THOMPSON RIVERS UNIVERSITY

Algorithms & Data Structure (Fall 2022) - SENG 3110

Household Equipment Repair Program

Luka Aitken (T00663672)

Chris Coulthard (T00661081)

Toma Aitken (T00663680)

December 1, 2022

Table of Contents

List of Figures

List of Tables

1 Abstract

When users need to get in contact with repair people for common household problems, it takes time for users to research and get in contact with the right repair person. This can waste a user's time and effort trying to look for the best possible repair person for their needs. Our goal through this report is to research and develop a household equipment repair program that will allow users to find the most qualified person at an affordable price in their area. The system will be developed using Visual Studios where our team will build a GUI by using multiple different frameworks for different functionalities to achieve user satisfaction. With the results, we were successful in developing the required algorithms and data structures indicated in the project outline but were not fully successful in a fully functional GUI for the users.

Keywords: GUI, C++, coding, programming, development

2 Introduction

Homeownership is becoming more prevalent every year, with homeowners making up approximately 65.5% of the housing market in the USA in 2021. Of that 65.5%, around 34% of people were first-time homebuyers [1]. The increase in home purchases has brought up issues relating to maintenance, especially with first-time buyers who don’t have much experience with owning a home. The types of issues can range anywhere from gas leaks, broken pipes, flooding, and a multitude of other issues, and the first action most people take is to search online for talents that can help solve these issues. There are many companies to pick from, each having multiple different qualified repair people. Searching for the correct talent that provides a good enough service without overcharging requires a lot of time and effort, time that could otherwise be used to fix issues in the house. Prices can also vary between different talents working at different household repair companies, and you may not know who you are getting. When getting these talents to fix household items, their qualifications may not line up with the task that needs to be performed. With this, they might not be able to do anything or might cause more damage making the situation worse. These talents could also charge higher rates to fix these repairs, as they may have a monopoly if they are the only talent qualified to repair something specific in the area [2]. This is likely to cause stress for homeowners that are trying to find the correct talent, and that is where our household equipment repair program will help homeowners communicate with the right talents.

To begin with our interactive household equipment repair program, we will come up with different possible solutions that require minimal effort and ease of use for users, while following the project requirements and constraints. The goal is to come up with a design/solution that can help homeowners find the right talents to fix their household repairs, while at an affordable cost. We will design the program using C/C++ code that will allow users to request help and connect users with the right talent. After choosing the final design, we will create a prototype version of the program, where we will do the necessary tests to ensure that they fit with the project requirements and constraints.

Through this report, we will discuss the problems we faced with the designing and building of the interactive program. We will also discuss other designs/solutions that weren’t used and how they inspire the final design/solution. Additionally, the environmental, societal, safety, and economic considerations, as well as sustainability are some other key factors that will be discussed throughout this report.

3 Design Problem

3.1 Problem Definition

There is a large selection of home repair solutions that homeowners can choose from, but a nearly infinite selection isn't always a good thing. When trying to find the perfect talent to fix the repairs, many options do not provide the right qualities for the problem. A lot of time is taken up by just searching through the plethora of options, looking for a talent that doesn't overcharge, provides a quality product, and can be trusted. All this time being used to find talents is being wasted as it could be used for the repairs themselves [3]. With a "Household Equipment Repair Program", all the time previously being used searching is now used for repairs, allowing homeowners to experience a house in top quality again.

3.2 Design Requirements

3.2.1 Functions

- F-1: The system should let the user add their request and connect with the right talent.
- F-2: The system should display the price for the talent, how much it would cost to hire talent, and talent rates.
- F-3: The system should display important information about the talent and skills the talent can perform by displaying their credentials and experiences.
- F-4: The system should stop users from connecting with unqualified talent to prevent unfair prices/rates and low-quality repairs.

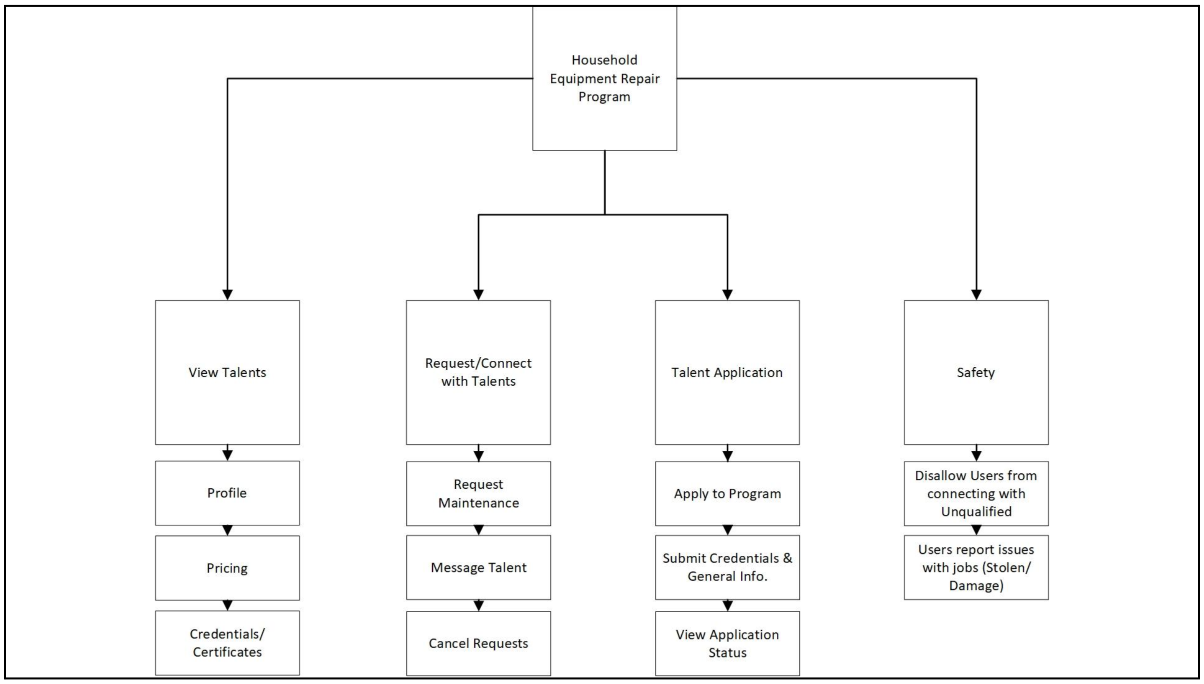

Household Equipment Repair Program
View Talents
Profile
Pricing
Credentials/ Certificates
Request/Connect with Talents
Request Maintenance
Message Talent
Cancel Requests
Talent Application
Apply to Program
Submit Credentials & General Info.
View Application Status
Safety
Disallow Users from connecting with Unqualified
Users report issues with jobs (Stolen/ Damage)

Figure 2.2.1 - Function Tree.

Performance	8
Safety	9
Simplicity	7
Maintenance	7
Affordability	8

Table 2.2.1: Importance Chart.

3.2.2 Objectives

- OB-1: Cost-effective System.
- OB-2: Easy to Use.
- OB-3: Provides an easy way to communicate with qualified talents.
- OB-4: Prevent Homeowners from requesting help from unqualified talent.

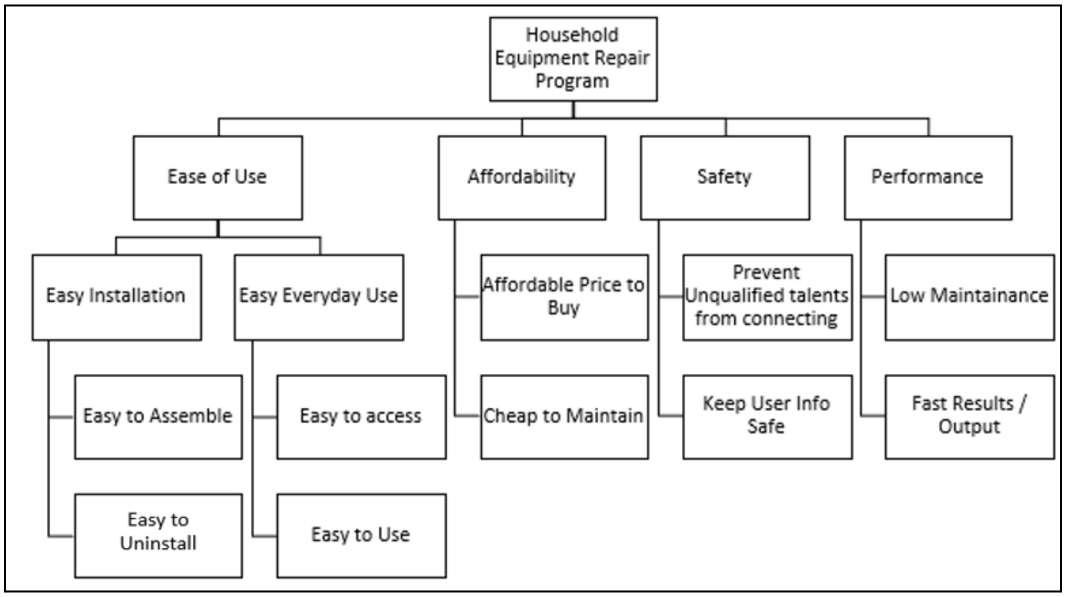


Figure 2.2.2 - Objective Tree.

3.2.3 Constraints

- C-1: The program must use a certain amount of proper algorithms and data structures in C/C++.
- C-2: The program should have GUI for better user interactions and ease of use.
- C-3: The program should be reliable and give correct information about the talent's details and pay rate.
- C-4: The program should be aesthetically pleasing and well organized for the users so it is easy to understand.
- C-5: The economic factors of the program should be cheap to maintain while providing the best services for the user.
- C-6: The program should be sustainable enough for all users using the program.

- C-7: The program should improve communication between users and talents.
- C-8: The program should keep users' information safe by regulating unqualified talents.

3.2.4 Functional Requirements

- FR-1: User Opens the program.
- FR-2: User chooses household repair assist type.
- FR-3: User chooses talent to help assist in repairs based on location, cost, and qualification.

3.2.5 Non-Functional Requirements

- NF-1: Should keep user information safe when using the program.
- NF-2: Give results of qualified talent efficiently and fast.
- NF-3: Should list the most unqualified at the bottom.

3.2.6 Scope & Complexity

- SC-1: The program will be on Windows desktop using C++.
- SC-2: The program will only be accessible in Kamloops, B.C.
- SC-3: The program will have different job options to fix repairs.
- SC-4: The database for the program will be accessed using a text file to save time and money.
- SC-5: The qualifications, location and cost will be assumed by us.

3.2.7 Design Ideas

For our design ideas, each group member was tasked in creating their own design idea for the household equipment repair program where the following functions, objectives, constraints, requirements, scope and complexity were taken into account.

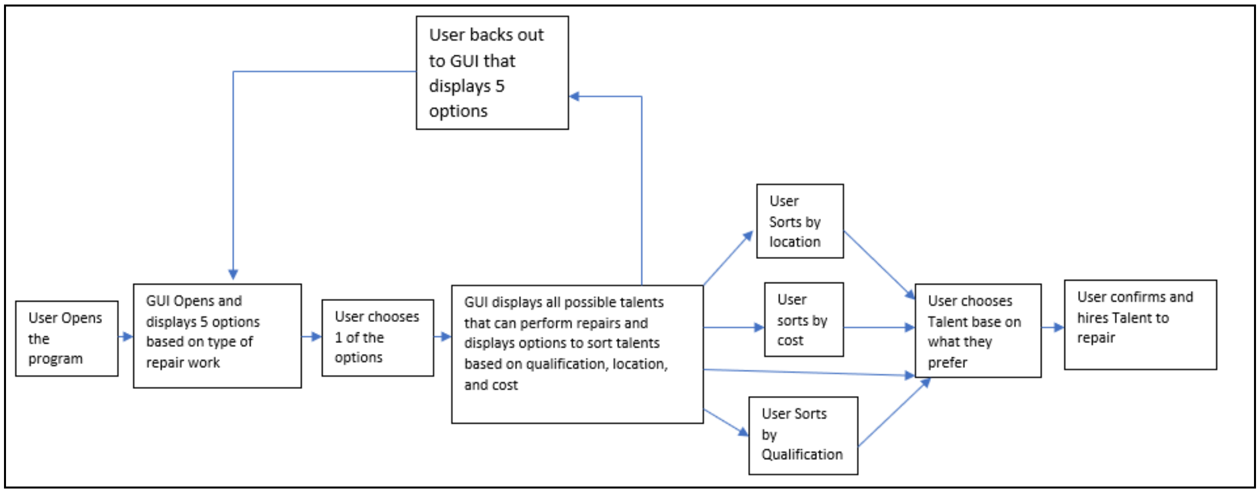

User backs out to GUI that displays 5 options
User Opens the program
GUI Opens and displays 5 options based on type of repair work
User chooses 1 of the options
GUI displays all possible talents that can perform repairs and displays options to sort talents based on qualification, location, and cost
User Sorts by location
User sorts by cost
User Sorts by Qualification
User chooses Talent base on what they prefer
User confirms and hires Talent to repair

Figure 2.2.3 - Toma's Design Idea.

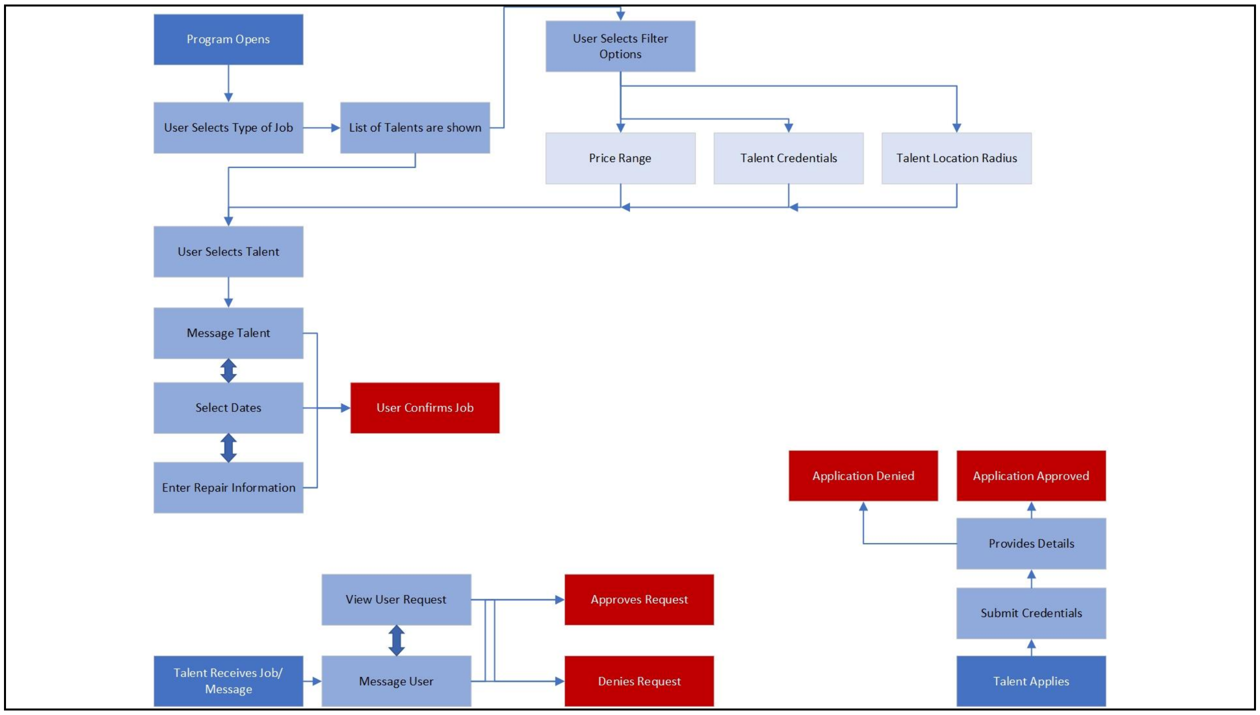


Figure 2.2.4 - Chris's Design Idea.

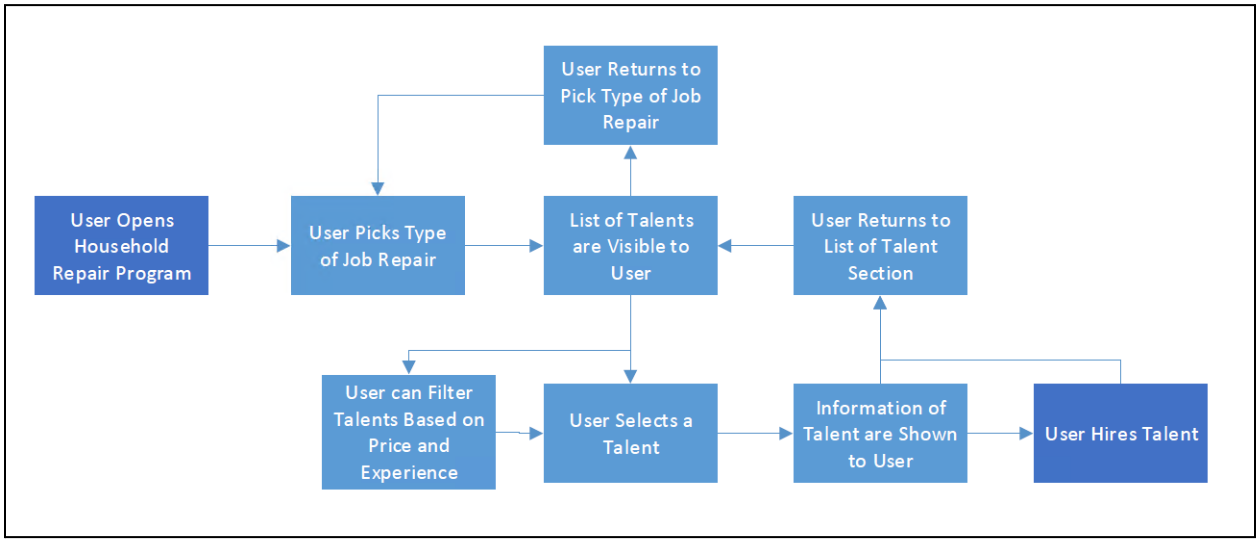

User Returns to Pick Type of Job Repair
User Opens Household Repair Program
User Picks Type of Job Repair
List of Talents are Visible to User
User Returns to List of Talent Section
User can Filter Talents Based on Price and Experience
User Selects a Talent
Information of Talent are Shown to User
User Hires Talent

Figure 2.2.5 - Luka's Design Idea.

With the first design idea from Figure 2.2.3, the user would be able to choose which job type they require. After the user picks a job type, the system will display all possible talents with that job type and the user can sort it by different attributes. After finding the talent, the user would be able to request that talent.

With the second design idea from Figure 2.2.4, the user would be able to choose which job type they require. After the user picks a job type, the system will display all possible talents with that job type and the user can sort it by different attributes. After the user chooses a talent, the user chooses a date for the repair and confirms the request. Also, the talent would get a message to approve or decline user requests. Additionally, the talent can register to the system.

With the third design idea from Figure 2.2.5, the user would be able to choose which job type they require. After the user picks a job type, the system will display all possible talents with that job type and the user can sort it by different attributes. The user is also able to go back if they require a different job type. After finding the talent, the user would be able to request that talent.

4 Solution

4.1 Original Solution

When beginning to create the code for our Household Equipment Repair program, our original idea was to create a linked list that carried each talent that was currently in the program. For each attribute, it would hold the values for each talent and be stored in special arrays depending on the attribute. With the cost of each talent, there would be a Cost array that holds the costs of each talent. With the talents, each talent in the system would be stored in their own array with the special attribute arrays. After each talent has been defined in their own array, there was a linked list to carry all the arrays and outputted each talent like a list, which included their name, job type and cost. After the program outputs, it would ask the user how they wanted the list to be sorted, which this program used as a selection sort.

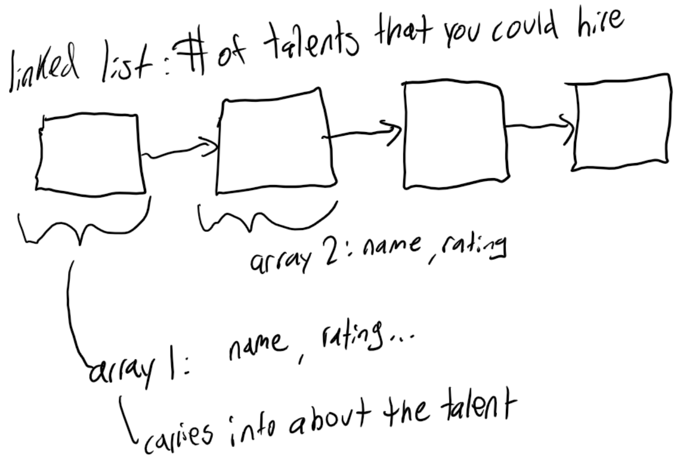

linked list: # of talents that you could hire
array 2: name, rating
array 1: name, rating...
carries info about the talent

Figure 4.1.1 - Original Solution.

The reason why we didn't use this idea is because of how the Household Equipment Repair Program would be sorted. When trying to sort the talents by cost, the cost value in each array would be strings rather than a numeric number, meaning that the system would need to convert the cost value from string to numeric in order to be sorted. Also, the selection sorting algorithm could only organize the attribute array and not the talent arrays, meaning that the program is only sorting by the attribute rather than by the talents. Additionally, selection sort is considered to have more time complexity making the system more unstable.

4.2 Main Solution

For our main solution, our approach was to implement aspects of the three design ideas that we came up with. When making the program to implement the design ideas, we used ideas to allow the user to sign in, choose which job type the user requires, and allow the system to sort the talents of the job type based on different attributes.

With our household equipment repair program, the user would open the application to start looking for talents. When the application opens, the user can either sign up to get access to the program or log in if the user already has an account. Once the user has logged in, they have access to browse all the talents that are currently on the program. The user can click on the job type button to choose which talent they require, which will show only the talents that can perform that particular job. The user can also sort the talents by attributes to view certain aspects of each talent. Once the user has found a talent, they can request that talent to perform the repair. If the user needs to go back to the login page, they can click on the back button to go back to the login page.

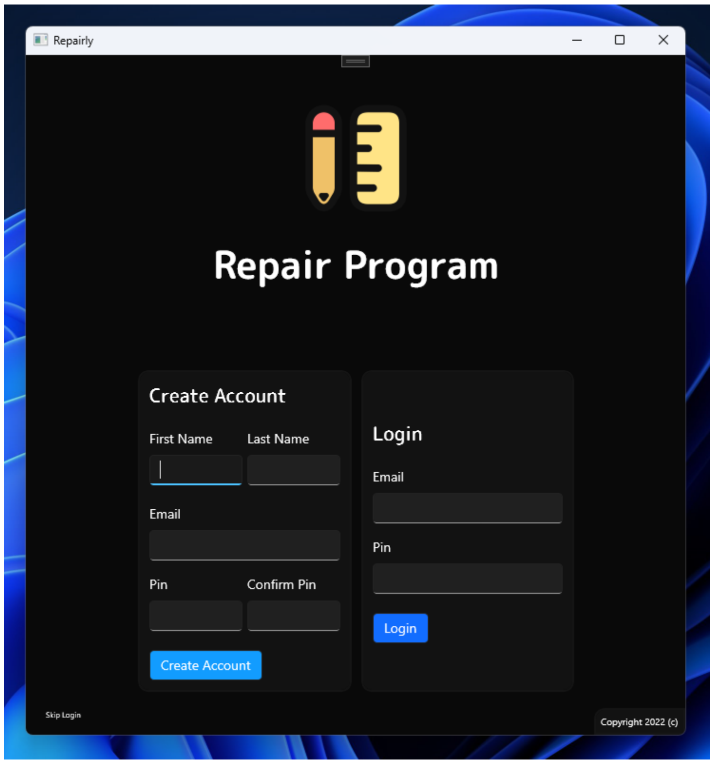
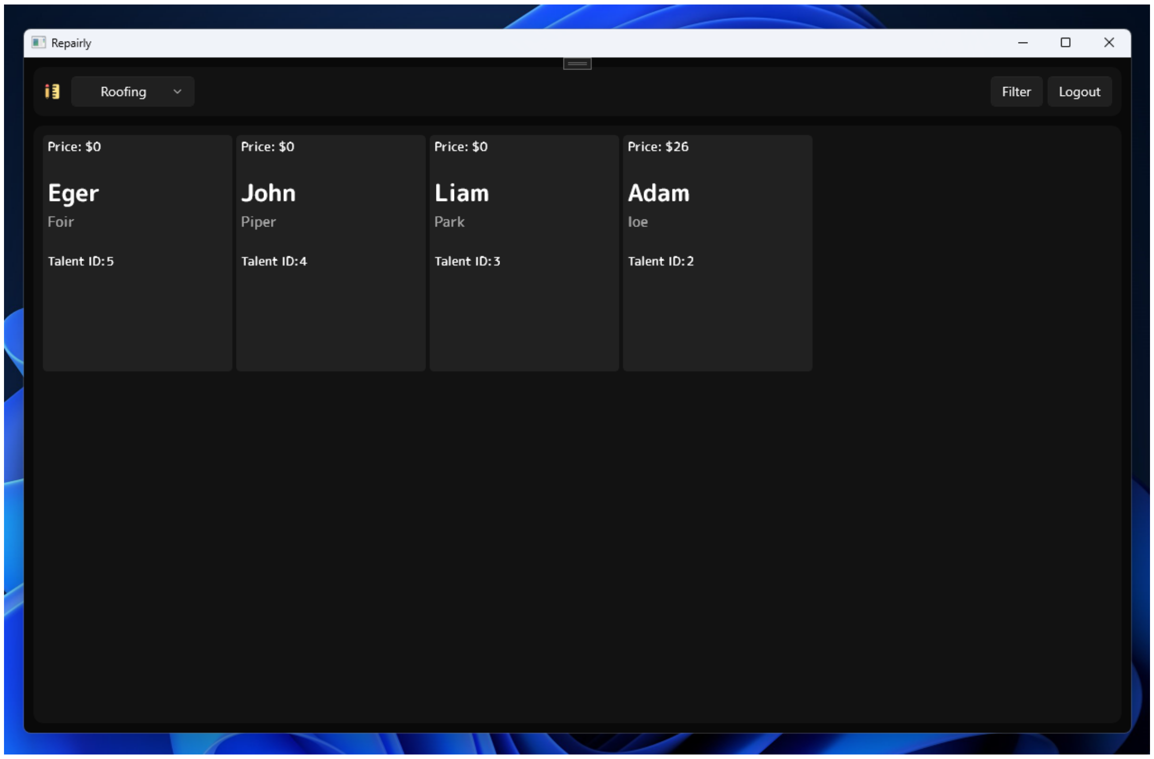


Figure 4.2.1 - Main Login Menu.



Price: $0
Eger
Foir
Talent ID:5

Price: $0
John
Piper
Talent ID:4

Price: $0
Liam
Park
Talent ID:3

Price: $26
Adam
Ioe
Talent ID:2

Figure 4.2.2 - Main Talent Page Menu.

With our code for this program, we used linked lists, classes, arrays, maps, and insertion sort to do the functions for our program. The linked list will hold all the talent information. The classes are used to define the node of the linked list and to display the information of the talents. The arrays are used to move around data of the talents in the maps and insertion sort. The maps are used to connect all the information of each talent and insert it into the linked list. The insertion sort is used to sort attributes depending on the user's requirement for the repair.

		Solutions			
		Original		**Main**	
Criteria	**Weight**	**Score**	**Partial Score**	**Score**	**Partial Score**
Aesthetics	0.25	6/10	0.150	7/10	0.175
Complexity	0.30	7/10	0.210	6/10	0.180
Simplicity	0.20	5/10	0.100	8/10	0.160
Performance	0.25	5/10	0.125	9/10	0.225
Sum	1.00		0.585		0.740

Table 4.2.1 - Decision Matrix.

4.2.1 Features

With our household equipment repair program, we included many features. When the system opens, the user can either sign up for the repair program or log in. If the user wants to sign up they can enter their personal information to get access and once they sign up, they can log in and get access to the system. The user would need to put an email address and a password of their choosing. Once the user creates an account they can log in to the household repair program. For development purposes, we have included a skip button that will skip the main login page and go straight to the talent list. They can view all the talents that are currently in the program to let the user view each talent's information. If the user wants a certain job to do repairs, the user can click on the job type button and choose a selected job type which will only show talents who are qualified to do that type of job. If the user wants to sort each talent by an attribute, they can click on the sort button to sort each talent by cost from cheapest to most expensive, allowing users to easily view how much each talent costs.

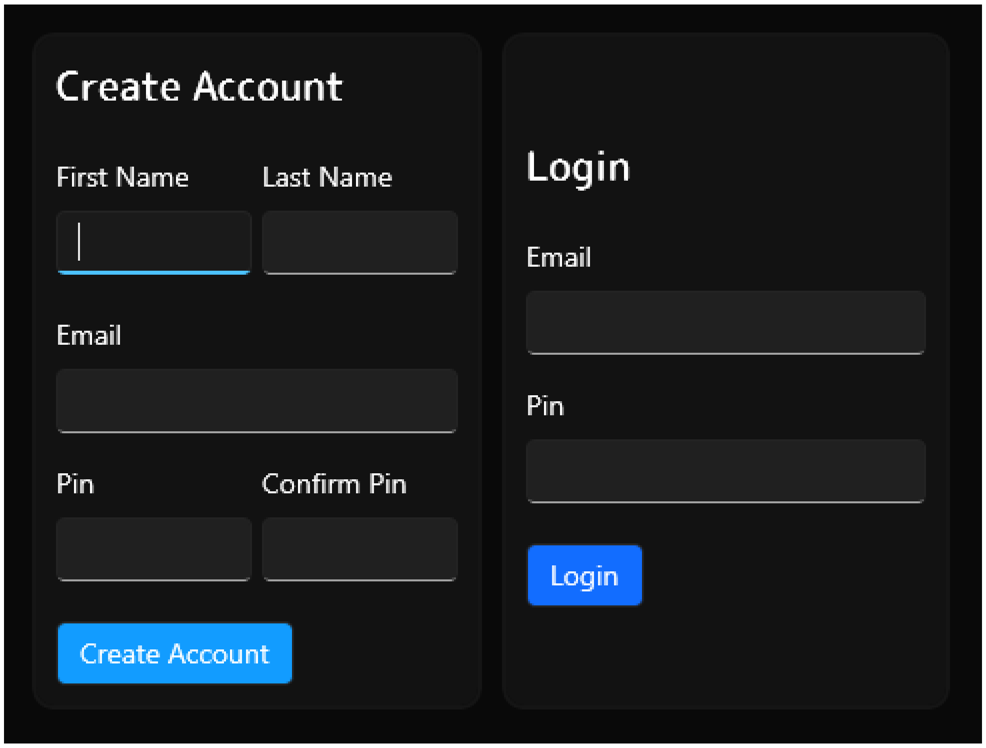


Figure 4.2.3 - Image of the Login and Create Account Feature.

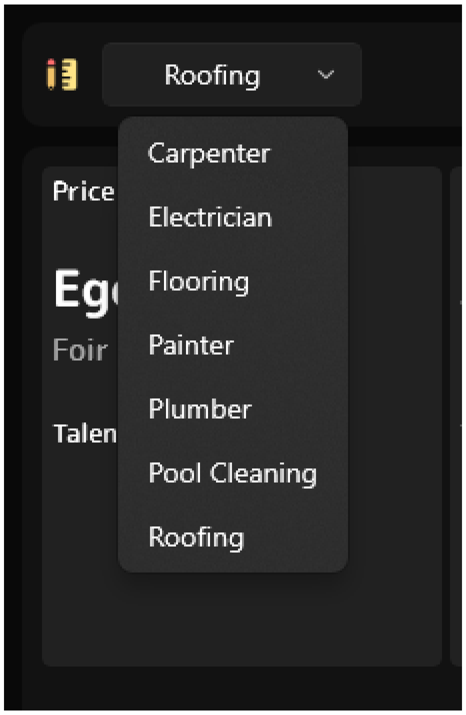
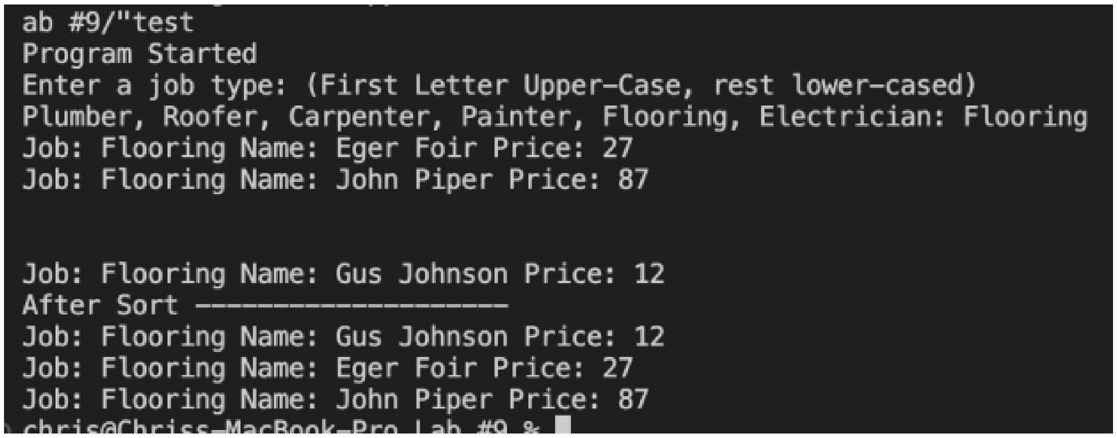


Figure 4.2.4 - Image of Job Type Tab.

Within the program version of the system, the user can ask the program what type of job type they require by allowing the user to type out the certain job they need. Once they enter the job type, the system will display all the talents with their information that can perform that job. It will also sort the talents by an attribute such as cost once the user enters the job type and the system will display the talents that can perform that job by talent Id and by cost.



```
ab #9/"test
Program Started
Enter a job type: (First Letter Upper-Case, rest lower-cased)
Plumber, Roofer, Carpenter, Painter, Flooring, Electrician: Flooring
Job: Flooring Name: Eger Foir Price: 27
Job: Flooring Name: John Piper Price: 87


Job: Flooring Name: Gus Johnson Price: 12
After Sort --------------------
Job: Flooring Name: Gus Johnson Price: 12
Job: Flooring Name: Eger Foir Price: 27
Job: Flooring Name: John Piper Price: 87
chris@Chriss-MacBook-Pro Lab #9 %
```

Figure 4.2.5 - Image of Working Output on Normal C++ File.

4.2.2 Environmental, Societal, Safety, and Economic Considerations

When designing our household equipment repair program, we took into consideration the system to make sure that it was environmentally friendly, safe to use, and inexpensive. We also wanted to take into consideration what elements we included to make the system much easier to use and understand.

For our environmental considerations, we used data types that will make the system faster and less harmful to the environment. If our program used slower data types and sorting algorithms, it would do more harm to the environment by consuming more energy and leading to a bigger carbon footprint. By using faster data types and sorting algorithms like insertion sort, it will take less time meaning that it would produce less energy and reduce our carbon footprint [4].

For our societal considerations, we made our GUI easy to use and easy to understand by using simplicity. By making the GUI user-friendly, using formatting and simple shapes can help the user understand what is happening on the screen as well as have an easier comprehension of the GUI. Also, it can make the GUI more attractive to use instead of other systems that do not use simple and easy-to-understand GUIs [5].

For our safety considerations, we implemented a sign-in screen on our GUI to only allow users with accounts to use the system. With a sign-in screen, we can allow users to fully use our household equipment repair program to check and view different talents and choose the best talent to complete their repair. Also, we can organize each talent by different attributes to reduce the chance of picking a talent that is not qualified for that repair [6].

For our economic considerations, we chose to make an inexpensive and cost-effective program that shows all the important information about each talent and the list of attributes that go along with each talent. Most household repair systems give the most information about each talent and can sort more than just numbers, but are usually behind a paywall and require a subscription to access all these features. Our household repair system is cheap, cost-effective, and doesn't require a subscription to access all the features.

4.2.3 Limitations

With prototyping and designing our household equipment repair program, there are bound to be limitations. For our program, we had a few limitations that affected the final results of our household equipment repair program. The main limitation that we faced was that we were unable to use the main C++ program containing the different algorithms and data structures to work perfectly with our GUI system. There were errors with the GUI to display the correct talents and the sorting algorithm did not function as it should. If given more time and research on GUI systems, we could have figured out the main error and successfully combined the algorithms and data structures with the GUI. There were also some ideas that we wanted to implement in the program, such as a search bar to search for specific talents and a text file that would be able to store all the talents in one file where the program would fetch the talent information from there. Due to time constraints, we were unable to implement these features in our household equipment repair system.

4.2.4 Life-long Learning

When developing this program, we had to learn how to make a GUI for our household repair program. For our program, we learned multiple parts to create our GUI. We chose to do this method because of limitations that occurred when making our program itself. When first choosing our GUI, we had to make sure that it can be developed on any platform such as Windows and IOS systems, and the GUI has lots of tutorials for creating the GUI. With this knowledge, we chose not to use WXWidgets due to the fact that there wasn't enough information on how to create a GUI with it as well as it is complicated to use and download the program itself. We also chose to not use Visual Studios' built-in GUI due to it only working on Windows systems and not on any IOS systems. Using the GUI that we chose, we were able to work with the GUI on both systems and there was a lot of information on how to create a GUI for our household repair program. Additionally, by using this GUI, we were able to make the GUI very user-friendly and very aesthetic compared to how Visual Studios and WXWidget make their GUI. For our GUI, we used three components. These components are Win32, C++/WinRT, and WinUI 3. Win32 is used as the main set of Windows APIs for developing 32-bit applications. C++/WinRT was used to project C++17 language for Windows runtime APIs. WinUI 3 was used as the native UI platform components that are bundled with the Windows App SDK.

5 Team Work

5.1 Meeting 1

Time: September 23, 2022, 4:30 pm to 5:30 pm

Agenda: A list of topics to be discussed in meeting 1

1. Review project requirements.
2. Brainstorm ideas and decide which team member wants to work on.

Team Member	Previous Task	Completion State	Next Task
Chris	N/A	N/A	Brainstorming Ideas
Luka	N/A	N/A	Brainstorming Ideas
Toma	N/A	N/A	Brainstorming Ideas

Meeting Minutes: A list of decisions made in meeting 1

1. All team members will do individual research on topics from the project and report back on ideas in the next meeting.
2. In the next meeting, distribution of tasks for deliverables 1-2: Introduction + Design Problem will take place.
3. Team members will come up with a list of tasks for project management (Gantt Chart).

5.2 Meeting 2

Time: September 26, 2022, 6:00 pm to 7:00 pm

Agenda: A list of topics to be discussed in meeting 2

1. Discuss what type of GUI will be used
2. Review of individual progress on project ideas.
3. Distribution of tasks for deliverables 1-2: Introduction + Design Problem.
4. Create a full list of tasks for the entire project.
5. Discuss possible solutions.

Team Member	Previous Task	Completion State	Next Task
Chris	Brainstorming Ideas	70%	Start writing problem definition. Research about the project.
Luka	Brainstorming Ideas	70%	Creating a Gantt chart. Begin introduction of the report. Research about the project.
Toma	Brainstorming Ideas	70%	Begin introduction of the report. Research about the project.

Meeting Minutes: A list of decisions made in meeting 2

1. All team members will do individual research on other systems similar to the main project.
2. Toma and Luka will begin work on the introduction of the report.
3. Chris will begin work on the problem definition.
4. Each team member will look and provide information for the functions, objectives, and constraints of the project in the report.

5.3 Meeting 3

Time: October 6, 2022, 5:00 pm to 6:30 pm

Agenda: A list of topics to be discussed in meeting 3

1. Reviewing introduction and problem definition of the report.
2. Reviewing and adding to the functions, objectives, and constraints of the project in the report.

Team Member	Previous Task	Completion State	Next Task
Chris	Start writing the problem definition. Research about the project.	90%	Review and add to the functional, non-functional, scope and complexity. Come up with own design idea
Luka	Creating a Gantt chart. Begin introduction of the report. Research about the project.	85%	Review and add to the functional, non-functional, scope and complexity. Come up with own design idea
Toma	Begin introduction of the report. Research about the project.	90%	Review and add to the functional, non-functional, scope and complexity. Come up with own design idea

Meeting Minutes: A list of decisions made in meeting 3

1. Each team member will go through and add to the functional, non-functional, scope and complexity of the project in the report.
2. Each team member will come up with their own design idea for the project and show the other team members in the next meeting.
3. Next meeting will discuss the beginning of the code and deciding on how the GUI will be built.

5.4 Meeting 4

Time: October 10, 2022, 7:00 pm to 10:30 pm

Agenda: A list of topics to be discussed in meeting 4

1. Review the functional, non-functional, scope and complexity sections in the report.
2. Review individual design ideas of each team member.
3. Perform any necessary edits before deliverables 1-2: Introduction + Design Problem are handed in.

Team Member	Previous Task	Completion State	Next Task
Chris	Review and add to the functional, non-functional, scope and complexity. Come up with own design idea	90%	Hand in Deliverable 1-2. Begin and research ideas for the project code.
Luka	Review and add to the functional, non-functional, scope and complexity. Come up with own design idea	90%	Hand in Deliverable 1-2. Begin and research ideas for the project code.
Toma	Review and add to the functional, non-functional, scope and complexity. Come up with own design idea	90%	Hand in Deliverable 1-2. Begin and research ideas for the project code.

Meeting Minutes: A list of decisions made in meeting 4

1. Next meeting will begin the coding of the project.
2. Each team member will look into GUI for C++.

5.5 Meeting 5

Time: October 14, 2022, 5:00 pm to 6:00 pm

Agenda: A list of topics to be discussed in meeting 5

1. Choose what kind of GUI program that we want to use.
2. Begin and choose what kind of data types we want to use.

Team Member	Previous Task	Completion State	Next Task
Chris	Hand in Deliverable 1-2. Begin and research ideas for the project code.	100%	Develop and code datatype to store information. Research on using wxWigets for GUI.
Luka	Hand in Deliverable 1-2. Begin and research ideas for the project code.	100%	Begin writing the solution and features on the report. Research on using visual studios for GUI.
Toma	Hand in Deliverable 1-2. Begin and research ideas for the project code.	100%	Develop and code datatype to store information. Research and write the considerations on the report.

Meeting Minutes: A list of decisions made in meeting 5

1. Chris will look more into using wxWidgets as the main GUI development.
2. Luka will look more into using visual studios as the main GUI development.
3. Toma will look into implementing three different data types for the main project.
4. Deliverables 1-2 corrections for the report will be added to and corrected in the report.

5.6 Meeting 6

Time: November 4, 2022, 7:00 pm to 10:00 pm

Agenda: A list of topics to be discussed in meeting 6

1. Finish developing the first and second datatype used in the project code (Arrays and Linked Lists).
2. Decide on what GUI will be used.

Team Member	Previous Task	Completion State	Next Task
Chris	Develop and code datatype to store information. Research on using wxWigets for GUI.	80%	Begin developing the starting menu of the GUI.
Luka	Begin writing the solution and features on the report. Research on using visual studios for GUI.	75%	Continue to write in the report. Provide assistance to the development of the GUI.
Toma	Develop and code datatype to store information. Research and write the considerations on the report.	80%	Begin implementing a third datatype to the project code.

Meeting Minutes: A list of decisions made in meeting 6

1. Team members decided best to use visual studios for the development of the GUI.
2. Datatype 1 will be using Arrays and datatype 2 will be using Linked Lists.

5.7 Meeting 7

Time: November 10, 2022, 4:00 pm to 5:00 pm

Agenda: A list of topics to be discussed in meeting 7

1. Review current progress of the project.

Team Member	Previous Task	Completion State	Next Task
Chris	Begin developing the starting menu of the GUI.	50%	Create a login menu to access the repair program.
Luka	Continue to write in the report. Provide assistance to the development of the GUI.	70%	Begin filling out other sections of the report and provide assistance to GUI programming.
Toma	Begin implementing a third datatype to the project code.	60%	Combine the 3 data types together into a program.

Meeting Minutes: A list of decisions made in meeting 7

1. Team members decided that once data types 1 and 2 are implemented that we will meet and figure out data type 3(sorting method).

5.8 Meeting 8

Time: November 17, 2022, 5:00 pm to 7:00 pm

Agenda: A list of topics to be discussed in meeting 8

1. Review current progress of the project.

Team Member	Previous Task	Completion State	Next Task
Chris	Create a login menu to access the repair program.	90%	Begin mapping process for talents in repair program.
Luka	Begin filling out other sections of the report.Provide assistance to GUI.	80%	Begin writing the conclusion of the report.
Toma	Combine the 3 data types together into a program.	85%	Begin mapping process for talents in repair program.

Meeting Minutes: A list of decisions made in meeting 8

Team members will meet next when arrays, linked lists, and insertion sort are implemented together.

5.9 Meeting 9

Time: November 17, 2022, 5:00 pm to 7:00 pm

Agenda: A list of topics to be discussed in meeting 9

1. Discuss the progression of the GUI.
2. Finish the implementation of the 3 data types (Arrays, Linked Lists, Insertion Sort).

Team Member	Previous Task	Completion State	Next Task
Chris	Begin mapping process for talents in repair program.	85%	Combine the mapping process to the 3 data types.
Luka	Begin writing the conclusion of the report.	85%	Begin writing the conclusion of the report.
Toma	Begin mapping process for talents in repair program.	85%	Combine the mapping process to the 3 data types.

Meeting Minutes: A list of decisions made in meeting 9

1. Team members will meet next meeting to combine all code into the GUI.

5.10 Meeting 10

Time: November 21, 2022, 5:00 pm to 8:00 pm

Agenda: A list of topics to be discussed in meeting 10

1. Figure out how to combine C++ code into the GUI.

Team Member	Previous Task	Completion State	Next Task
Chris	Combine the mapping process to the 3 data types.	95%	Implement and test repair program. Go through the report and update information.
Luka	Begin writing the conclusion of the report.	85%	Implement and test repair program. Go through the report and update information.
Toma	Combine the mapping process to the 3 data types.	95%	Implement and test repair program. Go through the report and update information.

Meeting Minutes: A list of decisions made in meeting 10

1. Once the mapping process is finished, all code will then be moved and implemented into the GUI.

5.11 Meeting 11

Time: November 28, 2022, 5:00 pm to 7:00 pm

Agenda: A list of topics to be discussed in meeting 11

1. Continuing to implement the main code into the GUI.

Team Member	Previous Task	Completion State	Next Task
Chris	Implement and test repair program. Go through the report and update information.	80%	Create PowerPoint and make any final changes to the program/report.
Luka	Implement and test repair program. Go through the report and update information.	80%	Create PowerPoint and make any final changes to the program/report.
Toma	Implement and test repair program. Go through the report and update information.	80%	Create PowerPoint and make any final changes to the program/report.

Meeting Minutes: A list of decisions made in meeting 11

1. Chris will continue to work on the code/GUI until it works to a degree.
2. Toma and Luka will begin to start making the PowerPoint slides for presentation.

5.12 Meeting 12

Time: November 29, 2022, 5:00 pm to 8:00 pm

Agenda: A list of topics to be discussed in meeting 12

1. Go over the entire project (Repair Program, Report, PowerPoint Slides).

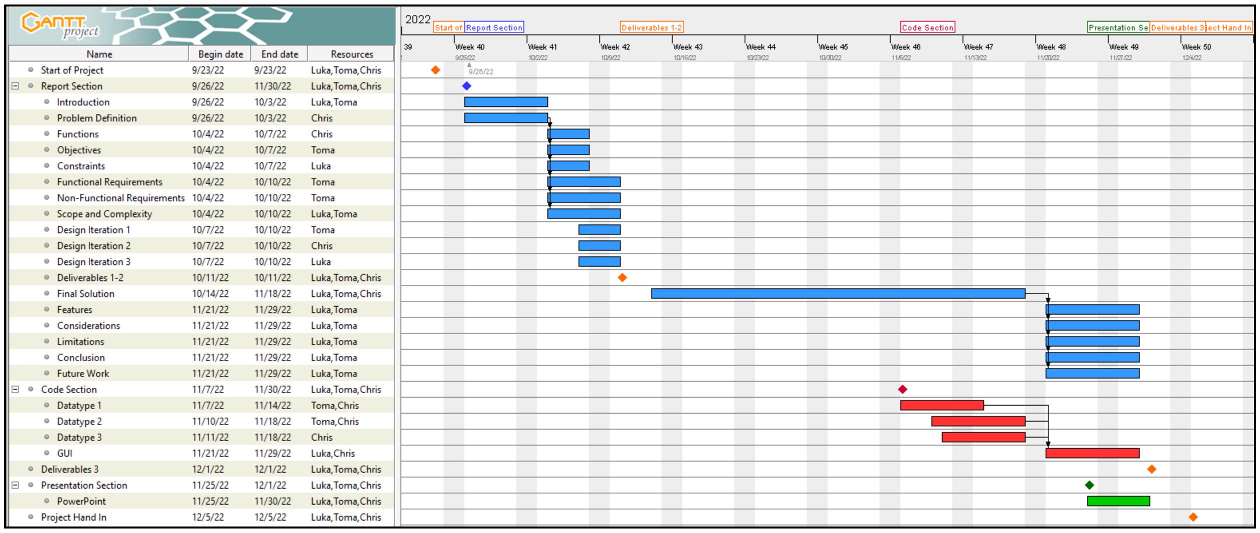
Team Member	Previous Task	Completion State	Next Task
Chris	Create PowerPoint and make any final changes to the program/report.	90%	Final Testing/Hand-In
Luka	Create PowerPoint and make any final changes to the program/report.	90%	Final Testing/Hand-In
Toma	Create PowerPoint and make any final changes to the program/report.	90%	Final Testing/Hand-In

Meeting Minutes: A list of decisions made in meeting 12

1. Due to the GUI not fully working, we will hand in both the C++ file with the different algorithms and data structures and the GUI.

6 Project Management

The following Gantt chart shows the different tasks that were done in the household repair project as well as the time each task should take by the end of the project hand-in. The slack time for each task is included with the main task.



Name	Begin date	End date	Resources
Start of Project	9/23/22	9/23/22	Luka,Toma,Chris
Report Section	9/26/22	11/30/22	Luka,Toma,Chris
Introduction	9/26/22	10/3/22	Luka,Toma
Problem Definition	9/26/22	10/3/22	Chris
Functions	10/4/22	10/7/22	Chris
Objectives	10/4/22	10/7/22	Toma
Constraints	10/4/22	10/7/22	Luka
Functional Requirements	10/4/22	10/10/22	Toma
Non-Functional Requirements	10/4/22	10/10/22	Toma
Scope and Complexity	10/4/22	10/10/22	Luka,Toma
Design Iteration 1	10/7/22	10/10/22	Toma
Design Iteration 2	10/7/22	10/10/22	Chris
Design Iteration 3	10/7/22	10/10/22	Luka
Deliverables 1-2	10/11/22	10/11/22	Luka,Toma,Chris
Final Solution	10/14/22	11/18/22	Luka,Toma,Chris
Features	11/21/22	11/29/22	Luka,Toma
Considerations	11/21/22	11/29/22	Luka,Toma
Limitations	11/21/22	11/29/22	Luka,Toma
Conclusion	11/21/22	11/29/22	Luka,Toma
Future Work	11/21/22	11/29/22	Luka,Toma
Code Section	11/7/22	11/30/22	Luka,Toma,Chris
Datatype 1	11/7/22	11/14/22	Toma,Chris
Datatype 2	11/10/22	11/18/22	Toma,Chris
Datatype 3	11/11/22	11/18/22	Chris
GUI	11/21/22	11/29/22	Luka,Chris
Deliverables 3	12/1/22	12/1/22	Luka,Toma,Chris
Presentation Section	11/25/22	12/1/22	Luka,Toma,Chris
PowerPoint	11/25/22	11/30/22	Luka,Toma,Chris
Project Hand In	12/5/22	12/5/22	Luka,Toma,Chris

Figure 6.1 - Gantt Chart.

7 Conclusion and Future Work

In conclusion, we achieved many considerable designs on how to create a household equipment repair program that can let users request talents to perform repairs. First off, we came up with a couple of design ideas that could be used to make the best possible household equipment repair program, while following all the constraints. Our objective was to build a program that is easy to use and integrated into a GUI for better aesthetics for the user.

From our research, the biggest challenges for homes are managing the home itself and the things that could go wrong. Some such incidents could be gas leaks, broken pipes, flooding, and a multitude of other issues. When an incident occurs, calling talent to repair the damages can help, but some talents could falsely advertise their services. Also, they can make repair costs higher than usual, making the homeowner pay more money for low-quality service. Furthermore, it can make the homeowner frustrated and confused in trying to pick the right talent to fix the repair. With the help of household equipment repair programs, we can reduce the number of unpleasant incidents that can occur. Using these programs can reduce the chances of choosing a talent that is not qualified to perform the repairs that the homeowner requires. Also, it can save money by showing recommended talents that can perform the repairs at affordable prices and no false advertisements [7].

For our future work, we would like to have a fully functional GUI with the algorithms and data structures that we created to be fully implemented with each other. To add more talents into our household equipment repair program, we would like to develop a database that will be able to add as many talents as you want to the program. This could be done by using a text file where the program can grab all talents information from one place. We also would like to have a menu that will allow talents to sign up onto the program and enter details about themselves such as job type, contact information, and pricing. When clicking onto a talent we would like to make a menu that will open up and show all the details of the talent like a profile page. If the user wants to hire the talent, there would be a hire button. When clicked it would tell the user that a request has been sent to the talent for hire. We would also like to have a search bar at the top of the program that would allow users to search up a particular talent instead of users searching through an entire list of talents.

8 References

1. R. Schnabel. “Homeownership facts and statistics 2022.” Bankrate. https://www.bankrate.com/insurance/homeowners-insurance/home-ownership-statistics/ (Accessed October 6, 2022)
2. H. Srinivasan. “Home Repair Costs Will Be Higher Next Year - Here’s What to Expect.” Better Homes & Gardens. https://www.bhg.com/home-improvement/advice/maintenance-repair/average-cost-of-home-repairs/ (Accessed October 6, 2022)
3. “Advantages and disadvantages of handyman services.” Lacrosseva. https://lacrosseva.org/advantages-and-disadvantages-of-handyman-services/ (Accessed October 10, 2022)
4. S. Podder. “How Green Is Your Software?.” Harvard Business Review. https://hbr.org/2020/09/how-green-is-your-software (Accessed October 29, 2022)
5. M. Roomi. “5 Advantages and Disadvantages of GUI | Drawbacks & Benefits of GUI.” Hitechwhizz. https://www.hitechwhizz.com/2021/02/5-advantages-and-disadvantages-drawbacks-benefits-of-gui.html (Accessed November 10, 2022)
6. “7 Benefits of Single Sign-On.” Dyanix. https://www.linkedin.com/pulse/7-benefits-single-sign-on-dyanix/ (Accessed November 12, 2022)
7. “Picking The Right Professional For House Repairs.” Commercial Construction & Renovation. https://ccr-mag.com/picking-the-right-professional-for-house-repairs/ (Accessed November 27, 2022)

9 Appendix

9.1.1 Functional Code on DevC++

```
#include <iostream>

#include <string>

#include <vector>

#include <map>

using namespace std;

class UserTalent;

class Node;

// Used for organizing talent information

class UserTalent {

public:

int userID;

string firstName;

string lastName;

// 'jobTypes' contains a jobType (string (key)) and a jobCost (int) map<string, int> jobTypes;

UserTalent() {

this->userID = 000;

this->firstName = "";

this->lastName = "";

};

UserTalent(int userID, string firstName, string lastName)

{

this->userID = userID;

this->firstName = firstName;

this->lastName = lastName;

}
```

```
void setFirstName(string newFirstName) {
this->firstName = newFirstName;
}
void setLastName(string newLastName) {
this->lastName = newLastName;
}
void addJobType(string jobType, int jobCost) {
jobTypes.insert(pair<string, int>(jobType, jobCost));
}
void removedJobType(string jobType) {
jobTypes.erase(jobType);
}
void updateJobType(string jobType, int newCost) {
jobTypes.erase(jobType);
jobTypes.insert(pair<string, int>(jobType, newCost));
}
int getJobCost(string jobType) {
try
{
return jobTypes.at(jobType);
}
catch(const std::exception& e)
{
std::cerr << e.what() << '\n';
return 0;
}
}
```

```
bool checkJob(string jobType) {
if (jobTypes.find(jobType) == jobTypes.end()) {
return false;
} else {
return true;
}
}
};
// Linked List Node
class Node {
public:
UserTalent userTalent;
Node* next;
};
class TalentLinkedList {
public:
Node* head;
Node* sorted;
void addTalent(UserTalent userTalent) {
Node* newTalent = new Node();
newTalent -> userTalent = userTalent;
newTalent -> next = head;
head = newTalent;
}
void insertionSort(Node *node, string jobType) {
sorted = NULL;
Node* current = node;
```

```
while (current != NULL) {
Node* next = current -> next;
//Check if Talent Offers Selected JobType
if ((current -> userTalent).checkJob(jobType)) {
if (sorted == NULL || (sorted -> userTalent).getJobCost(jobType) >= (current ->
userTalent).getJobCost(jobType)) {
current -> next = sorted;
sorted = current;
} else {
Node* newCurrent = sorted;
while (newCurrent -> next != NULL && (newCurrent -> next ->
userTalent).getJobCost(jobType) < (current -> userTalent).getJobCost(jobType)) {
newCurrent = newCurrent -> next;
}
current -> next = newCurrent -> next;
newCurrent -> next = current;
}
}
current = next;
}
node = sorted;
};
void displaySortedTalentList(Node* head, string jobType) {
while (head != NULL) {
UserTalent tempUser = head -> userTalent;
if (tempUser.checkJob(jobType)){
cout<<"Job: " <<jobType <<" Name: " <<tempUser.firstName <<" "
<<tempUser.lastName <<" Price: " <<tempUser.getJobCost(jobType);
}
head = head -> next;
cout <<endl;
}
}
};
int main() {
//Initiated Program
printf("Program Started\n");
//Create UserTalent Objects
UserTalent tal1 = UserTalent(001, "Gus", "Johnson");
UserTalent tal2 = UserTalent(002, "Adam", "Ioe");
UserTalent tal3 = UserTalent(003, "Liam", "Park");
UserTalent tal4 = UserTalent(004, "John", "Piper");
UserTalent tal5 = UserTalent(005, "Eger", "Foir");
//Add Jobs
tal1.addJobType("Plumber", 12);
tal1.addJobType("Flooring", 12);
tal2.addJobType("Carpenter", 43);
tal2.addJobType("Plumber", 39);
tal2.addJobType("Roofer", 26);
tal3.addJobType("Electrician", 30);
tal4.addJobType("Plumber", 43);
tal4.addJobType("Flooring", 87);
tal5.addJobType("Painter", 23);
tal5.addJobType("Carpenter", 14);
```

```
tal5.addJobType("Flooring", 27);
tal5.addJobType("Electrician", 32);
tal5.addJobType("Pool Cleaner", 120);
//Create Nodes
TalentLinkedList talentList;
talentList.head = NULL;
talentList.addTalent(tal1);
talentList.addTalent(tal2);
talentList.addTalent(tal3);
talentList.addTalent(tal4);
talentList.addTalent(tal5);
string jobX;
cout<<"Enter a job type: (First Letter Upper-Case, rest lower-cased) "<<endl;
cout<<"Plumber, Roofer, Carpenter, Painter, Flooring, Electrician: ";
cin>>jobX;
talentList.displaySortedTalentList(talentList.head, jobX);
talentList.insertionSort(talentList.head, jobX);
cout<<"After Sort --------------------" <<endl;
talentList.displaySortedTalentList(talentList.sorted, jobX);
return 0;
}
```