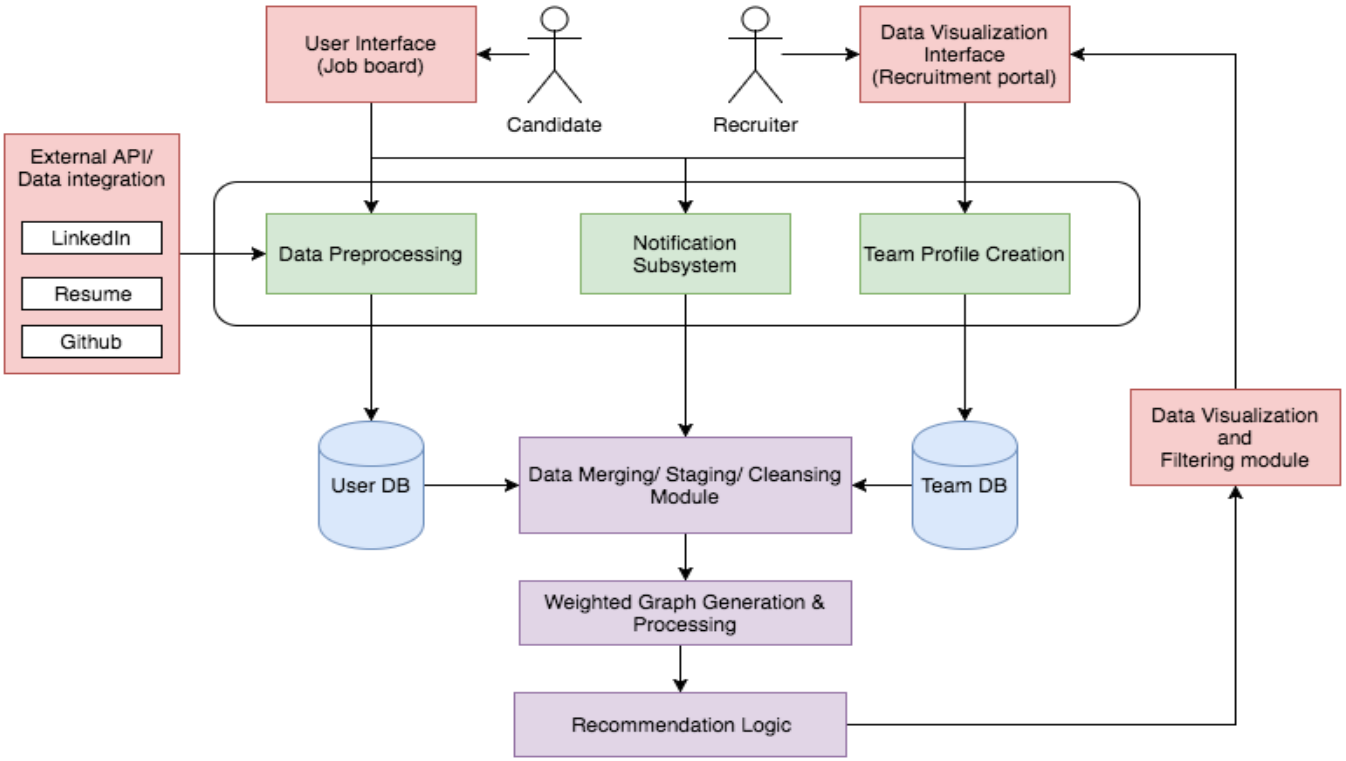


Figure 1 Architecture Diagram

User Interface

- The candidate and the recruiter will have separate user interface functionality to interact with the system.
- A candidate can create his or her profile and apply for any listed jobs.
- Some additional functionality for the candidate profile:
    - Upload resume
    - Upload cover letter or any other relevant documents.
    - Attach a social profile from LinkedIn or GitHub
- A recruiter creates a team profile based on various criteria:
    - Team size and roles
    - Skills required
    - Diversity
    - Geographic location

Business Logic

- Data preprocessing
    - Extract skills from the resume and the cover letter
    - Collect information from external sources
    - Prepare a combined profile for each candidate
- Team profile creation
    - Post a job based on team profile requirements
    - Attach role info, skill, designation and location information.

Data Storage

- Two types of data stores:
    - User profile database: Relational data store to persist all user information.
    - Team profile database: Relational data store to save team requirements and any updates.

Recommendation Engine

- Input will be fetched using the processing logic after applying various filters.
- Candidates will be ranked according to their skills and roles they are fit for. A multiple weighted team prediction graph will be calculated based on the collaboration weight generated by different parameters such as GitHub connections, location similarity, and diversity within the team

Data Visualization

- This module interfaces with the recruiter's dashboard. A recruiter may have further capability to apply additional filters, select teams, and notify candidates

Architecture Subsystems

1. Data aggregation (collection), transform and cleansing module

- The system will consume data from different database sources (NoSQL and Hybrid/Graph database)
- A NoSQL database will be used to extract non-structured data from different external APIs and the candidate's resume. We will use GraphDB to store team results, which the data visualization module will use.
- Databases of different types will output a common data format such as JSON in the transformation phase to build a common ground.
- Data will be combined and cleaned of redundancies or invalid entries.

2. Recommendation Module

The recommendation module comprises data analysis and team recommendation logic. The following diagram provides an overview of candidate interactions. The end of this process generates an expertise score and a social network graph.

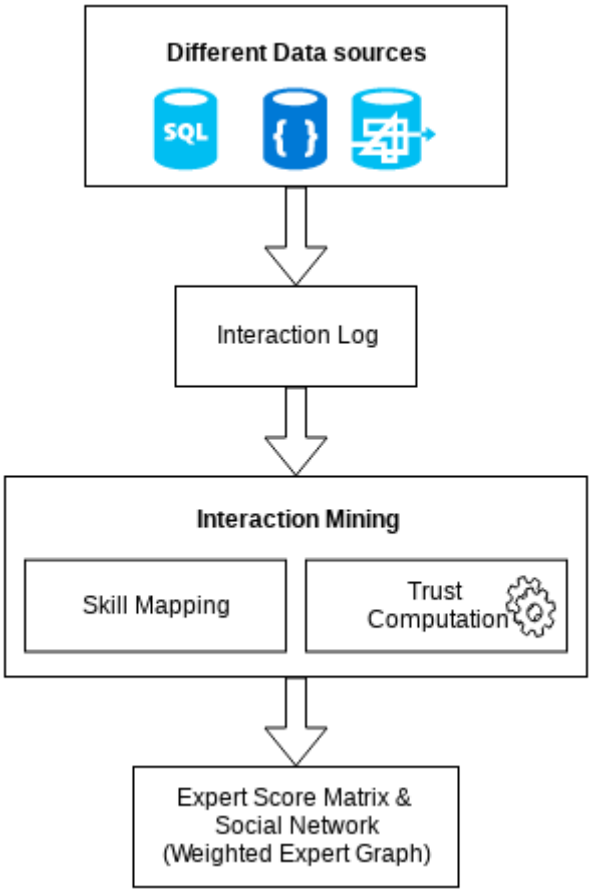


Figure 2 Weighted graph architecture

2.1 Data Analysis & Mining

Data analysis and mining is performed by the following two methods [4].

Knowledge competence

Knowledge competence is a process of finding the expertise scores of candidates for a given skill and the relationship strength between candidates.

- **Interaction log**

  An interaction log is a set of data used for knowledge competence.

  Table 1 is an example:

I_k	C_i	SC_i	C_j	SC_k	$R_{i->j}$	$R_{j->i}$	T_k	θ
1	Jack	Java	Paul	C++	0.62	0.43	1	20
2	Dan	Python	Peter	C	0.80	0.81	2	25
3	Ron	DBMS	Jack	Java	0.72	0.60	5	35
4	Paul	C++	Ron	DBMS	0.65	0.75	8	30
5	Ron	DBMS	Peter	C	0.66	0.78	10	40
6	Ron	DBMS	Paul	C++	0.67	0.79	11	30

Table 1 Interaction log

I_k − Unique ID
C_i − Candidate i
C_j − Candidate j
SC_i − Candidate i skill
SC_j − Candidate j skill
$R_{i->j}$ − Rating from candidate i to candidate j [0,1]
$R_{j->i}$ − Rating from candidate j to candidate i [0,1]
T_k − Task k
θ − Time since last known interaction between candidates (Days)

- **Interaction mining**

  Interaction mining is the process of mapping candidates to their interaction counts. An interaction count (IC) is number of interactions made by a candidate for a particular task. For example, in Table 2, Jack interacted on a Java-based project 8 times. The total interaction count for Java is 20 (total 20 interactions made using Java by all candidates).

The second table indicates the total number of interactions made using Java in that project.

Skill Mapping:

Candidate	Skill	Interaction Count
Jack	Java	8
Dan	Python	10
Ron	DBMS	7
Paul	C++	9
Peter	C	5

Skill	Interaction
Java	20
Python	15
DBMS	13
C++	15
C	8

Table 2 Skill Mapping

- **Expertise Score computation**

  In this process we calculate the expertise score of an individual for a given skill. For simplicity, we assume that a person is expert at only one skill. The expertise score X is computed from the interaction log. The expertise score for a candidate Jack is X_{jack}. The expertise score computation function is adopted from [4].

$$X_{jack} = IC_{jack} / \text{ Total IC}$$
$$= 8 / 20 = 0.4$$

Candidate	Skill	Expertise Score
Jack	Java	0.40
Dan	Python	0.66
Ron	DBMS	0.53
Paul	C++	0.61

Peter	C	0.62

Table 3 Expert Score

- **Finding connection strength**

The strength of a relation can be identified by linking two factors, frequency and recency [8]. Frequency is the number of interactions between two candidates and recency is the time since the last interaction between candidates. Interaction intensity [10] is calculated from the frequency and recency of the interactions between two candidates. Let sc'_{ij} be the interaction intensity between candidate i and j. The strength is computed by following formula [4]

$$sc'_{ij} = |I_{T(C_i \cap C_j)}| \,/\, \min\,(I_{T(C_i)}, I_{T(C_j)})$$

Here, sc'_{ij} represents the interaction intensity between candidate i and j.
$I_{T(C_i \cap C_j)}$ is number of tasks candidate i and j performed together

$I_{T(C_i)}$ & $I_{T(C_j)}$ is the total number of tasks performed by candidate i and j respectively.

This will calculate the overall collaboration strength between candidates, based on the maximum number of tasks they have performed together in the past.

In Table 1, Jack and Paul interacted for 1 task and total tasks they are involved individually is 2:

$$sc'_{ij} = 1 \,/\, \min\,(2,2) = 0.5$$

Recency according to [4]:

$$\phi_l = \begin{cases} 1; \theta_l - \theta \leq 180 \text{ days} \\ 1 - \left(\frac{\theta_l - \theta}{\lambda}\right) & \lambda > \theta_l - \theta > 180 \text{ days and } \lambda = 500 \text{ days} \\ 0; \theta_l - \theta \geq 500 \text{ days} \end{cases}$$

ϕ_l is a weight factor for sc'_{ij}which represents l^{th} interaction between candidate $C_i$and C_j and λ is a constant value. Consider Jack and Paul's example to find the final strength

factor based on frequency and recency. Referring to Table 1, ϕ_l for Jack and Paul is 1 as they interacted 20 days ago which is less than 180.

$$sc_{ij} = sc'_{ij} * \phi_l$$

$$= 0.5 * 1 = 0.5$$

The strength factor for Jack and Paul is 0.5. By following the process for every candidate, we can find the weighted social graph:

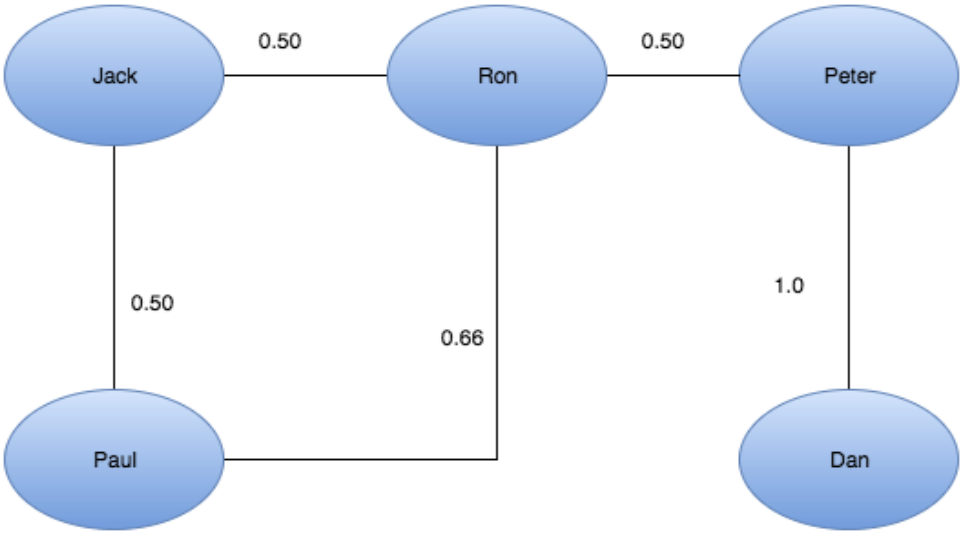


Figure 3 Weighted Social Network

- **Collaboration competence**

  Collaboration competence can be achieved by finding similarity between candidates. This solution is inspired from the FIRE (Fides and Reputation) model [4] for calculating interaction, trust and reputation. We can consider certain parameters such as candidate's location, past experiences, GitHub contribution, gender diversity, cultural diversity and build connection weights between the candidates.

  - We select candidates with top skill score and generate combinations of those candidates.

- For such combinations, total costs are calculated by adding parameter weights associated with the candidates involved in the combination.

  Total weight of the combination= $W_l + W_g + W_s + W_c$

  W_l = Weight based on location of the candidates

  W_g = Weight based on GitHub contribution made by candidates

  W_s = Weight based on gender of the candidate

  W_c = Weight based on race of the candidate

- Finally, a set of such combinations with higher weights is recommended.
- Weight factor can be dynamically changed using the scale on the user interface.

Example:

Team	**Combinations**	**Positions**	W_l	W_g	W_s	W_c	**Total Weight**
Team 1	Combination 1	Candidate A (Software Engineer 1)	2	2	4	4	12
		Candidate B (Software Engineer 2)					
		Candidate C (Software Engineer in Test)					
	Combination 2	Candidate D (Software Engineer 1)	1	2	2	4	9
		Candidate E (Software Engineer 2)					
		Candidate F (Software Engineer in Test)					
	Combination 3	Candidate G	2	2	2	1	7

		(Software Engineer 1)					
		Candidate H (Software Engineer 2)					
		Candidate I (Software Engineer in Test)					

Table 4:Example team weight calculation

Combinations with higher weights are recommended.

2.2 Recommendation Logic

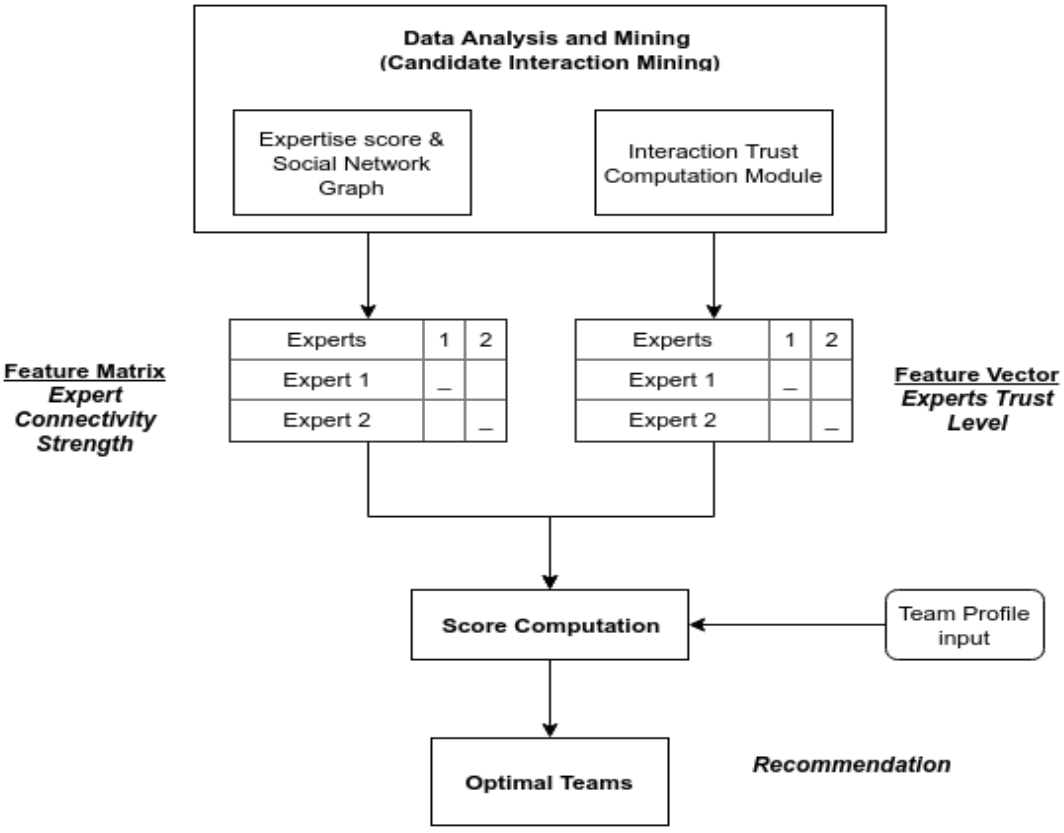


Figure 4 Recommendation architecture

As shown in Figure 4, team formation and recommendation are performed in the following steps:

- Obtain the weighted social network graph from the job seeker's profile.
- Create feature a matrix of each model (knowledge and collaboration competency).
- Calculate the expert enhanced connectivity score and the final trust value between a pair of experts.
- The output of the score computation will be mapped to the graph database, and it will be used by the data visualization module.

3. Web Application Module

This module includes the database and client interaction required for our application. The web application will be built on the MVC (model-view-controller) architectural design pattern. The software application is divided into three interconnected parts to present information accepted from a user.

Model: Responsible for managing the rules, data and logic.
View: Output information viewed in different formats such as tables and graphs.
Controller: Accepts user input and modifies it to command the view and the model.

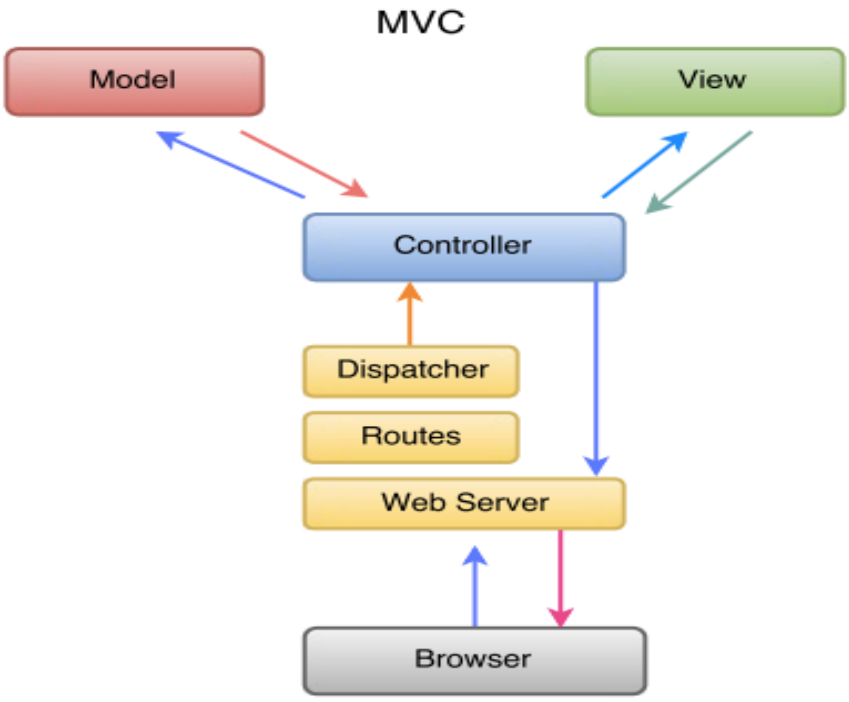


Figure 5 Web Application Module

Description:

- The user interacts [13] with the user interface using the web browser and web services. Process the client request and get data from the database.
- The web server routes the incoming request to the dispatcher to find the correct controller class handler.
- The controller handles the user input by using a callback or an event handler.
- The controller notifies the model about the user action, which results in changes to the model state.
- The view uses the model to display the result in the form of graphs. The model will not have any direct knowledge of the view.
- The user interface waits for the next user interaction to begin the next cycle.

Using this model reduces the complexity in design and increases the use and flexibility of the application. There are many advantages of the MVC architectural design pattern [12]:

Clarity of design: The MVC design makes the application easy to implement and maintain.
Efficient modularity: Components can be swapped in and out based on the programmer's desire. Changes in one part of the program do not affect other parts. Work can be done in parallel across all the components.
Multiple views: The model can be displayed in many ways. Views are designed in a scalable way.
Ease of growth: We can add new views as model grows.
Distributable: This MVC application can be easily distributed by only altering its startup method.
Powerful user interfaces: Provides a cleaner and friendlier user interface.

Chapter 3. Technology Description

This section, we describe tools and technologies which we are using to develop the project. This section is divided into three primary tiers and a generic section. Each section includes a tabular description of the tools and technologies, and details their intended usage and purpose.

Client Technologies

Name	Description	Reason to choose
AngularJS 2	A JavaScript framework for dynamic web application development	Familiarity and strongly typed language (Typescript)
D3/Google Charts/HighCharts	Display team and member interactions	One of the best open source and flexible data visualization tool
Html5/CSS/Bootstrap	Barebones structure to develop static website content	To transform mockup designs to actual web pages, and templates for the AngularJS framework.

Table 5 Client Technologies

Middle-Tier Technologies

Name	Description	Reason to choose
Node JS	Backend service for GUI service	Familiarity of this cross-platform JavaScript development environment
Python Flask	Backend service for databases and recommendation API	Lightweight framework with strong coupling with database tools and machine learning libraries
Apache Spark	Fast real-time recommendation processing to rank candidates	Distributed computing with support for machine learning libraries.
LinkedIn integration	Collect candidate profile information	Prominent job seeker's network with a wide API
GitHub integration	Collect skill and collaboration information	Widely used platform for code management

Table 6 Middle-Tier Technologies

Data-Tier Technologies

Name	Description	Reason to choose
MongoDB	Store data collected from various API integrations.	Document oriented datastore with many integration libraries
Redis	Cache responses for faster request processing	Lightweight and stable key-value store

Table 7 Data-Tier Technologies

Additional Technologies

Development Tools

Name	Description	Reason to choose
Atom	JavaScript IDE with Typescript support	Open source and strong integration with GitHub deployment
Network tools (SSH/Git)	Version tracking and remote machine interaction	Secure and required tools for the task.
Microsoft Office	Documentation and presentation	Best enterprise tool and familiarity with use.

Table 8 Additional Technologies (Development Tools)

Deployment Tools

Name	Description	Reason to choose
MongoLab	Cloud hosted MongoDB service	Familiarity and free instance availability
AWS EC2	Host all micro services in the cloud	Familiar web service for scalable cloud computing
Bitbucket	Project code deployment	Git revision control system with private repositories.

Table 9 Additional Technologies (Deployment Tools)

Chapter 4. Project Design

Client Design

1. Storyboard

A graphical representation or blueprint of this project includes a problem statement, a recommendation of the solution, and the implementation of Talent Acquisition System to provide a solution to the problem.

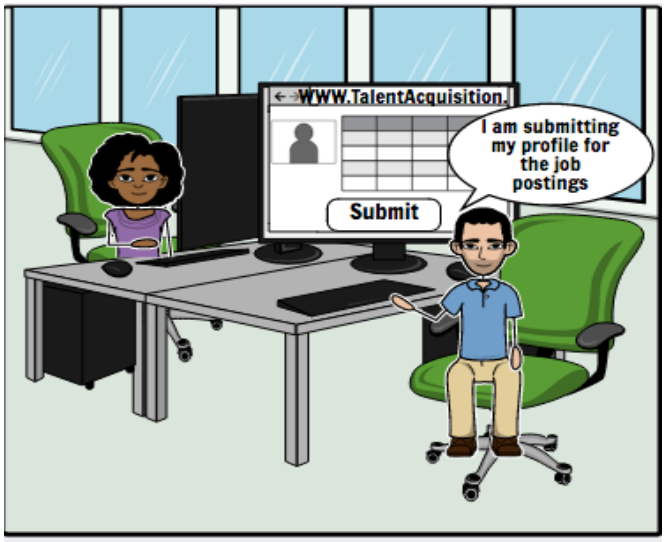
The purpose of the storyboard is to presents an outline of the design approach for the portal.

Here is a scenario-based design process of the web application portal:

- Scenario [1] Problem statement

- Scenario [2] Solution Recommendation

- Scenario [3-5] Solution Implementation



The Job seeker submits their profile to the Talent Acquisition portal for the listed job openings.

The Talent Acquisition portal will analyze the submitted applications and will provide team recommendation.

- Scenario [6] Solution Success

2. Wireframes

A screen blueprint of this project provides a visual guide to the basic design template of the web application without adding CSS styles. Wireframes are a skeleton of the website that includes all the functionalities and models of the web project. Using the wireframes, we can connect information through visual design.

Wireframes for Talent Acquisition System are as displayed:

Figure 6 User Login page blueprint

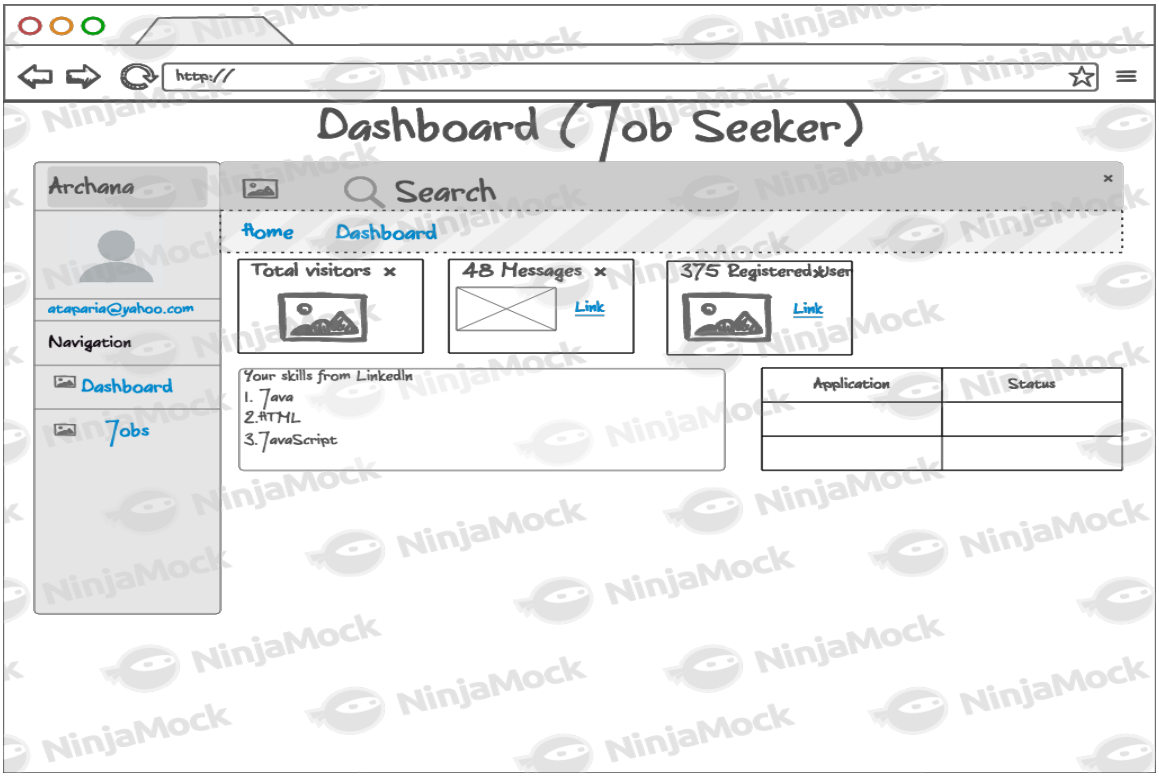


Figure 7 Job Seeker Dashboard page blueprint

Figure 8 Job Seeker Job page blueprint-1

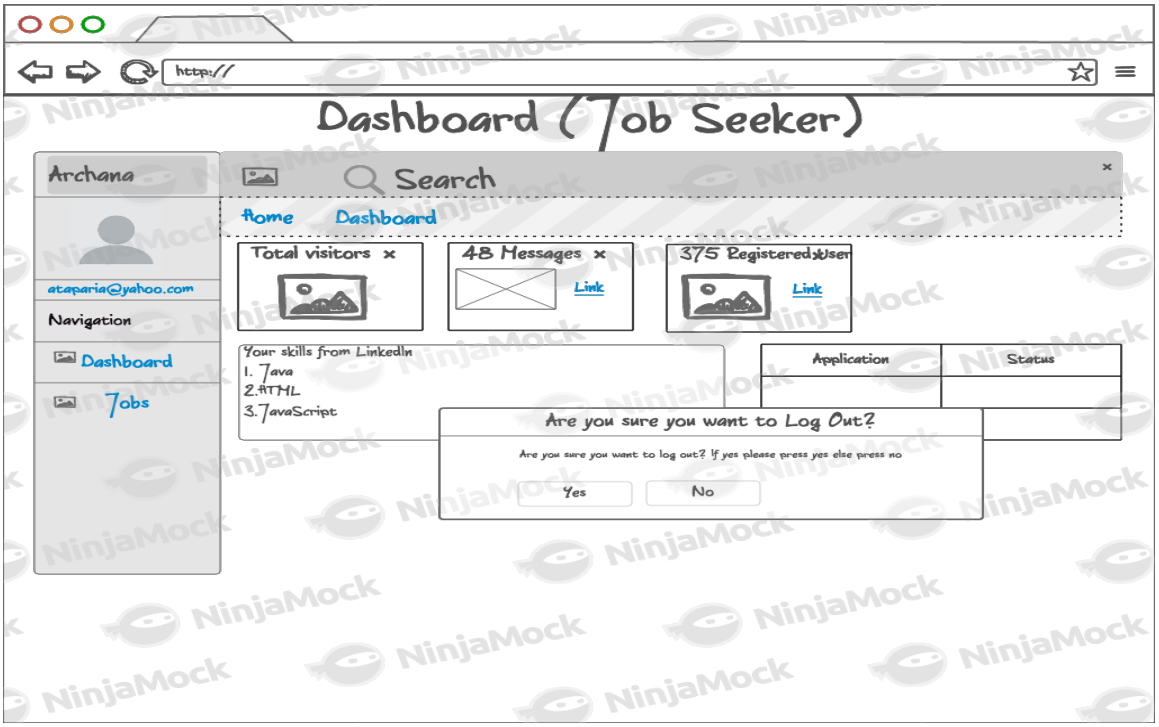


Figure 9 Log out warning blueprint

Figure 10 Job Seeker Job page blueprint-2

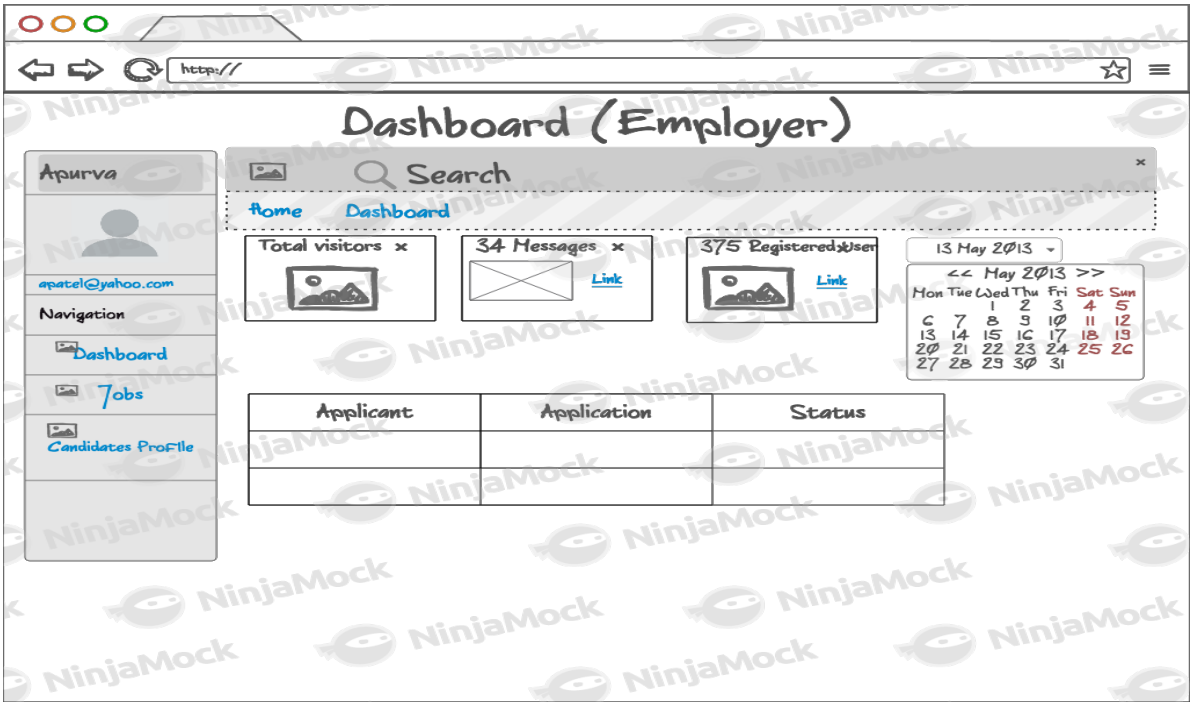


Figure 11 Employer Dashboard page blueprint

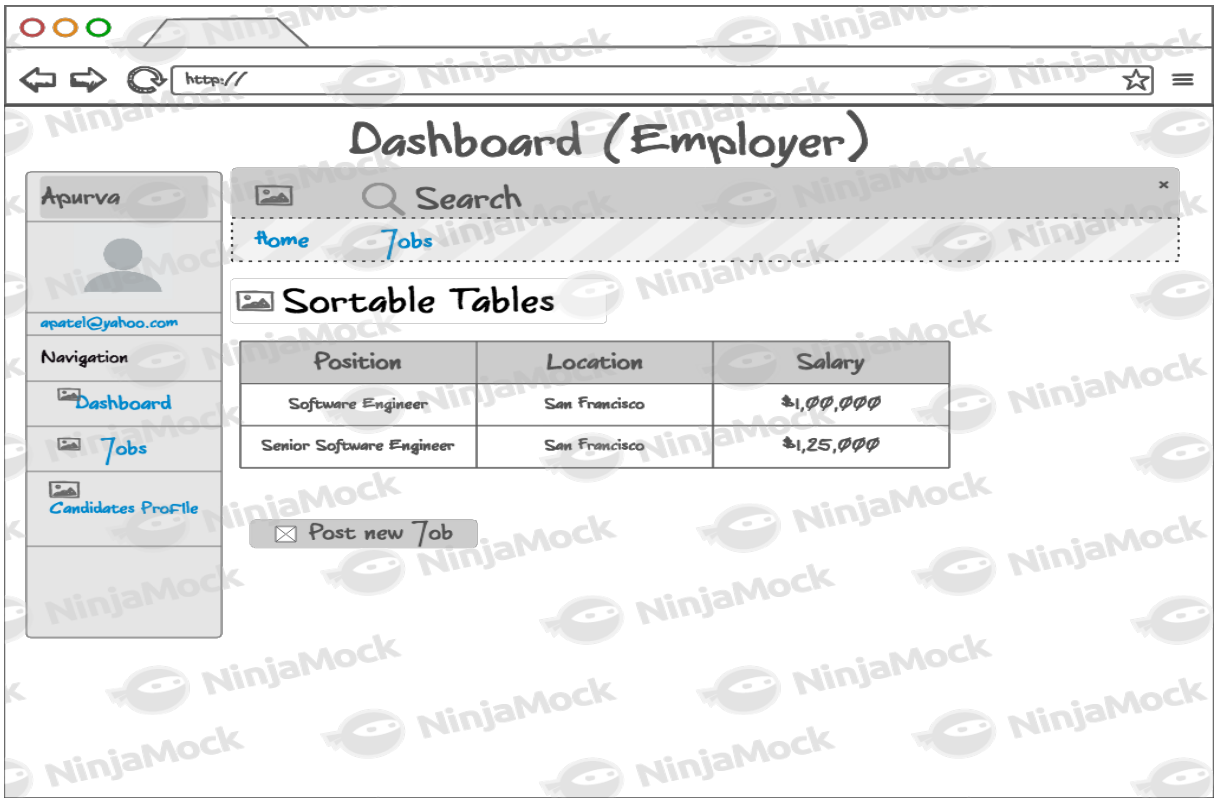


Figure 12 Employer Job page blueprint

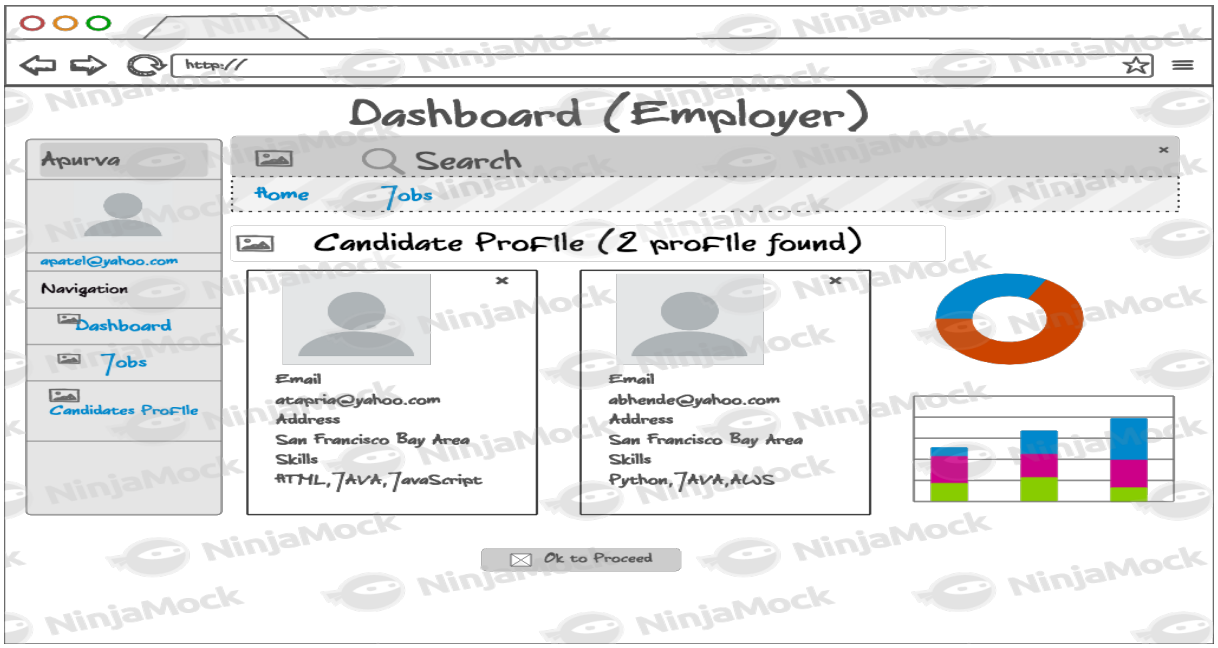


Figure 13 Employer Candidate Profile page blueprint

Middle-Tier Design

1. Use case and Sequence Diagrams

A use case refers to actions or events that define the interaction between a role (actor) and a system. A use case diagram is a graphical representation of the interactions between a user and the system. Sequence diagram is an ordered graphical representation of object interaction in a system.

Use case diagram for user authentication

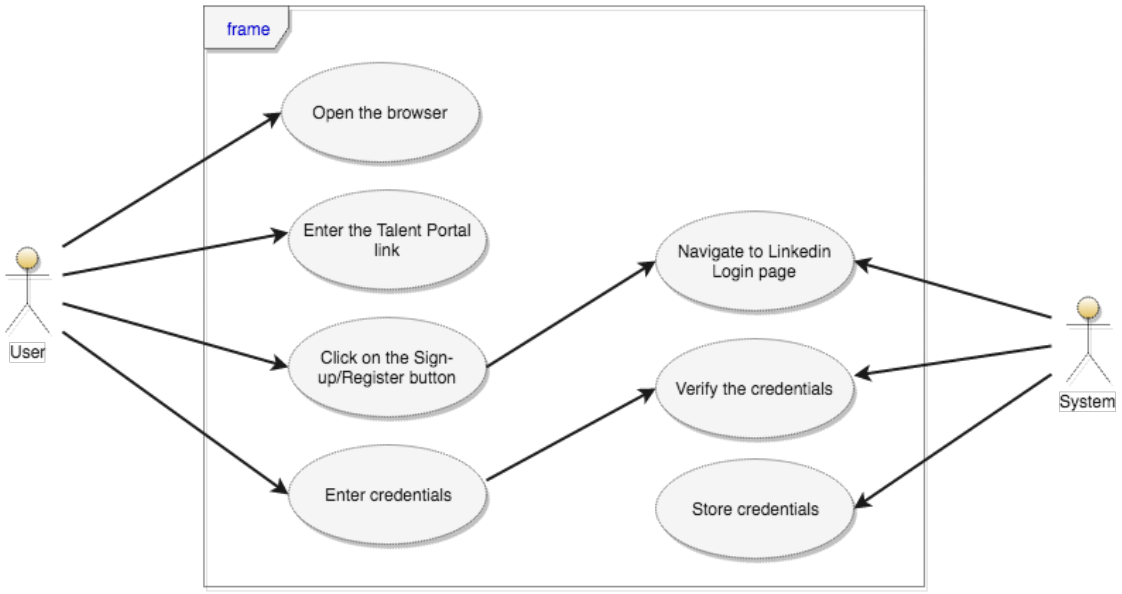


Figure 14 Use case specification for User Authentication

Use case specification for User Authentication

Use case Name	Login to the Talent Acquisition Portal through LinkedIn
Actors	User (Employer or Job Seeker), Talent Acquisition web server
Description	To access the portal user has to maintain a LinkedIn account. User log on to the existing LinkedIn account, if user is not having then LinkedIn account then he has to create a LinkedIn profile first before using Talent Acquisition Portal.

Basic Flow	1. The user clicks on the login button shown on the authentication window. 2. The Talent Acquisition web server will navigate the user to the LinkedIn login page. 3. The user enters LinkedIn credentials. 4. The User clicks on Login button.

Use case Name	Login verification
Actors	Talent Acquisition web server
Description	Server has to verify user's credential before providing access to the portal.
Basic Flow	1. The user logs into his or her LinkedIn account by entering a user-id and password. 2. The LinkedIn authentication system verifies user's credential. If something is wrong, the system goes back to the login window with error message. 3. The server uses LinkedIn user-id for unique identification of the user.

Use case Name	Store Login details
Actors	Talent Acquisition web server
Description	Talent Acquisition web server stores the user's credential to identify the user's profile after a successful verification.
Basic Flow	1. The Talent Acquisition web server verifies the user's credential. 2. Talent Acquisition web server stores the user's detail in the database if verification is successful.

Sequence diagram for User Authentication

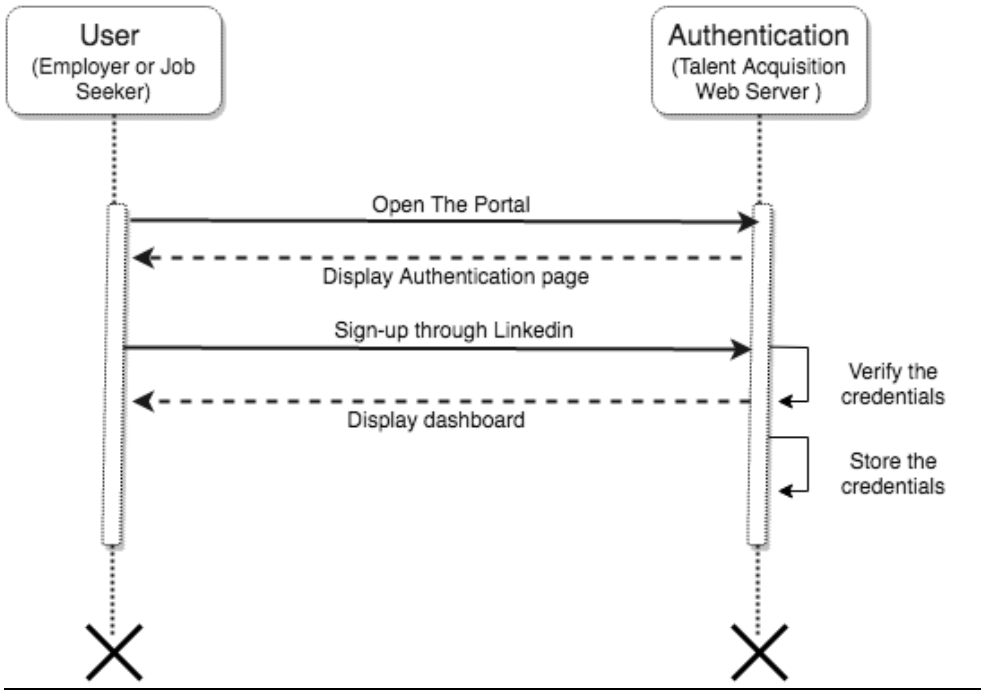


Figure 15 Sequence diagram for user authentication

Sequence diagram description:

- The user enters the link for web application and open the portal.
- Authentication component object of the system response by portal authentication page display.
- The user clicks on the LinkedIn button and sign-up through the LinkedIn authentication.
- Once the user's credentials are verified successfully then the server loads the dashboard.
- Upon successful login, system stores the user's data into the database for future use.

Use case diagram for Job Posting by Employer (User)

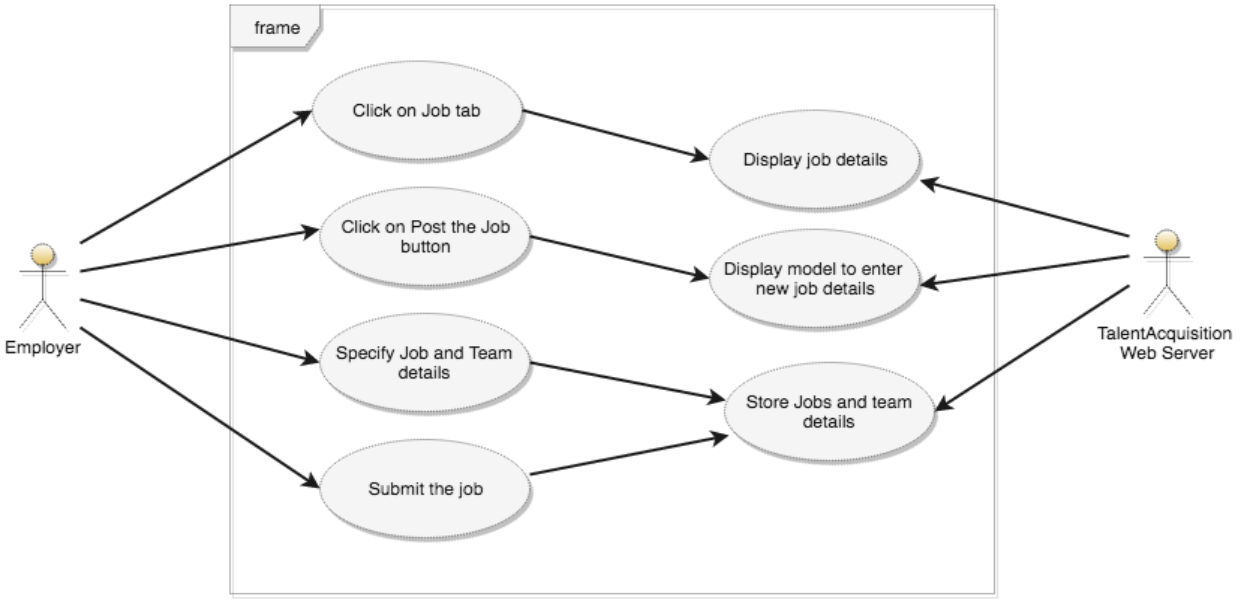


Figure 16 Use case diagram for Job Posting

Use case specification for Job Posting

Use case Name	Access the Job page
Actors	User (Employer), Talent Acquisition web server
Description	The employer posts the job on the job page of Talent Acquisition Portal
Basic Flow	1. After the user's verification, Talent Acquisition web server displays home page of the portal. 2. The employer clicks on the Job tab. 3. Talent Acquisition web server fetches the existing job details and display in a tabular format.

Use case Name	Specify Job and Team Details
Actors	User (Employer)
Description	The employer specifies job details, requirement, team name and location.
Basic Flow	1. The user clicks "Post the Job" button on the Job page. 2. Talent Acquisition web server opens a model to enter job details. 3. The user provides job details such as location, salary, required skills etc. 4. The user specifies the team and location for the opening. 5. The user press Ok button and submit the Job.

Use case Name	Store Job and Team details
Actors	Talent Acquisition web server
Description	Talent Acquisition web server stores jobs and team details in the system database.
Basic Flow	1. After job submission by the user, Talent Acquisition web server stores the details in the database. 2. Talent Acquisition web server repeats the process while the employer keeps posting jobs.

Sequence diagram for Job Posting

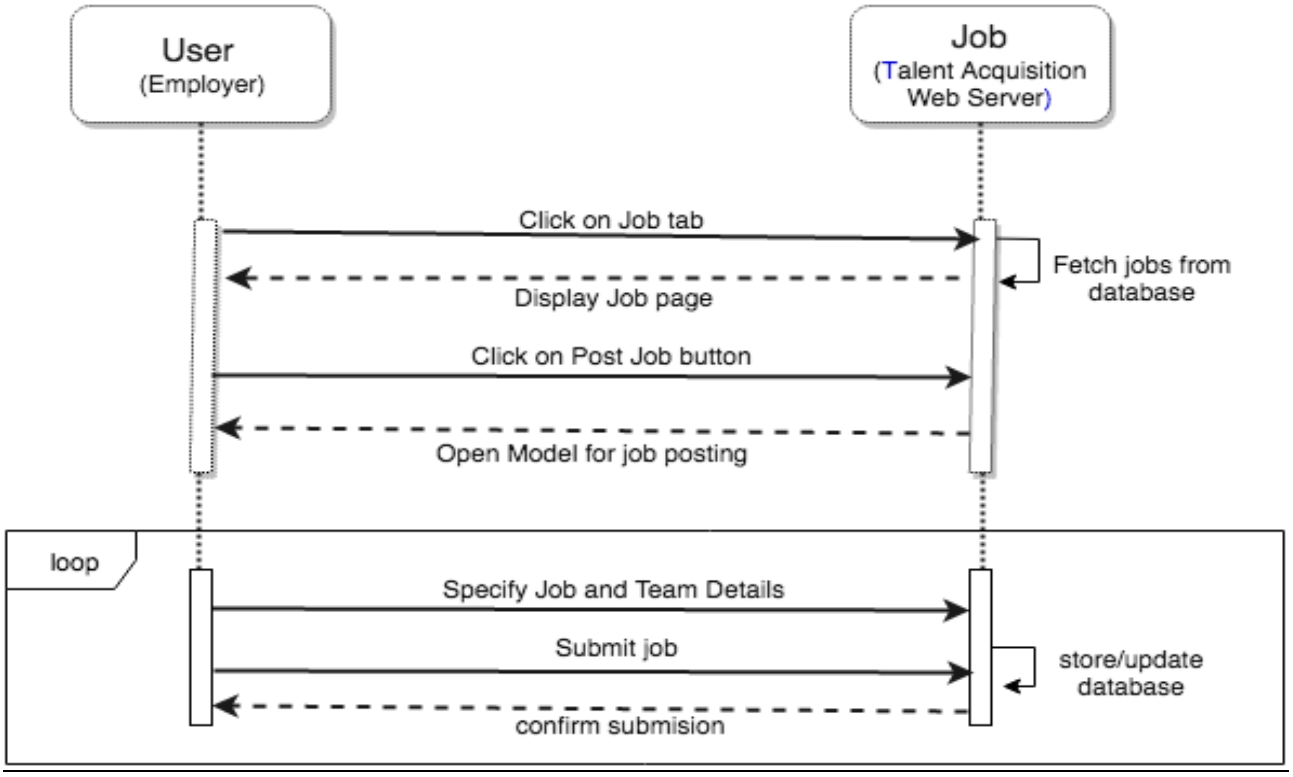


Figure 17 Sequence diagram for Posting a Job

Sequence diagram description:

- The user clicks on the Job tab and post the job.
- Job object of the system fetches details from database and display job page.

- The user clicks on the “Post the Job” button and Talent Acquisition web server displays a window to post the job.
- The employer specifies job specifications, team details and submit the job.
- Job object stores the information in the database and repeat the process until the employer keep on posting jobs.

Use case diagram for Profile creation by Job Seeker (User)

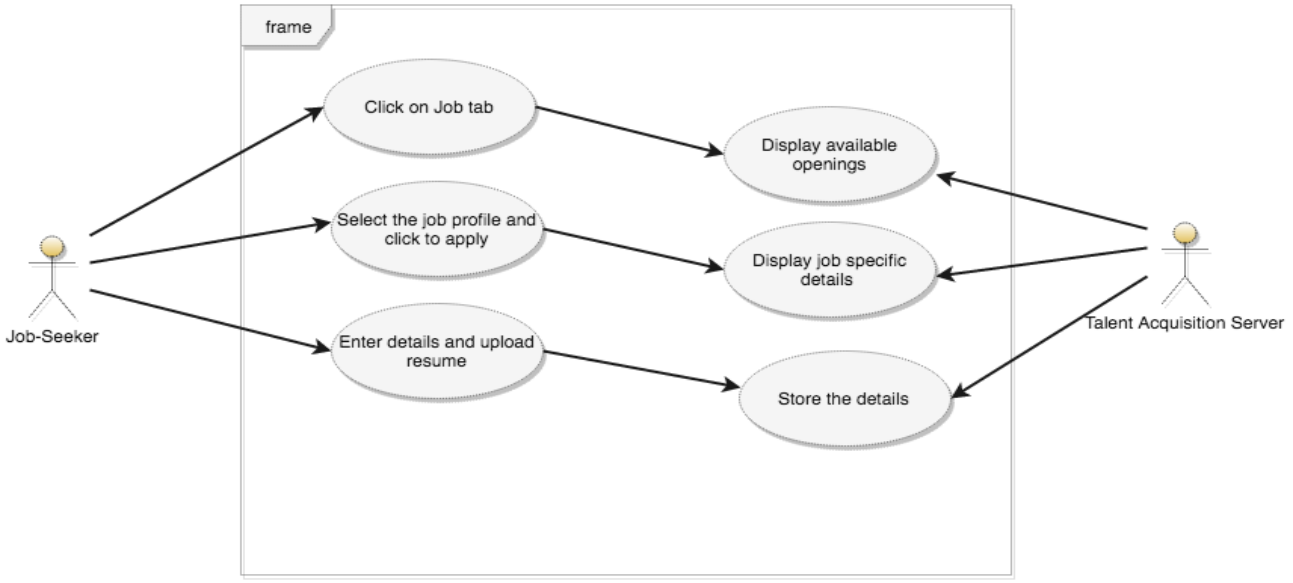


Figure 18 Use case diagram to create user profile

Use case specification for Profile Creation

Use case Name	Display available openings
Actors	User (Job Seeker), Talent Acquisition Server
Description	Talent Acquisition web server fetches open positions from the database and displays the information in tabular format on job page.
Basic Flow	1. The user clicks on the “Job” tab on the dashboard. 2. Talent Acquisition web server fetches available openings from the database. 3. Talent Acquisition web server displays open positions in tabular format.

Use case Name	Creating Job specific profile

Actors	User (Job Seeker)
Description	The job seeker can maintain multiple job profiles. The number of profiles are limited to total count of available openings.
Basic Flow	1. The user selects job opening on the "Job" page. 2. Job details page contains skills requirement and job profile information. 3. The user enters personal information such as name, E-mail, and skills etc. 4. The user uploads the resume to the portal.

Use case Name	Storing user profile
Actors	Talent Acquisition Server
Description	Talent Acquisition Server stores jobs related profiles and the user's resume in system's database.
Basic Flow	1. After profile submission by the user, Talent Acquisition Server stores the details in the database. 2. Once profile is stored it is available for next process.

Sequence diagram for Profile Creation

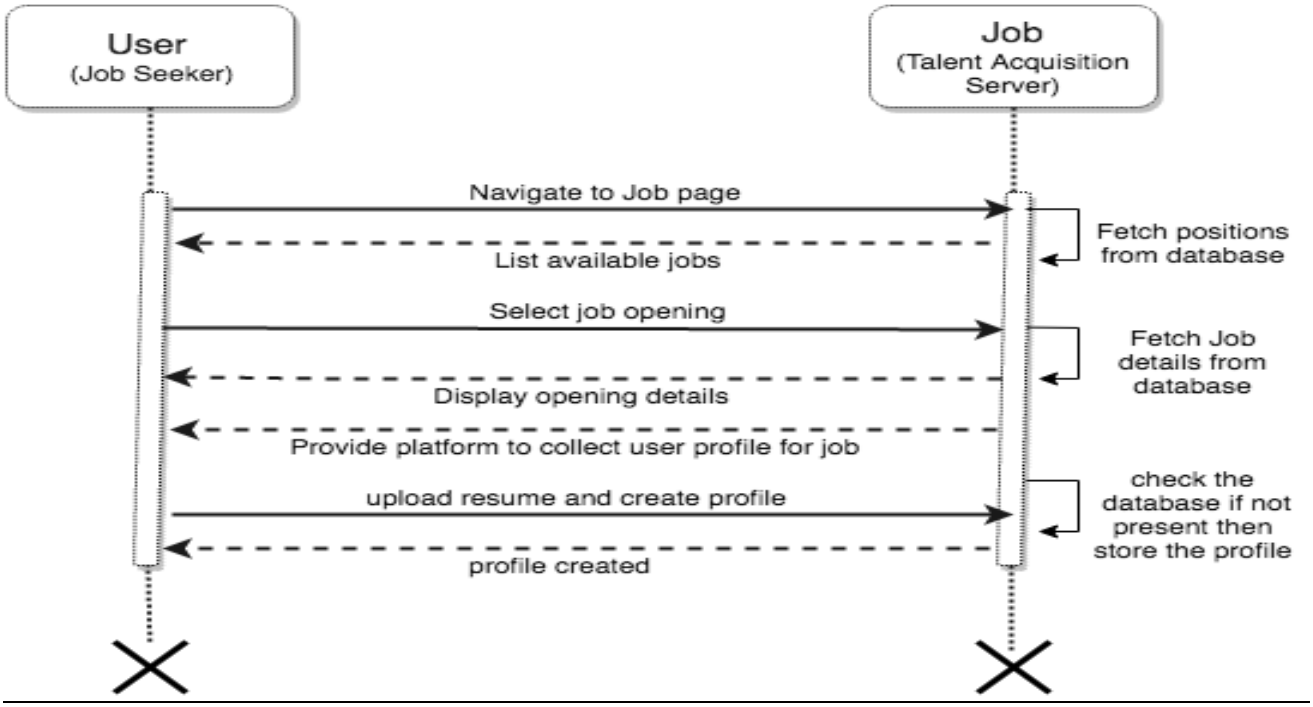


Figure 19 Sequence diagram for creating a profile

***Sequence diagram description**:*

- The job seeker clicks on the " Job" tab to view the open job position.
- Job object of the system fetches open position details from the database and displays the list on job page.
- The job seeker selects the Job opening.
- Job object display job specific details on the new page and provides platform to build profile for the available position.
- The job seeker submits the information and uploads the resume.
- The job object check the database, if the entry is not present in the database then new profile is created for the user and information stored in the database.

Use case diagram to Change Application status

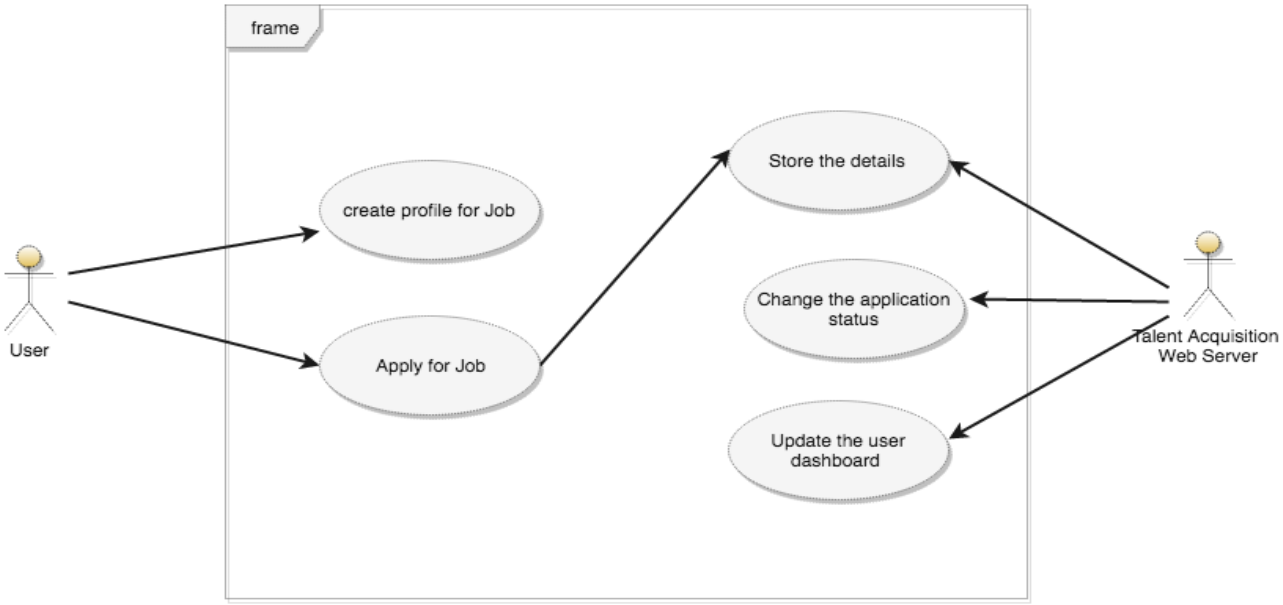


Figure 20 :Use case diagram to update application status

Use case specification to update application status

Use case Name	Apply for Job
Actors	User (Job Seeker), Talent Acquisition Web Server
Description	Talent Acquisition Server displays jobs on "Job" page and the user applies on a particular Job.
Basic Flow	1. The user clicks on the Job tab on the dashboard. 2. The user creates profile for job and hit ok. 3. User's information is stored in the database.

Use case Name	Change in status of the application
Actors	Talent Acquisition Web Server
Description	Talent Acquisition Web Server changes the status of the application on user's dashboard (both Employer and Job seeker) .
Basic Flow	1. Once the user applies for the job, Talent Acquisition Server changes the status of the application. 2. Talent Acquisition Server updates the database with new status. 3. Talent Acquisition Server updates the user's portal as well.

Sequence diagram to update application status

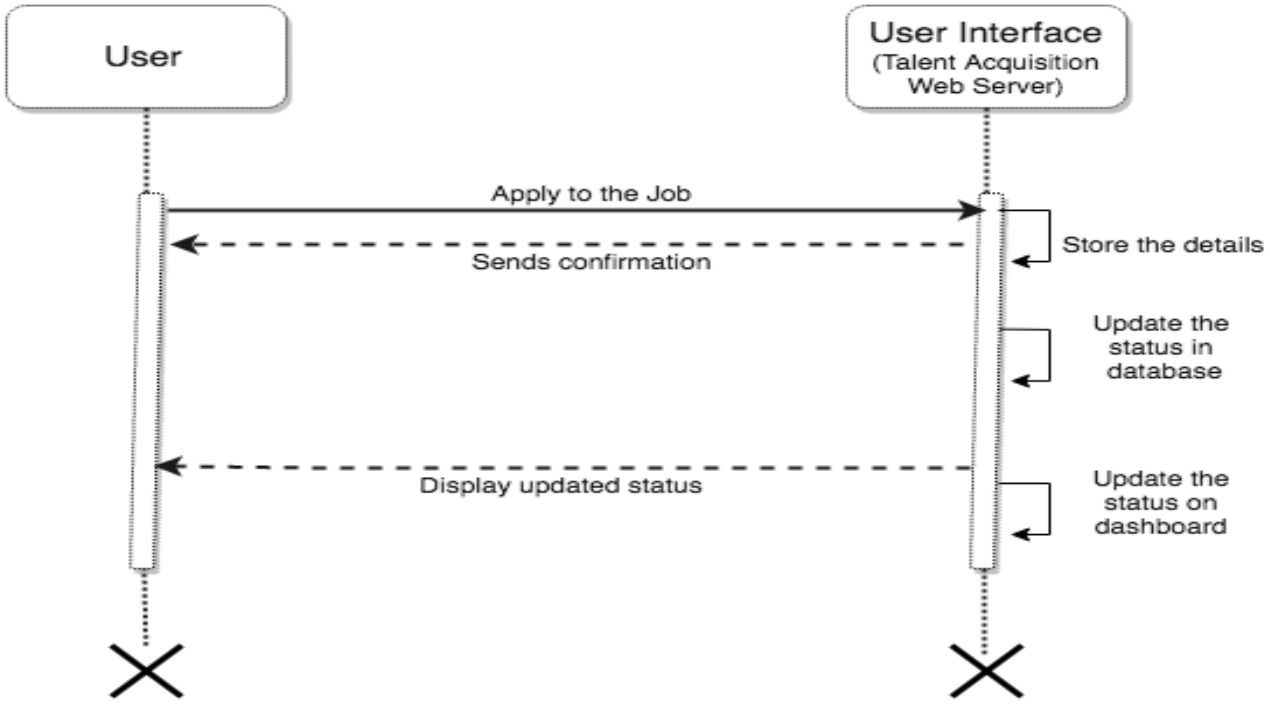


Figure 21: Sequence diagram to update application status

Sequence diagram description:

- The job seeker applies to the Job by creating profile.
- Talent Acquisition Server stores the details.
- User Interface component‘s object updates the database and dashboard for new status.
- User Interface component‘s object updates the application status on the user’s dashboard.

Use case diagram for Job application

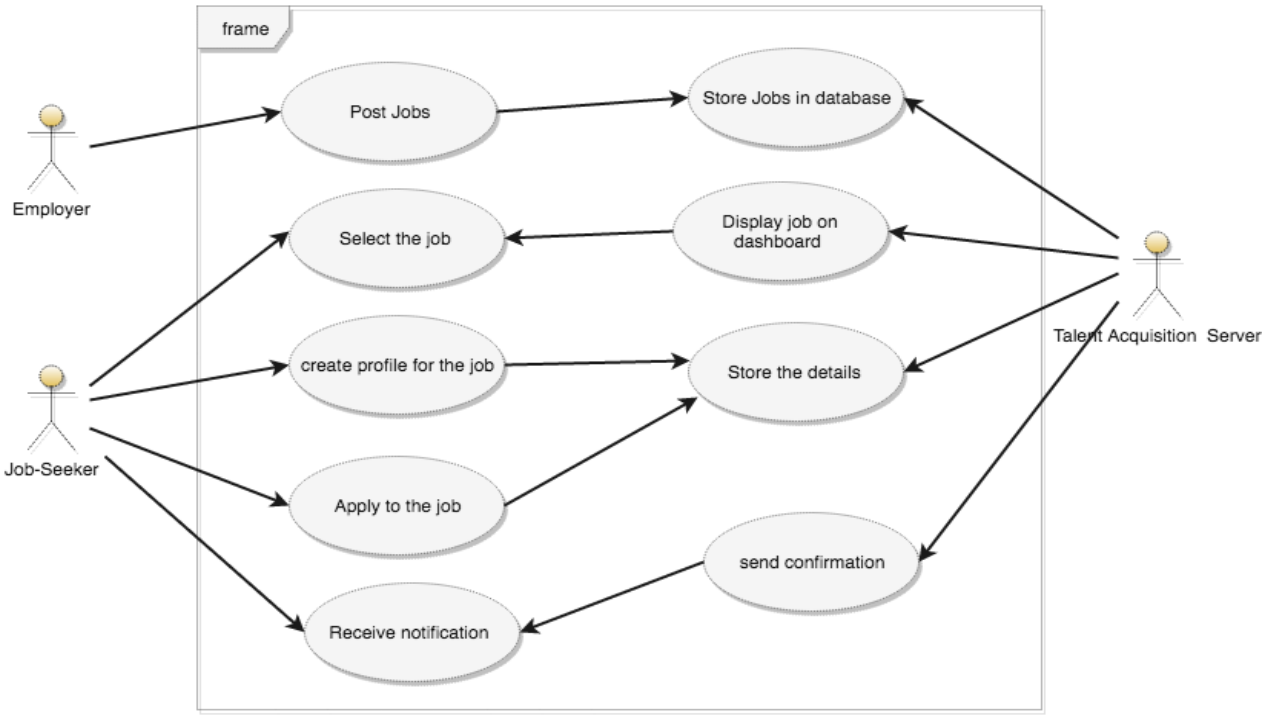


Figure 22: Use case diagram to apply for a job

Use case specification for Job application

Use case Name	**Post the job**
Actors	User (Employer)
Description	The employer posts a job opening on the portal.
Basic Flow	1. The user specifies the requirements on the job page. 2. The user submits those requirements to the system. 3. Talent Acquisition Server stores the requirements of the job.

Use case Name	Select the job and create profile for job
Actors	User (Job Seeker)
Description	The job seeker selects the job from the list and creates profile specific to the job.
Basic Flow	1.The user chooses job from the job pool. 2. The user creates profile for the job. 3. Talent Acquisition Server stores the profile.

Use case Name	Apply for the job
Actors	User (Job Seeker), Talent Acquisition Server
Description	The job seeker applies for the job.
Basic Flow	1. The user clicks on the job link. 2. The user submits the profile and applies for the job.

Use case Name	Store information in the database
Actors	Talent Acquisition Server
Description	To access the user's information and application details in future, Talent Acquisiti Server stores the details in the database.
Basic Flow	1. The user hits the "OK" button and applies for the job. 2. Talent Acquisition Server saves user's application in the database for future reference.

Use case Name	Receive notification or confirmation
Actors	User (Employer or Job Seeker), System (Talent Acquisition System)
Description	Talent Acquisition Server sends confirmation to the user once application has been submitted.
Basic Flow	1. Talent Acquisition Server receive request to saves information in the database. 2. Talent Acquisition Server verifies the request. 3. Once details are updated in the database, Talent Acquisition Server sends a notification to the user.

Sequence diagram for Job application

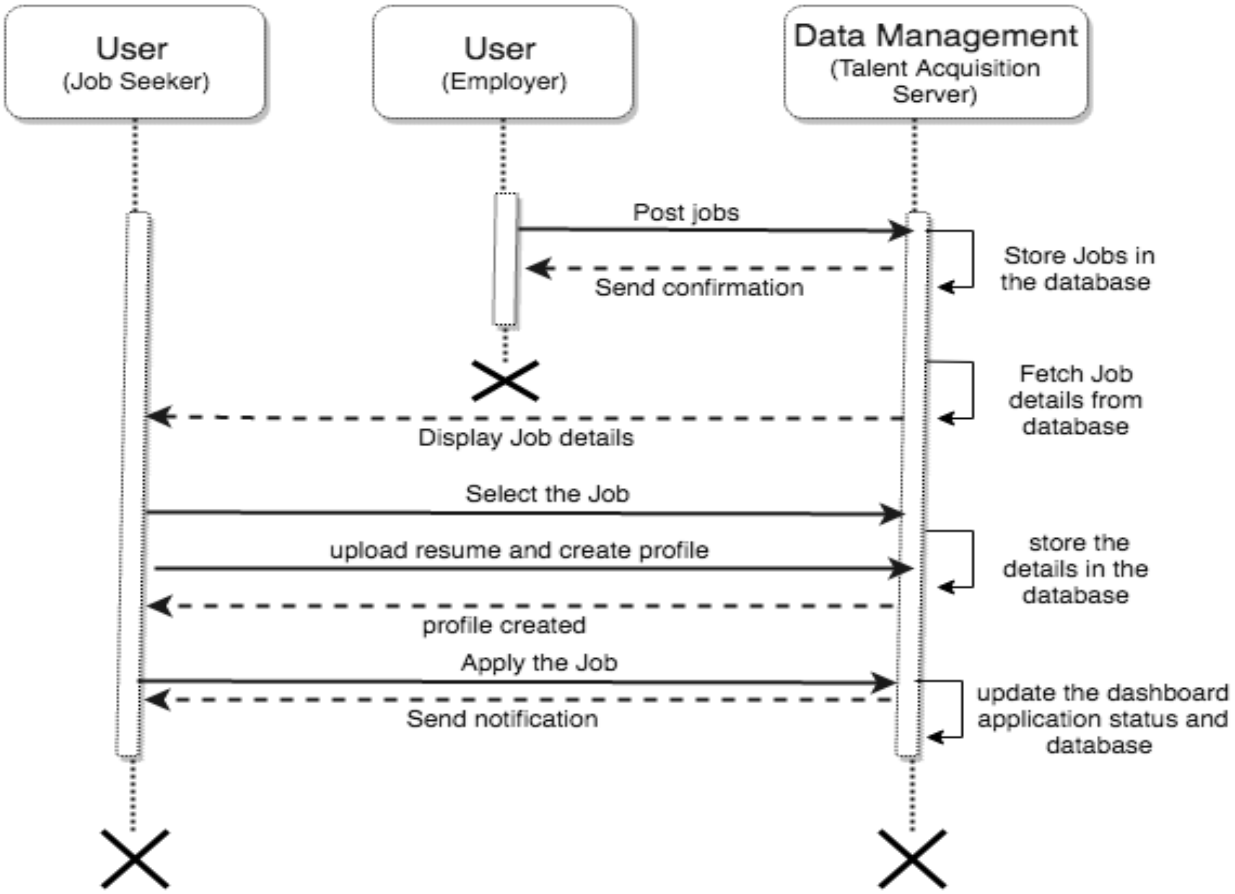


Figure 23:: Sequence diagram to apply for a Job

Sequence diagram description:

- The employer posts the job on the portal.
- Data Management object of the system updates the database and dashboard accordingly.
- The job seeker views updated dashboard and select the job.
- The job seeker creates the profile for the job and upload resume.
- Data Management object sends confirmation of the profile creation and stores the profile.
- The job seeker applies to the job.
- Data Management object updates the database and dashboard.
- Data Management object sends notification to the user.

Use case diagram to generate recommendation based on GitHub contribution and location

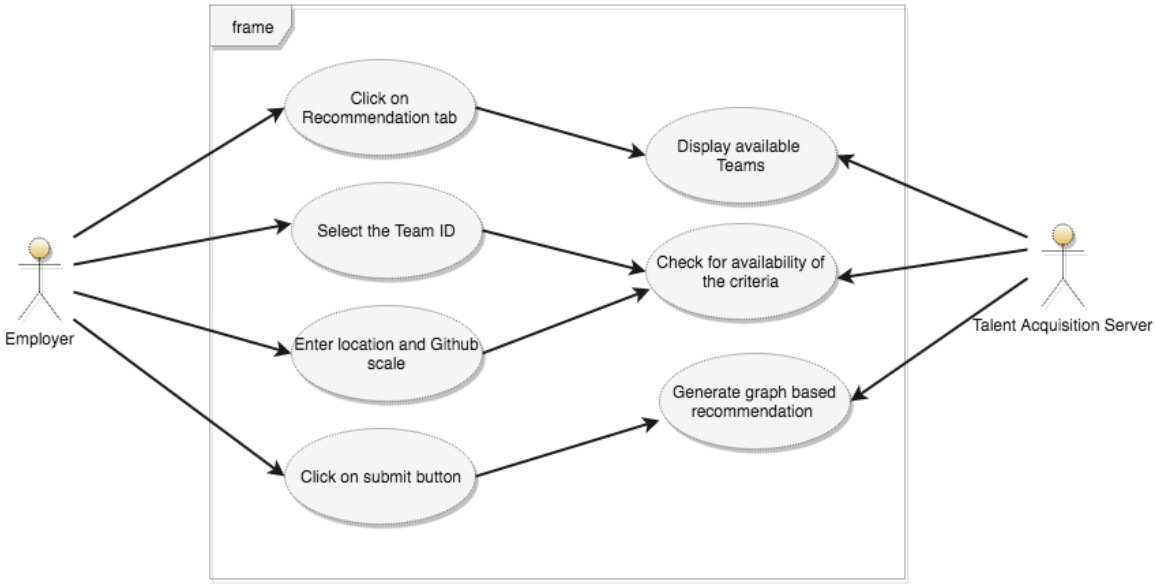


Figure 24: Use case diagram to generate recommendation based on GitHub and Location

Use case specification to generate recommendation based on GitHub and Location

Use case Name	Enter GitHub link and location
Actors	User (Job seeker), Talent Acquisition Server
Description	Job seeker provides GitHub link and location information.
Basic Flow	1. The user selects the Job tab. 2. The user provides the GitHub link on the job portal. 3. Talent Acquisition Server stores the information into the database and parse the informationto the recommendation engine as requested.

Use case Name	Select GitHub contribution and location
Actors	User (Employer), Talent Acquisition Server
Description	The employer selects GitHub contribution and location criteria for the recommendation.
Basic Flow	1. The user click on the Recommendation tab on the dashboard. 2. Talent Acquisition Server displays available Team ID from the database. 3. The userselects the GitHub contribution and location ratio based on scale.

Use case Name	Generate Recommendation based on GitHub contribution and location
Actors	User (Employer), Talent Acquisition Server
Description	Talent Acquisition Server generates recommendation for teams based on GitHub contribution and location criteria entered by the user.

Basic Flow	1. The user clicks on the submit button and Talent Acquisition Server verifies the threshold. 2. If there are not enough teams for recommendation Talent Acquisition Server generates error message else generate recommendation as per the input.

Sequence diagram to generate recommendation based on GitHub and Location

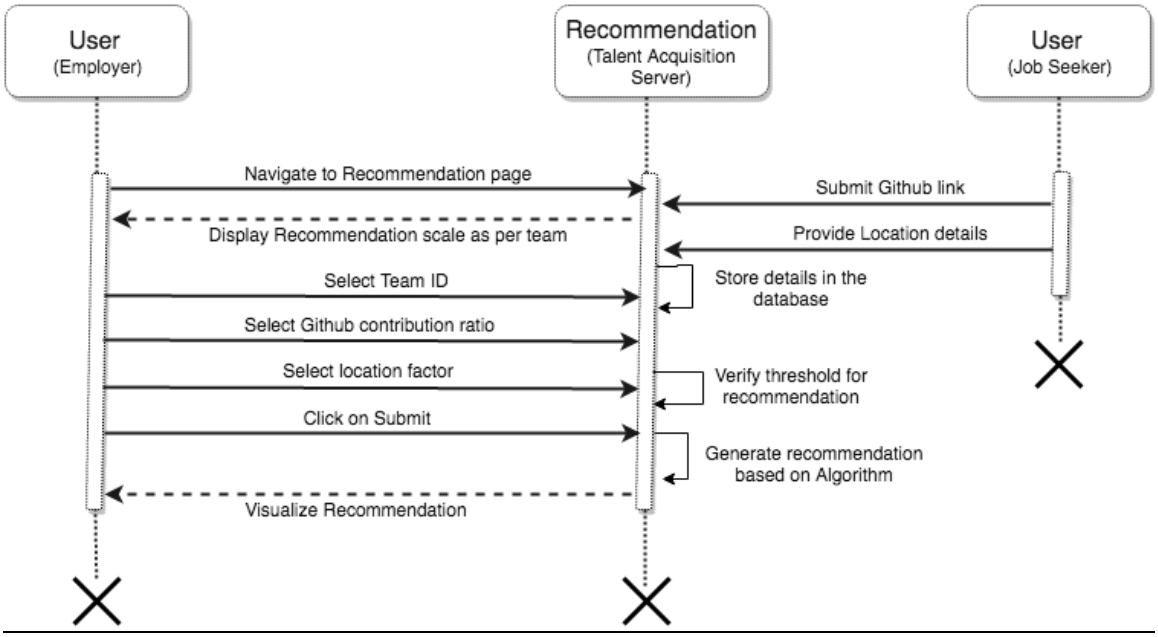


Figure 25: Sequence diagram to generate recommendation based on GitHub and Location

Sequence diagram description:

- The job seeker submits the GitHub link and location preference while submitting application for the job.
- Recommendation object of the system stores this information in the database.
- The employer navigates to the recommendation page.
- The recommendation object displays the recommendation scale for the team.
- The employer selects Team ID and provides factors to generate recommendation such as GitHub contribution and Location.
- The employer click on submit.
- Recommendation object verifies the threshold. If there is no team to compare, it generates an error message. Otherwise, it generates the visualization for recommendation.

Use case diagram to generate recommendation based on Diversity

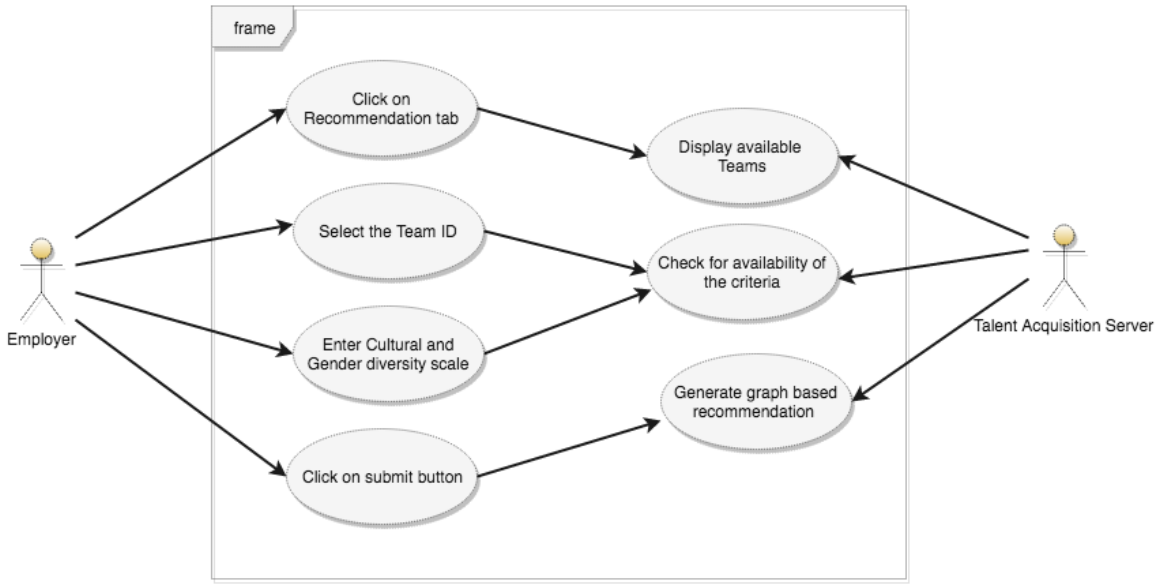


Figure 26: Use case diagram to generate recommendation based on Diversity

Use case specification to generate recommendation based on Diversity

Use case Name	Enter cultural diversity and gender diversity
Actors	User(Job seeker), Talent Acquisition Server
Description	Job seeker provides cultural (race) and gender information while applying for a job.
Basic Flow	1. The user selects the Job tab. 2. The user provides the gender and race information on the job portal. 3. Talent Acquisition Server stores the information into the database and parses the information for the recommendation engine as requested.

Use case Name	Select cultural diversity and gender diversity
Actors	User (Employer), Talent Acquisition Server
Description	The employer selects cultural diversity and gender diversity for the recommendation.
Basic Flow	1. The user clicks on the “Recommendation” tab on the dashboard. 2. Talent Acquisition Server displays available Team ID from the database. 3. The user selects the cultural and gender diversity ratio based on scale.

Use case Name	Generate Recommendation based on cultural and gender diversity
Actors	User (Employer), Talent Acquisition Server
Description	Talent Acquisition Server generates recommendation for teams based on cultural and gender information entered by user.
Basic Flow	1. Once the user clicks on the submit button Talent Acquisition Server verifies the threshold. 2.Talent Acquisition Server generate recommendation if teams are available above threshold value.

Sequence diagram to generate recommendation based on cultural and gender diversity

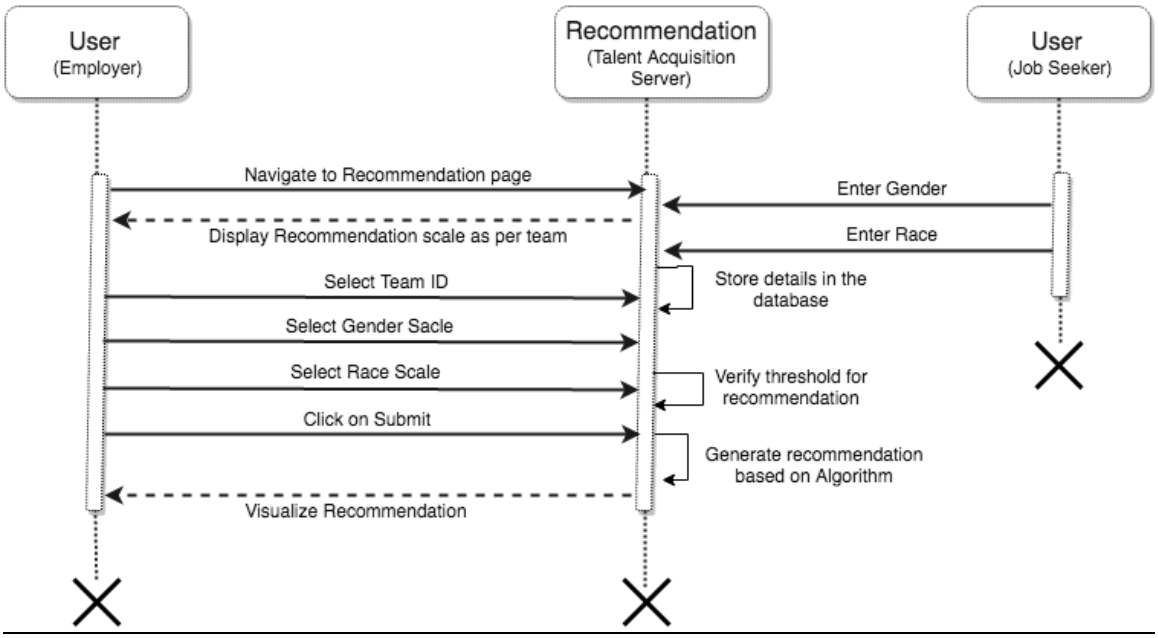


Figure 27: Sequence diagram to generate recommendation based on cultural and gender diversity

Sequence diagram description:

- The job seeker submits the gender and race details during job application submission.
- Recommendation component object of the system stores this information in the database.
- The employer navigates to the recommendation page.
- Talent Acquisition Server displays Recommendation scale as per the team.
- The employer selects Team ID and provides factors to generate recommendation such as gender or cultural ratio.
- The employer clicks on submit.

- Talent Acquisition Server generates a visualization for the recommendation if the number of teams is above threshold value in the system.

Use case diagram to visualize recommendation in graphical format

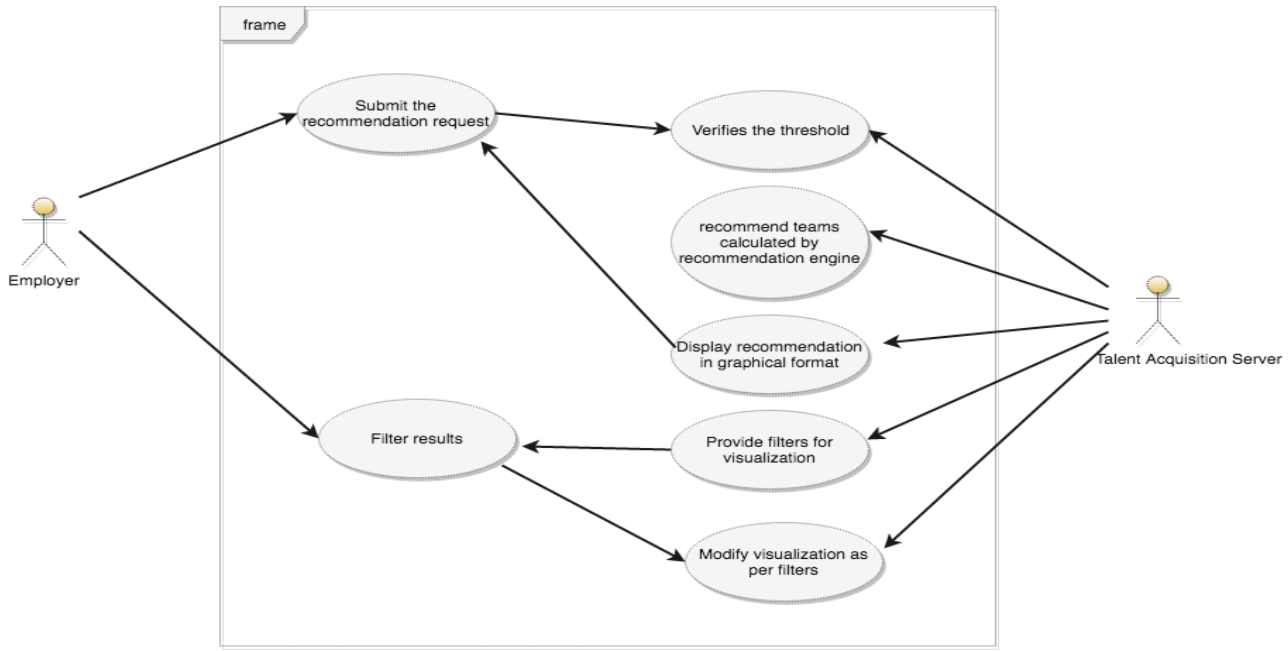


Figure 28: Use case diagram to visualize recommendation in graphical format

Use case specification to visualize recommendation in graphical format

Use case Name	Submit the recommendation request
Actors	User(Employer)
Description	The employer click on the submit button and request for team recommendation.
Basic Flow	1. The user enters criteria for Recommendation. 2. The user clicks on the submit button.

Use case Name	Generate team recommendation
Actors	Talent Acquisition Server
Description	Talent Acquisition Server provides team recommendation based on recommendation engine algorithm that verifies threshold.

Basic Flow	1. Once the user submits the recommendation request, Talent Acquisition Server check the database threshold value to generate recommendation. 2. If number of team profiles are above threshold then recommendation engine calculate strongest relations. 3. Talent Acquisition Server generate graph for team recommendation.

Use case Name	Display team recommendation on user dashboard
Actors	Talent Acquisition Server
Description	Talent Acquisition Server displays team recommendation based on diversity, GitHub and location through the graph.
Basic Flow	1. Talent Acquisition Server fetches information from the database. 2. Talent Acquisition Server weights the team as per the selected factors. 3. Talent Acquisition Server displays the recommendation in the graph format.

Use case Name	Modify visualization based on filters
Actors	User (Employer), Talent Acquisition Server
Description	The employers manage and modify visualization of the team recommendation as per their preferences.
Basic Flow	1. Talent Acquisition Server provides visualization of the recommended team as an interactive graph. 2. The user can filter the result as per their choice. 3. Talent Acquisition Server modifies the visualization (graph) based on the filter provided by the user.

Sequence diagram to visualize recommendation in graphical format

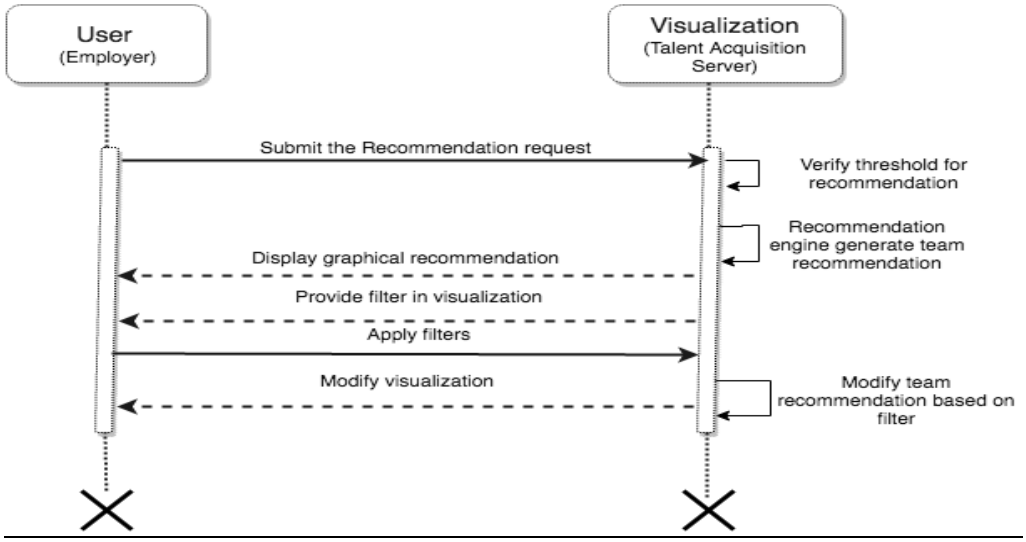


Figure 29: Sequence diagram to visualize recommendation in graphical format

***Sequence diagram description**:*

- The employer submits the recommendation request to the server.
- Talent Acquisition Server verifies the threshold and further instructs recommendation engine to generate team recommendation.
- Talent Acquisition Server displays graph for team recommendation on the user's dashboard.
- This graph also provides option to select filters.
- The user applies filters to modify the visualization.
- Visualization object of the system modifies the visualization as per the filter applied.

Data-Tier Design

Use case diagram for resume parsing

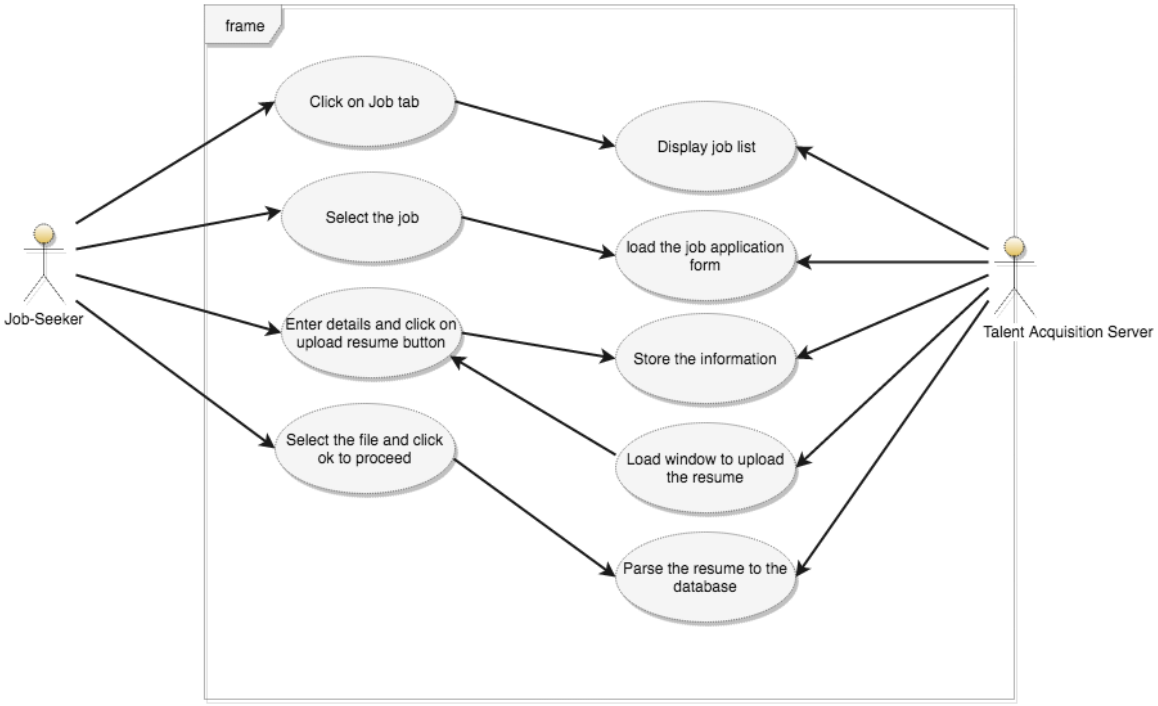


Figure 30: Use case diagram for resume parsing

Use case specification for resume parsing

Use case Name	Select and apply for the job from job list
Actors	User (Job seeker), Talent Acquisition Server
Description	Job seeker checks available job list for selection and apply for the job.
Basic Flow	1. The user click on the job tab. 2. The user's dashboard displays available job listing. 3. The user selects one job and clicks on the job name to apply for it. 4. After job selection, Talent Acquisition Server loads the job application form.

Use case Name	Enter personal information
Actors	User (Job seeker), Talent Acquisition Server
Description	Job seeker provides personal information such as name, email address, GitHub and phone number through the job application process.
Basic Flow	1. Talent Acquisition Server upload form to get user's information. 2. The user enters the information and Talent Acquisition Server stores the information in the database.

Use case Name	Upload resume to the server
Actors	User (Job seeker)
Description	Job seeker uploads the resume to the server.
Basic Flow	1. To upload resume user clicks on the upload button on the job application form. 2. User interface component lets user select desired file from the device to upload. 3. The user selects the file and click ok.

Use case Name	Parse resume data for the database
Actors	Talent Acquisition Server
Description	Talent Acquisition Server parses resume and upload the data to the database.
Basic Flow	1. After successful upload of resume Talent Acquisition Server interacts with external APIs(Application Programming Interfaces). 2. APIs parse the resume information to the database.

Sequence diagram for resume parsing

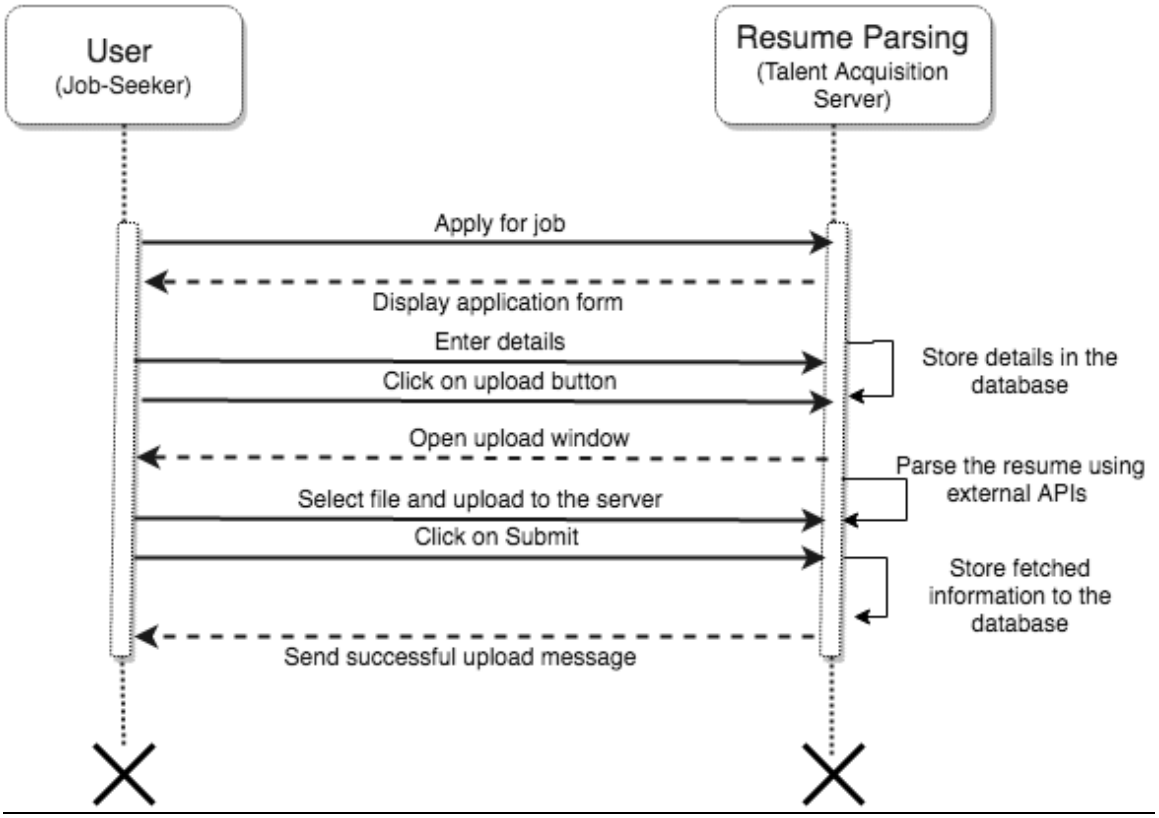


Figure 31: Sequence diagram for resume parsing

Sequence diagram description:

- The job seeker submits the personal information while submitting application for the job.
- The job seeker clicks on the upload button.
- User interface by server provides platform to upload the resume files.
- The user selects the file and uploads the resume to the server.
- Talent Acquisition Server parses the resume using external APIs and stores the fetched information into the database.

Data-Tier Design

1. ER Diagram

Talent Acquisition System has two business data models:

1. Recruiter data model: the company's recruiter will create and post a team profile for any particular location.
2. Candidate data model: the candidate will enter his details like resume and preferred role.

Our recommendation engine is one business process to provide results for both models. The recruiter and the candidate will not communicate directly. However, the recommendation engine will act as an interface for both models.

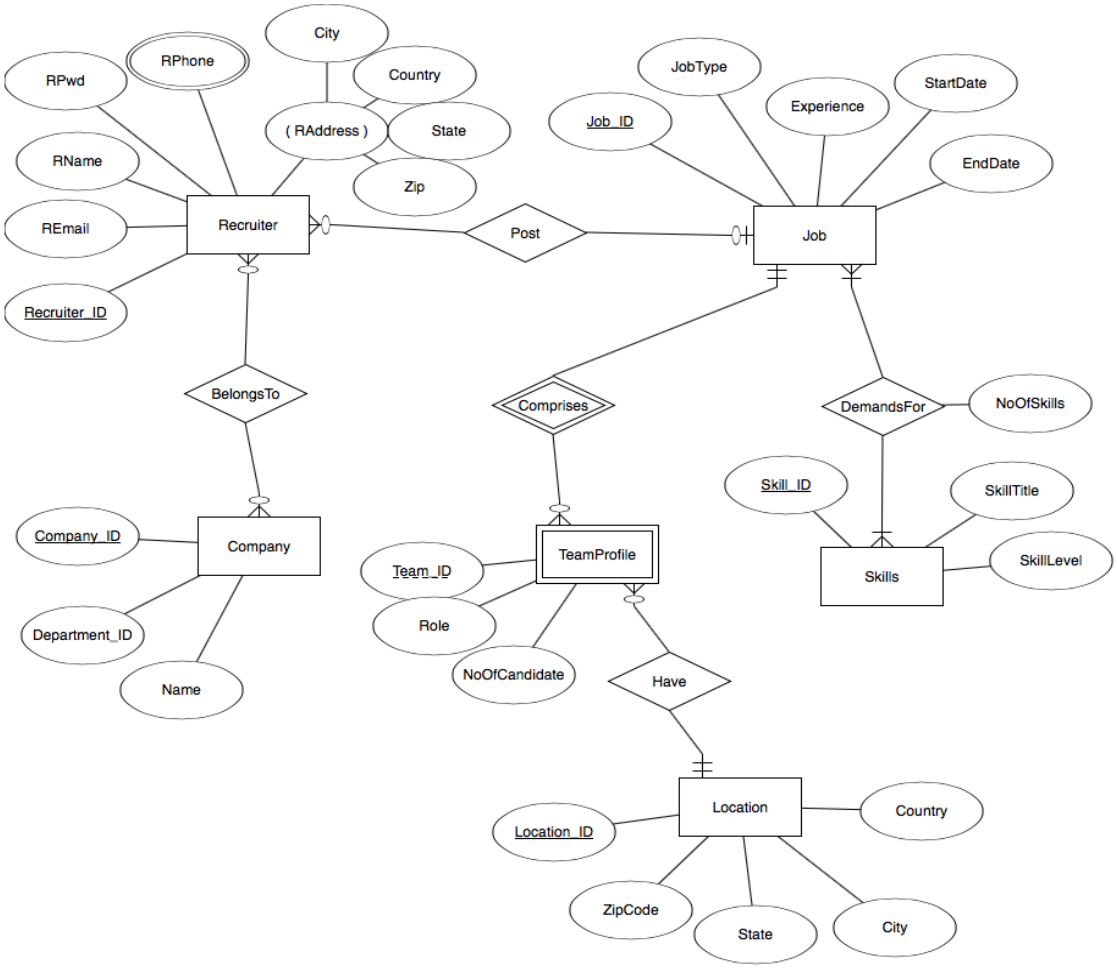


Figure 32: ER Diagram for Recruiter login

Figure 32 shows the entity-relationship diagram from the recruiter's perspective.

- Every company can hire one to many recruiters. Each company has a unique Company_ID and Name.
- The recruiter can create and post Jobs for the company. The Recruiter entity has various attributes like Email, Name, password and location details. A recruiter can have multiple phone numbers and every recruiter has a unique Recruiter_ID.
- Each job has a unique Job_ID along with title, date of employment, and required skills and experience. One Job requires many skills, or one skill is a requirement for multiple job openings.
- Multiple job openings for a specific location build a team profile. Every team has a partial id as teamID and is dependent on the job table.
- A team is dependent on a specific location provided by the recruiter as a team profile requirement specification.

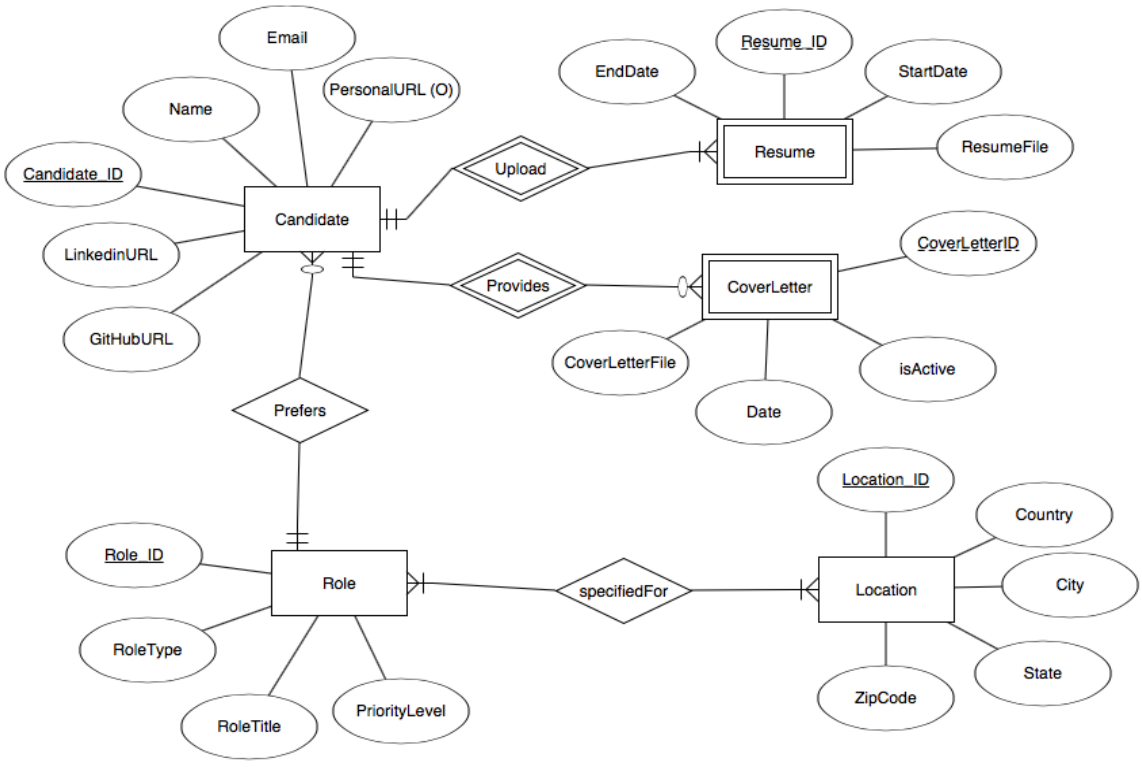


Figure 33:ER Diagram for candidate login

Figure 33 is the entity relationship diagram from a candidate's perspective.

- Every candidate has unique a Candidate_ID. Each candidate will provide a LinkedIn UTL and a GitHub UTL and can upload resumes and cover letters.
- One candidate can upload multiple resumes and cover letters. Uploaded resumes and cover letters will have unique partial ids as ressume_ID and Cover_letterID respectively.
- Candidates prefer a role for a particular location. Every instance of Role and Location are identified by unique Role_ID and Location_ID respectively

Chapter 5. Project Implementation

Client Implementation

Our application is based on MEAN stack technologies (MongoDB, Express, AngularJS and Node.js) and it is Model View Controller framework. The view of the application is completely isolated from the data model and the application logic. The client application is implemented using AngularJS 1.5. Each view is controlled by a controller.

1. Candidate Login Screen:

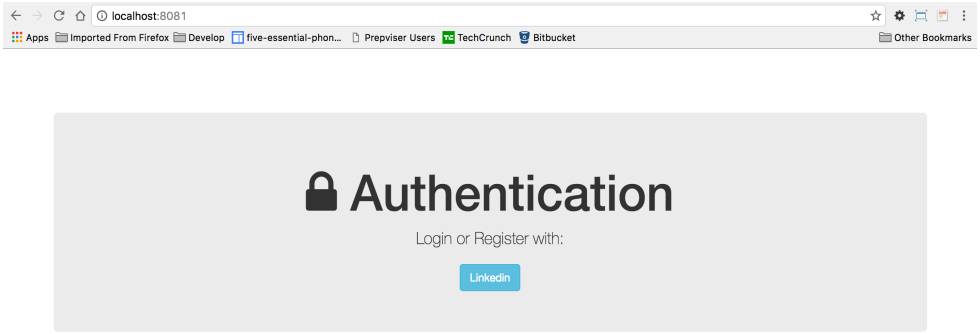


Figure 34: Candidate Login Screen

- The login screen enables job seekers to login with their LinkedIn login credentials. Once a job seeker clicks on the login button, the application calls a '/auth/linkedin' route in Node.js server.

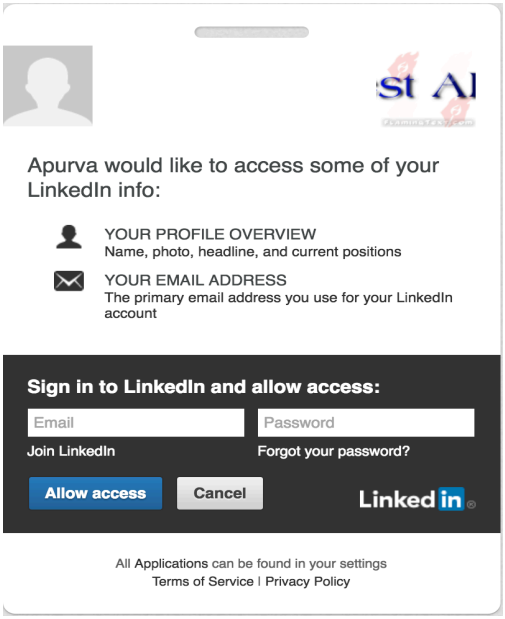
```
// send to linkedin to do the authentication
app.get('/auth/linkedin', passport.authenticate('linkedin',{ scope: ['r_basicprofile', 'r_emailaddress'] }));

// handle the callback after linkedin has authenticated the user
app.get('/auth/linkedin/callback',
passport.authenticate('linkedin', {
  successRedirect : '/profile',
  failureRedirect : '/'
}));
```

- Passport.js is an authentication middle ware for Node.js application. Different authentication strategies such as LinkedIn, Facebook, Twitter and Local login is defined in the passport file.

- LinkedIn route call LinkedIn strategy defined in Passport.js file. LinkedIn strategy fetch our LinkedIn application's clientID, clientSecret and callbackURL from our configuration file.

```
passport.use(new LinkedInStrategy({
  clientID: configAuth.linkedinAuth.clientID,
  clientSecret: configAuth.linkedinAuth.clientSecret,
  callbackURL: configAuth.linkedinAuth.callbackURL,
  profileFields: ['id', 'first-name', 'last-name', 'email-address','public-profile-url','picture-url','location'],
  state:true
},
```

- The LinkedIn strategy redirects a user to the LinkedIn authentication page.



- Once a user logs in with the LinkedIn credentials, the LinkedIn application calls the callbackURL which redirects user to the profile page. The LinkedIn strategy checks if the

user exists in our database and returns the user details from database. Otherwise, it creates a new user for the database.

```
function(req, token, tokenSecret, profile, done) {

  User.findOne({ '_id': profile.id }, function (err, user) {
    if(err) return done(err);

    if(user) {
      done(null, user);
    } else {
      var newUser = new User();

      newUser._id    = profile.id;
      newUser.username = profile.id;
      newUser.token = tokenSecret.access_token;
      newUser.name  = profile.name.givenName;
      newUser.email = profile.emails[0].value;
      newUser.location = profile._json.location.name;
      newUser.picUrl = profile.photos[0].value;
```


```
{
  "_id": "dNxo1R-Yo-",
  "picUrl": "https://media.licdn.com/mpr/mprx/
pnoHNs6ORLamtWosKRfVQR3NwaHcv",
  "location": "Wichita, Kansas Area",
  "email": "apurvalink@email.com",
  "name": "Test",
  "token": "AQXM1pepPsZERhmumqBJSD5KyMF85RReTP
XiI9l42VRWW9P8G7AG1QWbywya4VsSe-E",
  "username": "dNxo1R-Yo-",
  "skills": [
    "Java",
    "HTML",
    "CSS",
    "JavaScript"
  ],
  "__v": 0
}
```


- Once the authentication process is done, the LinkedInStrategy calls our profile app. Our profile app checks whether the user is already logged in or not, and if logged in, the application will redirect the user to the application profile page

```
// PROFILE SECTION =========================
app.get('/profile', isLoggedIn, function(req, res) {
  var user = { name: req.user.name, email : req.user.email, picUrl: req.user.picUrl, _id : req.user._id };
  res.render('../app_client/main.ejs', user);
});
```

2. Candidate Profile Screen:

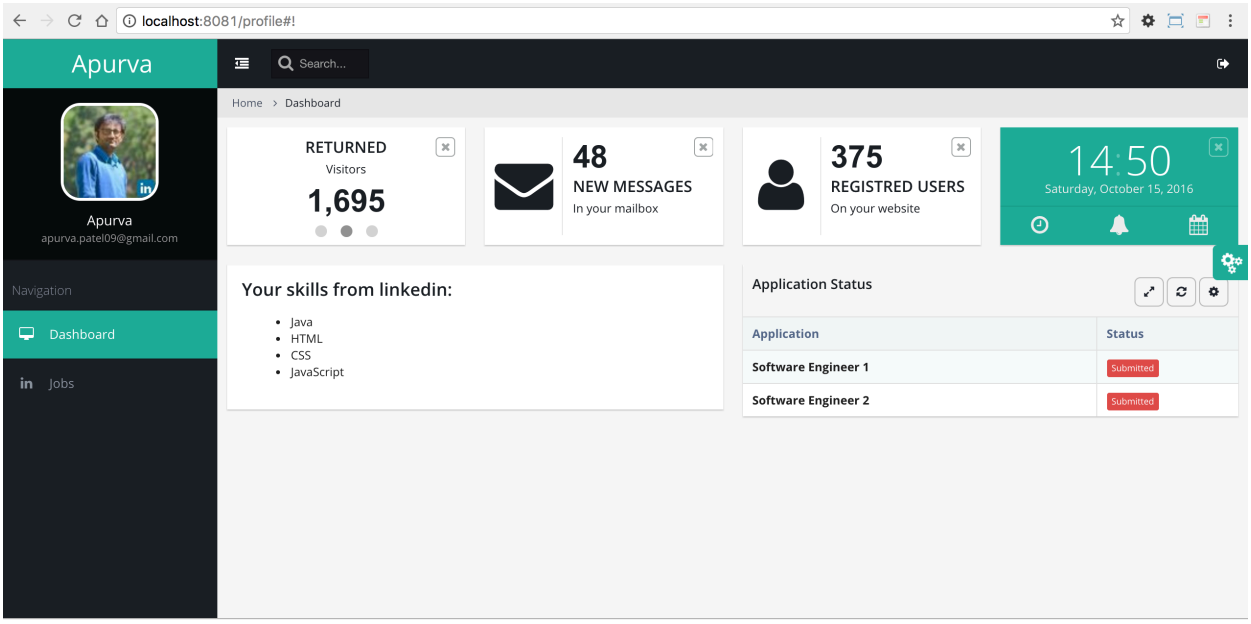


Figure 35: Candidate Profile Screen

- The profile app renders main.ejs file which loads our main.js controller which is an Angular controller. Main.js file defines all routes of our front end application.

```
(function () {

  angular.module('meanApp', ['ngRoute']);

  function config ($routeProvider, $locationProvider) {
    $routeProvider
    .when('/', {
      templateUrl: '/profile/profile.view.html',
      controller: 'profileCtrl',
      controllerAs: 'vm'
    })
    .when('/profile', {
      templateUrl: '/profile/profile.view.html',
      controller: 'profileCtrl',
      controllerAs: 'vm'
    })
    .when('/jobs', {
      templateUrl: '/jobs/jobs.view.html',
      controller: 'jobsCtrl',
      controllerAs: 'vm'
    })
    .when('/jobProfile/:id', {
      templateUrl: '/jobProfile/jobProfile.view.html',
      controller: 'jobProfileCtrl',
      controllerAs: 'vm'
    })
```

- '/profile' route loads profile.view.html and a 'profileCtrl' which is a controller for the profile page. The profile controller fetches user details by making an HTTP request to our Node.js server.

```
(function() {

  angular
  .module('meanApp')
  .controller('profileCtrl', profileCtrl);

  profileCtrl.$inject = ['$location','$http','$routeParams','$window'];
  function profileCtrl($location,$http,$routeParams,$window) {
    var vm = this;
    vm.user = {};
    var user = $window.localStorage['username'];

    $http.get('/user').success(function(data) {
      vm.user = data;
      $http.get('/application').success(function(data) {
        vm.application = data;
      });
    });
```

- The Node.js route for fetching the user details from database is:

```
router.get('/user', function(req, res) {
  User.findOne({'_id': req.user._id },function(err, userProfile) {
    res.send(userProfile);
  });
});
```

- Once the user details are successfully fetched, Angular application calls '/application' route from our server which returns the user's past applications from our database. Applications are stored in the following format

```
{
  "_id": ObjectId("58029344f2bf2650ee9755fc"),
  "resumeID": "58029344f2bf2650ee9755fa",
  "applicationStatus": "Submitted",
  "applicantLocation": "Wichita, Kansas Area",
  "applicantEmail": "apurvalink@email.com",
  "applicantGithubID": "apurvalink14",
  "applicantID": "dNxo1R-Yo-",
  "applicantName": "Test",
  "position": "Senior Software Engineer",
  "jobID": "57f825fc89df9c81888d7df9",
  "applicantGithubConnections": [ ],
  "applicantSkills": [
    "Java",
    "HTML",
    "CSS",
    "JavaScript"
  ],
  "__v": 0
}
```


- Once all the user details are fetched from the database, the user can see all the details on the dashboard.

3. Posted Jobs Screen:

A user can view posted jobs by clicking on the Jobs tab from the dashboard:

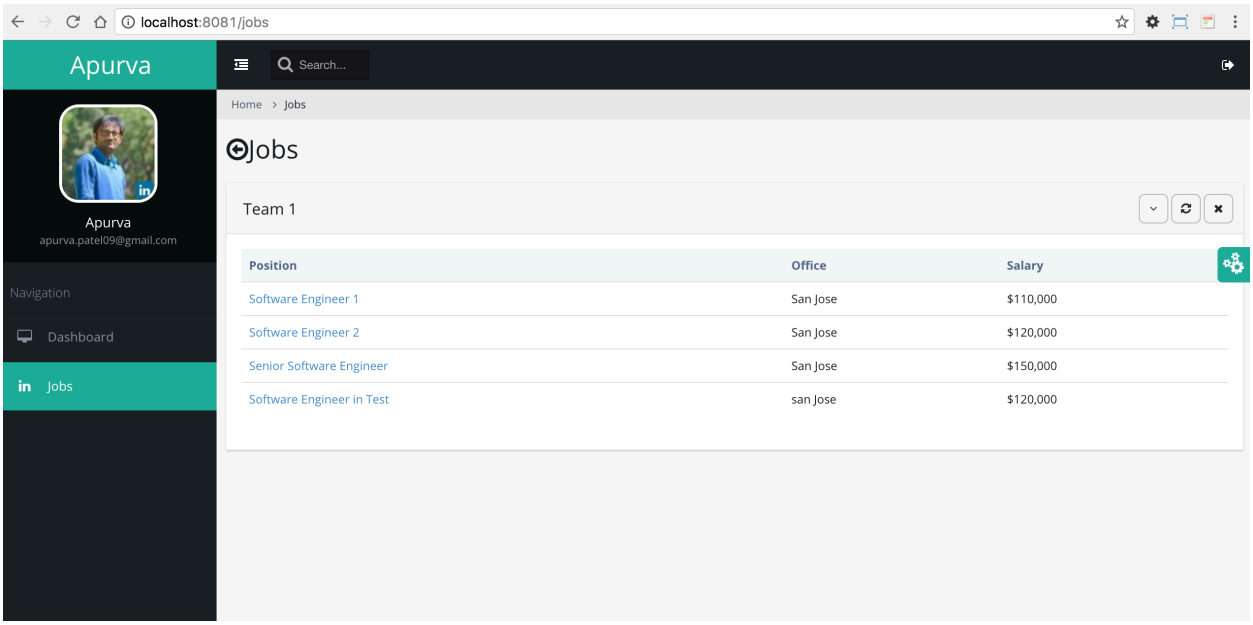


Figure 36:Posted Jobs Screen

- Once the user clicks on the Jobs tab, the application calls '/jobs' route from main.js

```
    .when('/jobs', {
      templateUrl: '/jobs/jobs.view.html',
      controller: 'jobsCtrl',
      controllerAs: 'vm'
    })
```

- /jobs route calls jobsCtrl which is a job controller. The job controller makes an HTTP request to the server to fetch all the posted jobs from the database and return all the posted jobs from the database:

```
$http.get('/job').success(function(data) {
    vm.jobs = data;
    console.log(vm.jobs);
});
```

- Jobs are stored in the database in the following schema:


```
{
  "_id": ObjectId("57f825fc89df9c81888d7df9"),
  "position": "Senior Software Engineer",
  "description": "Designs, develops, and implements software packages
's degree in a related area and at least 8 years of experience in the
s within a particular field (i.e., SQL, C++, HTML, CGI and JavaScript
iety of tasks. Works under general supervision. A certain degree of cr
  "salary": "$150,000",
  "location": "San Jose",
  "skills": [
    "Java",
    "JavaScript",
    "HTML",
    "CSS",
    "Node.js",
    "Python",
    "C++"
  ],
  "__v": 0
}
```

4. Job Profile Screen:

A user can see the detailed profile about the job by clicking on the job link on the posted job screen:

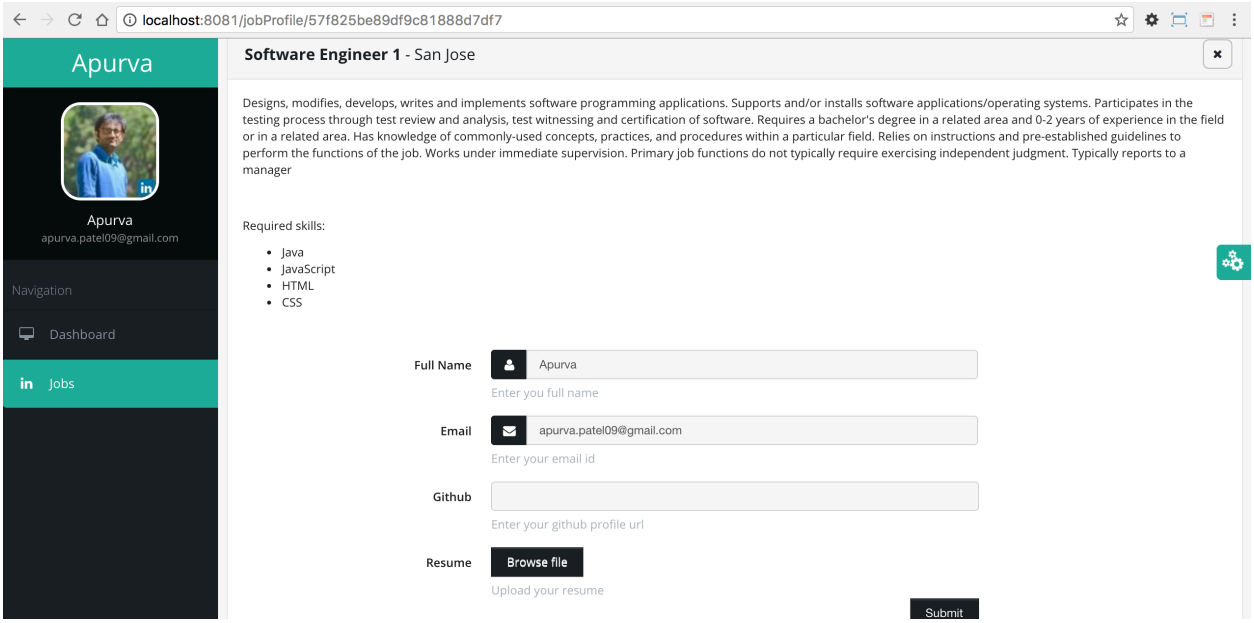


Figure 37: Job Profile Screen

- Once the user clicks on a particular job, the application calls '/jobProfile' route from main.js:

```
.when('/jobProfile/:id', {
  templateUrl: '/jobProfile/jobProfile.view.html',
  controller: 'jobProfileCtrl',
  controllerAs: 'vm'
})
```

- /jobProfile route calls 'jobprofileCtrl' which is a job profile controller. A job profile controller fetches a job profile by making an HTTP request with job id to the server to fetch particular job profile.

```
$http.get('/job/' + $routeParams.id).success(function(data) {
  vm.job = data;
});
```

- A user can add personal details, a GitHub profile URL and upload resume from local machine. Once the user clicks on Submit application, the controller fetches user details from the GitHub public API to get users' repositories and to find contributors from each repository.

```
$.ajax({
  url: 'https://api.github.com/users/'+ splits[1] + '/repos',
  type: 'GET',
  success: function(data){
    for(var i=0; i<data.length;i++){
      $.ajax({
        url: data[i].contributors_url,
        type: 'GET',
        success: function(contributers){
          var result = $filter('filter')(contributers, {login: splits[1]});
          if(result.length == 1){
            for(var j=0; j<contributers.length; j++){
              if(finalConstributers.indexOf(contributers[j].login) == -1){
                finalConstributers.push(contributers[j].login);
              }
            }
          }
        }});
    }
  }});
```

- Once the GitHub data is fetched, the controller makes an HTTP request to the server to apply for the job and store the user resume on mongoDB

```
var application = {
  jobID                 : vm.job._id,
  position              : vm.job.position,
  applicantName         : vm.user.name,
  applicantID           : vm.user._id,
  applicantEmail        : vm.user.email,
  applicantGithubID     : splits[1],
  applicantLocation     : vm.user.location,
  applicantSkills       : vm.user.skills,
  applicationStatus     : "Submitted",
  applicantGithubConnections : finalConstributers
};
$.ajax({
  url: '/upload/' + vm.job._id,
  type: 'POST',
  data: fd,
  processData: false,
  contentType: false,
  success: function(data){
    if(data == "We already have your application!"){
      alert(data);
    }
    else{
      application.resumeID = data;
      $http.post('/apply', application).success(function(data) {
        alert(data);
      });
    }
  }});
```

```
router.post('/apply', function(req, res) {
  var application = new Application();
  application.jobID = req.body.jobID;
  application.position = req.body.position;
  application.applicantName = req.body.applicantName;
  application.applicantID = req.body.applicantID;
  application.applicantGithubID = req.body.applicantGithubID;
  application.applicantEmail = req.body.applicantEmail;
  application.applicantLocation = req.body.applicantLocation;
  application.applicantSkills = req.body.applicantSkills;
  application.applicationStatus = req.body.applicationStatus;
  application.resumeID          = req.body.resumeID;
  application.applicantGithubConnections      = req.body.applicantGithubConnections;
  application.save(function(err) {
    if(err) {
      console.log(err);
      throw err;
    }
    console.log('New Application: ' + application + ' created!');
    res.send("Your application is submited!");
  });
});
```

5. HR Profile Screen:

The HR profile home page displays job applications, details about applications for a particular team, and job seekers' names.

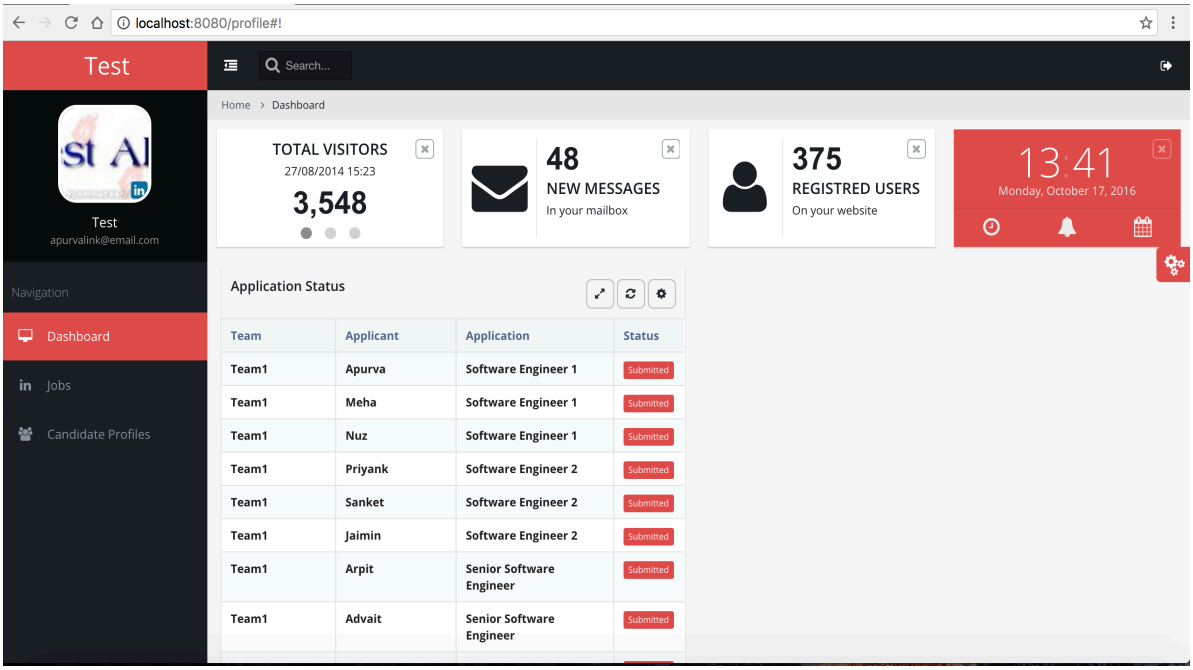


Figure 38: HR Profile Screen

- The profile controller makes an HTTP call to the server to get all the applications from the database. The following code snippet shows how http request is made:

```
(function() {

  angular
  .module('meanApp')
  .controller('profileCtrl', profileCtrl);

  profileCtrl.$inject = ['$location','$http','$routeParams','$window'];
  function profileCtrl($location,$http,$routeParams,$window) {
    var vm = this;
    vm.user = {};
    var user = $window.localStorage['username'];

    $http.get('/user').success(function(data) {
      vm.user = data;
      $http.get('/application').success(function(data) {
        vm.application = data;
      });
    });
```

6. HR Post Job Screen:

HR can post a job for the team and submit the details:

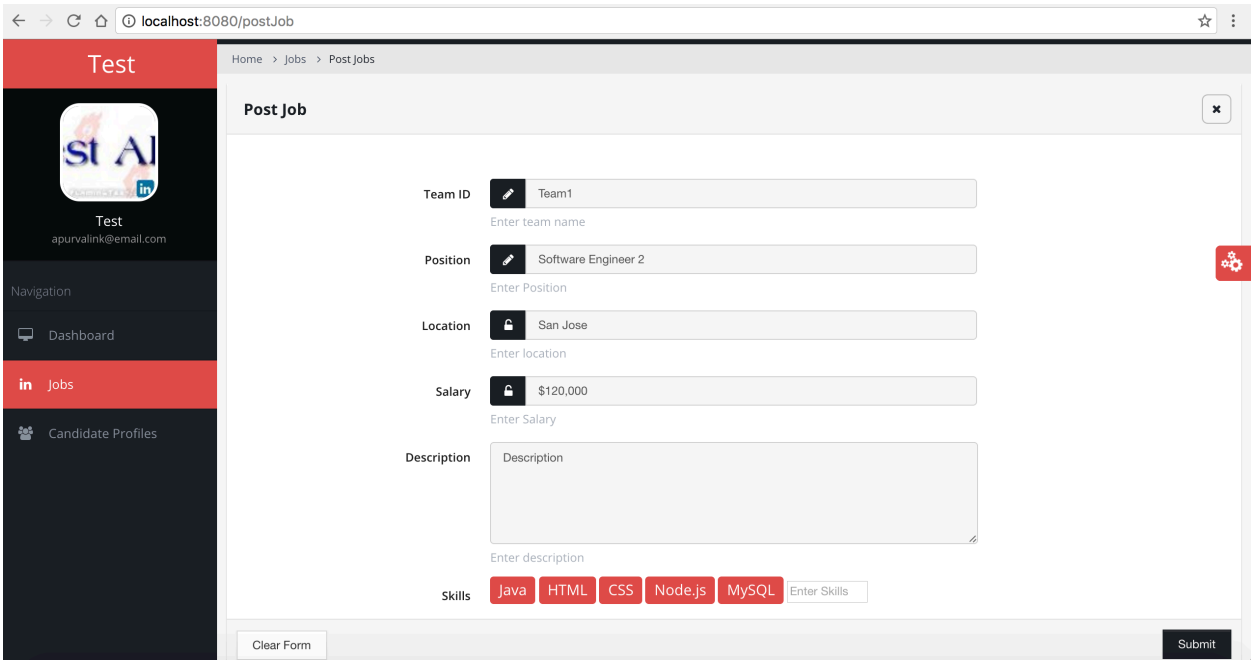


Figure 39: HR Post Job Screen

- Once HR adds all the details and submits the form, the post job controller makes a HTTP request to the server to store the details in the jobs collections.

```
vm.onSubmit = function(){
  var job = {
    teamID                : $scope.team,
    position              : $scope.position,
    description           : $scope.description,
    salary                : $scope.salary,
    location              : $scope.location,
    skills                : $("#tags2").tagsinput('items')
  };

  var team = {
    teamID                : $scope.team,
    position             : $scope.position
  };

  $http.post('/job', job).success(function(data) {
    alert("Job Posted!");
  });
  $http.post('/team', team).success(function(data) {
    console.log("Team stored!");
  });
}
```

7 View Candidate Profile Screen:

HR can view all the profiles of the candidates by clicking on the Candidate profile tab:

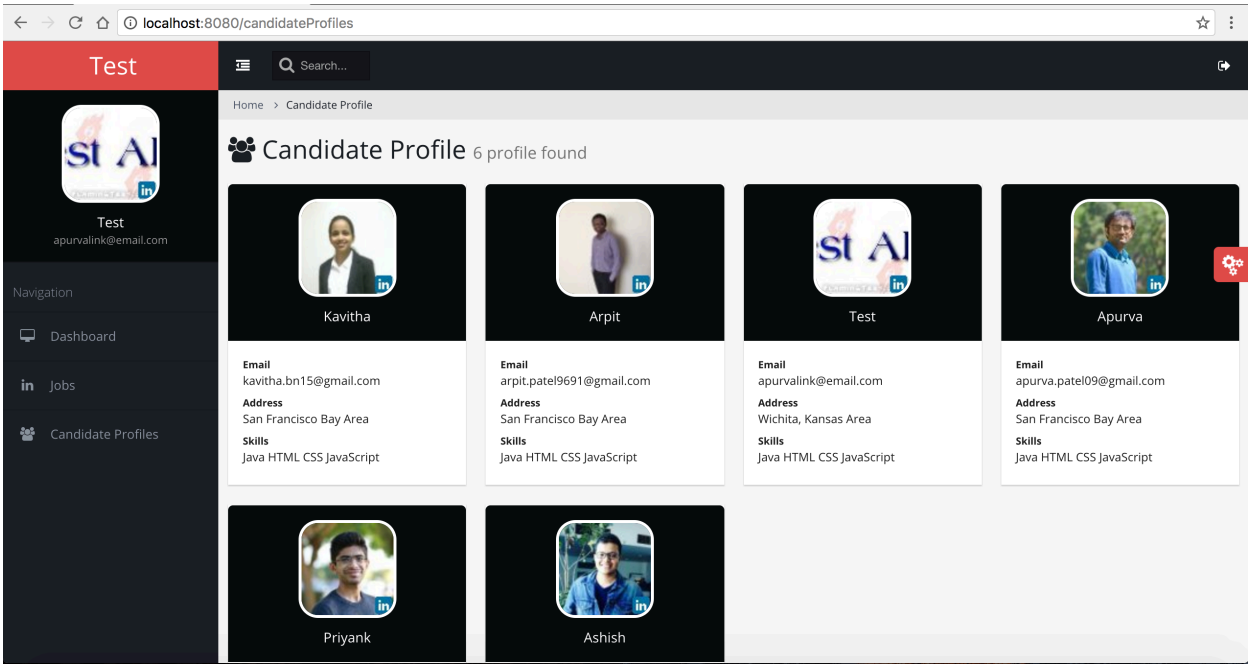


Figure 40: View Candidate Profile Screen

8 View Recommended Teams:

HR can select weights for different parameter and generate recommendations for a specific team.

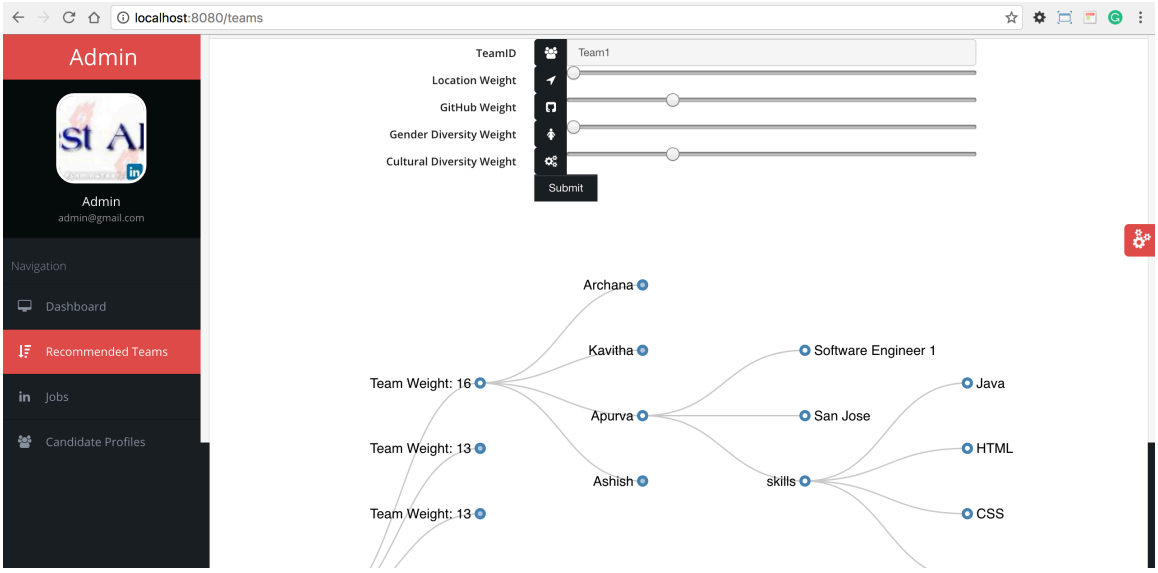


Figure 41:Recommended Teams Screen

- Once HR selects weights for the parameters and clicks on submit, the team controller makes an HTTP request to the server to generate a recommendation. Once the recommendation is generated, it is displayed in the screen using D3.js (D3.js is JavaScript based library to generate visual effects based on the data).

Middle-Tier Implementation

1. Backend API

The backend API is based on a service-oriented architecture where REST APIs are exposed to the client to fetch and store the data into the database. This section explains the RESTful services implemented in the project. REST APIs are created using a node-restful npm package which

allows us to create a common API for CRUD (Create, Read, Update and Delete) operations. Npm is a Node Package Manager used to add node programs or dependencies in the project. The backend application server is implemented using Node.js and the Express framework.

Job API

This API is used to post, update, get or delete a job from the mongoDB.
Request type: GET, PUT, POST, DELETE,
REST endpoint: /job

Application API

This API is used to post, update, get or delete an application from the mongoDB.
Request type: GET, PUT, POST, DELETE,
REST endpoint: /application

User API

This API is used to post, update, get or delete a user from the mongoDB.
Request type: GET, PUT, POST, DELETE,
REST endpoint: /user

```
Jobs.methods(['get', 'put', 'post', 'delete']);
Jobs.register(router, '/job');

Application.methods(['get', 'put', 'post', 'delete']);
Application.register(router, '/application');

User.methods(['get', 'put', 'post', 'delete']);
User.register(router, '/user');
```

2. Authentication API

A user is authenticated using LinkedIn OAuth. We used the user passport-linkedin-oauth2 npm package to authenticate a user. A user request for authentication is redirected to the LinkedIn

OAuth page where the user enters a username and a password. The LinkedIn OAuth sends an access token to our application which we store in our database to authenticate the user and make requests on behalf of the user.

LinkedIn API

This API redirect user to the LinkedIn OAuth page.

REST endpoint: /auth/linkedin

LinkedIn Callback API

This API calls the passport service once the LinkedIn OAuth redirects the user back to our application.

REST endpoint: /auth/linkedin/callback

```
// send to linkedin to do the authentication
app.get('/auth/linkedin', passport.authenticate('linkedin',{ scope: ['r_basicprofile', 'r_emailaddress'] }));

// handle the callback after linkedin has authenticated the user
app.get('/auth/linkedin/callback',
passport.authenticate('linkedin', {
  successRedirect : '/profile',
  failureRedirect : '/'
}));
```

3. Knowledge competence

Knowledge competence is a process of finding skill score of a candidate which indicates how strong a candidate is for the applied job based on the required skills of the job. The range of the skill score is between 0 to 1, where 1 indicates that the candidate does have all the required skills.

```
var filePath = path.join(__dirname, '../uploads/'+req.body.fileName);

extract(filePath, { splitPages: false }, function (err, text) {
  if (err) {
    console.dir(err)
    return
  }
  var skillScore = 0;
  var applicantSkills = [];
  var totalSkills = req.body.requiredSkills.length;
  for(var i in req.body.requiredSkills){
    if(text.toString().toLowerCase().includes(req.body.requiredSkills[i].toLowerCase())){
      applicantSkills.push(req.body.requiredSkills[i]);
      skillScore++;
    }
  }
  application.applicantSkills = applicantSkills;
  application.skillScore = skillScore/totalSkills;
  application.save(function(err) {
    if(err) {
      console.log(err);
      throw err;
    }
```

Once candidates apply with their resume, we parse the resume in a text format. Then we find key words from that text and increase the skill score if the required key word is in the resume. Once the skill score is computed, it is stored along with the application data.

4. Recommendation logic

Once a hiring manager submits a request from the recommendation page by selecting weights for the different parameters, the request goes to the '/recom' router on the server side.

```
router.get('/recom/:id/:locS/:gitS/:divS/:gdivS', function(req, res){
  var gitS = req.params.gitS;
  var locS = req.params.locS;
  var divS = req.params.divS;
  var gdivS = req.params.gdivS;

  Team.findOne({"teamID": req.params.id },function(err,team){

    var allArrays = [];
    var finlResult = [];
    var numberOfApplications = [];
    for(var i = 0; i < team.positions.length; i++){
      var pos = [];

      Application.find({"position":team.positions[i],"teamID": req.params.id,"skillScore":1},function(err,applications){
        pos = applications;
        allArrays.push(pos);
        numberOfApplications.push(pos.length);
      });
    }
```

The server finds the appropriate applications with skillScore = 1, which indicates that the applicant does have all the required skills in his/her resume, and it generates all possible combinations of the applicants for the posted jobs in a team.

```
        function allPossibleCases(arr) {
          if (arr.length == 1) {
            return arr[0];
          } else {
            var result = [];
            var allCasesOfRest = allPossibleCases(arr.slice(1));  // recur with the rest of array
            for (var i = 0; i < allCasesOfRest.length; i++) {
              for (var j = 0; j < arr[0].length; j++) {
                result.push('/' + JSON.stringify(arr[0][j]) + allCasesOfRest[i]);
              }
            }
            return result;
          }
        }

        for (var i=0 ; i< results.length; i ++){
          var string = results[i].split('/');
          var final = string.slice(1,string.length - 1);
          rankTeamBasedOnUserDetails(final);
        }
```

allPossibleCases() function generates combinations and pass that combinations to rankTeamBasedOnUserDetails() function.

```
function rankTeamBasedOnUserDetails(arr){
  var locationWeight = 0;
  var gitWeight = 0;
  var diversityWeight = 0;
  var genderDiversityWeight = 0;

  for(var i = 0; i< arr.length; i++){
    for (var j = i + 1; j < arr.length; j++){
      var obj1 = JSON.parse(arr[i]);
      var obj2 = JSON.parse(arr[j]);
      if(obj1.applicantLocation == obj2.applicantLocation){
        locationWeight = locationWeight + parseInt(locS) ;
      }
      if(obj2.applicantGithubConnections.indexOf(obj1.applicantGithubID) >= 0){
        gitWeight = gitWeight + parseInt(gitS);
      }
    }
  }
  var map = new Object();
  var gmap = new Object();

  for (var h = 0; h< arr.length; h++){
    var object = JSON.parse(arr[h]);
    if(map[object.applicantRace]){
      map[object.applicantRace]++;
    }
    else{
      map[object.applicantRace] = 1;
    }
    if(gmap[object.applicantGender]){
      gmap[object.applicantGender]++;
    }
```

rankTeamBasedOnUserDetails() function finds similarity between applicants by comparing their location, GitHub connections. It generates the weight of the team based on the weight parameters set by the hiring manager.

5. Recommendation based on diversity in the team

To recommend a team with gender and cultural diversity, we add weight to the team according to team members gender and their race.

```
var map = new Object();
var gmap = new Object();

for (var h = 0; h< arr.length; h++){
  var object = JSON.parse(arr[h]);
  if(map[object.applicantRace]){
    map[object.applicantRace]++;
  }
  else{
    map[object.applicantRace] = 1;
  }
  if(gmap[object.applicantGender]){
    gmap[object.applicantGender]++;
  }
  else{
    gmap[object.applicantGender] = 1;
  }
}
for(var val in map){
  diversityWeight = diversityWeight + parseInt(divS);
}

for(var val in gmap){
  genderDiversityWeight = genderDiversityWeight + parseInt(gdivS);
}
```

We generate a hash map to find how many different races are there with in the generated combination and we add the weight for every different race. The higher the number of races, the higher the weight of the team. Likewise, we generate weight for gender diversity.

Data-Tier Implementation

1. Database Collections

We have used MongoDB for data storage. REST endpoints store data to different Mongo collections based on application logic. Details about the collections are below:

User Collection:

This collection stores a user's basic profile details once the user is authenticated through LinkedIn. Profile details are fetched from user's LinkedIn profile.


```
{
  "_id": "dNxo1R-Yo-",
  "picUrl": "https://media.licdn.com/m
pnoHNs6ORLamtWosKRfVQR3NwaHcv",
  "location": "Wichita, Kansas Area",
  "email": "apurvalink@email.com",
  "name": "Test",
  "token": "AQXM1pepPsZERhmumqBJSD5KyM
XiI9l42VRWW9P8G7AG1QWbywya4VsSe-E",
  "username": "dNxo1R-Yo-",
  "skills": [
    "Java",
    "HTML",
    "CSS",
    "JavaScript"
  ],
  "__v": 0
}
```

Job Collection:

This collection stores different jobs posted by HR for team formation. Details include basic requirements and job descriptions.


```
{
  "_id": ObjectId("5803c552f8f7c495529c75b7"),
  "teamID": "Team2",
  "position": "Software Engineer 1",
  "description": "MS in Software Engineering. 1 year Professional experience
  "salary": "$110,000",
  "location": "San Jose",
  "skills": [
    "Java",
    "HTML",
    "CSS",
    "JavaScript"
  ],
  "__v": 0
}
```

Application Collection:

This collection stores a user's application for a particular job. It includes the user's basic details, skills fetched data from GitHub and resume details


```
{
  "_id": ObjectId("5829feeb4f55bd99882f26c7"),
  "resumeID": "5803f1b4db21e8add15214a2",
  "skillScore": 1,
  "applicationStatus": "Submitted",
  "applicantLocation": "San Francisco",
  "applicantRace": "Asian",
  "applicantEmail": "priyank14@email.com",
  "applicantGithubID": "priyank14",
  "applicantID": "dNxo1R-Yo-",
  "applicantName": "Priyank",
  "applicantGender": "Male",
  "teamID": "Team1",
  "position": "Senior Software Engineer",
  "jobID": "57f825be89df9c81888d7df7",
  "applicantGithubConnections": [
    "kavitha14",
    "archana14"
  ],
  "applicantSkills": [
    "Java",
    "HTML",
    "CSS",
    "JavaScript"
  ]
```

Team Collection:

This collection stores details about different positions for a particular team posted by HR.


```
{
  "_id": ObjectId("5803eb6c4ae25ca796ffa5a8"),
  "teamID": "Team1",
  "positions": [
    "Software Engineer 1",
    "Software Engineer 2",
    "Software Engineer in Test",
    "Senior Software Engineer"
  ],
  "__v": 0
}
```


Attachments.file Collection:

This collection stores meta data for the resume uploaded on the database


```
{
  "_id": ObjectId("582b49ca427a413cb10539f2"),
  "filename": "Apurva_Resume.pdf",
  "contentType": "application/pdf",
  "length": 53668,
  "chunkSize": 261120,
  "uploadDate": ISODate("2016-11-15T17:45:46.934Z"),
  "aliases": [ ],
  "metadata": {
    "jobID": "Team2",
    "applicantID": "iaMfkFePve"
  },
  "md5": "fec6cd1576606c9af6e764e9bb8a838b"
}
```

Attachments.chunks Collection:

This collection store the resume in BSON format (It is a data interchange format used mainly as a data storage and network transfer format in MongoDB)

Chapter 6. Testing and Verification

1. Project Test Plan

The test plan comprises the purpose, scope, schedule, approach, methods, models, and strategy required to test the project.

We performed various types of testing during each phase of software development.

Below are the some of the tests:

- Unit Testing
- Integration Testing
- System Testing

Testing Module	Scope
Web UI	Testing of the user interface of the web applications.
Back-end	White box testing of each of the functional component developed
Integration	Once each component is developed, it is integrated and tested.
System	The enter system is tested after the development and integration of all the components.

1. Unit Testing

In unit testing each individual component is tested for correctness in the development environment.

The table below describes in detail the unit test execution criteria

Testing Criteria	Methods
Entry	All the individual functionality should be completed Data should be valid and available The test plan should be well defined
Suspension	If there is a change in the requirement, then we need to recreate the test plan.
Exit	The unit test is complete and components are working as designed in the wireframe.

The table below shows the unit test cases for the project

Test Case Id	UT_01
Test Case Description	Check if the login API is working and Job seeker is able to login
Expected Results	Job Seeker should be able to login with the linkedin credentials
Actual Results	Job Seeker can login with his linkedin credentials
Status	Passed

Test Case Id	UT_02
Test Case Description	Should be able to connect to the Database and perform insert operation if the job seeker is using the application for the first time
Expected Results	Should perform insert record for new users
Actual Results	Can insert record in the DB
Status	Passed

Test Case Id	UT_03
Test Case Description	Check if the job seekers past applications are loaded from the database
Expected Results	Job seeker should be able to see the list of jobs applied
Actual Results	All the results are fetched from the database and posted on the dashboard
Status	Passed

Test Case Id	UT_04
Test Case Description	Check if the jobs API is working
Expected Results	All the jobs posted must be fetched from the database

Actual Results	All the posted jobs are fetched from DB and results are displayed on the dashboard
Status	Passed

Test Case Id	UT_05
Test Case Description	Check if each job can get the job description
Expected Results	On clicking the job name, we should be able to fetch the job profile
Actual Results	Detailed job profile description is visible on clicking the job name.
Status	Passed

Test Case Id	UT_06
Test Case Description	Check if GitHub public API is working
Expected Results	On giving the GitHub URL and submitting the job. GitHub public API should be able to fetch common contributions
Actual Results	We are able to get the contribution list
Status	Passed

Test Case Id	UT_07
Test Case Description	Check if all the application data if fetched from the database
Expected Results	We should be able to get the list of all candidates applied for the job saved in DB
Actual Results	We are able to get the list of candidates applied for the job
Status	Passed

Test Case Id	UT_08
Test Case Description	Check if job post API is working
Expected Results	The job posted by the HR should be saved in the DB.
Actual Results	All the posted jobs are saved.
Status	Passed

2. Integration Testing

Various components are integrated and tested together in the development environment.
The table below describes in detail the Integration test execution criteria.

Testing Criteria	Methods
Entry	All the individual functionality should be completed and integrated. Data should be valid and available. The test plan should be well defined. Unit tests should be completed.
Suspension	If there is a change in the requirement, then we need to recreate the test plan.
Exit	The integration test is complete and the components are working as the flow defined in the wireframes.

The table below shows the integration test cases for the project.

Test Case Id	IT_01
Test Case Description	Integrate all the Web pages
Expected Results	The flow and the display of the web pages should be correct
Actual Results	The flow of the application module works correctly after integration
Status	Passed

3. System Testing

All the components are integrated and entire system is tested for functionality against the requirement in the development.
The table below describes in detail the system test execution criteria.

Testing Criteria	Methods
Entry	All the individual functionality should be completed. Data should be valid and available. The test plan should be well defined Integration tests should be completed. Requirements should be well defined and complete.
Suspension	If there is a change in the requirement, then we need to recreate the test plan.
Exit	The system test is complete and the system is working as per the flow defined in the wireframes.

The table below shows the system test cases for the project

Test Case Id	ST_01
Test Case Description	Check the flow of User Portal
Expected Results	All the web pages should be displayed on each tab clicks
Actual Results	All the pages are displayed and the flow is from login, View jobs, apply jobs
Status	Passed

Test Case Id	ST_02
Test Case Description	Check the flow of HR Portal
Expected Results	All the web pages should be displayed on each tab clicks

Actual Results	All the pages are displayed and flow is from login, view candidates, select weights and view graph
Status	Passed

4. User Interface Testing

The system is tested to identify the defect in GUI in the development phase.

The table below describes in detail the UI test execution criteria

Testing Criteria	Methods
Entry	All the individual functionality should be completed. Data should be valid and available. The test plan should be well defined.
Suspension	Change in the requirement.
Exit	All the UI tests are approved and validated

The table below shows the UI test cases for the project

Test Case Id	UI_01
Test Case Description	Check if all the pages and layout are properly aligned
Expected Results	Enter values and test the page flow. Page flow should be correct
Actual Results	All the pages are properly aligned and displayed
Status	Passed

Test Case Id	UI_02
Test Case Description	Check if all the navigation buttons are working correctly

Expected Results	All buttons should work correctly
Actual Results	Buttons are working and necessary action like submit is performed.
Status	Passed

Test Case Id	UI_03
Test Case Description	Job Seeker should be able to log in with LinkedIn credentials
Expected Results	User Should successfully log in
Actual Results	User can login
Status	Passed

Test Case Id	UI_04
Test Case Description	After user login he should be able to see home page with list of submitted jobs and all his skills mentioned on linked in.
Expected Results	User should be able to see the skills and applied jobs.
Actual Results	We can see the jobs applied and skills.
Status	Passed

Test Case Id	UI_05
Test Case Description	User should be able to see the list of jobs in the jobs page
Expected Results	Can see all the jobs listed for various positions

Actual Results	Can see the jobs for all positions
Status	Passed

Test Case Id	UI_06
Test Case Description	For each job, the user should be able to see the job description and should be able to apply for jobs.
Expected Results	Should be able to submit jobs
Actual Results	Can apply for jobs and these applications are saved in MongoDB
Status	Passed

Test Case Id	UI_07
Test Case Description	HR should be able to see the list of applied candidates
Expected Results	Should display all applied candidates on dashboard
Actual Results	Results show the names of candidates and positions applied for.
Status	Passed

Test Case Id	UI_08
Test Case Description	HR should be able to post jobs and add skills.
Expected Results	Should be able to submit jobs.
Actual Results	All jobs can be submitted.
Status	Passed

Test Case Id	UI_09
Test Case Description	HR should be able to view the candidate profiles.
Expected Results	Should be able to view candidate details

Actual Results	Can view all candidate details
Status	Passed

Test Case Id	UI_10
Test Case Description	HR should be able to distribute weights for diversity, skill, location
Expected Results	Should be able to select the weights and view candidates based on the weights applied
Actual Results	Can see the team results based on weight selection
Status	Passed

Test Case Id	UI_11
Test Case Description	Should be able to plot graph based on the weights chosen
Expected Results	Graph should be displayed based on the result
Actual Results	Can see the graph with team details and weightage
Status	Passed

Chapter 7. Performance and Benchmarks

Performance

It is vital for a web application to be tested for responsiveness in view of stability and check how it behaves for various workloads. One way to check performance is by doing load testing. This helps us check how the system behaves for various loads of users who might be accessing the application at the same time. Using certain tools, we can do performance testing. They offer us some metrics by which we can predict how our system will behave.

In our project, we have considered performance testing for checking the load, stress, and latency time of the web application. Some of the features which need performance testing are the user and the HR portal where multiple users can log in at the same time to create and access their profiles also to apply for specific jobs of interest. Similar is the case of the HR portal where many HR departments can log in at the same time and send their requirements.

Tools and Technologies for evaluating performance and benchmark

Apache JMeter

It's used for load testing to analyze and measure the performance. We have used Eclipse to generate the request and this request is then feed to JMeter to check the latency for multiple requests at the same time.

Load balancing tool

We are using elastic load balancing to distribute the incoming traffic across multiple zones available. It makes our application available all time. We will be able to add or remove load balancers per our need without disturbing the request flow within the system. Elastic load balancing can work with another service like Amazon EC2 to move the application to the cloud, which improves the scalability and availability of web application.

Benchmarks

We have certain benchmarks to evaluate the project for it to work accurately.

- The algorithm should be able to detect best team under a given set of conditions like skills, location, education, and GitHub contributions.

Chapter 8. Deployment, Operations, Maintenance

Deployment

The web application is deployed on AWS EC2 for better scalability and availability.
Here are some of the components that are deployed in the cloud for successful execution of the web application.

Redis:

We used Redis for data caching. Redis is an in-memory data structure store used for caching, message brokering, and database. It has many data structures including strings, hashes, sets, lists, etc. We need to install redis manually on the AWS EC2 instance.

MongoDB:

MongoDB is a document-oriented database which we used in our application to store data. It provides many features like replication, load balancing, file storage, aggregation, indexing, etc. A MongoDB cluster will make use of the Amazon Elastic Cloud computing with replica sets. It provides three replica sets, primary, secondary0, secondary1, to increase availability of nodes. All the read and write operations are done at the primary. It also provides the flexibility to read from the secondary node but writes are always done at the primary node.

AWS EC2:

It is a web service that provides the computational capacity in the cloud to make the web application scalable. It helps increase or decrease the capacity within minutes. It provides the capacity to have thousands of servers simultaneously. Amazon EC2 works with a virtual private cloud with the specified IP range. It helps us decide which instance has to be exposed on the internet and which will remain private.

Operations and Maintenance

Our application components are loosely coupled, so that deploying and debugging the application are easy, and we can scale the application as necessary. Our application is based on the MEAN stack so maintaining the dependencies is easy through the package.json file.

Chapter 9. Summary, Conclusions, and Recommendations

Summary

Hiring a candidate and building a team from the hired candidates (or integrating the candidates with an existing team) is a very complex but intrinsic process for development success. In the wake of emerging online user activities and digital footprint to follow, it is possible to build a virtual candidate profile from various social network sites. Team formation is a well-known problem in the operational research field. There has been much research, and multiple solutions have been proposed over the years, to answer various facets of this problem. But most of these solutions focus largely on solving the cost metric in building a team (monetary or task distribution wise), and completely ignore the behavioral aspect of team candidates.

In our project, we attempt to build the most suitable teams (team formation problem is a known NP-hard problem, and has proximate solutions) for a provided team-requirements template. We considered various parameters like skill, publications, performance, experience etc., to pick domain experts and candidates. In addition, we used social network and graph results to find team members who are socially close.

Furthermore, we are also considering the diversity aspect in finalizing the team. Diversity parameters such as race, gender, active/veteran military duty, disability etc. are important factors that need to be considered. The diversity threshold is designed by the team-profile template and is largely controlled by the employer requesting the team.

Finally, the idea of using online social profile to build a right team is great, but there are certain legal caveats. We have been mindful of not violating any privacy or breaking any usage policy. Most of the user data gets barricaded by policy usage, and can hinder the system prediction accuracy. GitHub data is our primary source of open social data. We also query LinkedIn profiles (to some degree). The more data we can amalgamate, the better the system can do.

Conclusion

Our solution differs from a traditional job portal, which generally has job listings, candidate's profiles and interaction/networking modules. It takes a novel and a global approach to fulfil the entire team hiring requirements. We integrated data-visualization modules in the system portal. Data visualization is a great way to remove machine-recommendation anxiety, and helps to generate much transparent results with self-justifications (why the system thinks that this team would work well). This system can be integrated easily with any job portal or social network, for example LinkedIn.

Recommendation

Although the system answers the problem very well and establishes the proof of concept, we believe there is a great scope on improving three aspects:

Interactive data-visualization

Candidates' connected graphs give good details about their connectivity weight. Once the results are out, adding a more interactive feature such a Facebook/GitHub/LinkedIn profile links to quickly get in touch with the candidates or emailing the entire team members about their candidacy for the job process.

More data source input

Resume, GitHub or portal profile give the greater bulk of information about the candidates. But they alone can be inconclusive if the candidate hasn't filled information properly or has no social networking account. Forcing a candidate to supply the required information is not a complete solution, but more interaction with social networking sites can help to answer more general team hiring (not restricted to software/computer field).

Improving algorithms

Improving the solution and system performance is an ever evolving process. Calculating task distribution between candidates (equal workload), so that no one feels overwhelmed and

suggesting if more candidates are needed to complete the task (2-3 software engineers instead of just one?) is an avenue of research and implementation.

Glossary

API	Application Program Interface
App	Application
Activity	User's interaction with the system
Classifier	Grouping candidates (candidate pool) in different clusters
Session	Time period to perform the activity or Job availability period
UI	User Interface
Apache Spark	Framework to do in-memory data computation
Neo4J	Graph based database

References

[1] M. Ridwan, "Predicting Likes: Inside A Simple Recommendation Engine's Algorithms," *Toptal Engineering Blog*. [Online]. Available: https://www.toptal.com/algorithms/predicting-likes-inside-a-simple-recommendation-engine. [Accessed: 12-May-2016].

[2] D. Asanov, "Algorithms and methods in recommender systems," *Berlin Institute of Technology, Berlin, Germany*, 2011.

[3] Nagaraju K., Bende A., Patel A. & Taparia A. "Talent Acquisition System-TeamNike-Abstract". [Workbook document], March 2016.

[4] G. K. Awal and K. K. Bharadwaj, "Team formation in social networks based on collective intelligence – an evolutionary approach," *Applied Intelligence*, vol. 41, no. 2, pp. 627–648, Sep. 2014.

[5] T. De Pessemier, S. Dooms, and L. Martens, "An improved data aggregation strategy for group recommendations," in *3rd Workshop on Human Decision Making in Recommender Systems (Decisions@ RecSys 2013), held in conjunction with the 7th ACM Conference on Recommender Systems (RecSys 2013)*, 2013, vol. 1050, pp. 36–39.

[6] G. Beliakov, T. Calvo, and S. James, "Aggregation of Preferences in Recommender Systems," in *Recommender Systems Handbook*, F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, Eds. Springer US, 2011, pp. 705–734.

[7] C. Sofia Pereira and A. L. Soares, "Improving the quality of collaboration requirements for information management through social networks analysis," *International Journal of Information Management*, vol. 27, no. 2, pp. 86–103, Apr. 2007.

[8] M. Roth, A. Ben-David, D. Deutscher, G. Flysher, I. Horn, A. Leichtberg, N. Leiser, Y. Matias, and R. Merom, "Suggesting friends using the implicit social graph," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2010, pp. 233–242.

[9] E. Rahm and H. H. Do, "Data cleaning: Problems and current approaches," *IEEE Data Eng. Bull.*, vol. 23, no. 4, pp. 3–13, 2000.

[10] V. Agarwal and K. K. Bharadwaj, "A collaborative filtering framework for friends recommendation in social networks based on interaction intensity and adaptive user similarity," *Social Network Analysis and Mining*, vol. 3, no. 3, pp. 359–379, Sep. 2013.

[11] A. Farasat and A. G. Nikolaev, "Social structure optimization in team formation," *Computers & Operations Research*, vol. 74, pp. 127–142, Oct. 2016.

[12] T. Dong-Huynha, N. Jennings, and N. Shadbolt, “FIRE: An integrated trust and reputation model for open multi-agent systems,” in *16th European Conference on Artificial Intelligence, Valencia, Spain*, 2004, pp. 18–22.

[13] “Simple Example of MVC (Model View Controller) Design Pattern for Abstraction - CodeProject.” [Online]. Available: http://www.codeproject.com/Articles/25057/Simple-Example-of-MVC-Model-View-Controller-Design. [Accessed: 12-May-2016].

[14] “trinket: model view controller pattern.” [Online]. Available: http://cristobal.baray.com/indiana/projects/mvc2.html. [Accessed: 12-May-2016].

[15] A. Majumder, S. Datta, and K. V. M. Naidu, “Capacitated Team Formation Problem on Social Networks,” in Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA, 2012, pp. 1005–1013.