

## **CMPE 273: Enterprise Distributed Systems**

### **Class Project (Fall 2021) - Simulation of Indeed.com**



**Team size:** 6-7 people per team  
**Due date:** December 3rd, 2021

1. **API Documentation:** October 29<sup>th</sup>, 2021
  - Submit your API report document on Canvas before 11:59 PM.
  - You must turn in a document describing the interface of your backend server consisting of all the API request-response descriptions.
2. **Presentation:** November 19<sup>th</sup>, 2021
  - Submit your presentation document on Canvas before 3:00 PM
  - Group-wise presentation should be given in the class meeting.
  - Describe the system architecture used in your project.
3. **Project Demo:** December 3<sup>rd</sup>, 2021
  - Submit the project report before 3:00 PM.
  - Demo of your application in a class meeting.
4. **Peer Review:** December 7<sup>th</sup>, 2021
  - Submit the peer review document on Canvas before 11.59 PM.
  - Grade your team members except yourself on a scale of 10 and describe their contributions to the project.
  - This will be an individual submission and should keep the grades and comments confidential.

## **Indeed.com**

Indeed is an American worldwide employment website for job listings. The site aggregates job listings from thousands of websites, including job boards, staffing firms, associations, and company career pages. They allow companies for premium job postings and get the featured resumes. It allows job seekers to apply to the jobs directly.

### **Project Overview:**

In this project, you will design a 3-tier distributed application that will implement the functions of the Indeed website. You will use the techniques used in your lab assignments to create the system. In the Indeed prototype, you will manage and implement different types of users.

Though Indeed.com has a wide range of features, this project is limited to only a few important functionalities that involve writing reviews anonymously, post jobs, apply for jobs, reveal salaries tagged to a post, and implementing the functionalities that involve different roles.

For each type of object, you will also need to implement an associated database schema that will be responsible for representing how a given object should be stored in a database.

Your system should be built upon some type of distributed architecture. You have to use message queues as the middleware technology. You will be given a great deal of “artistic liberty” in how you choose to implement the system.

Your system must support the following types of entities:

- a. Job Seeker
- b. Employer
- c. Admin users

Your project will consist of three tiers:

- The client tier, where the user will interact with your system
- The middle tier/middleware/messaging system, where the majority of the processing takes place
- The third tier, comprising a database to store the state of your entity objects

## Tier 1 – Client Requirements

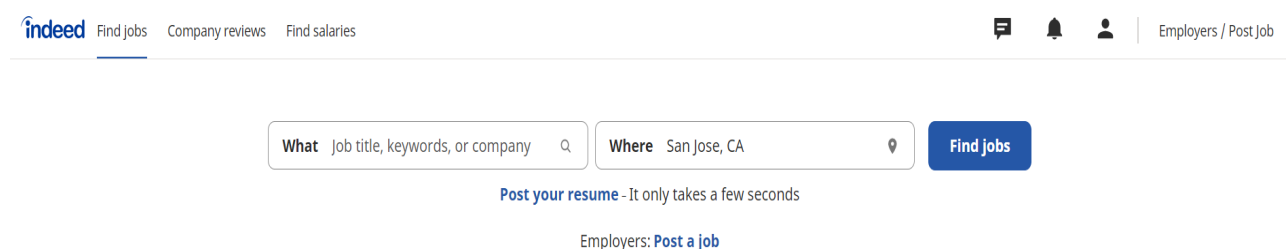
The client should have a pleasing look and be simple to understand. Error conditions and feedback to the user should be displayed in a status area on the user interface. Try to create a UI similar to Indeed.

Your application should allow users to search for jobs but on trying to apply for a job, it should ask a user to create an account first.

### Sign in & Sign up

- The user should be able to create a new account by providing his Email Address and Password.
- A user can sign up either as an Employer or a Job Seeker.
- The user should be able to log in using his email id and password.
- There should be a single login page for Job Seeker, Employer, and Admin.
- You can create credentials for admin from the backend.
- Based on their account type, they should be redirected to their respective pages.
- There is no need to implement the email verification functionality.

## Find jobs (Landing Page for Job Seekers)



The screenshot shows the Indeed job search interface. At the top, there's a navigation bar with the Indeed logo and links for 'Find jobs', 'Company reviews', and 'Find salaries'. On the right, there are icons for chat, notifications, and a user profile, followed by a link to 'Employers / Post Job'. Below the navigation bar is a search section with two input fields: 'What' (for job title, keywords, or company) and 'Where' (for location, currently set to 'San Jose, CA'). A blue 'Find jobs' button is to the right of these fields. Below the search fields, there's a link to 'Post your resume' with the text 'It only takes a few seconds'. At the bottom of the search section, there's a link for 'Employers: Post a job'.

- Job Seeker should be able to search for a job based on Job Title.(Type 3 alphabets and it should show the suggestions based on 3 alphabets)
- Job Seekers should be able to search for a job by entering a company's name.
- The location filter should help in searching for jobs based on those locations.
- The same page should allow a job seeker to post their resume.

### Job search by Company Name

- Job Seekers should be shown a list of job cards in the company.
- The following information should be displayed on all the job cards
  - o Role name

- o Company Name (should be a hyperlink pointing to the company's homepage within indeed)
    - o Rating
    - o Location (City, State)
    - o Salary details
    - o A glimpse of the job description
  - On selecting a job card, it should display the below details about the job:
    - o **Job Role**
      - Company and role name
      - Average Rating (can be based on the company rating)
      - Number of reviews
      - Location(City, State, ZIP)
      - Button to apply for the job (Should allow submitting resume)
      - Type of job (Full-time/ Part-time/ Remote)
      - A save button to save the job
      - A button to undo the save action for a job (visible only after saving a job)
    - o **Job details**
      - Salary Details
      - Job Type
    - o **Full Job Description**
      - This field will include the job details and complete description including
        - Location
        - Compensation
        - What You'll Do
        - Why You'll love working for a <job role>
        - What You'll Need
  - Pagination should be used to display search results. (Server-side recommended for better performance).
  - Click on the company name to open "Company Home Page"
- 

## Company Home Page

- There will be 6 sub-tabs on the company's page:
  - o Company Snapshot (The default tab)
  - o Why Join Us
  - o Reviews
  - o Salaries
  - o Photos
  - o Jobs

## Snapshot Tab

- The company's snapshot tab should have the following details about the company:

- o Average scores (Work Happiness Score, Learning, Appreciation)
- o About the company
- o CEO
- o Founded
- o Company size
- o Revenue
- o Industry
- o Company Description
- o Company Mission
- o Featured reviews selected by the company should be displayed on the profile.
- o 5 reviews to be displayed(at least 1 negative review)

## **Why Join Us Tab**

- This Tab should have the following features:
  - o About the company
  - o Work culture
  - o Company values

## **Reviews Tab**

- It should have the following features
  - o Review this company button to add a review. It should open a modal to accept the below details
    - Overall rating
    - Review summary
    - Your review
    - Pros
    - Cons
    - CEO Approval
    - How should I prepare for an interview at <company name>?
  - o Filter the jobs based on Helpfulness, Rating, and Date.
  - o Sort based on most helpful/unhelpful review, Rating.
  - o Review cards should have Review title, Reviewer's role, City, State, posted date, rating in number, and stars.
  - o Review card should also have a "was this review helpful" voting system.
  - o If the logged in user has posted any review it should be on top(Only in default view i.e., without any sort,filter).
  - o Implement pagination on this tab (server-side pagination preferred).

## **Salaries Tab**

- It should have the following features
  - o Add a salary button which will open Modal where you can add the below salary details
    - What's your company name?
    - Are you currently working at this company

- End date ( Invisible field, only to appear if the above question is answered No)
- What's your job title?
- Where's your job location?
- What's your pay at <company's name>?
- How many years of relevant experience do you have?
- Which benefits do you receive at <company's name>?
  - a. Paid time off
  - b. Health insurance
  - c. Life insurance
  - d. Dental/ vision insurance
  - e. Retirement/ 401(k)
  - f. Other benefits (Open a text box to fill other benefits, if chosen)
- o Display the salary reviews of different roles under categories. It should display the role name and annual salary.

### **Photos Tab**

- Office photos should be shared in this tab.
- Upload photos button that will open a modal where a job seeker can select one or more images to upload.
- Existing Photos should be displayed in a grid.

### **Jobs Tab**

- A list of job cards should be shared on this tab. It should include below details
  - Job Name
  - Location (City, State)
  - No. of days it has been posted for
- On selecting a job, it should below details
  - Apply button ( Should allow submitting resume)
  - Job details like Qualifications required, Responsibilities
- A filter to fetch jobs on the job title and location
- Pagination needs to be implemented (server-side preferred)

---






### **Company Reviews Tab**

- Job seekers should be allowed to search for a company based on its name and/or location.
- All the companies matching the search string should be displayed
- The display should include the company name (hyperlink pointing to the homepage of the company, ratings, Reviews (hyperlink pointing to the reviews tab of the company), Salaries(pointing to the salary reviews of the company), Jobs ( pointing to the active jobs tab of the company)

## Find salaries Tab

- Should have a search capability allowing to search by job title and location
- On searching for a job title, it should display the average salary of that job in that location.
- It should also display the highest paying 5 companies as per the role in the location as shown in the picture below.

### Top companies for Software Engineers in United States

	<b>Apple</b> 4.2 ★★★★★ 10366 reviews 33 salaries reported	<b>\$165,596</b> per year >
	<b>Facebook</b> 4.1 ★★★★★ 629 reviews 383 salaries reported	<b>\$158,727</b> per year >
	<b>Octo Consulting Group</b> 3.7 ★★★★★ 27 reviews 5 salaries reported	<b>\$153,511</b> per year >
	<b>Capital One</b> 3.9 ★★★★★ 9420 reviews 266 salaries reported	<b>\$152,798</b> per year >
	<b>Juniper Networks</b> 4.1 ★★★★★ 434 reviews 6 salaries reported	<b>\$145,627</b> per year >

## Job Seeker Profile & Activity

- Job seekers should be able to view their profiles, update their name and contact details. (Email ids have to be unique throughout the application)
- Job seekers should be able to download. Replace, delete their resumes
- Job seekers should be able to view the saved and applied jobs
- Job seekers should be able to view the list of reviews he has added for companies
- Clicking on each review should take the user to the review page and the logged in user's review should be on top.

## Employer

An employer will be having a different set of pages. He will not be able to view Job Seeker's pages.

## **Company Profile page**

- An employer should be able to view, edit and update his profile page containing his name, role in the company, and address.
- An employer should add/edit the following details for the company.
  - Website
  - Company Size
  - Company Type
  - Revenue
  - Headquarters
  - Industry
  - Founded
  - Mission & Vision
  - CEO Name

## **Review Page**

- An employer can see every rating and review made by job seekers of their company.
- An employer can mark featured reviews if he wants to show that review on the company's profile.

## **Job Posting Page**

- An employer can post the jobs from the job posting page.
- They should give the following details about the new job
  - Company name
  - Job title
  - Industry
  - Country
  - Remote/ In-person
  - Type (Part-time/ Full-time)
  - Street address
  - City
  - State
  - Zip
- Pagination should be used to display jobs.

## **Message**

- An employer can send a message to job seekers to show interest in their profile.
- Job seekers can reply to the employer and can not initiate the chat.

## **Applicant Page**

- An employer can go through each job posted by them.
- An employer can see the number of applicants for each job.
- The employer can click on the job title and can see the list of users who applied for the job.



- The employer can see the resume and cover letter attached by the applicant in front of their name.
- The employer can then click on the applicant's name and can see the profile page of an applicant with the job preference of that employee.
- The employer can change the applicant's status. This is for employer reference only
- Status of application (Submitted, reviewed, initial screening, Interviewing, Hired)

## **Report**

- The employer should be able to see the statistics of all the jobs posted by him last year, applicants applied to it, the applicant selected, and the applicant rejected.

## **Admin User**

They are the service staff working for Indeed who will have to keep a track of all the reviews and photos added by users on the website. They will be able to view a different set of pages.

## **Reviews and pictures**

- Admin should be able to filter out inappropriate reviews added by users.
- Admin should be able to filter out inappropriate photos added by users.
- Only after the admin approval, a review or image will be made public, before admin approval only the job seeker who submitted it can see the review and image.

## **Company Profile Page**

- Admin users should be able to see the list of companies in the system.
- They should be able to search for companies using the Company name.
- On clicking a Company name from results, they should be able to view all the reviews of the company(both approved reviews and rejected reviews).
- Admin users should also be able to see the statistics of job-related data of that particular company. For example (Number of the hired applicants, rejected applicants).

## **Analytics Dashboard**

- Admin users should be able to view different types of analytics graphs showing the following data:
  - o The number of reviews per day.
  - o Top 5 most reviewed companies.
  - o Top 5 companies based on average rating.
  - o Top 5 job seekers based on total accepted reviews made.
  - o Top 10 CEOs based on rating.
  - o Top 10 companies based on views per day.

## Tier 2 — Middleware

You will need to provide/design/enhance a middleware that can accomplish the above requirements. You have to implement it using REST-based Web Services as Middleware technology. Next, decide on the Interface your service would be exposing. You should ensure that you appropriately handle error conditions/exceptions/failure states and return meaningful information to your caller. Your project should also include a model class that uses standard modules to access your database.

Use Kafka as a messaging platform for communication between front-end channels with backend systems.

## Tier 3 — Database Schema

- You will need to design your database to store your data in both relational and NoSQL databases. Choose column names and types that are appropriate for the type of data you are using.
- You will need to decide which data will be stored in MongoDB and MySQL. (Hint: Choose the database for every information stored based on pros and cons of MongoDB vs MySQL)
- You need to implement SQL caching using **Redis**

## Scalability, Performance, and Reliability:

The hallmark of any good object container is scalability. A highly scalable system will not experience degraded performance or excessive resource consumption when managing large numbers of objects or processing large numbers of simultaneous requests. You need to make sure that your system can handle many users and incoming requests. Pay careful attention to how you manage “expensive” resources like database connections.

Your system should easily be able to manage 10,000 users and 10,000 reviews+photos. Consider these numbers as **minimum** requirements. Further, for all operations defined above, you need to ensure that your system is left in a consistent state if a particular operation fails. Consider this requirement carefully as you design your project as some operations may involve multiple database operations. You may need to make use of transactions to model these operations and roll back the database in case of failure.

## Testing:

To test the robustness of your system, you need to design test cases to test all the functionalities that a regular client would use. You can use your test cases to evaluate the scalability of the system as well by running the tests with thousands of products and users. Use these tests to debug problems in the server implementation.

- **Mocha Testing:** Implement testing on **Ten** randomly selected REST web service API calls using Mocha and include the results in the project report.

## **Presentation:**

1. Group number and team details
2. Database Schema
3. System Architecture Design Diagram
4. Performance comparison bar graphs that show the difference in performance by adding different distributed features at a time such as B (Base), S (SQL Caching), D (DB connection pooling), K (Kafka) for 100, 200, 300, 400 and 500 simultaneous user threads. (5 Bar graphs in total).

Combinations are:

- a. B
- b. B+D
- c. B+D+S
- d. B+D+S+K
- e. B+D+S+K+ other techniques you used

Note: Populate DB with at least 10,000 random reviews and measure the performance on an API fetching all the 10000 reviews.

5. Performance Comparison of services with below deployment configurations with the Load balancer
  - a. 1 Services server
  - b. 3 Services server
  - c. 4 Services server

## **Project Report:**

Your project report must consist of the following:

- A title page listing the members of your group
- A page describing how each member of the group contributed to the project. (One short paragraph per person)
- Screen capture of your database schema
- System Architecture Design diagram
- A short note describing:
  - o Your object management policy

- o How you handle heavyweight resources
  - o The policy you used to decide when to write data into the database
- Screen captures of your application (few important pages)
- A code listing your server implementation for entity objects
- A code listing of your server implementation of the security and session objects
- A code listing of your main server code
- A code listing of your database access or connection
- A code listing of your Mocha test and output result screenshots
- Observations on the performance of the application based on the comparative analysis at different deployment configurations with graph

## Scoring Points:

- **Functionality** (40 marks) – Every functionality mentioned in the requirement above carries some marks and they will be randomly tested along with the code and ESLint.
- **Scalability** (15 marks) – Your project will be tested with thousands of objects. There should not be any degradation in the performance of the application and it should not crash for any reason.
- **Distributed Services** (15 marks) – Divide the client requirements into different services and deploy them on the cloud. Each service should be running on the backend server connected using Kafka. Divide the data into MongoDB and MySQL and provide reasons for the choice of database for every object.
- **UI** (15 marks) – The user interface of your application must be similar to Indeed.com.
- **Report** (15 marks) – Performance testing results, Mocha testing results, and your explanations in Project Report carry marks.

## Notes:

1. Maintain a pool of database connections. Opening a database connection is a resource-intensive task. It is advisable to pre-connect several database connections and make use of them throughout your database class so you don't continually connect/disconnect when accessing your database.
2. To prevent a costly database read operation, you can cache the state of the entity objects in a local in-memory cache. If a request is made to retrieve the state of an object whose state is in a cache, you can reconstitute the object without reading the database. Be careful to update the cache, when the state of the object changes. You are required to implement **SQL caching using Redis**.
3. Don't write data to the database that didn't change since it was the last read.
4. Ensure to keep **store all the passwords in encrypted form**.
5. Focus on the implementation of all the features first. A complete implementation that passes all feature tests is worth as much as the performance and scalability part of the project.
6. Do not over-optimize. Groups in the past in an attempt to optimize the project, implemented complex optimizations but failed to complete the feature set.

7. Try to implement all the features with proper exception handling and user-friendly errors. Implement proper input validations on all pages. Do not give scope for the user to make the application crash.
8. Try to implement all the features as similar as it is in Indeed.com.