\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Title of the Project: Job Search Application

Name of the students: Brogan Murray, Corban Larson, Jake Cramer

1. Introduction:

Motivation: why do you choose this problem and this domain? The importance of the application being implemented:

Choosing this problem and domain of job searching was a logical choice when considering all group members being in a studious position and attending the University of Wisconsin Whitewater. Searching for a job is something all people must do to be successful in life, but it can be a daunting task with the number of job search websites and their confusing layouts. While they may show a number of jobs, they often flood the user with too many options that are potentially unrelated to the search, or draw the user away with the numerous UI elements and the websites’ layout. The importance of implementing a proper job search application is key for finding the right job for the user, with filters to narrow down the job search and keep searching simplistic. In this way, the user can get proper jobs for their career.

Describe the application:

Our application will serve as a job listing bulletin board–providing the job details, company information, and contact information. It allows users to create an account, save listings, add/delete/modify listings, and search for specific listings. The program's main purpose will be the search function, which will implement custom or predetermined search filters to tune results based on the Job Listing dataset. In this way, the user leaves the website with or without questions or a need to go on another job searching website. The application is intended to mirror other job search related websites, with an emphasis on providing the user with all of the information they need related to their job search, and preventing confusion around the application’s search and related UI traversal.

Describe the overall organization of the report and task assignment for each team member:

All team members meet 2+ times per week to discuss the current state of the project and what needs to be worked on for the week. During the last few weeks of the semester, group members were required to meet 4-5 times per week (2 during the week and 2-3 times over the weekend) to comply with the functionality goals and requirements for the project. Group members actively communicated with each other on duties required for the project, checking each others’ judgment as the application was built based on their specific duties on the specific day.

Brogan was in charge of the application development, as the work done on the database, database connectivity, and QT implementation was done on his computer 85% of the time. Brogan’s work contributed majorly to the project; setting up the job dataset connection to MariaDB, linking the dataset to the QT web server, and creating the overarching logical coding behaviors server-side and application side for the dataset allowed the building of the application to flow smoothly. Of course, other group members assisted in these overarching creations by assisting with questions or roadblocks that were encountered in the creation of the application and solving them before the next meeting time, but credit is due to Brogan and his efforts towards the project.

Corban’s job was assisting in Brogan’s queries, by being a second hand coder. Primarily, Corban implemented stored procedures and SQL queries within MariaDB in the application. Furthermore, Corban had followed alongside Brogan as the application was developed to ensure stored procedures were functioning properly before being implemented, editing any of the procedures as needed for proper execution within the backend of the database and application. Corban also looked over previous checkpoints, ensuring data was correct before implementation. Corban and Jake worked together with errors or questions that would arise during or after a meeting revolving around the application, figuring them out before the next scheduled meeting.

Jake’s contribution was a big part of being a project manager. Scheduling meeting times and places, as well as laying out what needed to be done within the meet was crucial for the application creations’ success. Jake worked on the back end parts of the project, which included the group presentation and final report documents’ substance. Ideas for the project were primarily passed through Jake, to ensure that the workload for the week/day wasn’t too much for the final implementation of the project. Besides assisting Brogans’ queries around the application creation, Jake also assisted with the stored procedure and database edits for the overall application flow, and ensuring that they work before implementing them into the application.

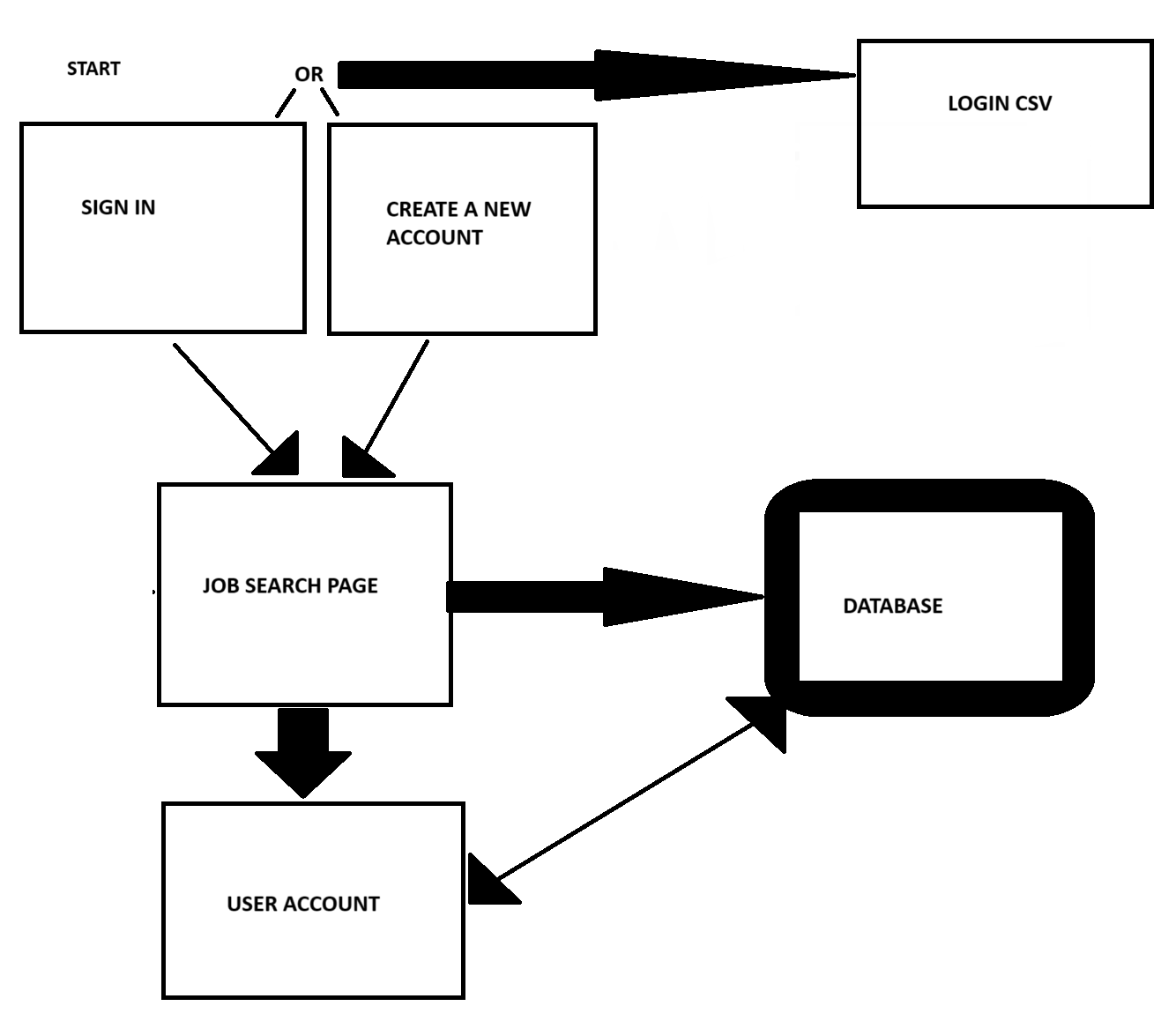
1. Our Implementation
   1. Description of the system architecture

Upon launching the application.exe file, the system architecture begins at the user sign in page. If the user has a preexisting account, they will enter their credentials and proceed to the job search page. If not, they should select the create account button and enter a new username and password. All username and passwords are stored locally in the login.csv file. These values are checked against the inputs for username and password supplied by the user. If incorrect, a windows error message will prompt the user to enter another password. Creating an account will store the newly created username/password pair into the csv file for future reference. Once an account connection has been established, the application opens the job search page.

This page allows the user to search the database for job listings that meet the criteria of the search filters. Three sets of drop down menus specify which job filter is being used, and what other filters can be used with them. Input boxes are paired with these dropdowns to allow for text-based search inputs. Once a user has specified which search filters they want to use and their inputs, they can press the submit button. This button is paired directly to the MariaDB database, allowing the program to submit the query created by the user. In a table box (QTableWidget in QT), the tuples that match the user criteria are displayed. Each column is distinguished by a header, matched to the table in the database. Users are able to click on specific rows on this table display and push the favorite button to add them to their favorites table. Once a user is done searching jobs, they can visit the user account page using the button in the top right corner.

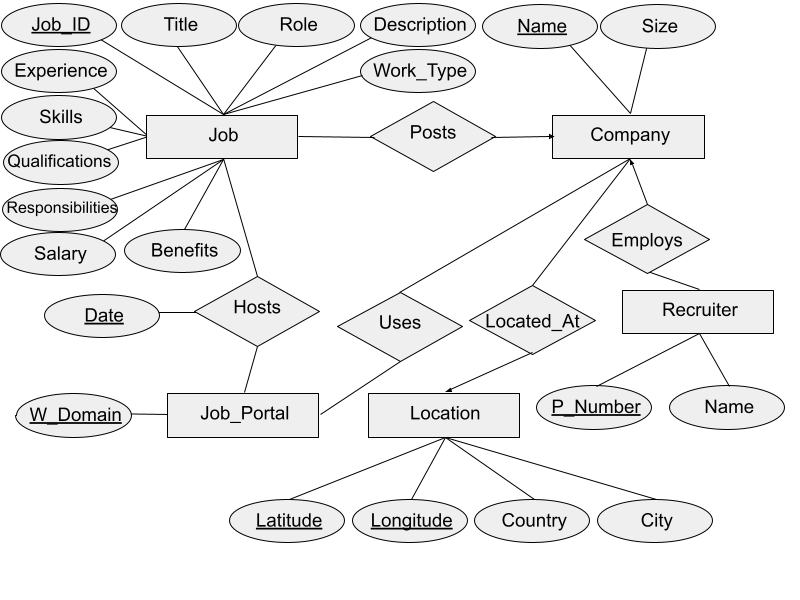
The user account page showcases the favorite listings the user has chosen from the job listing page. The account favorites are directly linked to the database, allowing the table display to showcase all of the listings. Selecting the ‘show’ button fills the table display with all of the tuples from the favorite table in the MariaDB database. If a user wishes to delete a tuple, they can click the row number on the left and select the delete button. This action automatically adjusts the favorites table. Each tuple has a comments attribute on the end of the row. A user can write comments within this cell and select the update button to save whatever information they wish to this tuple. Again, the database is automatically updated with the comments.

QT has enabled us to implement a variety of features within the system architecture itself. Automatic column resizing was a huge advantage to utilizing their built in table widget display. This allows us to dynamically allocate screen sizing to each tuple. Another benefit was the easy navigation due to Windows application pop-up windows. Every time a new page is visited, a new window is created to host the page. If a user wishes to go back, they can simply exit out of the window. Finally, deployment of the system utilizes Windows built in debugger to bundle the necessary libraries with the program. Accessing the program after exporting is as simple as sending the .exe file. Running this program will allow the user to access all of the previously described features.

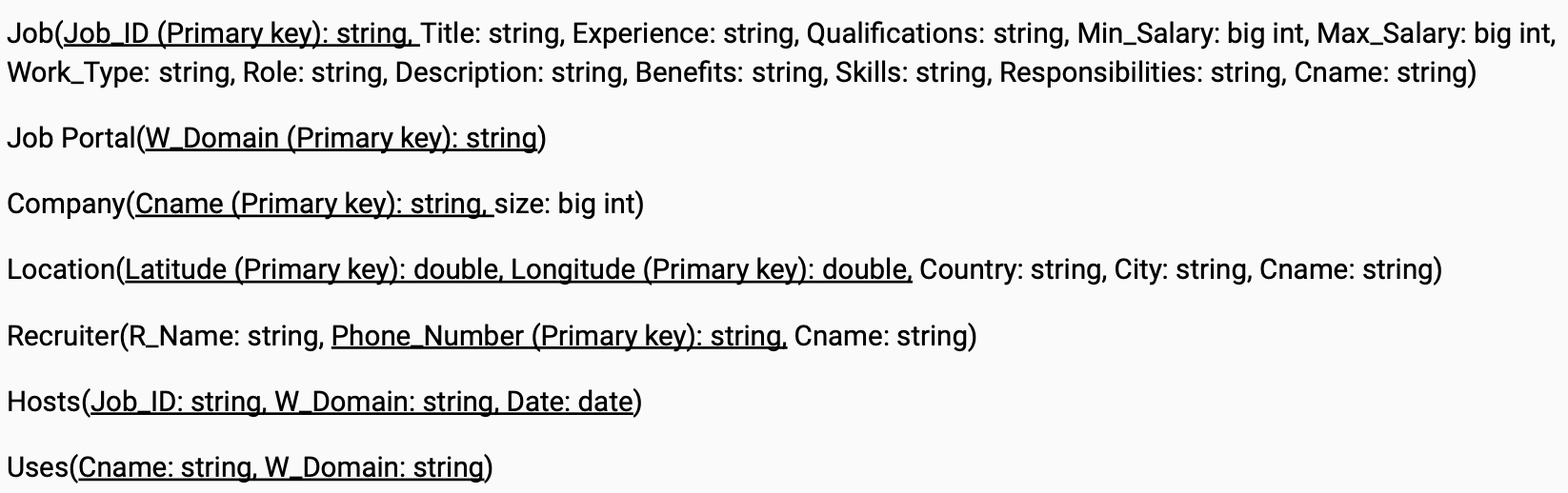
* 
  1. Description of the dataset

Our dataset chosen was the dataset titled “Job Dataset”. We found this dataset on Kaggle, searching for a dataset that allowed a dynamic creation of an application based on the descriptions given of the rows in the dataset. Within the base file, the dataset has 23 columns, ranging from job information (Experience, Qualifications, Salary Range, Benefits, Responsibilities and related Skills) to other specific related data around job information (Contact information, company location with Longitude and Latitude, Company Name and Profile). As a group, we wanted to narrow down some of the information within the dataset, as they weren’t relevant to our application development process (such as job applicant sex preference and the job posting date). Even so, the dataset we now have edited still consists of 1.6 million rows, 23 columns for each row and an overall size of 1.74GB, providing detailed information around the present rows. In this way, the application could be created to reflect the goals of another type of job search website, with a twist on the dataset within our own implementation.

* 1. ER diagram (final version from the previous checkpoint copied here)



* 1. Relational model (final version from previous checkpoint copied here)



* 1. Implementation: description of the prototype

As a group we felt the prototype was to be a working prototype, with as many working functionalities as possible. The prototype had functionality to allow the user to create an account/sign into the application. Upon entering their username and password (or creating a new username and password), the user was let into the application to begin their job search. Once in, the user could select their filter of the dataset which narrows down their search. The user could then type in a specific attribute relating to the filter of the data (example: Company, user types in the company name). If the user only has one filter, they search by that filter. Else, Company could stem with a location. The user would then type in their desired search query (example: Company X, with location in X). Clicking search shows the results with these filters. In the prototype, when the user selects their filter they are required to select “search” for each filter they implement, which then shows the filtered results in a filtered dataset below the search bar area. This had to be done in a linear order, from overfilter->more specific filter.

* 1. Evaluation: describe how you test your application (e.g create testing scenarios or queries or something else, running your application through these scenarios/queries/etc.., checking the returned results and see how the results make sense or not).

While creating the application, many tests had to be carried out to ensure proper execution of queries and connection from the database to the QT application. In our instance, we tested our queries within the MariaDB database before, and while implementing them in the QT application. First, the stored procedure (for example) was put into the database. Then, running a simple query on it was essential before attempting to implement it into the application, as the specified procedure could be tested and edited if needed. In this way, the application’s button behaviors, UI structural elements and overarching display could be edited based on the procedures’ behaviors. Of course, these could be changed if a procedure was not behaving as intended (example: Searching job listings by distance and qualifications). A restructure of a Query or stored procedure could be seen as we developed each one individually. Additionally, a ghost block of text or query call was used in some instances to see if the structure of a procedure was accurately created. Primarily these testing strategies were done within the database, and mirrored within the application which proved to be helpful throughout the creation process.

In addition to the tests mentioned above, we also did time tests on all of the queries as it seemed like some queries took a long time to display. The following table shows the recorded times for all of the currently supported queries. Most have two recorded runs. Some have three recorded times as they have greater variation in their recorded times.

| Query Combination | Time Test 1 | Time Test 2 | Time Test 3 (If Applicable) |
| --- | --- | --- | --- |
| Job Title | 0:25 | 0:15 | 0:17 |
| Job Title  Company Size | 3:30 | 3:25 | N/A |
| Job Title  RecruiterName | 0:21 | 0:19 | N/A |
| Company | 0:17 | 0:15 | N/A |
| Company  WebsiteDomain | 0:17 | 0:15 | N/A |
| Company Size | 4:57 | 4:58 | N/A |
| Company Size  Job Title | 3:14 | 2:40 | N/A |
| Qualifications | 0:14 | 0:24 | N/A |
| Website Domain | 0:20 | 0:20 | N/A |
| Website Domain  Minimum Salary | 0:14 | 0:20 | N/A |
| Website Domain  Company | 0:16 | 0:16 | N/A |
| Website Domain  Minimum Salary Maximum Salary | 0:18 | 0:19 | N/A |
| Recruiter Name | 2:15 | 1:00 | N/A |
| Recruiter Name  Job Title | 0:25 | 0:23 | N/A |
| City\* | - - | - - | N/A |
| City\*  Job Title | - - | - - | N/A |
| City\*  Minimum Salary | - - | - - | N/A |
| City\*  Country | - - | - - | N/A |
| City\*  Minimum Salary  Maximum Salary | - - | - - | N/A |
| Country | 0:10 | 0:03 | 0:15 |
| Country  City\* | - - | - - | N/A |
| Country  Skills | 0:07 | 0:07 | N/A |
| Job Type | 0:50 | 0:16 | 0:34 |
| Job Type  Minimum Salary | 0:34 | 0:35 | N/A |
| Job Type  Minimum Salary  Maximum Salary | 0:29 | 0:15 | 0:35 |
| Skills | 0:24 | 0:44 | 0:19 |
| Skills  Minimum Salary | 0:18 | 0:19 | N/A |
| skills  Minimum Salary  Maximum Salary | 0:16 | 0:15 | N/A |
| Recruiter Phone Number | 2:18 | 2:16 | N/A |

1. Conclusion

What do you learn from this project (both interesting and uninteresting points)? Have you found any relevant database knowledge you have learned in this course helpful and have you encountered any database relevant issues that have been discussed in this course?

A lot of information is to be taken away from this project. Working within a database is work enough, implementing an application with a large dataset is no easy task. Being able to create the connection between the dataset, database, and application is a process to be considered when searching for a future career involving databases, as it is a great resume builder and proves diligence and intelligence around database-related projects. It was interesting to see the different positions group members had to take, and how their individual contribution to the project dynamically changed and contributed evenly to others’ work. While working in the group setting, all group members agree that the newfound database knowledge around queries, connection, and overall implementation must contribute evenly to achieve the desired result of the application. Being enrolled in COMPSCI 366 proved the discussions revolving around databases were important, as they first put our foot into the door when beginning our project and solidified potential issues when creating a database application. One example being that databases are tedious, and have strict syntax can clearly be seen and reflected throughout the creation process. Another example being that databases provide a deeper understanding of information was definitely relevant, as all group members now know the dataset top to bottom throughout the creation process. Furthermore, the creation of the application through complex data manipulations and automation of tasks proved that a database management system must be present in any application.

1. References (if any) (One, the kaggle dataset)

<https://www.kaggle.com/datasets/ravindrasinghrana/job-description-dataset>