Salary Share

Arun Murugan

Saad Jeelani

Raul Cisneros

CS 157A

Jahan Ghofraniha

May 8, 2022

Table of Contents

Executive Summary

There is a drastic problem in the work industry: wage disparity. Employees are getting paid different salaries while having the same skills and experience for the same job position. This leads to potential undercutting by companies because their skills are not being recognized. A simple option around this is to look up salary values on job posting websites or when applying for a job, however the companies just don't show it in the job posting. This leaves job seekers with a lack of knowledge for baseline salary when they go to apply for the job. Another helpful solution would be to see what other employees make for the same position but unfortunately there is also a lack of compensation transparency on the web. As a result, when a job seeker is asked to negotiate the salary price during the interview, they do not have an accurate value to give.

To combat this issue, we decided to create a platform that would take in and post salary information for users to see. Users can voluntarily provide their current or past job information to the website and users can also pull up these results by filtering using certain information such as the job position or the salary amount. This would solve the lack of a baseline salary as well as compensation transparency since users would be able to see what others are making and use that information for their next negotiation.

Salary share takes in a user's data on a current or past job and stores it in its database. The database is designed to have 6 tables that are filled in from what the user provides in the job posting. The most important information that the tables require are always required by the user to provide on the front page to make sure invalid data is not inserted. The database also allows any query on itself so users will be able to find salaries posted such as for a certain job role or a company. With the information stored and queried this way, it allows for finding a baseline salary for certain requirements and it also fills in that lack of compensation transparency since users will be seeing information of other users' data.

For the project, we've gone through the entire design and deployment process. The details for design/implementation/testing/deployement are listed below. You can view the fully deployed application at: https://salaryshareapp.herokuapp.com/home

Background/Introduction

Currently, websites such as Indeed, Glassdoor, and many others all help job seekers find a job. These services allow job seekers the ability to effectively find the job that they're looking for. Essentially, these websites serve as the middle ground between job seekers and companies, allowing for fast and efficient communication. However, once a job is found, there's very limited information regarding the value that their job position demands on the market. This results in a wage disparity where two employees who work the same job and have the same market value will be getting paid two different amounts.

This problem is especially relevant to college students like us. Entering the industry, many college students (especially first-generation college students or first-generation students within their respective fields) don't know what salary they should be negotiating for. Having access to anonymous salaries from others in the industry can provide leverage and aid salary negotiations. To address this issue, we developed a platform which allows people to voluntarily report their salary information so that job seekers may use this information to recognize and demand for their market value.

Problem Statement

In several industry's there's a lack of compensation transparency which leads to a salary gap between employees who do the same job but get compensated a different value. A process or solution to bridge this salary gap is required for a fairer job market.

Purpose/Motivation

To solve this issue of compensation disparity, our group decided to develop Salary Share. This problem is especially motivating and relevant to us because a lot of us students will be entering the market through internships or full-time jobs and need an idea of what we're worth on the job market. Creating a platform such as salary share will allow us to become knowledgeable of our value and therefore get our worth.

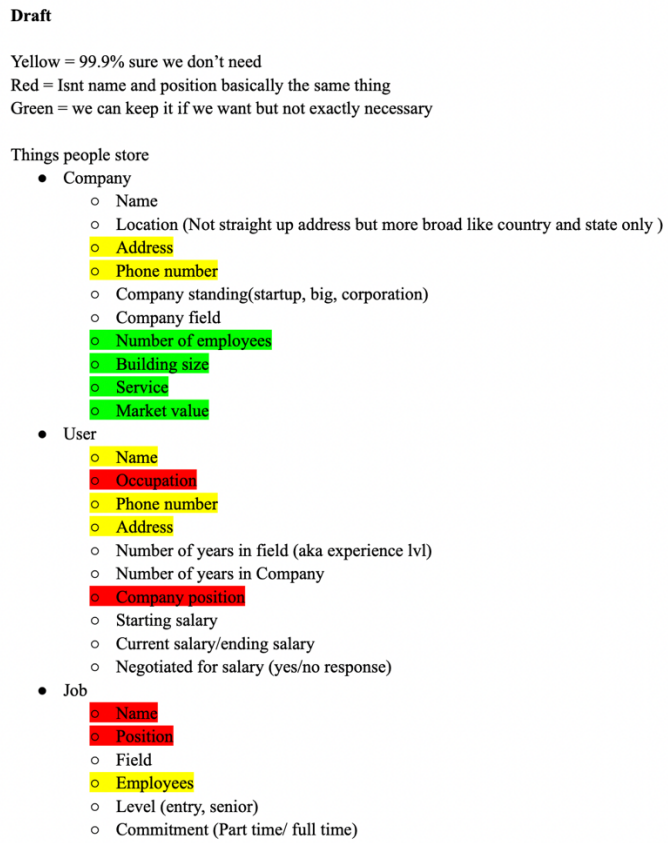
Furthermore, we noticed that there is a lack of information and guidance out there that can provide appropriate salary values. Websites such as Indeed can provide job listings for job seekers, but they do not provide a salary value that one can use for a basis. Our platform would allow any user to be able to post their job information while simultaneously allowing other users to also filter through the job postings. With knowledge of job postings from people in similar or exact criteria that one is in, users can find a baseline that they can use for interviews or resumes.

The Design Process

Requirements Gathering

Our requirements gathering phase primarily focused on three factors: the main distinguishing features for a job, the features of interest for a job, and what features in a job cause salary discrepancy. Keying in on the first two points allowed us to get an idea of what type of information job seekers may want to input to get salary values for those specific values. The third primary focus mentioned above allowed us to create business rules which would aid the user in finding their market value for their specific job of interest.

We went about gathering the distinguishing job features and features of a job that are of interest for the user by looking at information that's commonly used on websites such as indeed and glass door. We made a list of these job features and included them in our first few potential views.
Doing a requirement gathering on factors which cause salary discrepancy took more detailed researching. This researching included asking HR members within the industry for factors that decide a worker's compensation. Factors such as company that they work at, location of work (inflation/cost of living etc.), seniority etc. was collected. We then put together all this information into 3 broad tables which we felt would allow us to create the views that we desired.
Image of Requirement Gathering document:



Draft

Yellow = 99.9% sure we don't need
Red = Isnt name and position basically the same thing
Green = we can keep it if we want but not exactly necessary

Things people store

- Company
  - Name
  - Location (Not straight up address but more broad like country and state only)
  - Address
  - Phone number
  - Company standing(startup, big, corporation)
  - Company field
  - Number of employees
  - Building size
  - Service
  - Market value
- User
  - Name
  - Occupation
  - Phone number
  - Address
  - Number of years in field (aka experience lvl)
  - Number of years in Company
  - Company position
  - Starting salary
  - Current salary/ending salary
  - Negotiated for salary (yes/no response)
- Job
  - Name
  - Position
  - Field
  - Employees
  - Level (entry, senior)
  - Commitment (Part time/ full time)

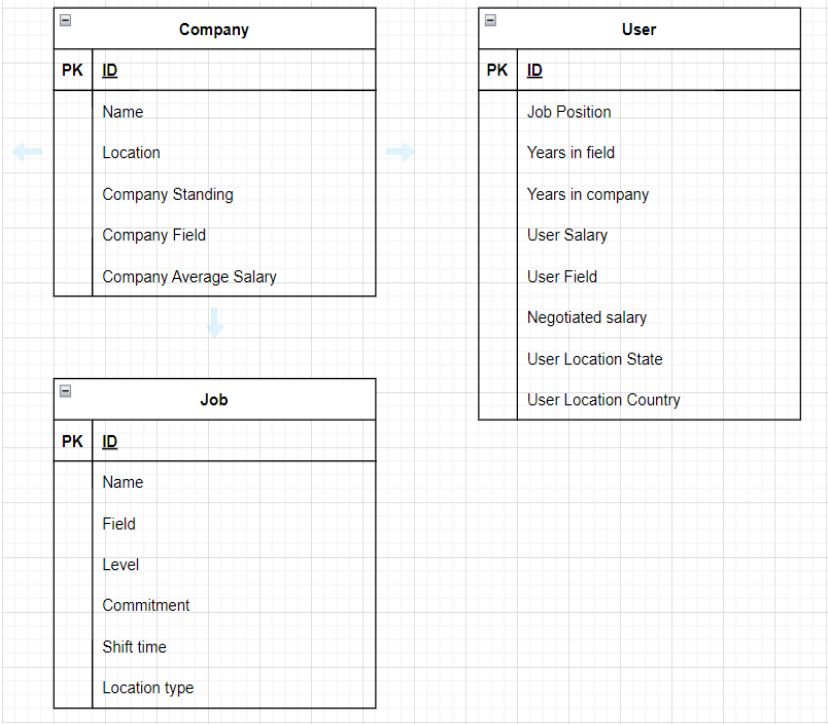
Final

Blue = Not straight up address but more broad like country and state only because pay can vary depending on the country and state.
Green = Basically all possible jobs they have like ex Apple has manager,software engineer,receptionist ect job positions

Things people store

- Company
  - Name
  - Location (Split into Country location and state location)
  - Company standing(startup, big, corporation)
  - Company field
  - Job positions
- User
  - Job position
  - Number of years in field (aka experience lvl)
  - Number of years in Company
  - Starting salary
  - Current salary/ending salary
  - Negotiated for salary (yes/no response)
  - Current year (depending what year, pay may have gone up or down)
- Job
  - Position
  - Field
  - Level (entry, senior)
  - Commitment (Part time/ full time)
  - Time of day (Aka night shift job or day shift job)
  - Job type (remote job/ in person job)
  - Stated salary (basically what was the salary on the job description, if they remember)

Here are the initial tables:



Conceptual Design

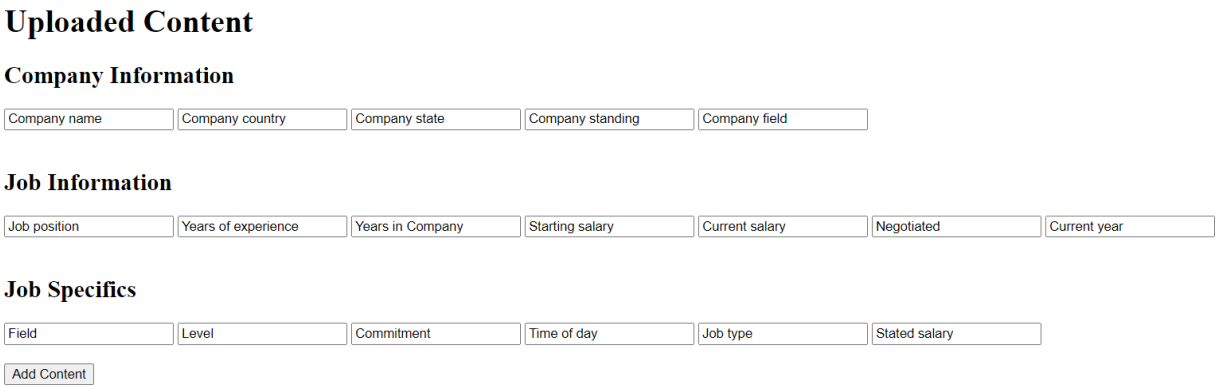
Business Rules:

1. Two jobs can have the same name because they may have different job attributes such as job level, location type etc.
2. Company location refers to company headquarters location
3. Two companies may have the same name – however companies with same name must have different location (Ex. ABC Pizzas in California, US and ABC Pizzas in Texas, US but can't have 2 ABC Pizzas in California…)
4. User input can have a different location than the company that it relates to (employee for Apple, which has a headquarters in California, can be working in Texas)
5. All user inputs are anonymous. We hold no information regarding the user's input for the sake of user anonymity
6. Field information is maintained about the job and company to allow for specific field-related queries

Design Screens/Form:

These forms/design screens represent the basic views and forms that we want represented on the website. The real front end will probably look more aesthetic than this:

Uploading Content Page:



Successful content uploaded page:

Uploaded Content

Company Information

Company name	Company country	Company state	Company standing	Company field

Job Information

Job position	Years of experience	Years in Company	Starting salary	Current salary	Negotiated	Current year

Job Specifics

Field	Level	Commitment	Time of day	Job type	Stated salary

Add Content

Content successfully uploaded!

Failed content upload page:

Uploaded Content

Company Information

Company name	Company country	Company state	Company standing	Company field

Job Information

Job position	Years of experience	Years in Company	Starting salary	Current salary	Negotiated	Current year

Job Specifics

Field	Level	Commitment	Time of day	Job type	Stated salary

Add Content

Unable to add content, please check input and try again

Search content page:

Search Content

Company Information

Company name	Company country	Company state	Company field

Job Information

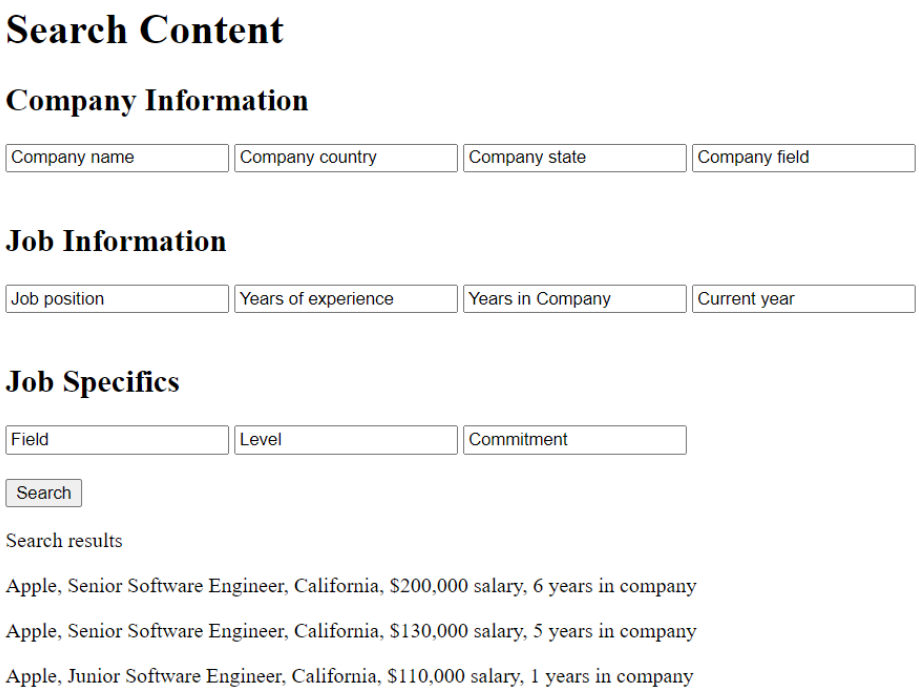
Job position	Years of experience	Years in Company	Current year

Job Specifics

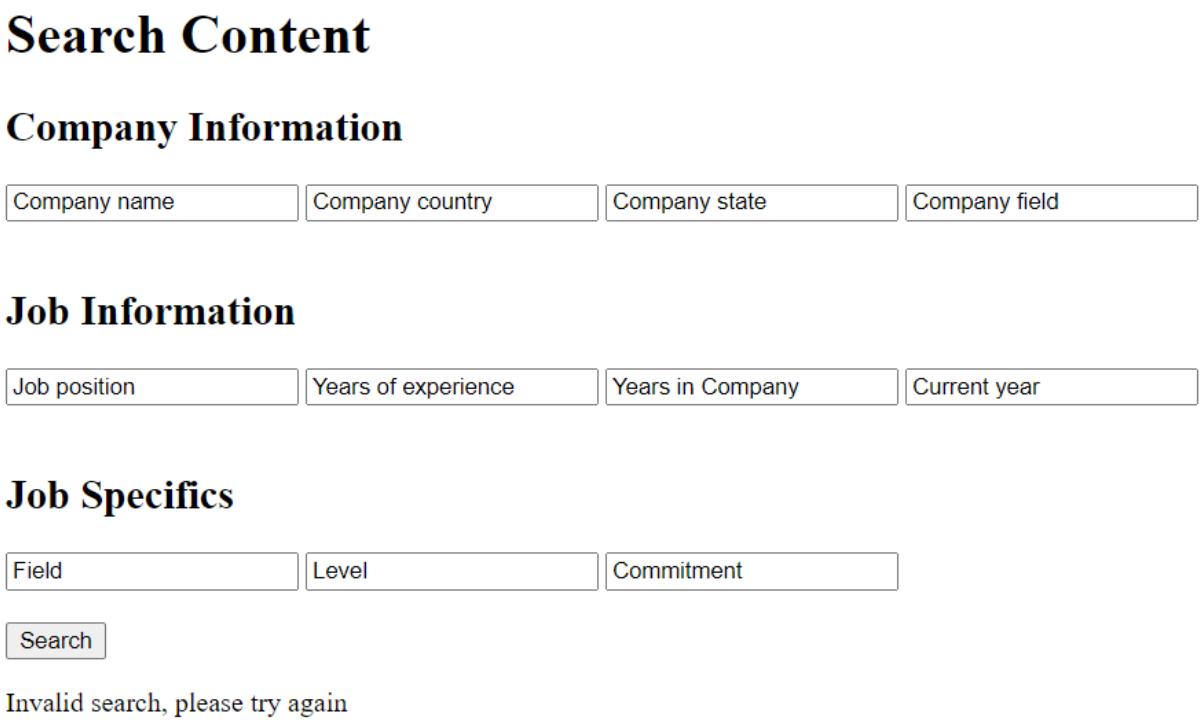
Field	Level	Commitment

Search

Successful search content page:



Failed search content page:



Unable to add due to lack of entries content page:

Search Content

Company Information

Company name	Company country	Company state	Company field

Job Information

Job position	Years of experience	Years in Company	Current year

Job Specifics

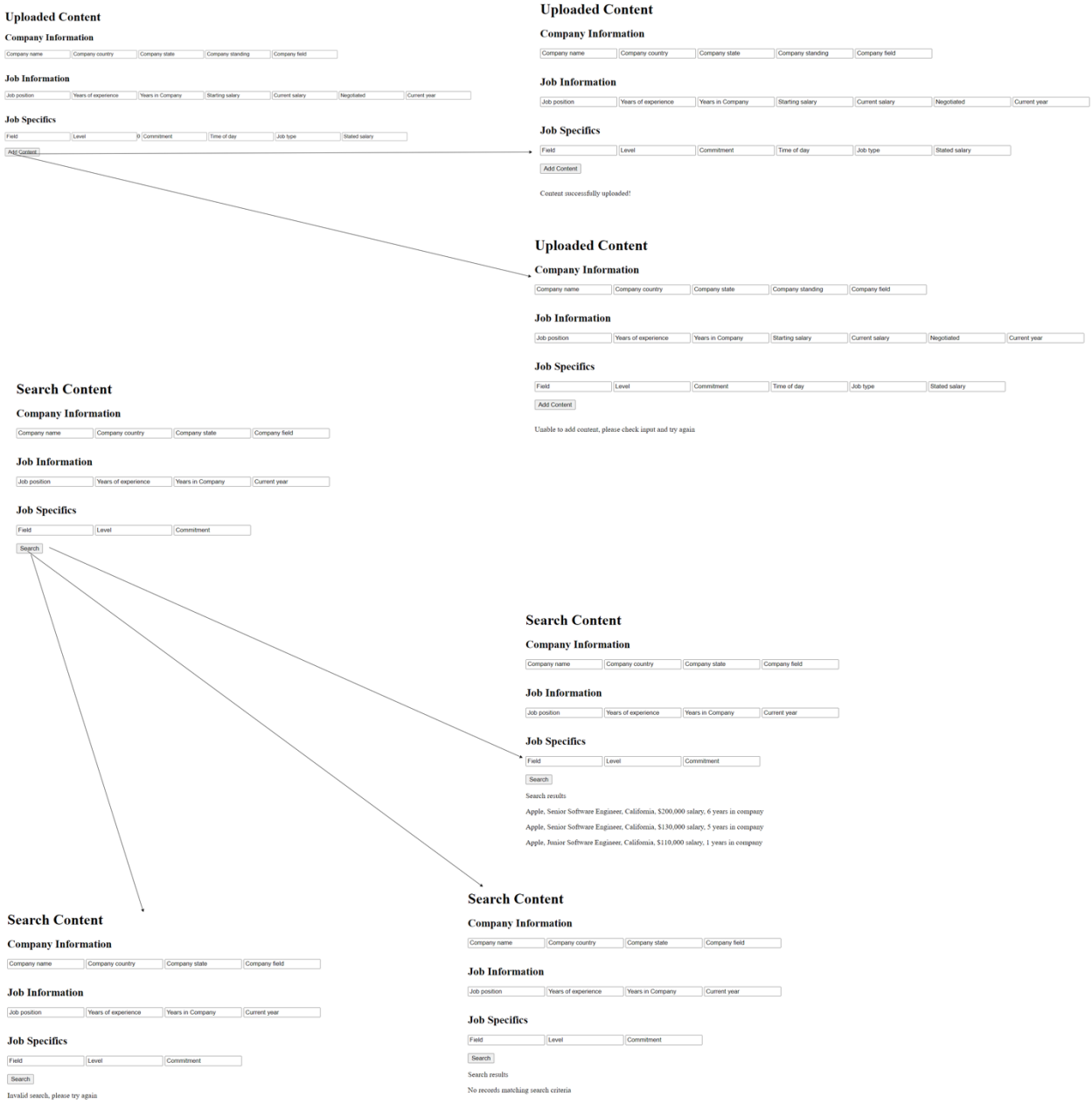
Field	Level	Commitment

Search

Search results

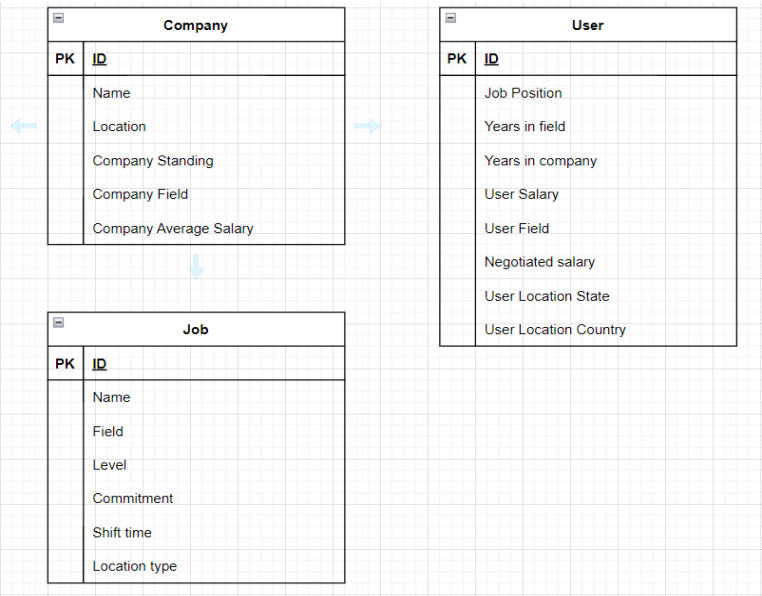
No records matching search criteria

Storyboards

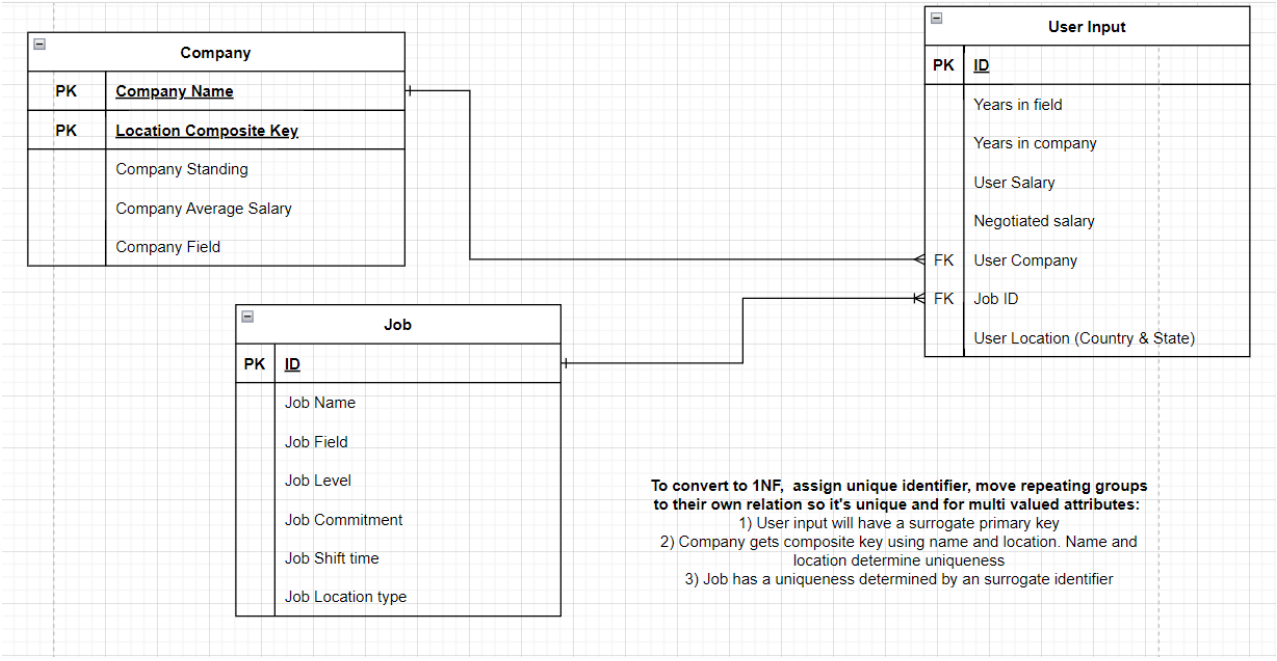


Logical Design/Normalization

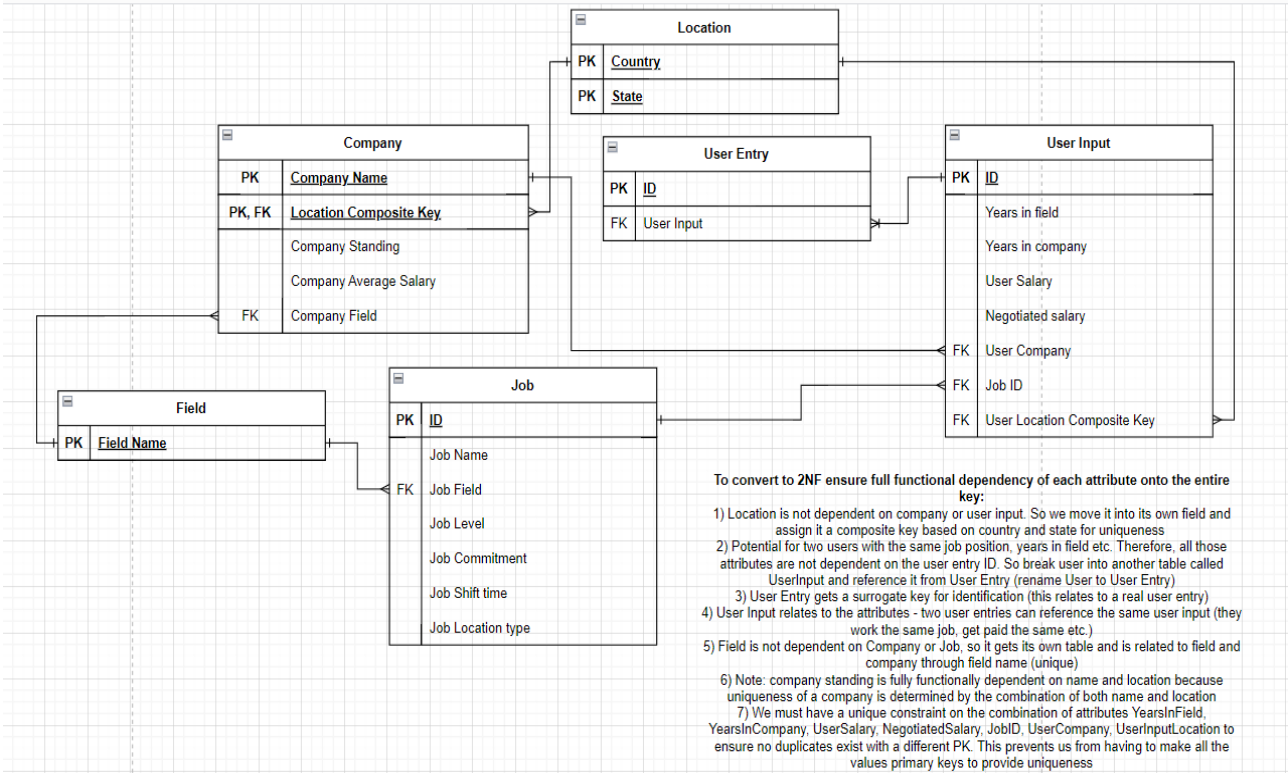
As mentioned below, after the requirements gathering phase, we came up with three main tables. Here's an image of our initial tables:



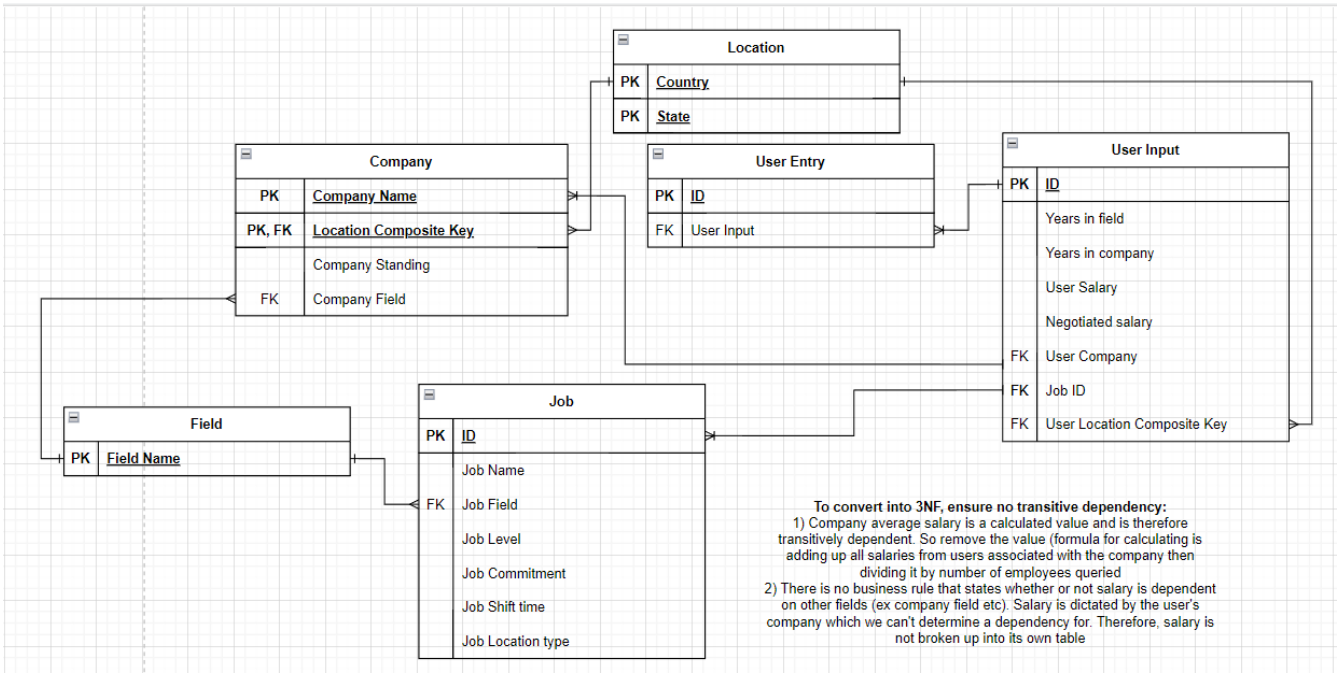
Once the initial tables were created, we proceeded to convert the table into 1NF as shown below. Since the initial company table had the location attribute as a multivalued attribute, it was made to become a composite key in the company table. We also assigned a numerical ID value as the primary key that would uniquely identify each entry in the tables, thus placing the appropriate primary keys as foreign keys in each table as required. The current ERD below is in 1NF because there are no multivalued attributes in any of the tables. Please read the notes inside the ERD for detailed descriptions.



Please read the notes in the ERD for a detailed description on how we got from 1NF to 2NF:



The final step in the normalization process was converting 2NF to 3NF - please read the notes in the image for more details:



Physical Design

The logical design was implemented using mySQL. First a local server was created in which we created a proper database. The development environment used was dBeaver on a Mac. First, we went about creating the Schema. A client-server scheme was used.

Testing the Physical Design

We pre-populated all the tables with sample values. Here's a screenshot of our populated tables – for the sake of conciseness, we only showed a few of the tables – most entries were hidden:
Job & Field tables:

	NAME
1	Business
2	Construction
3	Education
4	Engineering
5	Hospitality
6	Medicine
7	Military
8	Technology

	COUNTRY	STATE
1	Canada	Alberta
2	Canada	British Columbia
3	Canada	Manitoba
4	Canada	Ontario
5	Canada	Quebec
6	United States	Alabama
7	United States	Alaska
8	United States	California

Company Table:

	COMPANY_NAME	COMPANY_STANDING	COMPANY_COUNTRY	COMPANY_STATE	COMPANY_FIELD
1	ABC Pizzas	[NULL]	Canada	Alberta	Hospitality
2	Amazon	Corporate	United States	Washington State	Technology
3	Apple	Corporate	United States	California	Technology
4	Best Burgers	[NULL]	Canada	British Columbia	Hospitality
5	Fixerupper Hospital	Private	United States	New York	Medicine
6	Google	Corporate	United States	California	Technology
7	IBM	Corporate	United States	Georgia	Technology
8	Krispy Krabs	[NULL]	United States	Alabama	Hospitality

Job table:

	JOB_ID	JOB_NAME	JOB_L	JOB_C	JOE	JOB_L	JOB_FIELD
1	1	Software Developer	Tech Lead	Full-time	Day	On-site	Technology
2	2	Software Developer	Senior	Full-time	Day	On-site	Technology
3	3	Software Developer	Staff	Full-time	Day	On-site	Technology
4	4	Software Developer	Junior	Full-time	Day	On-site	Technology
5	5	Product Manager	Manager	Full-time	Day	On-site	Technology
6	6	Systems Architect	Senior	Full-time	Day	On-site	Technology
7	7	Quality Engineer	Junior	Full-time	Day	On-site	Technology
8	8	CEO	[NULL]	Full-time	[NULL]	On-site	Technology

User input table (not all attributes are shown due to lack of screen real estate):

		YEA	YEARS	SALAR	NEG	USE	USER_COU	USER_STATE	USER_COMP/	USER_COMPANY
1	1	2	2	11,000	[NULL]	16	Canada	Alberta	ABC Pizzas	Canada
2	2	2	2	13,000	[NULL]	16	Canada	Alberta	ABC Pizzas	Canada
3	3	1	1	13,000	[NULL]	16	Canada	British Columbia	Best Burgers	Canada
4	4	1	1	14,000	[NULL]	16	United States	Florida	Wicked Waffles	United States
5	5	2	2	10,000	[NULL]	16	United States	Alabama	Sizzling Steak	United States
6	6	5	3	22,000	[NULL]	17	Canada	Alberta	ABC Pizzas	Canada
7	7	7	4	26,000	[NULL]	17	Canada	Alberta	ABC Pizzas	Canada
8	8	1	1	26,000	[NULL]	17	Canada	British Columbia	Best Burgers	Canada

User entry table:

	ENTRY_ID	USER_INPUT_ID
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	8	8

Implementation & Test Report

Testing the database

Having populated the tables with over 150 entries, we moved to the testing process. The goal of our testing was to ensure that all entities relate to each other as our ERD called for. Furthermore, we wanted to ensure that our database is flexible enough to run various types of queries. We created 15 queries which would test our database integrity. Please refer to SQL code for the actual query code.

For each of the queries, there may be up to 145 entries. Therefore, not all entries are shown for the sake of conciseness – please refer to our test database code provided in the git to see the detailed view:

1) Get all inputs in the database and show their respective job/company/location attributes. Not all results are shown for the sake of conciseness:

Running Queries

	ENTRY_ID	COMPANY_NAME	COMPANY_COUNTRY	JOB_NAME	USER_STATE	SALARY
1	1	ABC Pizzas	Canada	Waiter	Alberta	11,000
2	2	ABC Pizzas	Canada	Waiter	Alberta	13,000
3	3	Best Burgers	Canada	Waiter	British Columbia	13,000
4	4	Wicked Waffles	United States	Waiter	Florida	14,000
5	5	Sizzling Steak	United States	Waiter	Alabama	10,000
6	6	ABC Pizzas	Canada	Waiter	Alberta	22,000
7	7	ABC Pizzas	Canada	Waiter	Alberta	26,000
8	8	Best Burgers	Canada	Waiter	British Columbia	26,000
9	9	Wicked Waffles	United States	Waiter	Florida	28,000
10	10	Sizzling Steak	United States	Waiter	Alabama	20,000
11	11	ABC Pizzas	Canada	Chef	Alberta	22,000
12	12	ABC Pizzas	Canada	Chef	Alberta	26,000
13	13	Best Burgers	Canada	Chef	British Columbia	32,000
14	14	Wicked Waffles	United States	Chef	Florida	30,000
15	15	Sizzling Steak	United States	Chef	Alabama	20,000

2) Show all users detailed job view – allows us to test that job attributes are being shown:

	EN	COMPANY_	JOE	JOB_	JOB_CO	JOE	JOB_LI	JOB_FIELD	USER_STATE	USER_COUNTRY	SALARY
1	1	ABC Pizzas	Waiter	Entry	Part-time	Day	On-site	Hospitality	Alberta	Canada	11,000
2	2	ABC Pizzas	Waiter	Entry	Part-time	Day	On-site	Hospitality	Alberta	Canada	13,000
3	3	Best Burgers	Waiter	Entry	Part-time	Day	On-site	Hospitality	British Columbia	Canada	13,000
4	4	Wicked Waffles	Waiter	Entry	Part-time	Day	On-site	Hospitality	Florida	United States	14,000
5	5	Sizzling Steak	Waiter	Entry	Part-time	Day	On-site	Hospitality	Alabama	United States	10,000
6	6	ABC Pizzas	Waiter	[NULL]	Full-time	Day	On-site	Hospitality	Alberta	Canada	22,000
7	7	ABC Pizzas	Waiter	[NULL]	Full-time	Day	On-site	Hospitality	Alberta	Canada	26,000
8	8	Best Burgers	Waiter	[NULL]	Full-time	Day	On-site	Hospitality	British Columbia	Canada	26,000
9	9	Wicked Waffles	Waiter	[NULL]	Full-time	Day	On-site	Hospitality	Florida	United States	28,000
10	10	Sizzling Steak	Waiter	[NULL]	Full-time	Day	On-site	Hospitality	Alabama	United States	20,000
11	11	ABC Pizzas	Chef	[NULL]	Part-time	Day	On-site	Hospitality	Alberta	Canada	22,000
12	12	ABC Pizzas	Chef	[NULL]	Part-time	Day	On-site	Hospitality	Alberta	Canada	26,000
13	13	Best Burgers	Chef	[NULL]	Part-time	Day	On-site	Hospitality	British Columbia	Canada	32,000
14	14	Wicked Waffles	Chef	[NULL]	Part-time	Day	On-site	Hospitality	Florida	United States	30,000
15	15	Sizzling Steak	Chef	[NULL]	Part-time	Day	On-site	Hospitality	Alabama	United States	20,000

3) Add criteria for user being in California and relating to the field of technology – allows us to test the relationships between all tables (relates company, job, field, location, user input and user entries):

	INPUT_ID	COMPANY_NAME	JOB_NAME	USER_STATE	SALARY	JOB_FIELD
1	45	Apple	Software Developer	California	500,000	Technology
2	47	Apple	Software Developer	California	530,000	Technology
3	50	Amazon	Software Developer	California	450,000	Technology
4	51	Meta	Software Developer	California	500,000	Technology
5	52	Apple	Software Developer	California	300,000	Technology
6	54	Amazon	Software Developer	California	400,000	Technology
7	55	Amazon	Software Developer	California	250,000	Technology
8	56	Meta	Software Developer	California	200,000	Technology
9	57	Meta	Software Developer	California	400,000	Technology
10	59	Netflix	Software Developer	California	300,000	Technology
11	60	Google	Software Developer	California	200,000	Technology
12	76	Apple	Software Developer	California	250,000	Technology
13	78	Amazon	Software Developer	California	350,000	Technology
14	79	Amazon	Software Developer	California	200,000	Technology
15	80	Meta	Software Developer	California	150,000	Technology

4) Junior software developer criteria – tests relevant business case of junior dev:

	INPUT_ID	COMPANY_NAME	JOB_NAME	JOB_LEVEL	USER_STATE	SALARY
1	64	Apple	Software Developer	Junior	California	200,000
2	65	Apple	Software Developer	Junior	New York	250,000
3	66	Amazon	Software Developer	Junior	California	300,000
4	67	Amazon	Software Developer	Junior	California	150,000
5	68	Meta	Software Developer	Junior	California	100,000
6	69	Meta	Software Developer	Junior	California	300,000
7	70	Netflix	Software Developer	Junior	New York	100,000
8	71	Netflix	Software Developer	Junior	California	200,000
9	72	Google	Software Developer	Junior	California	100,000
10	73	Google	Software Developer	Junior	New York	175,000
11	74	IBM	Software Developer	Junior	Georgia	90,000
12	75	IBM	Software Developer	Junior	Georgia	88,000

5) All entries with over 500 thousand salaries – another relevant business case. Allows us to test duplicates:

	INPUT_ID	COMPANY_NAME	JOB_NAME	JOB_LEVEL	JOB_FIELD	USER_STATE	SALARY
1	89	Apple	Product Manager	Manager	Technology	Washington State	550,000
2	91	Amazon	Product Manager	Manager	Technology	Washington State	500,000
3	110	Amazon	CEO	[NULL]	Technology	Washington State	4,000,000
4	45	Apple	Software Develop	Tech Lead	Technology	California	500,000
5	88	Apple	Product Manager	Manager	Technology	California	600,000
6	109	Apple	CEO	[NULL]	Technology	California	3,200,000
7	113	Google	CEO	[NULL]	Technology	California	5,350,000
8	51	Meta	Software Develop	Tech Lead	Technology	California	500,000
9	94	Meta	Product Manager	Manager	Technology	California	550,000
10	111	Meta	CEO	[NULL]	Technology	California	4,350,000
11	47	Apple	Software Develop	Tech Lead	Technology	California	530,000
12	90	Apple	Product Manager	Manager	Technology	California	630,000
13	112	Netflix	CEO	[NULL]	Technology	California	3,350,000
14	116	Fixerupper Hospital	Family Physician	[NULL]	Medicine	New York	535,000
15	114	IBM	CEO	[NULL]	Technology	Georgia	1,350,000

6) All users from a specific company – allows us to check multiple user input tuples. Will test against creating duplicate user inputs:

	INPUT_ID	COMPANY_NAME	JOB_NAME	USER_STATE	SALARY
1	35	Apple	Human Resource	California	80,000
2	36	Apple	Human Resource	California	85,000
3	37	Apple	Human Resource	New York	70,000
4	45	Apple	Software Develop	California	500,000
5	46	Apple	Software Develop	Washington State	350,000
6	47	Apple	Software Develop	California	530,000
7	52	Apple	Software Develop	California	300,000
8	53	Apple	Software Develop	New York	350,000
9	64	Apple	Software Develop	California	200,000
10	65	Apple	Software Develop	New York	250,000
11	76	Apple	Software Develop	California	250,000
12	77	Apple	Software Develop	New York	250,000
13	88	Apple	Product Manager	California	600,000
14	89	Apple	Product Manager	Washington State	550,000
15	90	Apple	Product Manager	California	630,000

7) All users who work in Human Resources (business field) – allows us to check specific job query:

	INPUT_ID	COMPANY_NAME	JOB_NAME	USER_STATE	SALARY	JOB_FIELD
1	35	Apple	Human Resource	California	80,000	Business
2	36	Apple	Human Resource	California	85,000	Business
3	37	Apple	Human Resource	New York	70,000	Business
4	38	Amazon	Human Resource	New York	72,000	Business
5	39	Meta	Human Resource	California	88,000	Business
6	40	Meta	Human Resource	Alabama	45,000	Business
7	41	Washington Hospital	Human Resource	New York	60,000	Business
8	42	Fixerupper Hospital	Human Resource	New York	62,000	Business
9	43	Mayo Clinic	Human Resource	California	66,000	Business
10	44	Sunshine Hospital	Human Resource	California	65,000	Business

Views Implemented

Queries 1, 2, 6 and 7 were implemented in the web application. In addition to those, we also implemented a view which allows users to check for all entries from a specific location. However, the nature of our web application implementation was semi-dynamic. Users were able to enter the field/company/job etc. that they wanted to search up. The web app would query the database and return/display the results

Database export

On the git repository, please find a ".sql" database dump file. To import it into your database development environment (Ex. workbench or dBeaver), select the import from a single stand-alone file option and select start import. Please do check configurations of utf-charset before importing. The database dump uses utf8 and utf8_general_ci. If the development environment you're using uses a different charset, change the keywords in the dump.sql file that you import then import it into your development environment. All entries should auto populate when you import from the dump file.

Implementation of Web Application

Tech Stack

The initial implementation (prototype) which was shown in class was created in node.js. However, the initial implementation was built on a flawed logical design. After redoing the design, we deemed it best to create our new web app using Flask. This decision was made solely based on our time constraint as Flask is more lightweight than Node.js and easier for pushing development code easier.

Framework: Python/Flask
- Form management: WTF-forms
- ORM: SQLAlchemy
- MySQL Plugin: PyMySql
- Frontend/templating: HTML/Jinja2

Database: MySQL

Website Pages

The final deployed application features 4 pages. The home & about us pages are static pages with a basic overview of the project. The upload user entries page allows us to enter data into our database. We can then query these entries from the view database entries page.

The upload user entries page features two separate forms – one for adding company information and another for adding a specific entry. Splitting our forms into two forms allows for a use case in which a company may self-report information about themselves which allows their employees to report their info more easily. If an employee tries to enter their information before adding company info, they will be redirected with an error message

The query database page allows us to view up to 5 views. The views are pre-defined as dynamic query generation is beyond the scope of this project (due to time constraints). The first query (general overview of all entries) is auto loaded. However, the other queries will only generate when the user fills out a form requesting for a specific search.

Workflow for Web App

There are two separate workflows for the web app – one for the user who's uploading their information and the second for the user who wishes to query the database for some information. The two workflows are listed below.

The user who wishes upload their information will visit the upload information page. They will then fill out their specific job role and salary. Some fields are required – if not filled out they will be prompted to refill the form with the additional information. The company that they enter will be verified to see if it exists in the database. If the company doesn't exist, they will be prompted to first enter their company information. If the company already exists, they will get a success message letting them know that their data has been entered.

If some job is entered which already exists in the database, the routes will check to ensure that duplicates aren't entered which prevents the database from throwing us an error. Duplicates are also checked for in the user input route. If the user enters a field that already exists, the input will be linked to that field, else a new field will be created.

The second user who wishes to query salary information will visit the view database page. They will automatically be able to see all the entries without any filters. They can then use the drop downs to query using a specific filter.

Input Validation

There is built in input validation for the type of data that can be uploaded. There is also input validation to ensure that specific fields are filled out. This is done both, on the front end (using wtf-forms) and on the backend (using querying the database in our routes). However, there is no formal validation for the integrity of information. Validating the integrity of information will require more strict business rules and additional processes which haven't been implemented.

Possible Improvements for Web App & Database design integration

Improvements can be made in various aspects of the web application. Processes are needed for data integrity validation. Furthermore, an ORM has already been used, therefore dynamic query generation would allow for a more dynamic and appropriate user experience. Utilizing an ORM for dynamic query generation will allow for dynamic filtration of our data. Our database (through logical design) can scale to handle an array of different queries – being able to take advantage of that in our application would be the next step. Furthermore, our application can extend beyond simple user input – we can use web scraping to populate our data from reliable sources (ex. indeed and other salary sharing websites). This would allow users to get a better sense of the market average.

Another inconvenience during development arose in the nature of the keys that we used during our database design. For job and user input tables, surrogate keys were used. Although they are valid, they are somewhat inconvenient. Using composite keys (even if all attributes were made to be part of the composite key for uniqueness) would save us some routing overhead such as querying for jobID/userInputID. This would save a few queries and in turn save us run time.

Deployment

Our application is fully deployed and accessible at salaryshareapp.herokuapp.com. When adding user input, please be weary of overloading our database. Since we are using a free service, the maximum storage utilization is 5 MB. More on deployment in the next subsections.

Technologies

For the sake of our quick deployment, we used heroku for the application. Heroku features compatibility with Flask applications which makes deployment simple. The second part of deployment was creating hosting our database somewhere. For cost effectiveness and compatibility (since flask has built in compatibility for postgress and not MySQL) we used clearDB and gunicorn. ClearDB allows for a free database deployment through their ignite service. Steps of deploying to heroku and clearDB are listed below.

Deployment process

The steps for deploying our web app to heroku are as follows:

1) Create a heroku account and download heroku onto our machine
2) If not already tracking our application through a git repository, create a repository
3) Create a requirements.txt file to specify our virtual environment dependencies
4) Commit to our git repo
5) Push our repo to heroku master – our app is now deployed

During the deployment process, we must ensure that our package structure matches that which is specified in the flask docs. Furthermore, a Procfile is necessary in our Flask app directory.

The steps of connecting our web app to clearDB database are as follows:

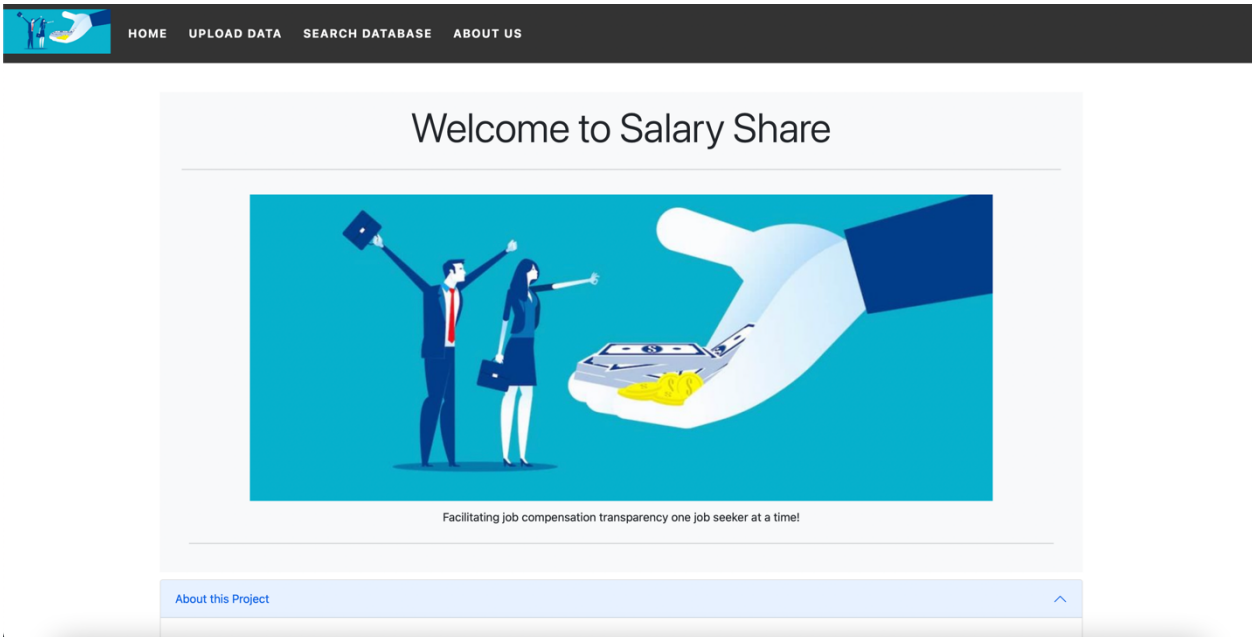
1) Create a clearDB database through heroku and add it to our app.
2) Install clearDB addon to our heroku app through the command line
3) Set clearDB paths in heroku app
4) Connect to clearDB application through a databse development environment (Ex. workbench or dBeaver)
5) Create a database dump for our testing database
6) Import the database into our clearDB-hosted database
7) Add new URI for clearDB into our flask App
8) Use gunicorn to step mySQL path
9) Re deploy heroku app

Deployment shortcomings/future improvements

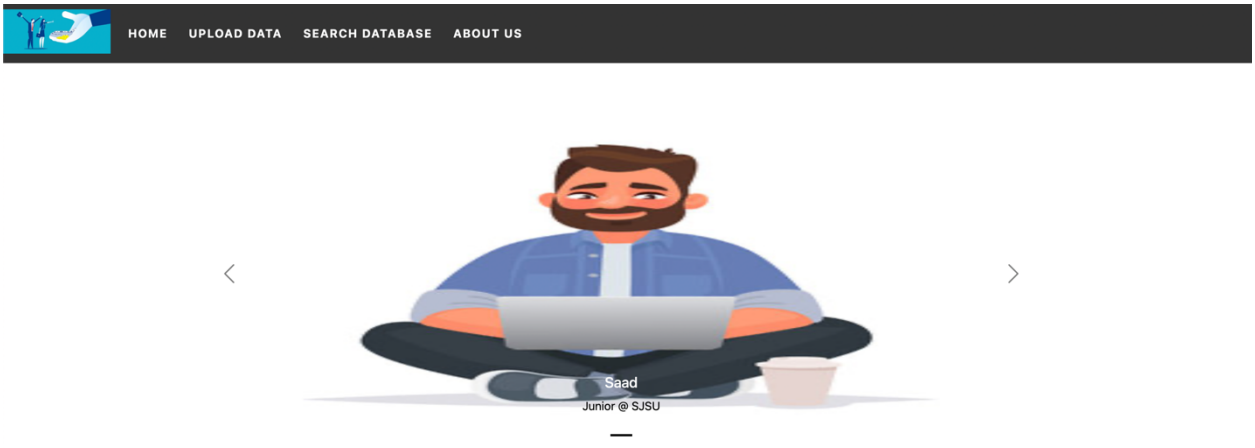
Due to the nature of our application, we used a free service for deployment. Therefore, there's multiple short comings in the production web application. Firstly, the connection to the database times out within a few minutes (variable – no exact time out time). Once the connection times out, the application will throw an error 500 "Internal Server Error". Furthermore, like we mentioned earlier, our database can only store a maximum of 5 MB worth of information. To address both these issues, we simply must refresh which will cause our database connection to be restored. This can be addressed by using a more reliable database service.

Screenshots of the Web App

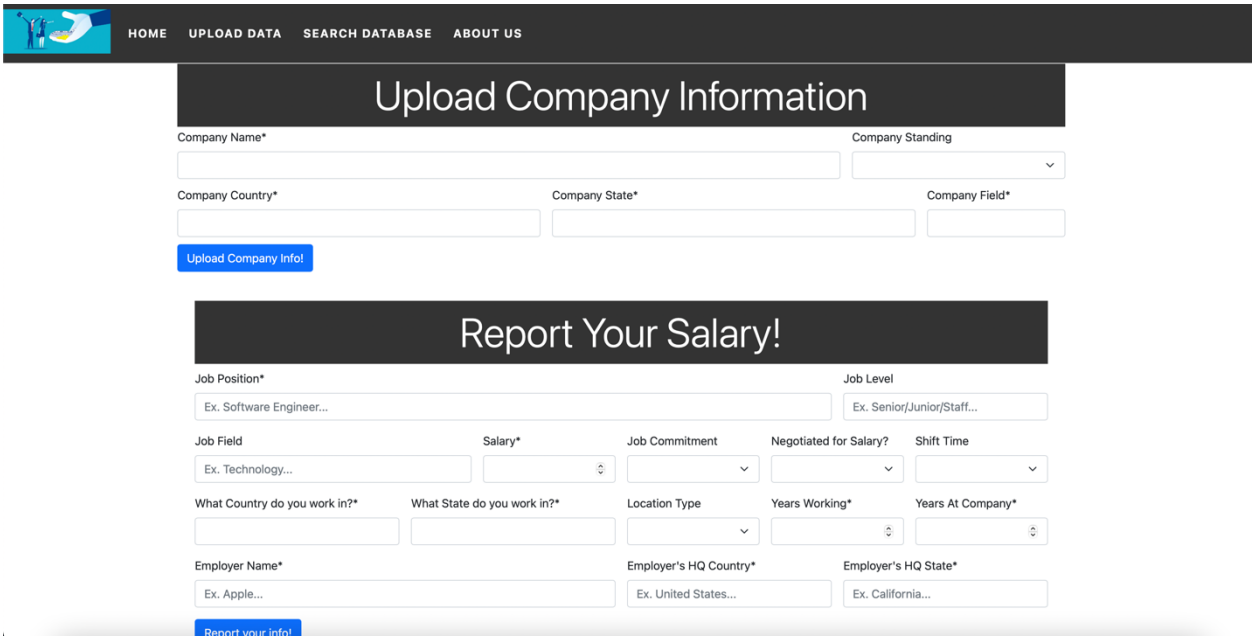
Home page:



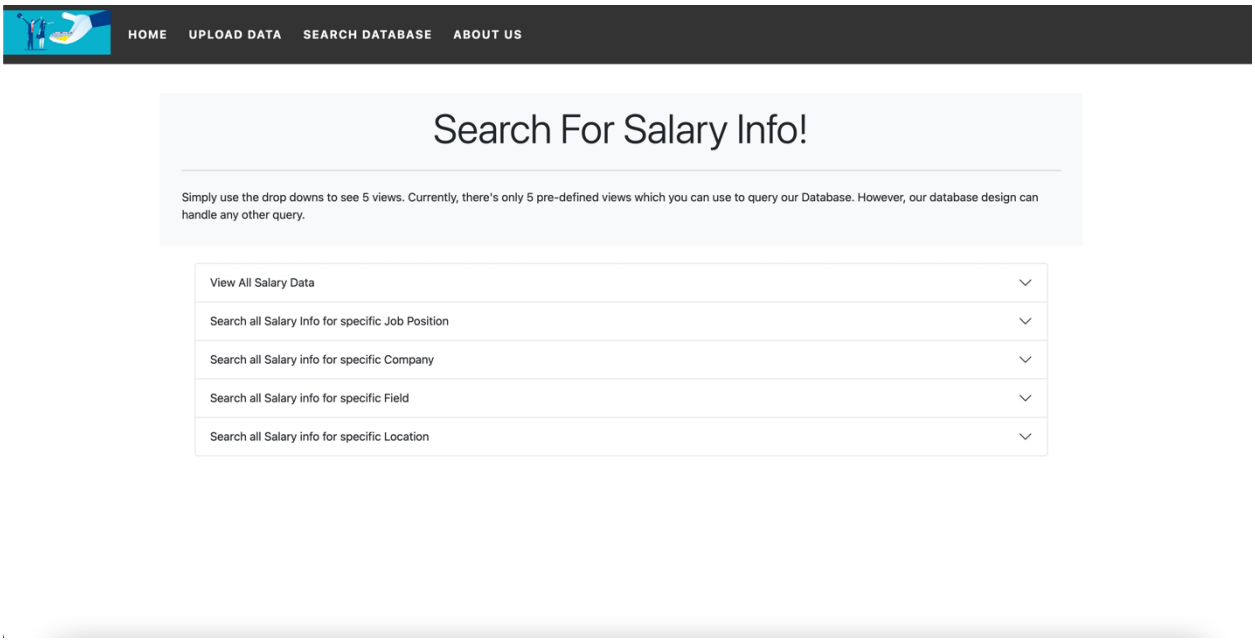
About us page:



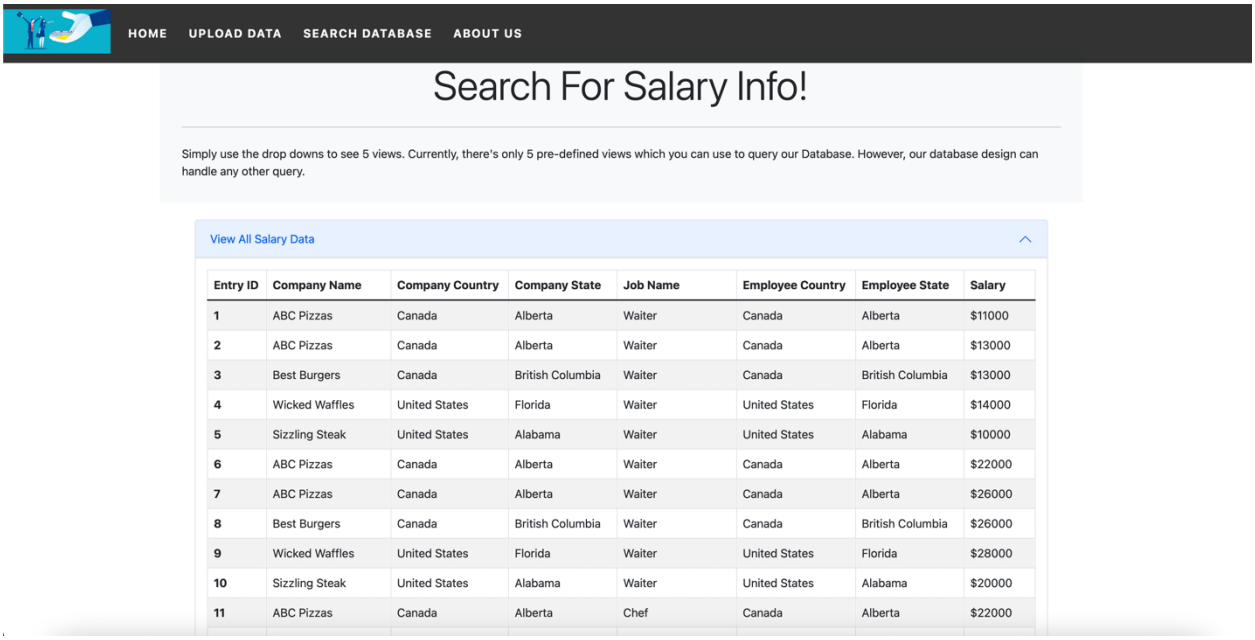
Data upload page:



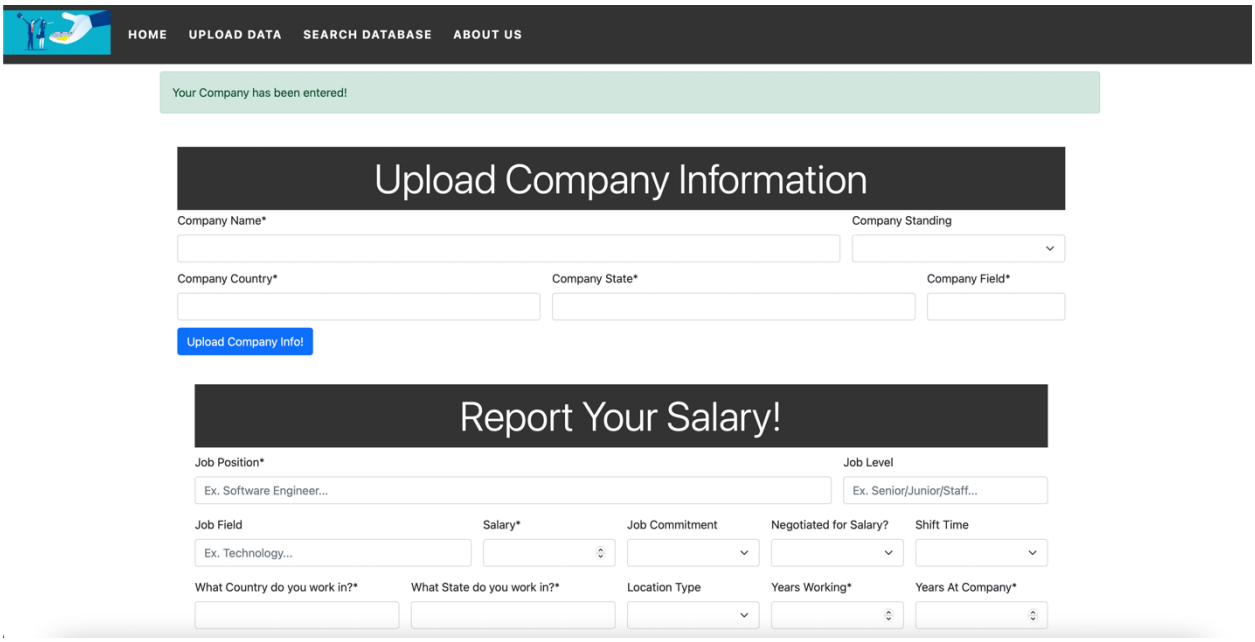
Search database page:



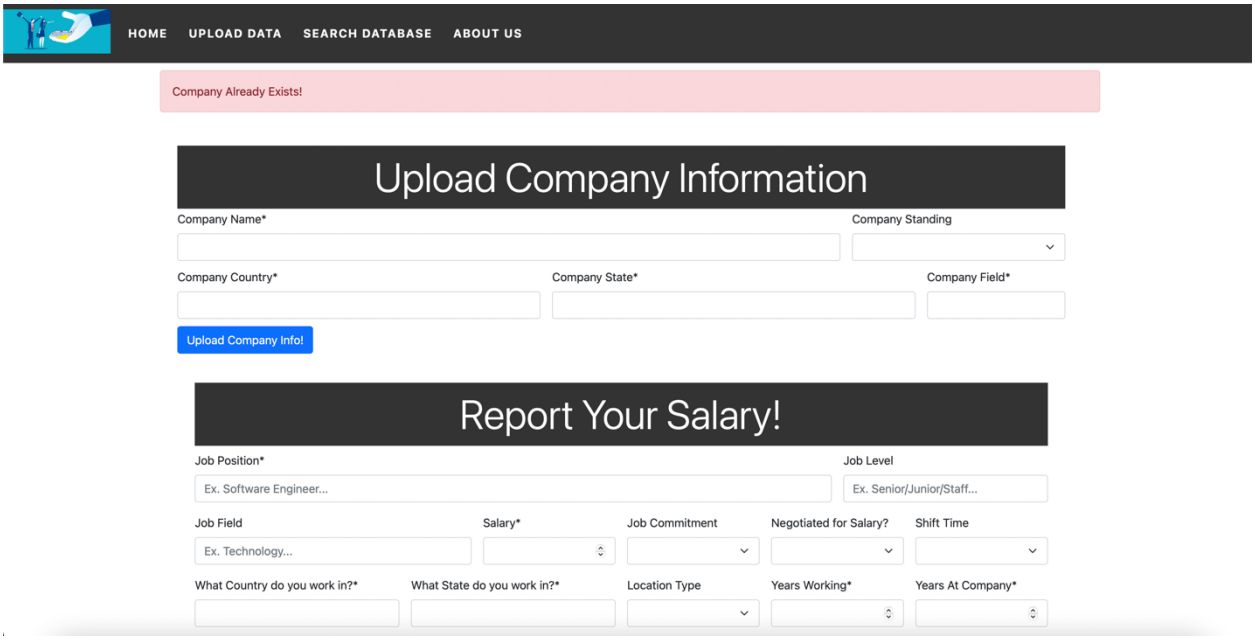
Database general query view:



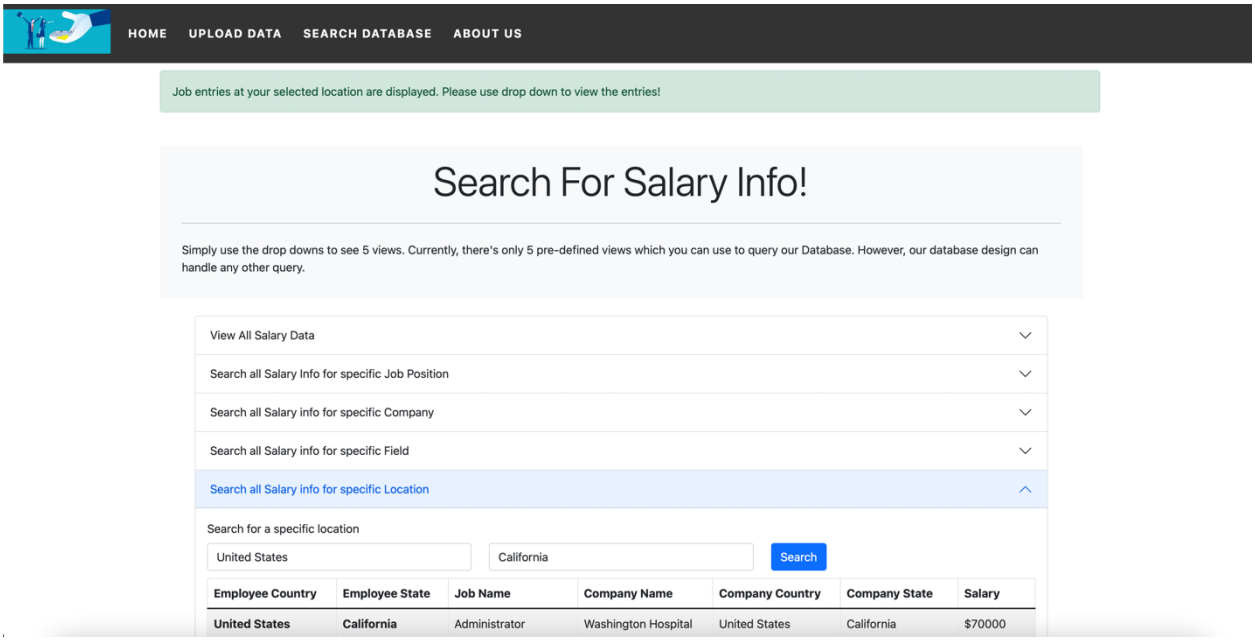
Successful company entry update view:

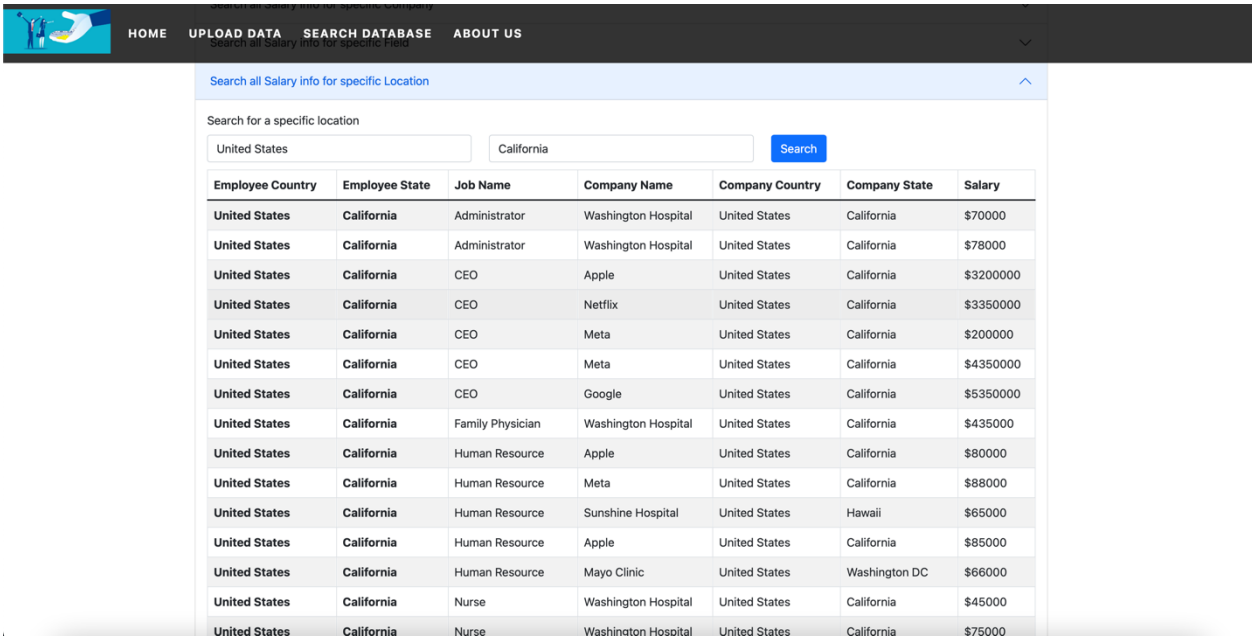


Failed company entry view (due to duplicate data):



Sample query for a specific location entry:





Employee Country	Employee State	Job Name	Company Name	Company Country	Company State	Salary
United States	California	Administrator	Washington Hospital	United States	California	$70000
United States	California	Administrator	Washington Hospital	United States	California	$78000
United States	California	CEO	Apple	United States	California	$3200000
United States	California	CEO	Netflix	United States	California	$3350000
United States	California	CEO	Meta	United States	California	$200000
United States	California	CEO	Meta	United States	California	$4350000
United States	California	CEO	Google	United States	California	$5350000
United States	California	Family Physician	Washington Hospital	United States	California	$435000
United States	California	Human Resource	Apple	United States	California	$80000
United States	California	Human Resource	Meta	United States	California	$88000
United States	California	Human Resource	Sunshine Hospital	United States	Hawaii	$65000
United States	California	Human Resource	Apple	United States	California	$85000
United States	California	Human Resource	Mayo Clinic	United States	Washington DC	$66000
United States	California	Nurse	Washington Hospital	United States	California	$45000
United States	California	Nurse	Washington Hospital	United States	California	$75000

Conclusions

Salary share is fully deployed using heroku. Please refer to implementation & testing sub section for steps and details of deployment. Overall, this project was smoothly executed from the logical design all the way through to the web app creation and deployment. Due to time constraints, many possible features were skipped.

However, having gone through all the required processes, our team came to the realization of how important our design process was. Due to our design process, we were able to implement a database design which allowed us to be more flexible with our front end/web app development. While there were a few features in the design process which may have been done more efficiently (key recognition etc. – please refer to appropriate sections for retrospect and analysis), following the overall process resulted in a headache free development experience.

Appendix

Websites used for requirements analysis:

- Indeed.com
- Glassdoor.com

Github repository link – this contains all scripts used for database design and testing. The database dump (which will allow you to create an instance of our test database) is also included as a database dump file). To view SQL Scripts, please visit github linked below -> click on scripts -> all scripts & queries available:

https://github.com/codermanz/salaryShareProject/

Deployed application link:

https://salaryshareapp.herokuapp.com/home