Task Management System

*Nathan Gillingham*

**Computer Science Projects Check Sheets**

IS THE FINAL PROJECT OF AN ‘A’ LEVEL STANDARD?

(a) Definitely yes.

(b) Probably yes but a little more depth required.

(c) Potentially yes but the depth of objectives and/or technical solution need improving.

(d) At this stage no, however, there is potential for an appropriate level.

(e) Definitely no, a significant amount of work required on the objectives and/or technical solution.

**ANALYSIS – 9 marks**

|  |  |  |  |
| --- | --- | --- | --- |
| **Background to the Project (Core)** | **Pg No?** | **Yes** | **No** |
| Give a clear statement that describes the problem area and the specific problem being solved or investigated. |  |  |  |
| Describe the problem area in sufficient detail to enable a third party to understand the problem being solved or investigated. |  |  |  |
| Description of the organisation |  |  |  |
| Number of employees (could only be one) |  |  |  |
| Nature of work the organisation carries out |  |  |  |
| What the ACTUAL PROBLEM is |  |  |  |
| **Evidence of Analysis (Core)** | **Pg No?** | **Yes** | **No** |
| Interview(s) |  |  |  |
| Observation Notes |  |  |  |
| Analysis of Existing Paperwork |  |  |  |
| Surveys/Questionnaires |  |  |  |
| Any other form of evidence analysis |  |  |  |
| **Objectives (Core)** | **Pg No?** | **Yes** | **No** |
| Functionality Objectives |  |  |  |
| Input Objectives |  |  |  |
| Processing Objectives |  |  |  |
| Storage Objectives |  |  |  |
| Output Objectives |  |  |  |
| Any other form of Objectives |  |  |  |
| **Log/Diary of Research (Core)** | **Pg No?** | **Yes** | **No** |
| Key points of research carried out |  |  |  |
| Book/Magazine research |  |  |  |
| Websites accessed |  |  |  |
| Prototyping taken place |  |  |  |
| Critical path analysis |  |  |  |
| **Description of the Current System (Desirable)** | **Pg No?** | **Yes** | **No** |
| A description of the current system along with where it’s being used, e.g. A library database within a school environment. |  |  |  |
| How are the computers used? |  |  |  |
| An outline of the current problems (probably a bullet pointed list). |  |  |  |
| Outline the specific areas of the problem that you are going to tackle. |  |  |  |
| **Identification of the end-user/supervisor (Desirable)** | **Pg No?** | **Yes** | **No** |
| End-user name |  |  |  |
| Name of your supervisor and their interest in your problem area. |  |  |  |
| **Prospective user(s) (Desirable)** | **Pg No?** | **Yes** | **No** |
| List of different types of users + number of each type of user |  |  |  |
| Purpose of system for each type of user |  |  |  |
| **Modelling of the Problem (Desirable)** | **Pg No?** | **Yes** | **No** |
| Data Flow Diagram (Current and Proposed Systems) |  |  |  |
| ER Model |  |  |  |
| Analysis Data Dictionary |  |  |  |
| State Transition Diagram |  |  |  |
| Scientific/Mathematical Models or Formulae |  |  |  |
| Graph or Network Model |  |  |  |
| **Any Further Research (Desirable)** | **Pg No?** | **Yes** | **No** |
| Example 1 |  |  |  |
| Example 2 |  |  |  |
| Example 3 |  |  |  |
| **Proposed Solution Details (Supplementary)** | **Pg No?** | **Yes** | **No** |
| Software requirements of proposed system |  |  |  |
| Hardware requirements of proposed system |  |  |  |
| **Acceptable Limitations (Supplementary)** | **Pg No?** | **Yes** | **No** |
| Limitations agreed with your end user. |  |  |  |
| **Data Volumes (Supplementary)** | **Pg No?** | **Yes** | **No** |
| Input volumes |  |  |  |
| Output volumes |  |  |  |
| Storage volumes |  |  |  |
| Estimation of File Size |  |  |  |

**DOCUMENTED DESIGN – 12 marks**

|  |  |  |  |
| --- | --- | --- | --- |
| **High Level Overview (Core)** | **Pg No?** | **Yes** | **No** |
| Structure/Hierarchy Chart(s) + explanation. |  |  |  |
| System Flowchart(s) + explanation. |  |  |  |
| IPSO Chart + explanation. |  |  |  |
| Data Flow Diagram(s) + explanation. |  |  |  |
| Object/Class Diagram(s) + explanation. |  |  |  |
| **Description of Algorithms (Core)** | | | |
| **PROCESSING** algorithms – in pseudo code or structured English + explanation. |  |  |  |
| **Description of Data Structures (Core)** | | | |
| Data Structure Designs and explanations of each, e.g. array of records, queues, etc. |  |  |  |
| **Database and/or File Designs (Core)** | | | |
| File Structure and Organisation designs + explanations, e.g. text files, direct files. |  |  |  |
| Database Designs, e.g. DDL Designs, Table designs, ER diagrams. |  |  |  |
| Query Designs, e.g. Cross table SQL designs. |  |  |  |
| **Design of User Interface (Core)** | | | |
| Samples of ANNOTATED screen shots + explanations. |  |  |  |
| **Hardware Selection/Design *(if appropriate)* – (Desirable)** | | | |
| Explanation of the suitability of the hardware to be used. |  |  |  |
| Explanation of how the hardware is to be used within the project. |  |  |  |
| **System Security and Integrity of Data (Supplementary)** | | | |
| User Names and Passwords – purpose described. |  |  |  |
| Design of any encryption used. |  |  |  |
| Integrity of data. Validation of user inputs. |  |  |  |
| Integrity of data. Maintaining referential integrity within a database. |  |  |  |
| **Log of stages taken during the investigation (Investigative project only).** | | | |
| For an investigative project, log the stages taken during the investigation and how this affected the coding. |  |  |  |

**TECHNICAL SOLUTION – 42 marks *(see note at end of Group C grid)***

|  |  |  |  |
| --- | --- | --- | --- |
| **Completeness of Solution (15 marks)** | Pg No? | Yes | No |
| Have all objectives, as set out in the Analysis section, been completed? |  |  |  |

***NOTE – Even if you have completed all your objectives, the final mark for this section is based on the completion AND the suitability of the objectives.***

|  |  |  |  |
| --- | --- | --- | --- |
| **Techniques Used (27 marks)** | Pg no? | Yes | No |
| Program commented to highlight **programming techniques** used. |  |  |  |
| Appropriate naming conventions used. |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **GROUP A : Have you successfully included the following programming techniques in your project?**  **If Yes, input name of unit** | Which  Form?  Pg No? | Yes | No |
| **Creating a complex data model in database (e.g. several interlinked tables)** |  |  |  |
| Cross table parameterised SQL.  *(Querying your database using more than one table).* |  |  |  |
| Aggregate SQL Functions  *(Use of MIN, MAX, SUM, AVG, etc. e.g. SELECT AVG(Salary) FROM Employee;)* |  |  |  |
| User generated DDL script.  *(Creating your database using Data Definition Language).* |  |  |  |
| **Hash tables, lists, stacks, queues, graphs, trees or structures of equivalent standard.** |  |  |  |
| Graph/Tree Traversal |  |  | 0 |
| List Operations |  |  |  |
| Stack/Queue Operations |  |  |  |
| Linked List Maintenance |  |  |  |
| **File(s) organised for Direct Access** |  |  |  |
| Hashing |  |  |  |
| **Complex Scientific/Mathematical/Robotics/Control/Business Model** |  |  |  |
| Advanced Matrix Operations |  |  |  |
| Recursive Algorithms |  |  |  |
| Complex user-defined algorithms (e.g. optimisation, minimisation, scheduling, pattern matching) or equivalent difficulty. |  |  |  |
| Merge Sort or similar efficient sort. |  |  |  |
| **Complex user-defined use of Object-Oriented Programming (OOP) model.** |  |  |  |
| Dynamic generation of objects based on complex user-defined use of OOP model including: |  |  |  |
| Classes |  |  |  |
| Inheritance |  |  |  |
| Composition |  |  |  |
| Polymorphism |  |  |  |
| Interfaces |  |  |  |
| **Complex Client-Server Model** |  |  |  |
| Server-side scripting using request and response objects and server-side extensions for complex client-server model. |  |  |  |
| Calling parameterised Web service APIs and parsing JSON/XML to service a complex client-server model. |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **GROUP B : Have you successfully included the following programming techniques in your project? – If Yes, input name of unit** | Which Form?  Page No? | Yes | No |
| **Creating a simple data model in databases (e.g. 2 or 3 interlinked tables)** |  |  |  |
| Single table or non-parameterised SQL.  *(Querying your database with no parameters using one table).* |  |  |  |
|  |  |  |  |
| **Multi-dimensional arrays, dictionaries, records** |  |  |  |
| Bubble Sort |  |  |  |
| Binary Search |  |  |  |
| **Reading and writing to/from a Text File** |  |  |  |
| **Files organised for sequential access** |  |  |  |
| **Simple scientific/mathematical/robotics/control/business model** |  |  |  |
| Simple user-defined algorithms (e.g. a range of mathematical/statistical calculation). |  |  |  |
| **Simple OOP Model** |  |  |  |
| Generation of objects based on simple OOP model including: |  |  |  |
| Classes |  |  |  |
| Inheritance |  |  |  |
| **Simple Client-Server Model** |  |  |  |
| Server-side scripting using request and response objects and server-side extensions for simple client-server model. |  |  |  |
| Calling Web service APIs and parsing JSON/XML to service a simple client-server model. |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **GROUP C : Have you successfully included the following programming techniques in your project? – If Yes, input name of unit** | Which Form?  Page No? | Yes | No |
| **Single-dimensional array** |  |  |  |
| Linear Search. |  |  |  |
| **Appropriate choice of simple data types** |  |  |  |
| Simple mathematical calculations (e.g. average) |  |  |  |
| **Single table database** |  |  |  |
| Non-SQL table access |  |  |  |

**NOTE for Technical Solution (Techniques Used)**

You will gain the highest marks for programming techniques used in **Group A**.

If your program mainly uses **Group A** programming techniques then you will probably not use any **Group C** techniques, but may still use some Group B programming techniques.

**TESTING – 8 marks**

***You do NOT need to provide evidence of testing all aspects of the system, you should provide evidence of testing of those aspects which clearly demonstrate that the project fulfils its purpose.***

|  |  |  |  |
| --- | --- | --- | --- |
| **Suitable list of tests that include:** | Pg no? | Yes | No |
| Description of its purpose; |  |  |  |
| The test data used; |  |  |  |
| The expected test outcome; |  |  |  |
| The **actual** outcome with a sample of the evidence, e.g. ANNOTATED screen shots of before and after the test. |  |  |  |
| The **actual** outcome with a sample of the evidence, e.g, Video evidence of before and after the test – time indicated on test table |  |  |  |

**EVALUATION – 4 marks**

|  |  |  |  |
| --- | --- | --- | --- |
| **Explanations/discussions relating to the final project as laid out below.** | Pg no? | Yes | No |
| Suitable explanations of how well the objectives have been met and how this was achieved. |  |  |  |
| Discussion on how the solution could be improved. |  |  |  |
| Analysis of feedback from the third party who was involved at the analysis stage. |  |  |  |

# 

[**Analysis**](#_dxf4yxnlg3v5) **12**

[Background to Project (Core)](#_x88toe5s84m0) 12

[Description of Current System (Desirable)](#_9n767s2r2ns) 12

[Identification of the end user/supervisor (Desirable)](#_29h3hqs4luof) 12

[Prospective users (Desirable)](#_t4r1yorf4zuz) 12

[Evidence of Analysis (Core)](#_5zaikkekmtng) 12

[Further Research (Desirable)](#_wyfz34iptavp) 12

[Modelling of the Problem (Desirable)](#_oy6543wcyb9g) 12

[Proposed Solution Details (Supplementary)](#_kn6g6thluel6) 12

[Acceptable Limitations (Supplementary)](#_av1hjz3ixm5v) 12

[Data Volumes (Supplementary)](#_md9mq5kfone9) 13

[User Requirements](#_a043pdv5a1k0) 13

[Objectives (Core)](#_ijzct25f7ggu) 13

[Main Objective](#_wsh1x1r5bh3p) 13

[Input Objectives (numbered)](#_5xvqguukq160) 13

[Output Objectives](#_f1mwiplnbkab) 13

[Processing Objectives](#_2pe1couwli5t) 13

[Storage Objectives](#_qv8kvrqm9vb5) 13

[Additional Objectives](#_gy44anwvr4ui) 13

[Log/Diary of Research (Core)](#_vtwsspr52rag) 13

[**Documented Design**](#_8zwsf1sgq90r) **14**

[System Flow Chart/Diagram(s)](#_9cnr055tf2kh) 14

[Structure/Hierarchy Chart – Modular Structure Diagram](#_kfhhgyq18yvq) 14

[IPSO chart(s)](#_u0eg5n18cnpo) 14

[Data Flow Diagram (covered in Analysis) and/or Object/Class diagram](#_ilmjr1vtvcpz) 14

[Description of Algorithms](#_fyyti2deo2ly) 14

[Description of Data Structures](#_mrwydo1qsntw) 14

[Database and/or File Design](#_wzwz2f7w95dx) 15

[Design of User Interface](#_hi5ceo25h9o9) 15

[System Security and Integrity of Data](#_qez398dgkbpw) 15

[Investigative Project](#_nd8y4sbtw06g) 16

[**Technical Solution**](#_54rxiab9jihh) **16**

[**Testing**](#_wz3blcyn28d0) **18**

[Test Table](#_8sa7ezh9xkjj) 18

[Screenshots (annotated)](#_vcr1hoyg983b) 19

[Final Testing](#_sxyyy9j34dgj) 19

[Test Table](#_pna6vuc4lgpw) 19

[**Evaluation**](#_6g89ivz4mewe) **20**

# 

# **Analysis**

## **Background to Project (Core)**

The project I am to undertake aims to create a more organised and resourceful task manager that fills gaps many other programs fail to. Features such as priority queues and complex hierarchy user systems are not included in popular programs such as Google Classroom. These limited capabilities mean businesses are not suited to using existing task managers, as current hierarchy systems only allow for closed groups that suit a school system, not a business with a completely different structure. Furthermore, the inability to weight tasks means many can be distracted and overwhelmed with many small tasks while important, large tasks are buried and the user underdelivers. Weighting in a priority queue promises to fix this problem by properly showing which tasks should be undertaken first and the most time is given, but no such feature is evident in existing projects.

The project will also feature various groups that users can be added to. This allows setting tasks to multiple users at once as opposed to a laborious process. Features like this are available in similar projects, however for my purposes this must be adapted slightly. As the user system is based around a hierarchy system, only users of a higher role will be able to add users to their group. This is the feature which is missing from competitors. Additionally, tasks cannot be set to individuals easily, they mut be added to a lone group first. My software will eliminate this problem, as users will be able to set tasks to each other, which will, in turn, allow users to set themselves tasks to stay organised in themselves.

There will also need to be a login system as the tasks will be specific to the user, the specific credentials for each employee can be added by an admin of the company or automatically if a large data set of employees is sent but can also be amended while the system is in use when new employees join the company.

In conclusion, the system I am to implement is a needed improvement for a company because current systems of task setting, such as google classroom and Microsoft teams, do not allow priority queueing of tasks, hierarchies of users, and setting tasks to individual users.

## **Description of Current System (Desirable)**

Mitie currently uses a task management system completely independently of any internal infrastructure or intranet. This means each user must set all their own tasks and manage them independently. The most important objective is to connect all users so tasks may be shared.

The software used is not bespoke and is completely separate from Mitie as a company. This means that any issues with the program or changes required must be brough up with the third-party, which is incredibly slow and usually gets no response or change in the software. This was an issue raised in the survey sent out, users felt their feedback had no influence on the software, and so no issues they had were addressed. Of course, this issue will be completely solved by a bespoke piece of software, and with the trial with a subset of users, I will be able to add features, and remove any bugs which may affect user satisfaction.

## **Identification of the end user/supervisor (Desirable)**

Mitie Group PLC is a British strategic outsourcing and energy services company. It provides infrastructure consultancy, facilities management, property management, energy and healthcare services. It has a head office at The Shard in London and more than 200 smaller offices throughout the United Kingdom and Ireland. They are the end user for this project, as their current task management system is unsuitable for a linked business and can only be used independently by separate employees, so all tasks must be individually set, which isn’t time efficient and allows for mistakes in the specification in tasks, as opposed to one centrally set task to multiple workers. For example, a manager could give a task in a meeting and if one employee misconstrues the objective, and then details the task wrong in their organisational tool, they have nothing to refer to. What prevents this is to be given, in writing, the task in a system which can be constantly referred to.

## **Prospective users (Desirable)**

The program will initially be trialled with 50 employees in a sub office to identify any immediate errors with the system before it is rolled out to the wider company. Of these employees there will be two main prospective users, the general manager and the employees who will make up the bulk of the users.

The general manager, and some positions below them, will mainly focus on setting tasks and creating groups. They will not have many tasks set to them unless it is from themselves, so they will focus on efficiently and accurately setting tasks to users and groups. In the initial stages of the program’s use, there will be 5 employees of manager level.

The lower-level employees, however, will focus on the displaying of tasks being set to them, and the functionality of completing those tasks. There will be 45 employees of around this level, although some will obviously be slightly more senior than others.

## **Evidence of Analysis (Core)**

I first conducted an interview with a manager at Mitie named Jon. This will allow me to focus directly on the needs of the user. The manuscript is below and can also be found as a separate document.

Interview – Thursday 8th September 2022:

Nathan: What are the faults with your current task management system used within the company?

Jon: The current system lacks several key features such as the ability to set priority tasks, this severely diminishes productivity as it makes it harder to sort by which tasks must be done sooner. This can cause several tasks to be missed out or to be done too early which can cause further changes to the schedule. This situation is further worsened by the fact that there is no proper calendar provided, this adds to the complexity of managing when tasks must be done.

Nathan: For implementing priority, should this figure be set by the person assigning others the task, or by individuals?

Jon: I would largely prefer for the priority of tasks to be set by management as this would mean that the whole team would be on the same page as to what needs to be done. This will largely streamline the work as it will ensure that everyone is working on the same task.

Nathan: How many tasks on average can you get in a week?

Jon: Anything ranging from 5-12 depending on size so the workload can get very cluttered.

Nathan: How should we set permissions for who can set who tasks?

Jon: Within the company, there is a hierarchy, people that are more important such as project managers can set tasks for their team. People should be able to set their own tasks, however, people shouldn’t have the ability to set tasks for those above them.

Nathan: And does the same apply for adding users to groups?

Jon: Yes, I think it’s best if employees can only add people they manage to their groups, not above them.

Nathan: And how do you want to manage the groups you have made?

Jon: I definitely want to be able to see who’s in them, and what tasks I’ve set to each group.

Nathan: Do you have an existing ID within the company for logins?

Jon: Yes, each employee is given an ID, from 0001 to 9999 which identifies them, but a password should also be in place.

Nathan: Going back to priority, how should we display which tasks need to be done first?

Jon: Nothing crazy, I think listing is probably best, maybe colour coding the most important task but that’s not a necessity.

Nathan: Okay I think that’s it, thanks.

Jon: No problem

End

The main feedback of the interview can be condensed down to:

* Hierarchy system for setting tasks
* Being able to self-set tasks
* Build on the existing ID system for a login system.
* An ordered list of tasks based on their priority
* A GUI that avoids clutter
* Display of users and tasks which belong to groups

I will use these to form my objectives for the project as a whole and will need to consider how these will be implemented considering limitations due to time, and coding ability.

I felt that for a representative perspective on the needs of lower-level employees, due to the sheer number and impracticality of interviewing a substantial proportion of them, it was better to send a survey, where the results could then be collated and summarised.

The following survey was sent to the employees’ part of the initial test:

* Are you satisfied with current system in place?
* What are your main problems with the current system?
* How many tasks would you be set in an average week?
* In which ways would you like tasks to be sorted?
* How many different people usually set you tasks?
* Are there any features you particularly wish to see in the new model?

The general summary of the responses was as follows:

92% of employees weren’t satisfied with the current system. This question was chosen to confirm the need for new system in place of the old.

The main problems with the current system are:

* Necessity to spend time individually setting each task assigned to employees
* Inability to sort efficiently through tasks set
* Tasks aren’t weighted in their priority
* There isn’t a central system so issues can’t be raised with Mitie, the software is simply third party and non-bespoke.

This question was chosen to ensure the end software fixed all these problems, so that users were not still dissatisfied.

The average employee is set 8 tasks in a week, ranging from short, hour-long tasks, to ones that are done in segments taking the whole week. This was chosen for data volume information.

The top three ways in which tasks should be sorted is:

* Priority
* Date Set
* Date Due

This can be directly implemented in the displaying of tasks in the project

Employees usually have three managerial figures who regularly set them tasks, although this figure is very volatile, some employees have one manager setting them all their tasks, whilst some are regularly set tasks by 5 managers. This was again asked for data volume purposes.

The features which users wish to see in the new software is:

* Ability to set-self tasks
* To be part of groups
* Print current tasks to paper, sorted.

Being able to print tasks had not been discussed previously, and upon speaking to employees it’s clear many find it helpful to have a physical representation of what they need to do in the day/week. Whilst I cannot print from the code, I will be able to write tasks to a file which can be printed and provide ways of sorting them.

Apart from this, it’s clear that most of the proposed features fulfil the necessary requirements for users, and since many employees responded with “N/A” or answers to that effect, I’m sure the scope is sufficient.

## **Further Research (Desirable)**

The best example of a task manager that already exists, I believe, is Google Classroom. Whilst some features would not work for my project, or some are missing, Google Classroom is a good example to follow.

One great feature is the ability of one person to set a task to multiple people easily. This is done in the form of a classroom, where there are people with permission to set tasks and those who are in the classroom receive them. Whilst I require a more complicated hierarchy system, the concept is there and all that needs to be added is different levels of permissions.

Chart

Description automatically generated with medium confidence

Example of the classroom.

Features that google classroom does not include, are the ability to set yourself tasks, and the ordering of tasks based on priority. Tasks are ordered purely by the due date, but this is not always helpful as people can get distracted by small tasks and are unable to dedicate time to important tasks, as detailed in the interview.

To ensure nothing was missing from my solution, I also researched some necessities of projects like this from [www.ntaskmanager.com](http://www.ntaskmanager.com). Some of the most important features in task manages listed by this website are:

Task Scheduling- this has been planned for as tasks require a date due and will take record of when the task was set, so that tasks can be scheduled and sorted accordingly.

Priority- This feature has been detailed thoroughly in the project analysis and will be fully implemented in both tasks and users.

Task Integrity- This means that once tasks have been completed, they will still be stored in the database. This is so that referring to previous tasks for key dates and information is available and will be implemented in the code.

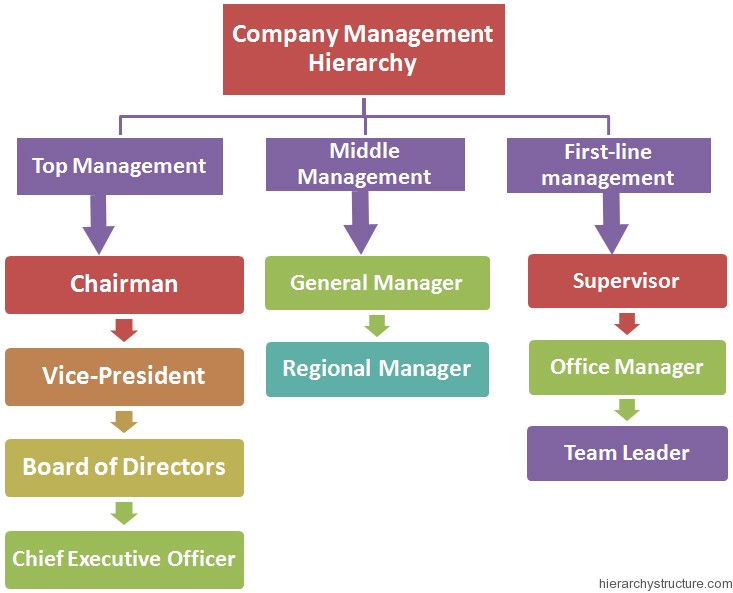
Another website, buildarray.com, listed Intuitive organisation. Whilst this seems to be met with sorting of tasks by important factors, they actually detail filtering instead of sorting. Whilst I believe these two work well in conjunction, filtering by itself wouldn’t necessarily help see which task is the most important, or due first. However, once sorting is enabled, filtering should be fairly easy to implement.

Sorting

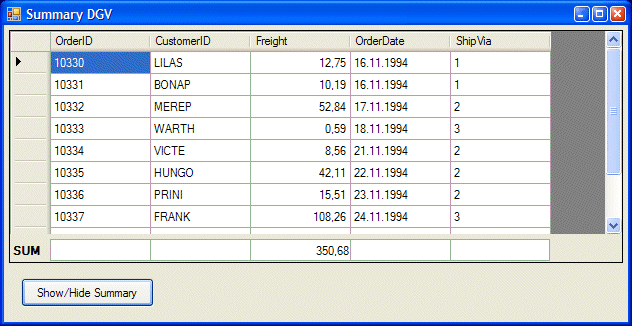
From research on sorting algorithms from [www.brilliant.org](http://www.brilliant.org), I have decided to use quick sort and modifications I will make to it in order to sort my tasks for outputs. Quicksort is a quick and effective sorting algorithm that organises a list of objects using a divide-and-conquer strategy. It is frequently used in practise and is generally regarded as one of the most effective sorting algorithms. Quicksort is preferred over other sorts for a variety of factors, including its effectiveness, simplicity, and capacity to sort lists in-place, which eliminates the need for additional memory. Additionally, Quicksort maintains the relative order of items with equal keys, making it a stable sort. This can be helpful in some situations where it's crucial that items with identical keys are arranged in relation to one another.

## **Modelling of the Problem (Desirable)**

Modelling is a visual representation of problems and their associated solutions and can be easier to understand than written forms in many cases. In my project, one problem that can be well represented is the hierarchy system of users.



Shown is a flow diagram of the tiers of employees within the company. One way the hierarchy system could work is the rows being given different permissions. If the end users feel this is too restrictive, as perhaps a supervisor would have the same permissions as an Office manager, then each job role can be divided individually so that no two job roles exist as the same, unless this is desired for specific roles that sit on the same level. An addition to the code that would be a feature of Google Classroom is the ability to create work groups separate from other staff so that one manager can isolate a group of staff and share tasks and messages with them.

Another visual feature of the project is the grid view. I will keep a simple design as I believe involving complex design and colour will slow the project and add far less substance to the result than ensuring all features work properly. Within each row representing a task, the user will be able to add their own task to be shown. In addition, tasks will be chosen that can be displayed. This will ensure the most important tasks are shown whilst avoiding a display that is too busy, as requested in the brief, furthermore a simple black and white display, I believe, assists this feature to ensure there is never too much going on so that the user feels distracted or that they cannot clearly see which important tasks need to be completed and when.

## **Acceptable Limitations (Supplementary)**

Whilst I will try to complete the project to as high a quality as possible, due to such constraints as time, there must be clear expectations, and a clear endpoint that is not aimed to be surpassed.

To create a GUI with anything more advanced than WinForms I believe would take up too much time, and whilst it would look nicer for the user, the actual back-end substance of the project would falter, so I will keep a simple GUI.

The login system will include example entries and admin roles in a database of entries, however, to create an entire list for all employees would take up an incredible amount of time and storage. Therefore, I will ensure the login system works, and then it shall be required of the user to create their associated account with their ID.

Finally, there will be no file attachment options in the project, unlike Google Classroom, and there I no way to transport the files without linking e-mails which would require access to the company intranet.

## **Objectives (Core)**

I-Input P-Process S-Storage O-Output

If subsection IPSO isn’t specified, same value as greater task, i.e., 1 is P, if 1.3 is undefined it is also P.

1. (S)Maintain a database of employees, tasks, groups
2. Use DDL to create multiple tables to contain all database entries in normal form.
   * 1. User Table Containing:

* UserID
* Hierarchy
* HashValue
* Name
  + 1. Task Table Containing:
* TaskID
* Title
* Priority
* Description
* DateDue
* DateSet
  + 1. Group Table Containing:
* GroupID
* GroupName
* OwnerID
  + 1. Link Tables containing composite foreign keys

1. Use DDL to create relationships between tables to link certain entities.

* User-Group
* Task-Group
* User-Task

1. (O)Create a WinForms UI to add entries to database.
2. (P)Test database integrity by addition and deletion of test entries before reaching the client.
3. (P)Create an employee log-in system
4. Use a hashing algorithm and use this to store hashed passwords in a table

* Take password input and store hashed result in User table

1. (O)Use WinForms to create a GUI for employee login where username and password can be entered.
2. (P,O)Check the login against the hashed value in the corresponding database entry.

* Check if stored hash is equal to hash of input
* Check if username is equal to stored username

1. Ensure employee attributes cannot be changed from within the account for changed access level
2. (P)Implement hierarchy system in creation of tasks
3. (S)Use DDL to create an access level attribute based on the employee’s level within the company.
4. Link this attribute to the assignment of tasks to other employees.
5. Implement a checker to query the database for the two employees’ access levels.
6. Use this to allow or deny the assignment of tasks between employees (tasks can only be given to those on a lower level).
7. (P,O)Allow tasks to be created and displayed by users from their account
8. Use WinForms the create a GUI that displays the tasks and dates of the employee logged in.
9. Allow the employee to navigate through the grid to any value using an SQL search embedded within the GUI

* Search with user defined input parameter.
* Search name, priority, hierarchy

1. Allow the employee to create tasks which can be linked to users and groups
2. Allow the employee to create their own personal tasks added only onto their grid

* Follows as part of hierarchy logic – can set to those of equal level.

1. (P,O)Allow organisation of tasks based on the date due, and a given priority
2. Use WinForms to establish a table with Title, Description, Due date, and priority etc attributes.
3. Sort these based on selected parameters

* Priority, the most important appearing at the top
* DateDue, the soonest appearing at the top
* DateSet, the most recent appearing at the top

1. Display all this without the need to refresh the display separately.

* Changing the display should be inbuilt in the function, not separate to sorting

1. (P)Allow users to create groups from a form accessed in their account
   1. Users can name the group and add users to the groups

* Name must be unique
* Users are referenced with their ID
* IDs are separated by commas, and this will be suggested in input box
  1. Users can only be added if their hierarchy value is higher than the owner
* Using a similar comparison as tasks
* Also applies to those of same hierarchy
  1. Tasks can be added to the group by the owner
* Check to see if user is owner
* If not, then task is still created as other relationships may be valid
* User notified of error

1. (P,O)Allow tasks to be completed and cleared
   1. User can select task to be completed and it will disappear from grid

* Refreshes grid without need to manually update by re-logging in or by other means
  1. Common tasks remain for other users when another completes it
* Includes tasks set to multiple individuals
* Includes tasks set to a group of multiple individuals
  1. Tasks remain in database once completed – Contributes to data integrity

1. (P,O)Allow users to write their current tasks to a local file
   1. Can sort by Priority, Date Due, Date Set, through sorting algorithm

* Quick Sort model is used
* Quick sort is adapted based on user input
* Uses polymorphism and abstraction
  1. Can present in ascending or descending order through use of stack or queue
* Decided based on selection of user
* Method for printing data structure overridden using polymorphism
  1. User can print file, must be in plain text format

# **Documented Design**

High Level Overview (Delete any not relevant)

### **System Flow Chart/Diagram(s)**

Flow Chart of entire system:



### 

This flow chart shows how the program goes through each form from user input. Initially the program launches in Login. If valid, goes to account, from there the user can select from different options such as set tasks.

Login Flowchart:



This flowchart starts, as seen in the previous flowchart, when the program is launched. The user must input their username and password and press login, the corresponding hash is fetched from their username log in the database, and their input is hashed, if the two match, access is granted, if not, the user if informed and the process repeats.



The AddTask flow starts when the corresponding button is pressed in the Account window. Once the input is recognised, the validity is checked, if valid, a task is then created with the data given. That task is then to be set to users and groups, subject to the setter being of a higher role that the recipient, and the setter being the owner of the group being set the task.

Create group and sort do not require a flow diagram, creating of a group is simply name, users(in the same method as setting users a task); sort is identifying which sort was selected, and then using a WinForms function, now SQL is needed

### **Structure/Hierarchy Chart – Modular Structure Diagram**

Admin Subsystem:

Admin Form

Access login window

Add users

Create tables in database

Create database

The admin form will allow users with administrative access to create a database, create tables in the database, add new users/ employees, and then return to the login.

Account Subsystem:

Account Form

Access Create Task Window

Access Create Group Window

Complete Tasks

Sort Tasks

Display Tasks

The account form will display a user’s tasks, sort and complete them, and allow the user to access the create group and task windows.

Create Group Subsystem:

Create Group Form

Name Group

Add Users to group

Add to GroupID table

Add to User\_Groups table

The Create Group form will allow users to create a group, name it, add users, and then store this information in the program database.

Create Task Subsystem:

Create Task Form

Add to TaskID table

Name Task

Set Priority

Set Dates

Set Description

Add to Task\_Groups table and HasTaskID

Add task to users and groups

The Create Task form will allow users to create a task, add a name, description, priority, and dates, set this task to users, and then store this information in the program database.

### **IPSO chart(s)**

|  |  |
| --- | --- |
| Input:  User Details:  Username, Password, Hierarchy  Task Details:  Name, Priority, Description, Date Set, Date Due  Users, Groups  Group Details:  Name, Users  Sort Details  By Priority, By Date Due, By Date Set  Create Database button  Create Tables button | Processes:  Add User  Add Task  Validate Task Setting  Add Group  Get GroupID  Get UserID  Get TaskID  Display Tasks  Sort Tasks  Complete Tasks  Validate Login  Get User Tasks |
| Storage:  Tables:  UserID  TaskID  GroupID  HasTaskID  User\_Groups  Task\_Groups | Outputs:  Login Form  Account Form  Create Task Form  Create Group Form  User Tasks:  TaskID, Title, Priority, Description, Date Due, Date Set  Exception Message Box: Login failed, task setting failed, group creation failed, complete task failed |

### 

### **Object/Class diagrams**

|  |
| --- |
| Account |
| Attribute |
| String User  String Connection String  String Username  String UserID |
| Methods |
| ExecuteSqlDisplay CompleteTask  TasksDone Order  IsTaskCompleted ParameterisedTasks  DisplayTask  PersonalTasks  TasksFromGroups  AllTasks  IncompleteTasks  LoadTasks  SetupGrid  DisplayData  Account  GetUserID  HasBeenSet |

|  |
| --- |
| QuickSort |
| Attributes |
| String User  String Connection String |
| Methods |
| SortList  getTaskPriority  getDateDue  GetDateSet |

### 

### 

## 

|  |
| --- |
| DateSetSort |
| Attributes |
|  |
| Methods |
| SortList |

|  |
| --- |
| PrioritySort |
| Attributes |
|  |
| Methods |
| SortList |

|  |
| --- |
| DateDueSort |
| Attributes |
|  |
| Methods |
| SortList |

|  |
| --- |
| SQL |
| Attributes |
| String Connection String |
| Methods |
| ExecuteSql  ExecuteSqlReturn |

|  |
| --- |
| Encryptor |
| Attributes |
| String text |
| Methods |
| Hash |

|  |
| --- |
| DataStructure |
| Attributes |
| String User  String connection string |
| Methods |
| SaveStructure  SetupWriter  GetTaskInfo  WriteTask |

|  |
| --- |
| MyQueue |
| Attributes |
| Int[] QueueArray  Int tail  Int head  Int length |
| Methods |
| Enqueue  Dequeue  IsEmpty  SaveStructure |

|  |
| --- |
| MyStack |
| Attributes |
| Int toppointer  Int[] StackArray  Int size |
| Methods |
| Push  Pop  Peek  IsEmpty  SaveStructure |

|  |
| --- |
| AddTask |
| Attributes |
| String User  String connection string  Int UserID |
| Methods |
| GetTaskID  SetTask  GetGroupOwner  GetGroupName  AddTaskGroup  GroupTask  ValidRecord  GetHierarchy  CanSetTask  AddTaskUser  IndividualTask  SetupUsersGrid  DisplayData |

|  |
| --- |
| CreateGroup |
| Attributes |
| String user  String connection string  Int UserID |
| Methods |
| ExecuteSqlDisplay  SetupGroup/User/TaskGrid  DisplayData  GetGroupID  GetGroupOwnerID  AddUsersGroup  AddGroup  AddTaskGroup  AddUsers  DisplayOwnedGroups  DisplayGroupTasks/Users |

## **Description of Algorithms**

The most used subroutines in the project as a whole are those which query within SQL. The SQL class includes two methods, ExecuteSql, which accepts a string as a parameter, and queries within the database, without returning a result. This is used for insertion of new records, and creation of tables.

The SQL in the project works on SQL Server. The Connection String contains the database which needs to be queried against, UserDatabase.mdb.

The subroutine works by creating a new instance of OleDbConnection, passing the connection string to the object. It then invokes the command method and tries to open a connection with the database. If this connection is established, the query is then executed. If the query syntax is not accepted, or the connection isn’t opened, an exception is thrown to prevent the project runtime crashing from an unhandled exception.

public void ExecuteSql(string Query)

{

using (OleDbConnection connection = new OleDbConnection(CONNECTION\_STRING))

{

using (OleDbCommand command = new OleDbCommand(Query))

{

command.Connection = connection;

try

{

connection.Open();

command.ExecuteNonQuery();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

}

}

The subroutine is notedly void because it doesn’t return any values from the queries it executes. Applications of this subroutine will be noted later in the design.

ExecuteSqlReturn

public OleDbDataReader ExecuteSqlReturn(string Query)

{

OleDbConnection connection = new OleDbConnection(CONNECTION\_STRING);

connection.Open();

OleDbCommand command = new OleDbCommand();

command.Connection = connection;

command.CommandText = Query;

OleDbDataReader reader = command.ExecuteReader();

return reader;

}

Although this subroutine looks majorly different upon first glance, both open a new instance of OleDbConnection. The actual difference is this function calls the ExecuteReader function, which returns an array. This array is of type OleDbDataReader and works by storing each column of returned data in one index, which then needs to be iterated through for all data to be accessed, as seen in code. There is no try catch, as if failed, the reader will simply be empty, so the try catch is implemented when trying to read the array, not fill it.

One version of SQL execution has been left in a form, not a method is the SQL object, as it is specific to that form. This is the ExecuteSqlDisplay subroutine:

public void ExecuteSqlDisplay(String sSqlString)

{

DataTable dt = new DataTable();

using (OleDbConnection connection = new OleDbConnection(CONNECTION\_STRING))

{

using (OleDbCommand command = new OleDbCommand(sSqlString))

{

using (OleDbDataAdapter dataAdapter = new OleDbDataAdapter(command))

{

command.Connection = connection;

try

{

connection.Open();

dataAdapter.Fill(dt);

DisplayData(dt);

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

}

}

}

}

This functions exactly the same, for the most part, as the first function. The only difference being after querying, it fills a data table with the data found in the query. This is used to fill the data table in the Account form.

Before displaying the data, the data grid must be set up. For each field wished to be displayed, a column is defined with an index, title, and format. This is done in the SetupGrid subroutine, called when a new account object is created. For example:

dgvTasks.Columns[2].Name = "Date Due";

dgvTasks.Columns[2].DefaultCellStyle.Format = "dd/MM/yyyy";

The data table form ExecuteSqlDisplay is passed to DisplayData as a parameter. The table is then iterated through, and each relevant column is given values from the returned data:

dgvTasks.Rows[n].Cells[1].Value = \_dt.Rows[i][1];

This extract fills the 2nd column (index starts at 0) with the Title, the data from the second column of the parameter. The n variable takes the value of the first non-filled row, so that there is no overlap of data, or missing rows, this value is obtained from Add function.

int n = dgvTasks.Rows.Add();

The filtering of which tasks to actually display is done majorly by two key functions, TasksFromGroups, and PersonalTasks. TaskFromGroups both identifies which groups the user is part of, and which tasks belong to those groups, as will be shown. PersonalTasks then finds which tasks belong to the user-not through groups- and then returns the set not overlapping with TasksFromGroups.

private List<int> TasksFromGroups()

{

string sSqlString = $"SELECT TaskID FROM Task\_Groups INNER JOIN User\_Groups ON Task\_Groups.GroupID = User\_Groups.GroupID WHERE UserID={UserID}";

var reader = Query.ExecuteSqlReturn(sSqlString);

List<int> GroupTasks = new List<int>();

while (reader.Read())

{

GroupTasks.Add(Convert.ToInt32(reader[0]));

}

return GroupTasks;

}

The query executed works by first creating an inner join between Task\_Groups and User\_Groups, where the GroupID field in both the tables is the same. It could look something like:

|  |  |
| --- | --- |
| UserID | GroupID |
| 1 | 2 |
| 2 | 3 |

|  |  |
| --- | --- |
| TaskID | GroupID |
| 5 | 3 |
| 1 | 3 |
| 6 | 1 |

|  |  |
| --- | --- |
| UserID | TaskID |
| 2 | 5 |
| 2 | 1 |

From this newly created temporary table, the records where UserID is the same as the current user’s ID, are selected, and from there, the TaskIDs are returned.

The query used in PersonalTasks is comparatively simple:

SELECT TaskID FROM HasTaskID WHERE UserID={UserID}

Simply returning the tasks which have been set specifically to the user aside from groups. However, the extra logical step comes in overlapping tasks. Since it is possible to be set the same task through a group and specifically, there must be logic to ensure the same task isn’t displayed twice.

This is done by invoking TasksFromGroups inside of PersonalTasks. Tasks returned from the query are stored in a temporary array, which is then iterated through; if a task is in the array returned from TasksFromGroups, it is not added to the final array, if not, then it is added. This means the arrays returned can be displayed simply, instead of having to use logic when they are displayed to avoid duplication.

foreach (int Task in TempIndividualTasks)

{

if (!GroupTasks.Contains(Task))

{

IndividualTasks.Add(Task);

}

}

return IndividualTasks;

The only other hurdle preventing tasks being displayed, is the user’s ability to complete tasks, before a task is displayed, a check must take place to see if it is completed, as explained previously, the task cannot simply be deleted.

When a task is completed, the task is added to the relevant table, along with he UserID of the user who completed it. From there, the subroutine TasksDone returns the array of tasks which have been completed, if any exist, from the database:

SELECT TaskID FROM Task\_Completed WHERE EXISTS (SELECT TaskID FROM Task\_Completed WHERE UserID={this.UserID})

Each task which is attempted to be displayed can then be checked against this array by passing to the IsTaskCompleted array.

Finally, the user has the option to sort their tasks visually by three different constraints:

* Priority
* Date Set
* DateDue

Whilst this obviously can be done with the SORT BY feature of SQL, since queries are being executed and put in arrays constantly, the trouble comes in when this sorting should be done. However, WinForms has an inbuilt sort feature which can be used one the tasks are displayed. An Example:

dgvTasks.Sort(dgvTasks.Columns[4], ListSortDirection.Ascending);

The grid is now sorted by priority, with the most important (value closest to one) being displayed first.

From the Account form, the main features are creating groups and setting tasks.

The majority of creating a group is simple and done by inserting the new values into the GroupID table, with a try catch in case the input was of the wrong format. The only complication of the creation is adding new users to a group.

The decided upon format that was easiest to implement was referencing users, separated by commas. From there, string handling is used to separate users into an array, which each user stored in a separate index of the array. From there, the text is trimmed to avoid space errors, and inserted into User\_Groups.

foreach (string Member in Members)

{

try

{

Member.Trim();

int MemberID = Convert.ToInt32(Member);

AddUserGroup(GroupID, MemberID);

}

catch

{

}

}

Adding tasks is slightly more complicated and requires many subroutines, however, since many of them function is a very similar way to previously explained functions, their explanations will be omitted.

The users and groups that are be set the tasks are subject to the respective constraints: the user setting the task has a higher hierarchy (lower value) that who is being set the task, and the personal assigning the task to the group is that group’s owner.

For the first constraint, CanSetTask accepts two UserIDs as parameters. The GetHierarchy function then returns the hierarchies of the two users. If the first user’s hierarchy is of a lower value than the second’s, then the Boolean value true is returned, if not, false is returned. This can be then used for each target user to deem if the task can be set.

private bool CanSetTask(int UserID, int TargetID)

{

int UserHierarchy = GetHierarchy(UserID);

int TargetHierarchy = GetHierarchy(TargetID);

if ((UserHierarchy != 0) && (TargetHierarchy != 0))

{

if (UserHierarchy <= TargetHierarchy)

{

return true;

}

else

{

return false;

}

}

else

{

return false;

}

}

To determine whether a user can set a task to a group, the function GetGroupOwner is used, passing the GroupID as a parameter, the following SQL query is executed:

SELECT OwnerID FROM GroupID WHERE GroupID={GroupID}

This value is then compared to the current UserID, if matching, the task is added to Task\_Groups.

Another algorithm used in my project is quick sort. This used recursion to sort a list of integers, dates, or any object with quantifiable quantities. The algorithm uses “divide and conquer” which recursively divides the parameter list into smaller sub-lists, which are ultimately re-combined to form the sorted list.

The algorithm uses a pivot element, which is compared to all other elements in the list. Those elements less than the pivot are moved left of the pivot, in their original order, and those greater than the pivot are moved to the right of the pivot.

The variables Lindex and Rindex are used to define the bounds of indexes used in the algorithm, are usually 0, the first index as the left most, and the length of the parameter list – 1, the right most index.

public virtual List<int> SortList(List<int> list, int Lindex, int Rindex)

{

var n = Lindex;

var m = Rindex;

var fulcrum = list[Lindex];

while(n <= m)

{

while(list[n] < fulcrum)

{

n++;

}

while(list[m] > fulcrum)

{

m--;

}

if(n <= m)

{

int temp = list[n];

list[n] = list[m];

list[m] = temp;

n++;

m--;

}

}

if(Lindex < m)

{

SortList(list, Lindex, m);

}

if(n < Rindex)

{

SortList(list, n, Rindex);

}

return list;

}

The reason why it cannot be assumed the Lindex will be 0, and the right will be the right most index, is because the list is recursively sorted with different indexes and fulcrums(pivots), which is the essence of the divide and conquer concept.

The use of quick sort in my project is writing tasks to a file, sorted based on user input. The user can choose sort based on task priority, date due, or date set. An SQL query could theoretically be used, but would require a parameterised 3 way join, where quick sort is a more efficient and easier way of sorting the tasks based on parameters.

The SortList function is virtual, and the QuickSort class contains functions GetTaskPriority, GetDateDue, and GetDateSet, which are all inherited by the modified quick sort algorithms. An example is the PrioritySort, which uses similar code as the base quick sort, but overrides with a method which, instead of comparing the values of the list indexes, the value of GetTaskPriority is compared when the index is a parameter- the same principal applies for Date Due and Date Set. For example:

while(list[m] > fulcrum)

{

m--;

}

Is replaced with:

while (GetTaskPriority(list[m]) > GetTaskPriority(fulcrum))

{

m--;

}

Resulting in a list of sorted tasks, based on priority, not on their ID value as would be the case with the base quick sort.

## **Description of Data Structures**

A stack is a data structure of type Last in First Out (LIFO) and is used to store, manage, and export data, such as integers, strings, or objects. When a series of data is put in a stack, the first to be put on the stack will be the last to be removed. Stacks are useful in accessing data in a LIFO manner, which is needed in my project.

The class MyStack represents a integer stack data structure and can be instantiated with a size, as the structure is built on an array, which is static, so when a stack is to be made, which is to hold tasks in my case, the number of tasks needs to be known, which doesn’t actually pose a problem in my project.

Stacks work with a top pointer, which keeps track of the next available space for data to be stored, it is indexed at 0, as this is the first index an array can store data in. The basic operations of a stack are Push, Pop, Peek, and a check to see if the stack is empty.

Push takes an integer and checks to see if the stack is full, with the top pointer index, if it is, then storing the data will result in a stack overflow, so an exception is thrown. If the stack has space, then the integer is stored in the next available space, and the top pointer is incremented.

public void Push(int number)

{

if (topPointer == this.size)

{

throw new Exception("Stack Overflow");

}

else

{

StackArray[topPointer] = number;

topPointer++;

}

}

The pop function is the exact opposite of push, it checks to see if the stack is empty, with the aforementioned IsEmpty function. If there is data to be outputted, then the top pointer is decremented, and the corresponding top integer is returned to where the function was called, which is why the return type is an integer, not void.

public int Pop()

{

if (topPointer == 0)

{

throw new Exception("Stack Underflow");

}

else

{

topPointer--;

return StackArray[topPointer];

}

}

Peek is the same as pop, it returns the data value on the top of the stack, however, the function doesn’t increment the top pointer, meaning the integer is not taken off the stack, merely looked at.

public int Peek()

{

if (topPointer == 0)

{

throw new Exception("Stack Underflow");

}

else

{

return StackArray[topPointer - 1];

}

}

A queue is a data structure of type First in First Out (FIFO) and is used to store, manage, and export data, such as integers, strings, or objects. When a series of data is put in a queue, the first to be put on the queue will be the first to be removed. Queues are useful in accessing data in a FIFO manner, which is needed in my project.

The class MyQueue represents an integer queue which needs a length, like the stack, to be instantiated because of the static nature of arrays, the structure it is built on. Queues require both a heads pointer and a tail pointer to keep track of which integers are still in the queue and which have been released. Queues have similar functions to stacks, with Enqueue, Dequeue, IsEmpty as methods.

Enqueue checks is the tail pointer is bigger than the length, enqueueing here would result in overflow, like in the stack, and throws an error. If there is space, the parameter integer is stored in the tail index, and the tail in incremented.

public void Enqueue(int number)

{

if(tail > length)

{

throw new Exception("Queue Overflow");

}

else

{

this.QueueArray[tail] = number;

tail++;

}

}

Dequeue is exactly like pop, after checking the queue isn’t empty, which would be the case if the tail pointer is in the same index as the head pointer, it returns the value of the head pointer, which is the integer first enqueued.

These structures are used in the organisation of tasks when writing to files. Once files have been sorted, there needs to be an efficient way of writing these tasks to a file sequentially, and depending on which the user has selected, ascending or descending order. With a method, queues and stacks can be completely cleared and return every value they have in order, which means these tasks can be written to a file incredibly easily. Since the sorting algorithm sorts in ascending order, if the user selects ascending, the tasks are put into a queue to retain order, is descending, a queue is used to reverse the order which they will be outputted.

Since the program doesn’t know which thew user will pick, a virtual method must be used to be overwritten by the selected structure, polymorphism of the DataStructure class. Both structures inherit from the DataStructure class, an abstract class which cannot be instantiated alone. DataStructure has methods to Write a task to a file, shown in file handling, and to get the information about the task required, which are the same for a queue and a stack.

The virtual method is SaveStructure, which is executed differently depending on which structure the data is stored in. Since the structures have different methods for removing the next integer, but the functions do the same thing, the SaveStructure only differs in this aspect.

public override void SaveStructure()

{

StreamWriter WriteStream = SetupWriter();

while(!IsEmpty())

{

WriteTask(GetTaskInfo(Pop()/Dequeue()), WriteStream);

}

WriteStream.Close();

MessageBox.Show("Tasks Saved!");

}

Depending on the structure, the dequeue or pop function is used to write the returned task to a file, until the structure is empty.

## **Database and/or File Design**

All data about tasks, groups, and users, will be stores in a database within respective tables. Databases have been chosen as opposed to file storage because of the key feature, which will ensure there is no overlap in certain key fields, such as UserID.

Additionally, relationships between tables can be kept simple (many-to-one or one-to-many) with the creation of middling tables between a many-to-many relationship, these types of relationships can rarely be used effectively in code. However, with query functions such as joins used in conjunction with these middling tables, all appropriate data can be reached, as well as any overlaps, as will be seen in the query section of the design.

The project utilises seven key tables in order to store all necessary data, maintaining correct relationships. The tables, along with their fields, data types, purpose etc are shown below.

UserID:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field | Constraint | Data Type | Validation | Notes |
| UserID | Primary Key  Auto Increment | Integer |  |  |
| Name | Unique | String | <30 |  |
| Hierarchy |  | Integer | 0<x<100 | Can be updated for uses by different company size |
| HashValue |  | Integer | 128 bits | Generated using MD5 hashing algorithm, cannot be reversed |

The UserID table will hold User information relevant to the system. UserID will be an identifier within the system, however Name can also be used as it has been specified to be unique. Hierarchy is designed to have the most access or highest role at 1, so that as more positions or levels within a company are made, hierarchies can expand outwards, not having to change every other level if a new one is added, which would be the case if the lowest level was 1. The hash value is an encrypted password, and the details of how it is generated will be detailed in the later sections.

TaskID:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field | Constraint | Data Type | Validation | Notes |
| TaskID | Primary  Auto Increment | Integer |  |  |
| Title | Unique | String | <30 | Can include numbers |
| Priority |  | Integer | <10 |  |
| Description |  | String | <60 | Can include numbers |
| DateDue |  | Date | Date | Selected with integrated calendar |
| DateSet |  | Date | Date | Selected with integrated calendar |

The TaskID table will hold all information on tasks, similar to UserID, the TaskID field will be the primary identifier for records within the code, however, this could be done with the Title field as it is also unique. Priority operates in the same way as hierarchy, with 1 being the most urgent and 9 being the least, if the client feels this is too little or large of a scope, the range can be reduced or increased to fit the need of the client. Although DateDue originally has logic coded to ensure it was always after Date Set:

int result = DateTime.Compare(DateSet, DateDue);

if (result< 0)

return false

else

return true

Upon consulting the client, they opted out of this logic as tasks which take only a few minutes could be delayed in being set, and although I suggested changing the priority in this case, they wished to allow setting of tasks regardless of this.

GroupID:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field | Constraint | Data Type | Validation | Notes |
| GroupID | Primary Key  Auto Increment | Integer |  |  |
| GroupName | Unique | String | <30 |  |
| OwnerID | Foreign | Integer |  | Reference: UserID |

The GroupID table will hold all relevant information about Groups of Users. Each group needs an owner, since a group can only have one owner, it isn’t necessary to have a middling table because the relationship is many-to-one. However, upon the creation of a group, it makes sense to ensure there actually is a user to be referenced as the owner. The simplest way to do this is to make OwnerID a foreign key, if there is no associated UserID, the record will not be made, and an exception will be thrown.

After these three key tables have been explained, it becomes obvious that the relationships between them are three many-to-many relationships. As previously mentioned, these relationships are incredibly hard to query against and work with, so the following tables were created to solve this issue.

User\_Groups:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field | Constraint | Data Type | Validation | Notes |
| GroupID | Primary Key  Foreign Key | Integer |  | Reference:  GroupID |
| UserID | Primary Key  Foreign Key | Integer |  | Reference:  UserID |

Task\_Groups

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field | Constraint | Data Type | Validation | Notes |
| GroupID | Primary Key  Foreign Key | Integer |  | Reference:  GroupID |
| TaskID | Primary Key  Foreign Key | Integer |  | Reference:  TaskID |

HasTaskID

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field | Constraint | Data Type | Validation | Notes |
| UserID | Primary Key  Foreign Key | Integer |  | Reference:  UserID |
| TaskID | Primary Key  Foreign Key | Integer |  | Reference:  TaskID |

Each of these three tables have a composite key, made up of the two fields. This means that whilst there can be multiple records with the UserID the same (in the User\_Groups table), there cannot be repeats of a record where the UserID and GroupID are the same. This means- for example- a user can be in multiple groups, groups can have multiple members, but a user cannot be part of a group twice. This is particularly useful in the HasTaskID, ensuring that a user can’t be set the same task twice.

Another feature requested by the user was to be able to complete tasks, which would then not appear in the task view. Because tasks can be set to groups, but the tasks are completed individually, the task record can not be simply deleted from the database; as soon as one user has marked the task completed, it would be completed for all. Additionally, looking back at previous tasks wouldn’t be possible, which would most likely be needed for confirmation of certain dates etc.

To solve this problem, the following table has been implemented:

Task\_Completed:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field | Constraint | Data Type | Validation | Notes |
| TaskID | Primary Key  Foreign Key | Integer |  | Reference:  TaskID |
| UserID | Primary Key  Foreign Key | Integer |  | Reference:  UserID |

The table is of the same format as HasTaskID, but records which tasks have been completed.

When displaying tasks, logic is used to only display tasks that haven’t been completed, as will be explained in the subroutines section.

DDL

The following DDL has been used to create the database as previously explained.

"CREATE TABLE UserID("

+ "UserID INT IDENTITY(1,1),"

+ "Name VARCHAR(30) UNIQUE,"

+ "Hierarchy NUMERIC(2),"

+ "HashValue VARCHAR(60),"

+ "PRIMARY KEY(UserID)"

+ ")"

"CREATE TABLE TaskID("

+ "TaskID INT IDENTITY(1,1),"

+ "Title VARCHAR(30) UNIQUE,"

+ "Priority NUMERIC(1),"

+ "Description VARCHAR(60),"

+ "DateDue DATE,"

+ "DateSet DATE,"

+ "PRIMARY KEY(TaskID)"

+ ")"

CREATE TABLE GroupID("

+ "GroupID INT IDENTITY(1,1),"

+ "GroupName CHAR(30) UNIQUE,"

+ "OwnerID INT,"

+ "PRIMARY KEY(GroupID),"

+ "CONSTRAINT OwnerID FOREIGN KEY (OwnerID) REFERENCES UserID

(UserID)"

+ ")"

CREATE TABLE HasTaskID("

+ "UserID INT,"

+ "TaskID INT,"

+ "CONSTRAINT FKUser FOREIGN KEY (UserID) REFERENCES UserID (UserID),"

+ "CONSTRAINT FKTask FOREIGN KEY (TaskID) REFERENCES TaskID (TaskID),"

+ "PRIMARY KEY(UserID,TaskID)"

+ ")"

"CREATE TABLE User\_Groups("

+ "GroupID INT,"

+ "UserID INT,"

+ "CONSTRAINT UserID FOREIGN KEY (UserID) REFERENCES UserID (UserID),"

+ "CONSTRAINT GroupID FOREIGN KEY (GroupID) REFERENCES GroupID (GroupID),"

+ "PRIMARY KEY(UserID,GroupID)"

+ ")"

"CREATE TABLE Task\_Groups("

+ "GroupID INT,"

+ "TaskID INT,"

+ "CONSTRAINT TaskID FOREIGN KEY (TaskID) REFERENCES TaskID (TaskID),"

+ "CONSTRAINT FK\_Group FOREIGN KEY (GroupID) REFERENCES GroupID (GroupID),"

+ "PRIMARY KEY(TaskID,GroupID)"

+ ")";

"CREATE TABLE Task\_Completed("

+ "TaskID INT,"

+ "UserID INT,"

+ "CONSTRAINT FK\_User FOREIGN KEY (UserID) REFERENCES UserID (UserID),"

+ "CONSTRAINT FK\_Task FOREIGN KEY (TaskID) REFERENCES TaskID (TaskID),"

+ "PRIMARY KEY(UserID,TaskID)"

+ ")"

Constraint names such as FK\_User have been used in place of UserID, as you cannot have two constraints with the same name in the same database, so they have been used as alternative, but similarly descriptive, placeholders.

Relationship Model:

The relationships previously described can be seen in the following model, particularly that no tables have a many-to-many relationship.

Diagram

Description automatically generated

File Design

To write data to a locally stored file in my project, I have used StreamWriter. StreamWriter is a .Net class which writes text to a stream, which in turn accesses a file, network or other stream which supports string handling. I have used the StreamWriter method for writing strings, instead of char types, and use WriteLine to start a new line after every record.

Writing to file is used when tasks need to be stored on the user’s computer, by request of employees. The format of the file is

TaskID:{Reader[0]} | Title:{Reader[1]} | Priority:{Reader[2]} | Description:{Reader[3]} | DateDue:{Reader[4]} | DateSet:{Reader[5]}

Each task occupies a new line in the file, and all it’s properties are written sequentially for ease of reading.

The Stream is to a file named Tasks.txt and will rewrite the data if one is already existent. The stream is setup and then returned with a function, so it can be used in other functions.

StreamWriter sw = new StreamWriter("Tasks.txt");

return sw;

The file is written to using the WriteLine function:

sw.WriteLine(writeText);

The result of tasks being written will look like:

A picture containing Word

Description automatically generated

## **Design of User Interface**

An important element of the project is the graphical user interface, it must enable the user to access all features, and must be intuitive to use so that extensive training isn’t required to use the system

From research in other similar programs, such as Google Classroom, the following forms have been created. Whilst they aren’t as aesthetic as other programs, largely due to the limitations of google forms (and my designing abilities), I believe they are fit for purpose.

The AddUser from, accessible by admins and those with total control of the system.

Text Boxes for user input

Graphical user interface

Description automatically generated

Labels for text boxes

Close form, can be omitted as forms have banner with close and minimize

Account form accessible after login.

Writes a user’s tasks to file

Graphical user interface

Description automatically generated

Parameterized search for tasks

Data grid view where task will appear

Ability to complete task, removing from view

Radio buttons for task sorting

Links to other forms

Calendar for user organization

AddTask from accessible via button in account form

Graphical user interface

Description automatically generated

Data grid View for lookup of User IDs

Larger text box due to bigger character allowance of description

Date selector that allows user to pick from calendar

Automatically appearing text to show desired input

Named group of elements

CreateGroup form accessible via button in account form

Graphical user interface

Description automatically generated

Quick add of tasks to a group

Display info of chosen group as seen in SetTask

Displays for Owned Groups, Tasks in groups, users in groups

Suggested input as seen in SetTask

Form accessible only by administrators allowing creation of users.

Graphical user interface, text, application

Description automatically generated

Links to forms used by regular users as seen previously.

### 

## **System Security and Integrity of Data**

Since this project requires different accounts for different users, there will need to be some form of security to ensure only valid access to accounts is granted. This is done with a hashing algorithm, specifically in this project, the MD5 algorithm.

The MD5 algorithm accepts strings of any length and acts as a hash function, producing a 128-bit hash value. Hashing is a one-way type of encryption; one cannot decipher the original input from the hash.

The algorithm breaks up the input into 512-bit blocks and pads the message to be a multiple of 512, by adding a single 1, and then as many zeros as needed. The algorithm operates on 4 32-bit words, with 4 stages of 16 operations. The four functions:

Text, letter

Description automatically generated

This algorithm is accessed through the Microsoft Security.Cryptography library, which contains many hashing algorithms. MD% was chosen because, whilst there are vulnerabilities, the hardware requirements are sufficiently low, and since, as of 2019, a quarter of content management systems still use MD5, it proves sufficient for this purpose, and can even be updated later in the project’s lifetime.

The encryptor class contains the Hash method which is where MD5 is accessed.

Upon invoking the method, a new instance of MD5 in instantiated, the hash is computed and returned as a byte array. This is then changed to hexadecimal to be stored as a hash, and returned:

public static string Hash(string text)

{

MD5 md5 = new MD5CryptoServiceProvider();

//compute hash from the bytes of text

md5.ComputeHash(ASCIIEncoding.ASCII.GetBytes(text));

//get hash result after compute it

byte[] result = md5.Hash;

StringBuilder password = new StringBuilder();

for (int j = 0; j < result.Length; j++)

{

//change it into 2 hexadecimal digits

//for each byte

password.Append(result[j].ToString("x2"));

}

return password.ToString();

}

This method is invoked in adding users, and in the login. For the purposes of security, the login logic and the use of MD5 will be shown.

When a new user is added, their password is stored as a HashValue in the UserID table, this is then queried against to retrieve when a user is logging in.

SELECT HashValue FROM UserID WHERE Name='{Username}

Where username is the name entered. If no username or hash is retrieved, null is returned. The password entered by the user is then hashed:

string Hash = GetHash(Password);

Finally, if the two hash values match, a new account object is created, if they do not match, then user is informed through a message box, and if a hash value was not retrieved, the user is informed the username is not stored on the database.

if ((Hash == Password) && (Hash != "null"))

{

var Account = new Account(Username);

Account.Show();

MessageBox.Show("Login Successful!");

}

else if (Hash == "null")

{

MessageBox.Show("Username not found!");

}

else if ((Hash != Password) && (Hash != "null"))

{

MessageBox.Show("Username and Password do not match!"); }

# **Technical Solution**

Copy all annotated code and place here – ensure it is single line spacing

# **Testing**

Testing will be carried out to ensure the functionality of the program is that which was intended when the project was proposed. The indexing of these tests will refer to the project objective which it tests, those being the core components/requirements of the program. The evidence for these tests will be in video format with YouTube links corresponding to each test. The tests will not detail every part of the system, but should be representative of its functionality, and it’s meeting of core objectives.

Changes to the code in response to tests will also be shown after the tests, for example if new validation is required, then the amendments to the code will be shown and tested again.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test | Test Data | Expected Outcome | Actual Outcome | Action Needed | Link |
| (Objective)1.1,1.2  Use DDL to create tables to contain all database entries.  (Normal) | Creation of new program instance | Database and tables created ,program resumes | As seen in video, all tables detailed in documented design are made, with relationships | None | 00:00:05  https://youtu.be/tnD1LKw7FsQ |
| (Objective)1.4  Add new user as admin to database  (normal) | Input username and password of length in expected range | Message Box “user created”  user present in database | User Created message box appears, and user record appears in database to be used in login | None | 00:01:58  https://youtu.be/tnD1LKw7FsQ |
| (Objective)1.4  Add new user as admin to database  (erroneous) | Input no username and password | Message Box  “Input valid info!” | Message Box appears as expected, no new records appear in database | None | 00:01:34  https://youtu.be/tnD1LKw7FsQ |
| (Objective)1.4  Add new user as admin to database  (erroneous) | Input hierarchy as a string | Message Box  “Input integer Hierarchy!” | Message Box  “Input integer Hierarchy!” | None | 00:01:51  https://youtu.be/tnD1LKw7FsQ |
| (Objective)4.2,4.4  Use user information to log in to system  (normal) | Input previously added user data and click login button | Message Box  “Login Successful”  Account window opened | As seen in video, Message box is displayed, as is the account form | None | 00:02:29  https://youtu.be/tnD1LKw7FsQ |
| (Objective)4.2,4.4  Use user information to log in to system  (normal) | Input username and wrong password | Message Box  “Username and Password don’t match!” | Message Box is displayed, program continues and allows another login attempt | None | 00:02:16  https://youtu.be/tnD1LKw7FsQ |
| (Objective)2.2,2.4  Use user information to log in to system  (erroneous) | Input invalid username | Message Box  “Username not found!” | Message box is displayed, no new entries in database | None | 00:02:24  https://youtu.be/tnD1LKw7FsQ |
| (Objective)4.1,5.1  WinForms display displaying tasks set  (normal) | Login to a user account | Data table with: Title, TaskID, Priority, Date set and due, description | Example task shown, with correct details displayed, with slidable interface | None | 00:00:08  https://youtu.be/16gtLSY43\_Y |
| (Objective)4.3,3.4  Allow users to set tasks to other users  (normal) | Create a new task and set the task to lower priority users | Message Box  “Task Set”  Should appear in those user’s account grid | Message Box appears, and task appears in relevant account view, as well as in database | None | 00:00:47  https://youtu.be/16gtLSY43\_Y |
| (Objective)4.3,3.4  Allow users to set tasks to other users  (erroneous) | Create a new task and set to users of higher priority | Message Box  “Unable to set to one of these users”  No link in table | Message Box, Task is created in database, but link is not created between task and user. | None | 00:01:04  https://youtu.be/16gtLSY43\_Y |
| (Objective)4.3,3.4  Allow users to set tasks to other users  (boundary) | Create a new task and set to users of the same priority | Message Box  “Task Set”  Should appear in those user’s account grid | Message box is displayed, and task appears in task grid. | None | 00:01:35  https://youtu.be/16gtLSY43\_Y |
| (Objective)5.2  Sort tasks based on date due  (normal) | Click date due radio button in account window | Tasks become sorted based on date due, closest first | Tasks become sorted based on date due, closest first | None | 00:02:24  https://youtu.be/16gtLSY43\_Y |
| (Objective)5.3  Sort tasks based on Priority  (normal) | Click priority radio button in account window | Tasks become sorted based on priority, most important first | Tasks become sorted based on priority, most important first | None | 00:02:29  https://youtu.be/16gtLSY43\_Y |
| (Objective)6.1,6.2  Users can create group and add users to it  (normal) | Create group and add users of lower hierarchy | Message box  “Group created”  “Users added” | Message box  “Group created”  “Users added” | None | 00:03:02  https://youtu.be/16gtLSY43\_Y |
| (Objective)6.1,6.2  Users can create group and add users to it  (erroneous) | Create group and add users of higher  hierarchy | Message box  “Group created”  “Some users couldn’t be added” | Group was created and user was added to group | Review and change logic in who can be added to groups, relationship shouldn’t have been created | 00:03:26  https://youtu.be/16gtLSY43\_Y |
| (Objective)6.1,6.2  Users can create group and add users to it  (boundary) | Create group and add users of same hierarchy | Message box  “Group created”  “Users added” | Message box  “Group created”  “Users added” | None | 00:03:50  https://youtu.be/16gtLSY43\_Y |
| (Objective)6.4  Users can add tasks to groups they own  (normal) | Create task and add to group owned by user | Task created, appears for users in that group | Task was created, added task in group window, not in task window, despite this the relationship was created | None | 00:04:55  https://youtu.be/16gtLSY43\_Y |
| (Objective)6.4  Users can add tasks to groups they own  (erroneous) | Create task and add to group not owned by user | Task created, Message Box  “You do not own group x” | Task created, Message Box  “You do not own group x” | None | 00:00:22  https://youtu.be/Rir7fcj8-Qg |
| (Objective)7.1,7.3  Users can complete tasks  (normal) | Complete set task from user account | Message Box  “Task complete”  Task disappears  Task still in database | Message Box  “Task complete”  Task disappears  Task still in database | None | 00:05:36  https://youtu.be/16gtLSY43\_Y |
| (Objective)7.1  Users can complete tasks  (erroneous) | Complete task not set to user | Message Box  “You haven’t been set that task” | Message Box  “You haven’t been set that task”, no relationships are added or created | None | 00:05:45  https://youtu.be/16gtLSY43\_Y |
| (Objective)7.2  Common tasks remain for users once completed by others  (normal) | Complete task set to user through group, login to another user in group | Task disappears from first account, remains in second | Task disappears from first account, remains in second | None | 00:00:50  <https://youtu.be/uDrAAioHJ1I> |
| (Objective)8,8.1,8.2  Writing tasks to local file, sorted, and ordered  (normal) | Select Priority and Ascending | File accessible with current tasks, sorted and in order | File accessible with current tasks, sorted by priority, and in ascending order, complete tasks are omitted | None | 00:06:16  https://youtu.be/16gtLSY43\_Y |
| (Objective)8,8.1,8.2  Writing tasks to local file, sorted, and ordered  (erroneous) | Don’t select options and press save tass button | Message Box  “Select Order and Sort condition!” | Program Crash | Conversion of selected item to string must be encased in try catch | 00:06:35  https://youtu.be/16gtLSY43\_Y |

Sample of database values, showing relationships made and not made:

Graphical user interface, application, table

Description automatically generated

 Changes Made:

string OrderBy = cmboOrderBy.SelectedItem.ToString();

string AscDesc = cmboAscDesc.SelectedItem.ToString();

DataStructure Tasks = AscDescFunction(AscDesc, Order(OrderBy));

Tasks.SaveStructure();

Encased in try catch loop:

try

{

string OrderBy = cmboOrderBy.SelectedItem.ToString();

string AscDesc = cmboAscDesc.SelectedItem.ToString();

DataStructure Tasks = AscDescFunction(AscDesc, Order(OrderBy));

Tasks.SaveStructure();

}

catch

{

MessageBox.Show("Select Order and Sort condition!");

}

Working evidence:

Graphical user interface

Description automatically generated

Changes Made:

CanAddToGroup method created to evaluate if a user can be added to a group by another, returning false if any values are 0:

private bool CanAddToGroup(int UserID, int TargetID)

{

int UserHierarchy = GetHierarchy(UserID);

int TargetHierarchy = GetHierarchy(TargetID);

if ((UserHierarchy != 0) && (TargetHierarchy != 0))

{

if (UserHierarchy <= TargetHierarchy)

{

return true;

}

else

{

return false;

}

}

else

{

return false;

}

}

This is used in logic of creating relationships, if returned true, the user is added, if returned false, the user is informed the link wasn’t made, as desired in the initial tests:

if (CanAddToGroup(this.UserID,MemberID ))

{

AddUserGroup(GroupID, MemberID);

}

else

{

MessageBox.Show("Some users couldn’t be added");

}

Working Evidence:

Graphical user interface

Description automatically generated

# **Evaluation**

|  |  |  |
| --- | --- | --- |
| Objective | Objective Met | Comment |
| 1.1) Use DDL to create multiple tables to contain all database entries in normal form. | Yes | This objective has been met by checking if a database is already in existence upon runtime, if not, a connection is established and the DDL shown is used to create a database with UserID, TaskID, GroupID, User\_Groups, Task\_Groups, and HasTaskID. The database features no data duplication, and all data is dependant on a key, whether that be a standalone primary key, or composite foreign key. |
| 1.2) Use DDL to create relationships between tables to link certain entities. | Yes | If only UserID, GroupID, TaskID are used, each table has a many-to-many relationship which is not normal form, and cannot be efficiently used and queried against. Therefore, link tables are used to reduce these relationships to many to one. The relationships are achieved by the means of composite foreign keys referenced from the relevant main tables, i.e., User\_Groups has UserID, GroupID as a foreign key. |
| 1.3) Create a WinForms UI to add entries to database. | Yes | This is achieved by the AddUser form which is accessible through the admin window, which is accessible through the login form when login with the admin account is used. The admin can input a username, password, and hierarchy which are stored in UserID table. The UserID field is autogenerated and autoincremented, with an index of one. |
| 1.4) Test database integrity by addition and deletion of test entries before reaching the client. | Yes | In testing, at least one entry into each tables was created, and then linked together, for example, adding a user to a group and setting that group a task. The need for removing tasks has been removed by including the Task\_Completed table. This means that even once a user has completed a task and no longer wants to see it, integrity is still maintained and there is no risk of removing the task without removing dependant entries in other tables. |
| 2.1) Use a hashing algorithm and use this to store hashed passwords in a table | Yes | The Encryption class encases the algorithm used to hash passwords. MD5 is used from a library to take input of any length(bar null due to external logic). This value is stored in the UserID, in the corresponding user field. A possible extension to this part of the project is making the hash unique, but this would have to be researched further in case this would compromise security. |

|  |  |  |
| --- | --- | --- |
| Objective | Objective Met | Comment |
| 2.2) Use WinForms to create a GUI for employee login where username and password can be entered. | Yes | This is achieved through the login form, accessed when the program is initially launched if a database already exists. |
| 2.3) Check the login against the hashed value in the corresponding database entry. | Yes | The inputs are the username and password, and after checking the data is valid, the corresponding hash value to the username is retrieved, this is checked against a hashed version of the password input, if matching, access is granted, and if not, an error message is given depending on which part of the logic was failed. |
| 2.4) Ensure employee attributes cannot be changed from within the account for changed access level | Yes | There is no SQL query which allows the updating of user fields such as the hierarchy or UserID. Therefore, even with SQL injection, it would not be possible to update values from inside the program, only if access is given to the database through a third-party software such as access |
| 3.1) Use DDL to create an access level attribute based on the employee’s level within the company. | Yes | This is achieved through the priority field in the UserID table. This is a small integer value corresponding to an employee’s position in the company. This value in supplied when doing checks for task setting, group adding etc. The highest access is one, which is automatically given to the admin, who is the only account who can add users, but this can be later changed for access to be granted to those of a certain level at the companies’ request. |
| 3.2) Link this attribute to the assignment of tasks to other employees. | Yes | When an employee tries to set another a task, both of their hierarchy values are obtained. These are then compared to each other and if the target user has a higher value(implying they hold a lower position) then the task will be set to them, if not however, no such link will be created. |

|  |  |  |
| --- | --- | --- |
| Objective | Objective Met | Comment |
| 3.3) Implement a checker to query the database for the two employees’ access levels. | Yes | This is achieved through the function CanSetTask, which takes two UserID value, checks the hierarchy values, and returns a Boolean expression used in if else statements to call a function that creates a link between the user and task, or throws an error |
| 3.4) Use this to allow or deny the assignment of tasks between employees (tasks can only be given to those on a lower level). | Yes | As mentioned in 3.3, after CanSetTask is implemented, the Boolean value determines the action taken and whether the task is set or not. |
| 4.1) Use WinForms the create a GUI that displays the tasks and dates of the employee logged in. | Yes | This is achieved through the account form. A data grid view displays any tasks set to the current user which haven’t been completed, this incudes all attributes in the TaskID table, such as priority, date set etc. |
| 4.2) Allow the employee to navigate through the grid to any specific date and time using an SQL search embedded within the GUI | Yes | This is also achieved in the account form. The user can select the parameter by which they search by, such as title. Then the user inputs their desired value such as ‘Test1’, the data grid view will then display the task Test1 if it exists, if the field isn’t unique, then all the tasks of matching value will be displayed, such as a priority of 4. |
| 4.3) Allow the employee to create tasks added to other users, again using embedded SQL methods. | Yes | This is achieved through the set task form, which takes all fields in the TaskID tables as input, bar the ID, and then uses aforementioned logic to decide whether the task should be set, and then creates the deemed correct links between user and task. This form also takes groups as input, and if the user owns the group, can be added to the group |

|  |  |  |
| --- | --- | --- |
| Objective | Objective Met | Comment |
| 4.4) Allow the employee to create their own personal tasks added only onto their grid, again using embedded SQL methods. | Yes | This is achieved in exactly the same way as setting other users tasks but relies on being able to set users of equal hierarchy tasks, since this is possible, and a user has the same hierarchy as themselves, it is possible. |
| 5.1) Use WinForms to establish a table with Title, Description, Due date, and priority etc attributes. | Yes | This is achieved through the account form. A data grid view displays any tasks set to the current user which haven’t been completed, this includes all attributes in the TaskID table, such as priority, date set etc. |
| 5.2) Sort these based on the Due date, with the nearest being first. | Yes | This is done using WinForms functions and uses the quantifiable values of columns. Dates can be sorted based on closeness, or earliness and so these are sorted based on which has been/ will come first |
| 5.3) Sort the tasks of an individual based on the priority. | Yes | This is done using WinForms functions and uses the quantifiable values of columns. Integers can be sorted based on absolute value, and so those with the lowest value, and in term the highest priority, are displayed first. |
| 5.4) Use WinForms to display this without the need to reload the display | Yes | Since this isn’t requiring an SQL query, the data grid view doesn’t need to be reloaded for the sort to be displayed, once the sort is executed (from the user button press) the tasks appear in order. |

|  |  |  |
| --- | --- | --- |
| Objective | Objective Met | Comment |
| 6.1) Users can name the group and add users to the groups | Yes | This is done through the CreateGroup form, the name of the group must be unique, and a field for the IDs of users to be added is included. |
| 6.2) Users can only be added if their hierarchy value is higher than the owner | Yes | Once the group is created, the string of ID’s is handled and dealt with individually, getting hierarchy values and comparing each one, so that is only one person is unable to be added, the group and other users are still connected. |
| 6.3) Tasks can be added to the group by the owner | Yes | This is an error thrown in the CreateGroup form and the AddTask form, if a task is to be assigned to a group, a check is made that the current user is the owner of that group, and from there the link is made or not made, if in the Add task form, the task is still made even if not assigned to the group. |
| 7.1) User can select task to be completed and it will disappear from grid | Yes | This is done through the user inputting the id of a task from their data grid view, the task is added to the Task\_Completed table with the UserID referenced alongside it, and a query for completed tasks is used when displaying tasks so that this wont show up for the user, the task can only be completed if it was set in the first place, to maintain data integrity. |
| 7.2) Common tasks remain for other users when another completes it | Yes | This is achieved through the Task\_Completed table. Instead of tasks being deleted once they are completed, a record is made is Task\_Completed and this is checked when displaying tasks, meaning tasks are always displayed for users who haven’t completed them no matter if others have. |

|  |  |  |
| --- | --- | --- |
| Objective | Objective Met | Comment |
| 7.3) Tasks remain in database once completed – Contributes to data integrity | Yes | As mentioned previously, this is achieved by means of the Task\_Completed table. Data integrity insured as there will never be entries in the link tables which don’t have a corresponding main table foreign key. |
| 8.1) Can sort by Priority, Date Due, Date Set, through sorting algorithm | Yes | A quicksort method was created to sort through integer values from lowest value to highest value. This is then overridden by different child classes dependant on user input which all compare different values, returned from functions such as GetDateDue. |
| 8.2) Can present in ascending or descending order through use of stack or queue | Yes | Once tasks are sorted, they need to be printed, they are put into a stack or queue to maintain or reverse order, and then using data structure methods, emptied, and written to a file in ascending or descending order accordingly. |
| 8.3) User can print file, must be in plain text format | Yes | The file takes the form of a text file, which can be easily printed, or even copied and pasted into a more formatted document, such as a diary, to give the user full control over how the data is used. |
|  |  |  |

User Feedback

Jon, the manager from the initial requirements, used the program as well as the aforementioned employees. Generally, they were very happy with the functionality of the software. Used for a trial, Jon found it very easy to set tasks to different users and groups and keep track of which users and tasks belonged to which groups with the data grid view- this was an element he was particularly impressed with as it wasn’t included in initial discussion, but he found it immensely improved his ease-of-use and experience with the software.

Additionally mentioned was the versatility in writing the tasks to file. Employees commented they really enjoyed the ability to choose between ascending or descending, a feature they hadn’t considered overall but proved useful in reducing the bulk of tasks by completing small, unimportant ones first when they became flustered or had too much to do.

Jon suggested that the program could be expanded upon, with means such as critical path analysis of tasks which would help plan out larger projects, but as far as the discussed scope was concerned, the client was very happy.

Improvements

Based on the user feedback given by Jon and the rest of the team, it appears the following can be added to the program to improve its versatility, and also some improvements I have found could be made from my own use:

* Critical path analysis of compound tass for larger projects regarding their dependencies and time scale.
* A user lookup in the AddTask form, so that switching between that and the CreateGroup form isn’t required
* Ability to add users based on username
* Linear programming models, such as simplex, for budget allocation