**Team Tracker Project**

**By George Driver**

**Analysis**

**The End-User & Background**

The manager of an engineering development team, Jon Driver, overlooks 30 full time employees who work in developing and supporting various software programs. This team is split into two groups, a development team and an operational team. The development team works to develop new software and to add features to existing software and the operational team works to support existing software and to deal with the needs of users. There are roughly 100 users of the supported software programs who frequently feedback to the team with bugs, technical difficulties, requests, and new feature ideas.

However, the team is often overwhelmed by work. The existing system of managing user feedback as well as the development of new software and features leads to backlogs of work, difficulties keeping track of work, and difficulties prioritising essential tasks. Additionally, communication between the two teams is weak, often leading to various issues when a new program or feature is released and the responsibility of its upkeep is transferred to the operational team.

**The Analysis Interview**

**The Interview With Mr. Jon Driver**

**1 - “What are some of the issues that you’ve faced with the current system?”**

We have found it difficult prioritising more important tasks which occasionally leads to deadlines being missed which is obviously a big issue for us. Additionally, communication between our two teams is lacklustre.

**2 - “What features are you looking for in a project management program?”**

The most important features I’m looking for are the ability to create, and edit tasks as well as being able to keep track of these tasks in a visually intuitive way. We should be able to keep track of who is working on a task and its stage in the process of completion.

**3 - “What do you hope this program will offer that isn’t offered by other existing programs?”**

We hope that by using a specialised system to organise our team, the program will contain only the features that are important to us and that the program will have in-depth support for everything we need.

**4 - “How big of a concern is security for you and your team?”**

Security is of the utmost importance as the programs and databases we create contain sensitive information from many of our clients. Because of this, information about the programs we create and their inner workings should be kept confidential to ensure weaknesses in the system aren’t broadcasted, and consequently taken advantage of to gain access to our clients’ data. Additionally, we want to ensure that our competitors don’t have in-depth information on our programs and what we are doing.

**5 - “How would you like to have control over your employees accounts?”**

Ideally, I would be able to verify my identity and then control which accounts have administrative permissions and be able to add/remove accounts.

**6 - “How should the program support better communication between the operational and development teams?”**

I hope that the program will support tasks that apply and are visible to both teams as well as tasks that are only visible to the team that needs to view them. Additionally, it would be helpful to allow the teams to set tasks with instructions for each other.

**7 - “What sort of information should be included when recording new tasks?”**

Obviously, as we have to deal with various different types of tasks, the exact information that needs to be recorded will vary significantly. It would be helpful to include who is taking on a task as well as the very basic stuff like title, deadline, and description.

**8 - “What sort of work input do you receive?”**

We communicate with clients primarily through emails, calls, and occasionally in-person meetings. This is where the vast majority of our work input comes from although, this work often must be decomposed significantly into separate tasks in order to tackle them individually. We hope the program will be able to store these decomposed tasks in a draft-like state.

**Insights From The Interview**

**1** - In his answer to question 1, Mr. Driver mentions that the team has struggled to prioritise tasks to ensure deadlines are met. This could be solved by implementing a system to mark essential tasks as more important. Or possibly, tasks with deadlines approaching could trigger a prompt to the team. At the very least, the program should clearly display deadlines on a timeline such that the team is aware of when something must be completed.

**2** - It is essential that tasks can be created and edited and that these tasks are displayed in an intuitive manner. There should be a system in place to keep track of the state of tasks and who is responsible for them.

**3** - To ensure that this program fulfils its worth, it must be a more fitting program than others that are already available. So that this is the case, the program should avoid the clutter of features that will not be useful for this team and should make sure that features that are useful have in-depth support.

**4** - A login system is necessary to ensure only authorised personnel have access to the system. In addition to this, levels of access may be required to ensure that employees only have access to the information and features that they need.

**5** - The answer to question 5 further reinforces the need for levels of access and displays a need for a system within the program to control accounts. As suggested by Mr. Driver, the system to control accounts should be able to control which accounts have administrative permissions and add or remove accounts.

**6** - In the answer to question 6, Mr. Driver suggests that the program should support two different types of tasks: team-specific tasks and shared tasks. Team-specific tasks should only be shown to employees that are part of the applicable team. In contrast, shared tasks should appear to all employees.

**7** - To ensure that all necessary information for a large variety of tasks can be represented, a long description should be supported. Ideally, these descriptions would be viewable from the main task page so the layout of the program should leave significant room for descriptions. Also, as mentioned before, the program should keep track of whoever is responsible for a task.

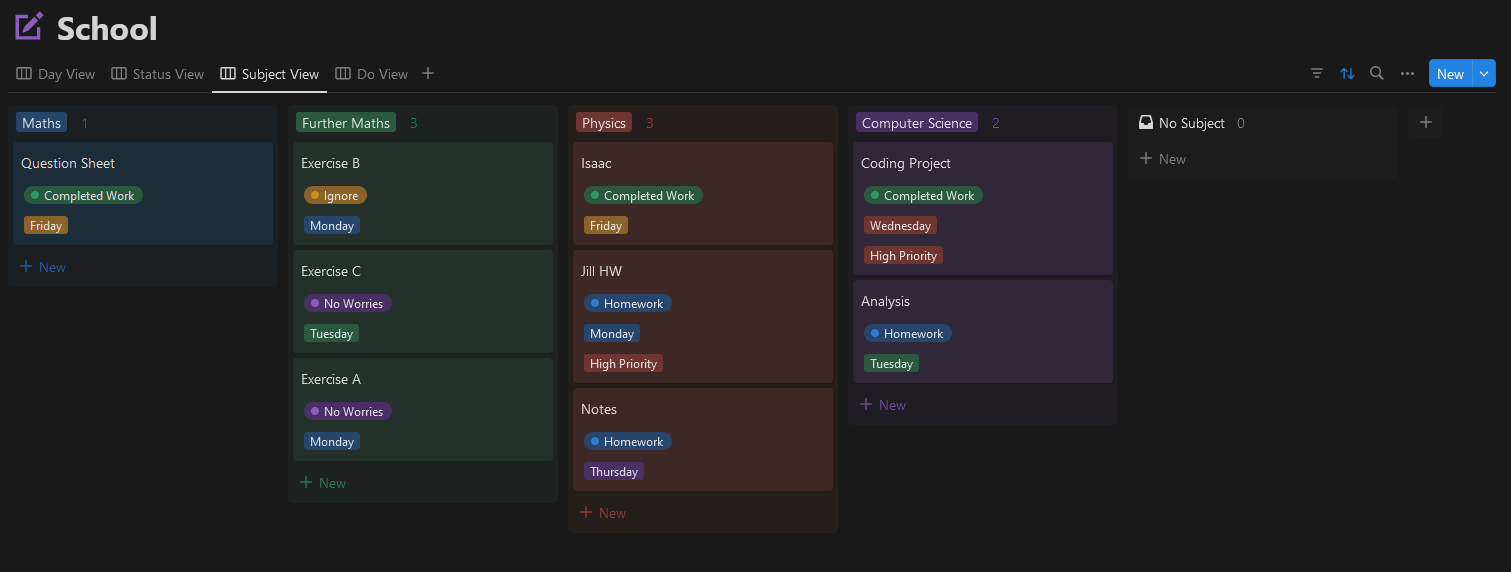
**8** - The program should support the ability to create tasks without filling in all fields as it may not be practical to rush users into decisions about the importance of tasks or who the tasks are being assigned to.

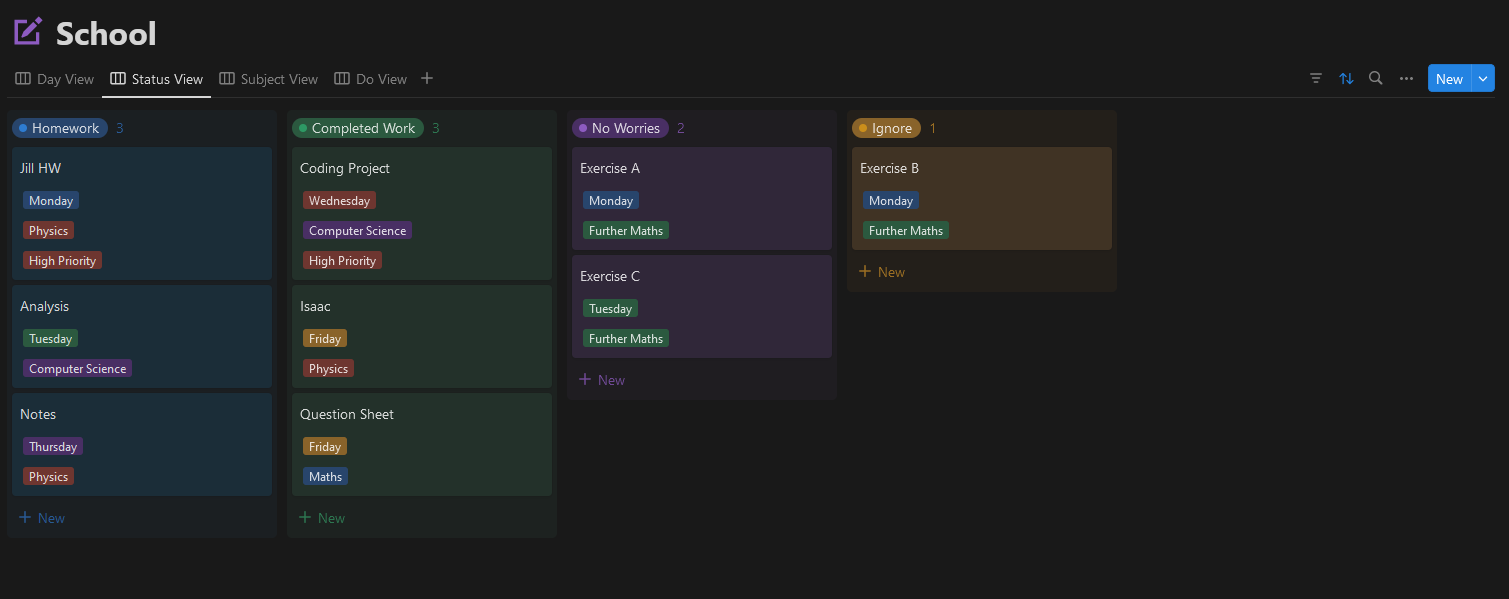
**Existing Solutions**

The issue in question could be tackled in various ways. For example, tasks could be filed physically with written instructions for each of the team members. Or tasks could be assigned and recorded through emails. Despite being familiar and avoiding the need to set up new software systems, these solutions can often make it hard to keep track of lots of tasks as instructions can be lost, incomplete, or confusingly organised. In contrast, a specialised software solution would clearly organise tasks, allowing members of the team to easily access all the necessary information to complete their tasks.

**Notion** - [notion.so](http://www.notion.so)

One possible software solution is Notion. Notion is an organisational tool that helps teams and groups to keep track of deadlines, meetings, to-do lists, and much more. The strength of Notion is its flexibility in that you have lots of control over what information is being held and how it is being displayed. However, this flexibility is in sacrifice of the ability to process the data in any meaningful way. Additionally, Notion is very manual with any data being visualised having to be inputted by hand. Notion also lacks a few important features such as the specialised handling of accounts and permissions, along with an intuitive way to handle work input.



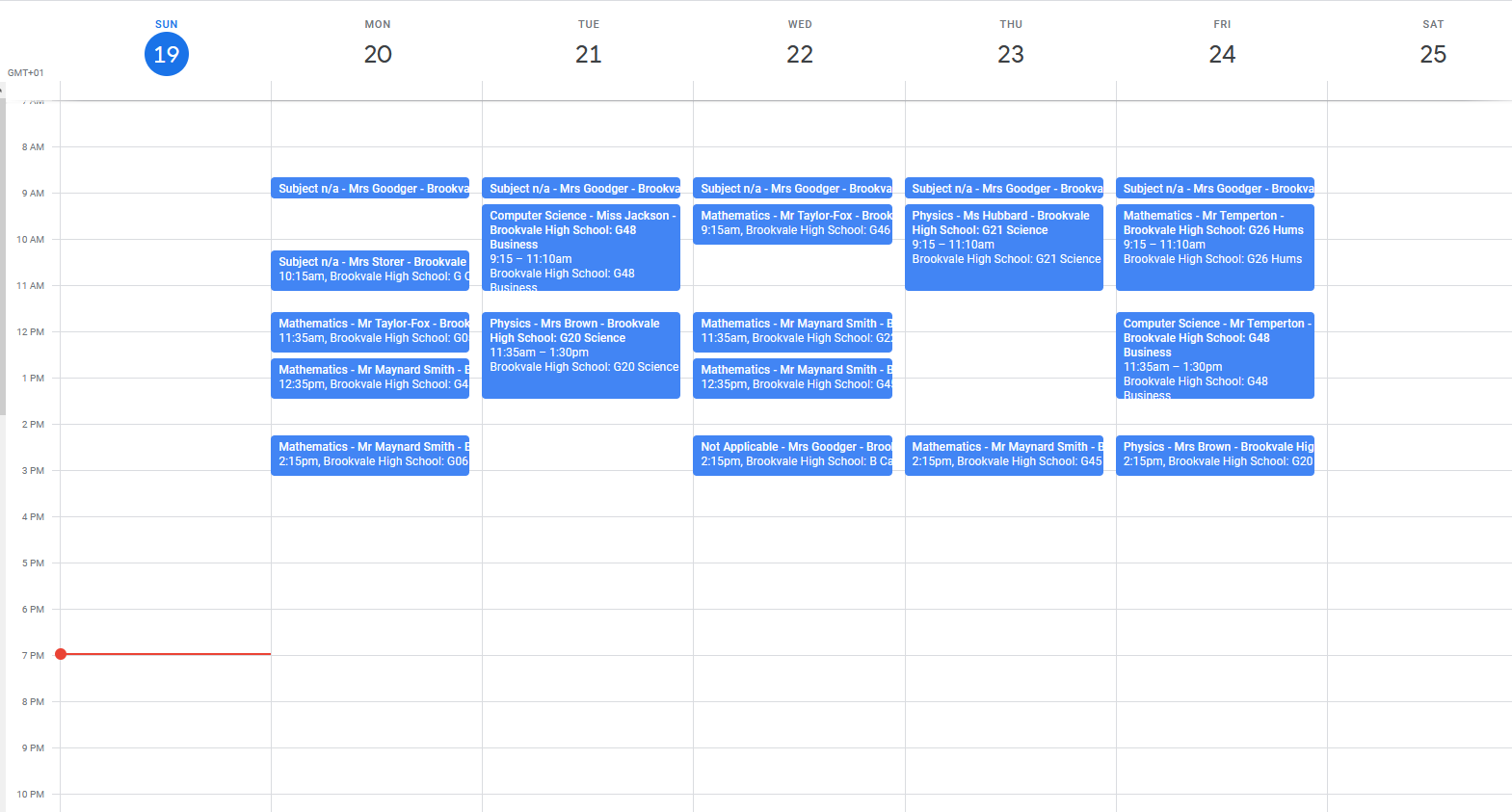


*Screenshots showing two different ways of viewing items on a school to-do list on Notion.*

Despite these pitfalls, the way in which you can create and organise items is nice. As shown in the image above, card-like widgets represent each of the individual tasks and include all the necessary information about the task. Additionally you can create different “views” which sort and display the information differently. There is a lot I like about this; in my project I hope to display tasks in this card-like way and to provide flexibility in how these cards are presented. Another thing I like about this setup is the ability to interact with these cards directly to perform actions like editing.

**Google Calendar** - [calendar.google.com](http://www.calendar.google.com)

It may seem obvious that an online calendar is a vastly ineffective solution to this problem, however it holds some important benefits that could help tackle parts of the problem. One of the main issues that the teams are facing is with meeting deadlines. Google calendar helps to combat this by providing a clear visual to see deadline dates. Not only does this ensure that employees are aware of deadlines, it also makes it easier to plan when and for how long you should spend doing certain things.



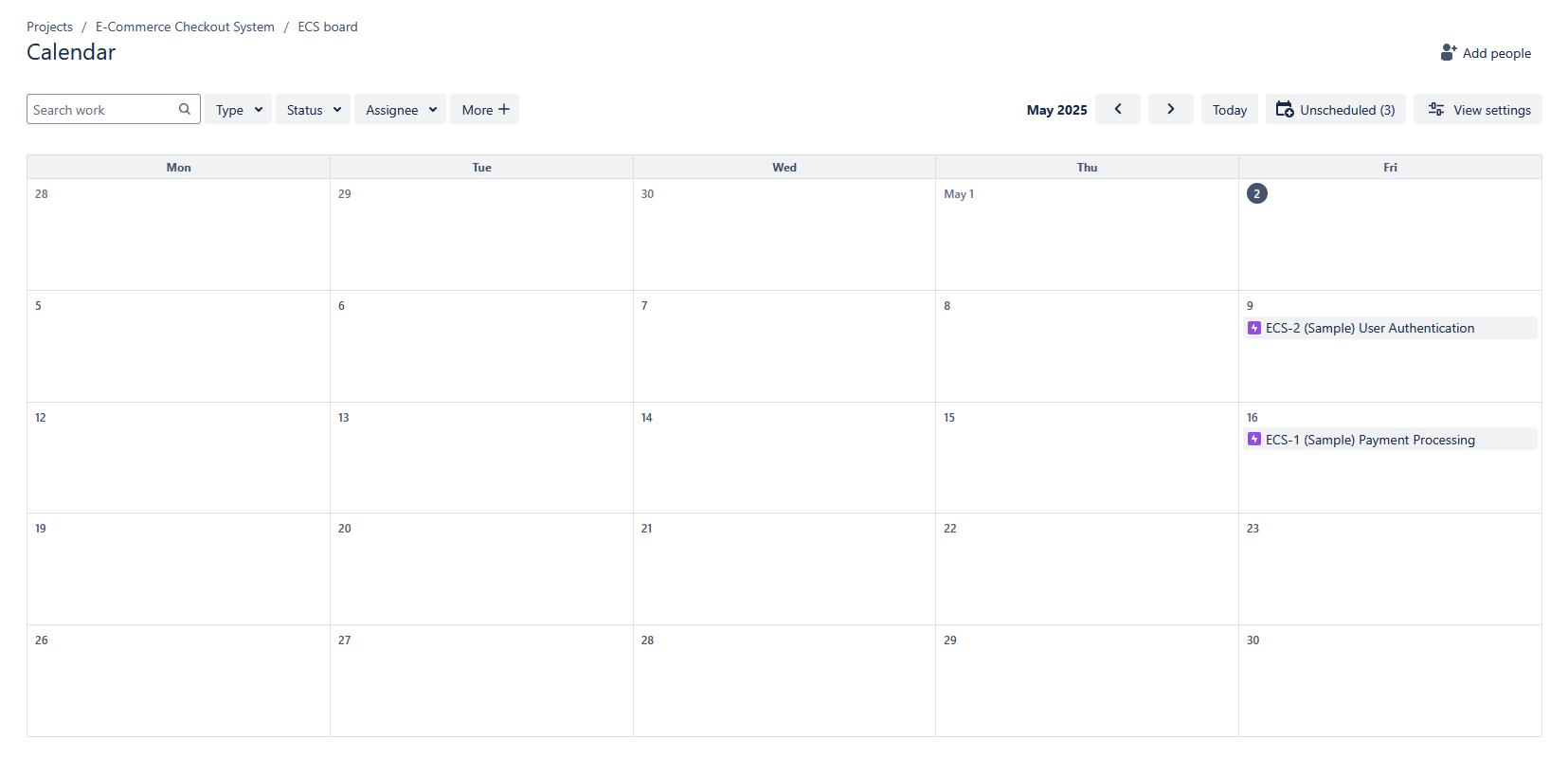
*A screenshot of a school timetable on Google Calendar.*

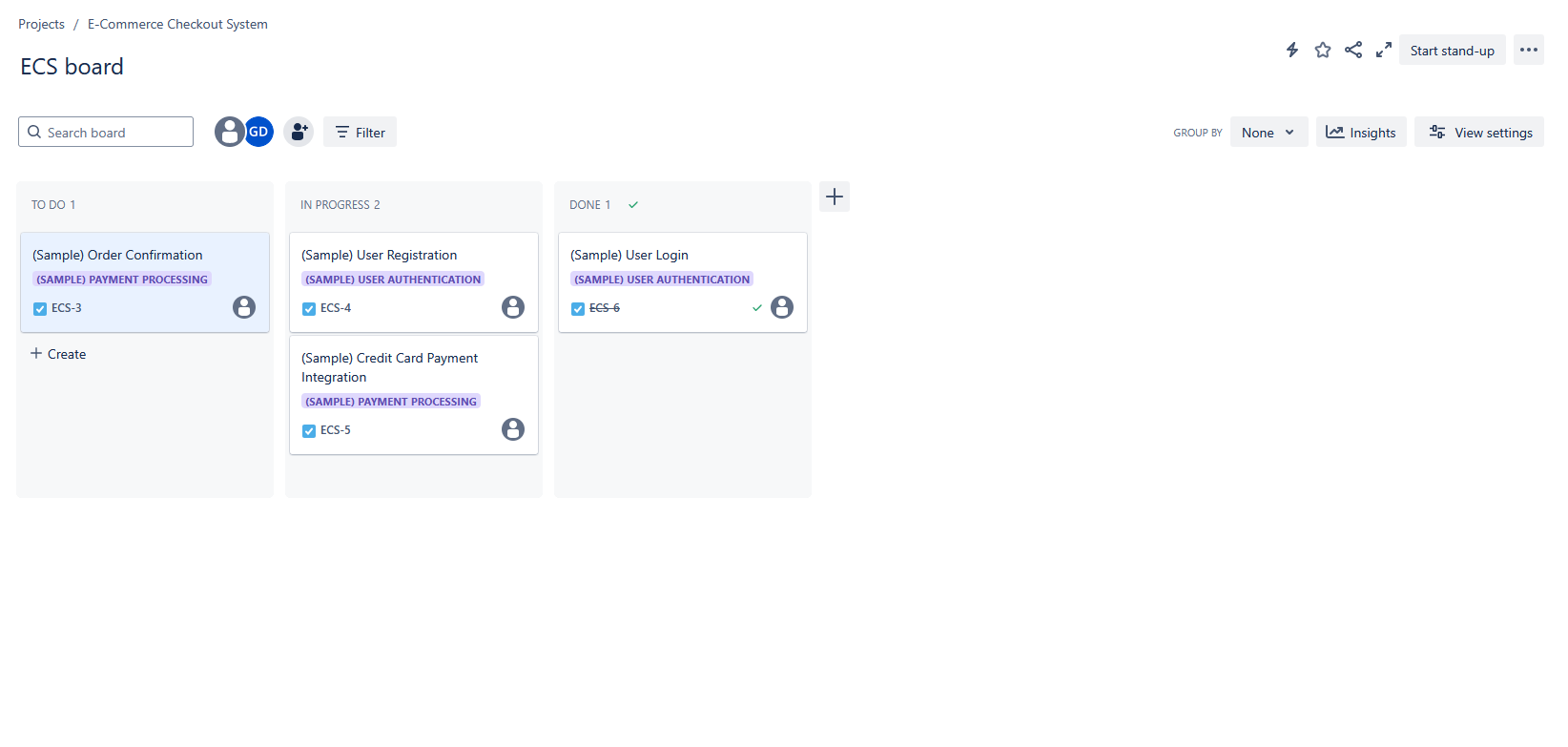
Google Calendar also allows you to break up your day as shown in the timetable above. This rigid structuring of time could yet again assist in ensuring deadlines are met, however, tracking things to this level of detail is probably an unnecessary sacrifice of flexibility for the problem at hand.

It should be noted that, being simply a calendar app, Google Calendar is very limited in functionality; no intuitive way to handle work input, only a single way of viewing the deadlines, little space for detailing other information besides the deadline and its date, etc. Although, in theory, Google Calendar could be used as part of a bigger system, it would make more sense to have this functionality be part of a more capable software program as to allow all important information to be accessible from one place.

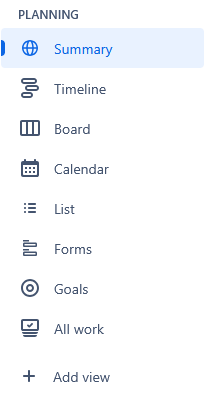
**Jira** - <https://www.atlassian.com/software/jira>

At first glance, Jira seems like the most promising solution of the three. It combines numerous features, including a calendar like the one from Google and a card-based Kanban board a bit like what's available in Notion. Originally, it was even designed specifically for software development teams like Mr. Driver’s.





*Screenshots of the calendar and Kanban board features of Jira.*



However, its complexity and customizability is a double-edged sword. For example, it includes many other interfaces for viewing and interacting with aspects of the project, as shown in this image. To get effective use of Jira, all employees must learn how to use each and every one of these proficiently which presents an extraordinarily steep learning curve, especially for people who aren’t already familiar with similar programs (which it seems Mr. Driver’s team isn’t). Implementing a solution like this would likely require team-wide training which would be costly and time consuming. This becomes even more problematic when considering that the team is already struggling to keep up with their deadlines.

On top of this, the project layout on Jira would likely take a lot of tinkering in order to specialise it for the needs of the team. This runs the risk of over-engineering, spending too much time customizing the layout, and hence harming productivity.

As a final complaint of Jira, the application is built to work best with other Atlassian tools, forcing the team to either bite the cost of paying for and learning to use the other Atlassian tools or committing to a subpar experience by using more familiar tools.

**Final Insight**

Although the solutions above are plausible aids to the problem at hand, they each have their own issues. I believe these applications and all other similar applications can be grouped into two broad categories; either the application is too simple and hence misses vital features to the effective functioning of the team, or the application contains so much unnecessary complexity that it becomes impractical to implement and/or harmful to productivity. It is these kinds of issues that I hope to avoid with a light, specialised application.

**Objectives**

1. **A login system is present**
   1. The user can enter a username and password
      1. The password is replaced with “\*”s when entered
      2. A button to enter the login is present
         1. Logins that match stored logins allow the main program to load up
         2. Invalid logins trigger a pop-up to notify the user
   2. First time logins prompt the user to change their password before the main program loads up
      1. The user can enter a new password
         1. The password is replaced with “\*”s when entered
         2. The password must be entered twice to confirm it is correct
      2. The password is only accepted if it is strong enough
         1. The password must be 8 or more characters in length
         2. The password must contain a special character
         3. The password must contain a number
         4. The password must contain an uppercase letter
         5. The password must contain a lowercase letter
      3. Weak or disagreeing passwords trigger a pop-up to notify the user
      4. The database is updated with the new password
         1. The password is encrypted before being stored
   3. The main program is loaded up after a successful login
      1. Admins load up in admin mode (i.e. they have the ability to control the accounts of employees)
      2. The program is loaded up in accordance with the team of the user
      3. Tabs allow the user to switch between the available interfaces
         1. Tabs are reloaded with each tab change
2. **An interface to view, edit, and create tasks is present**
   1. All necessary tasks are present and visible
      1. The tasks’ IDs are visible
      2. The tasks’ titles are visible and stand out
      3. The tasks’ descriptions are visible
      4. The tasks’ deadlines are visible
      5. The tasks’ importances (colour-coded signifiers of a tasks importance to aid prioritisation) are visible
         1. The importance of tasks is emphasised if the deadline is approaching or missed
   2. Tasks are organised such that their completion status is clear
      1. A “Backlog” column is present
         1. The importance of tasks in this column isn’t visible
      2. A “To Do” column is present
      3. A “Doing” column is present
      4. A “Done” column is present
      5. It is possible to scroll down the columns to view all tasks
      6. Tasks are organised by importance within columns
   3. Tasks can be created
      1. A button to enter a “New Task” interface is present
      2. All necessary information can be entered by the user
         1. A title can be entered
         2. A team can be entered
         3. A description can be entered
         4. A deadline can be entered
            1. A button to open the deadline widgets is present such that it is clearly optional
      3. A button to create the task is present
         1. Valid tasks are added to the database with the “backlog” state
            1. A title is present
            2. A team is chosen
            3. The deadline, if present, is a real future date
         2. Invalid tasks trigger a pop-up to notify the user
      4. Once completed, the task interface is reloaded
   4. Tasks can be edited
      1. A button to enter an “Edit Task” interface is present
      2. All of the tasks’ details can be edited (except for the task’s ID number)
         1. The description can be edited
         2. The deadline can be edited
         3. The importance can be changed
         4. The team can be changed
      3. A button to save changes is present
         1. Valid changes are saved to the database
            1. The deadline is a future date
         2. Invalid changes trigger a pop-up to notify the user
      4. Once completed, the task interface is reloaded
   5. Tasks can be deleted
      1. A button to delete tasks is present
         1. The user is asked to confirm that they want to delete this task before progressing
         2. The task is deleted from the database
         3. The task is removed from the interface
   6. Tasks can be progressed (i.e. moved to the next state)
      1. A button to progress tasks is present for tasks in the “backlog”, “to do”, or “doing” state
         1. For tasks in the “backlog” or “to do” state, a “Progress Task” interface is entered
            1. All necessary information can be entered

For tasks progressing to “To Do”, an importance and a deadline can be entered

Deadline entry widgets are only present if a deadline is yet to be picked

For tasks progressing to “Doing”, assignees can be entered

* + - * 1. A button to confirm the progression is present

Valid information is added to the database

An importance is chosen (“Backlog”)

The deadline is a future date (“Backlog”)

At least one assignee is chosen (“To Do”)

Invalid information triggers a pop-up to notify the user

* + - * 1. Once completed, the task interface is reloaded
      1. For tasks in the “doing” state, the user is asked to confirm that they want to progress this task
         1. The task’s state is changed to “done”
         2. The task is moved to the next column

1. **An interface to view, edit, and create events is present**
   1. All necessary events and deadlines are present and visible
      1. The events’ titles are visible
      2. The deadlines’ titles are visible
      3. The deadlines’ importances are visible
   2. Events and deadlines are organised such that their dates are clear
      1. Events and deadlines are organised in a 7x4 grid showing 4 weeks of dates
         1. The grid is coloured to make it clear which days have passed
         2. The grid is coloured to make it clear which days are weekdays and which are weekends
   3. Events can be created
      1. A button to enter a “New Event” interface is present
      2. All necessary information can be entered by the user
         1. A title can be entered
         2. A date can be entered
      3. A button to create the event is present
         1. Valid events are added to the database
            1. A title is present
            2. The date is a real future date
         2. Invalid events trigger a pop-up to notify the user
      4. Once completed, the event interface is reloaded
   4. Events can be deleted
      1. A button to enter a “Remove Event” interface is present
      2. An event to delete can be entered
      3. A button to delete the event is present
         1. If an event has been selected, it is deleted from the database
         2. If an event hasn’t been selected, trigger a pop-up to notify the user
      4. Once completed, the event interface is reloaded
2. **An interface for admins to control the accounts of employees is present**
   1. All accounts are present and visible
      1. The accounts’ full names are visible
      2. The accounts’ usernames are visible
      3. The accounts’ teams are visible
      4. Whether an account is an admin account is visible
      5. Whether an account is active (i.e. whether or not the user will be required to change their password upon login) is visible
   2. Accounts are organised in a list
      1. It is possible to scroll through the account list
      2. Accounts are organised alphabetically by full name
   3. Accounts can be created
      1. A button to enter a “New Account” interface is present
      2. All necessary information can be entered by the user
         1. A first and last name can be entered
         2. A team can be entered
         3. The account can be made admin
         4. A temporary password can be entered
      3. A button to create the account is present
         1. Valid accounts are added to the database
            1. A first and last name is present
            2. A team is chosen
            3. The temporary password passes the strength check referenced under 1.2.2
            4. A username is generated automatically from first and last name

Username should have the form: first letter of first name + last name + smallest available positive integer

* + - 1. Invalid accounts trigger a pop-up to notify the user
    1. Once completed, the account interface is reloaded
  1. Accounts can be edited/deleted
     1. Widgets to edit the account are present
        1. An “Admin” checkbox is present
        2. A “Active” checkbox is present
        3. A team dropdown menu is present
        4. A button to save changes is present
           1. Changes are saved to the database
     2. A button to delete accounts is present
        1. The user is asked to confirm that they want to delete this account before progressing
        2. The account is deleted from the database
        3. The account is removed from the interface

**Potential Algorithms**

**A Sorting Algorithm**

It is highly likely that a sorting algorithm will be required by the program (e.g. to sort accounts or tasks). Below, I consider the pros and cons of insertion sorts, bubble sorts, and merge sorts.

| **Sorting Algorithm** | **Pros** | **Cons** |
| --- | --- | --- |
| Insertion sort | * Easy to understand and implement * Performs well on nearly sorted or small lists (O(n) time complexity in the best case) * Doesn’t require additional memory beyond the list being sorted (i.e. no pointers) | * Performs badly on large lists (O(n²) time complexity in the worst case) * Performance deteriorates significantly if the input array is not partially sorted |
| Bubble sort | * Very easy to understand and implement * Performs well on small lists (O(n) time complexity in the best case) | * Performs badly on large lists (O(n²) time complexity in the worst case) * Involves a large number of redundant comparisons |
| Merge sort | * Performs well on large lists (O(n log n) time complexity in all cases) | * Relatively complex to implement |

Although the sorting algorithm is unlikely to sort large quantities of data, merge sorts are still the most efficient for smaller quantities of data. Additionally, the use of a merge sort would lend to the scalability of the application, allowing efficiency to be maintained even if the user base greatly increases. Hence, despite the additional complexity, it would be worthwhile to use a merge sort over an insertion or bubble sort.

The following sort takes a key (a function that defines the metric used for sorting) to allow it to be used more widely within my program.

| **Merge Sort** | |
| --- | --- |
| def merge\_sort(data, key=lambda x: x):  if len(data) <= 1:  return data  mid = len(data) // 2  left = merge\_sort(data[:mid], key)  right = merge\_sort(data[mid:], key)  return \_merge(left, right, key)  def \_merge(left, right, key):  # underscore signals the subroutine is for use internally  result = []  i = j = 0  # i is left index, j is right index  while i < len(left) and j < len(right):  if key(left[i]) < key(right[j]):  result.append(left[i])  i += 1  else:  result.append(right[j])  j += 1  while i < len(left):  result.append(left[i])  i += 1  while j < len(right):  result.append(right[j])  j += 1  return result | |

**SQL Algorithms**

As I will be using lots of SQL, it would be very useful to have model algorithms for selecting, updating, and deleting as well as a model DDL algorithm.

| **Admin Check** |
| --- |
| A function to check whether an account stored in the database is an admin. |
| def admin(username):  conn = sq.connect("team\_tracker.db")  cur = conn.cursor()  cur.execute("SELECT Admin FROM Account WHERE Username = ?", (username,))  admin = cur.fetchall()[0][0]  cur.close()  conn.close()  return bool(admin) |

| **New Password** |
| --- |
| A function to update the password associated with an account. |
| def set\_new\_password(username, new\_password):  conn = sq.connect("team\_tracker.db")  cur = conn.cursor()  cur.execute(  "UPDATE Account SET Password = ?, Active = 1 WHERE Username = ?",  (new\_password, username),  )  conn.commit()  cur.close()  conn.close() |

| **Task Deletion** |
| --- |
| A function to delete a task and its corresponding assignments from the database. |
| def delete\_task(task\_id):  conn = sq.connect("team\_tracker.db")  cur = conn.cursor()  cur.execute("""DELETE FROM Assignment WHERE TaskID = ?""", (task\_id,))  cur.execute("DELETE FROM Task WHERE TaskID = ?", (task\_id,))  conn.commit()  cur.close()  conn.close() |

| **Create Event Table** |
| --- |
| A function to create the event table of the database. |
| def event():  conn = sq.connect("team\_tracker.db")  cur = conn.cursor()  cur.execute(  """CREATE TABLE "Event" (  "EventID" INTEGER,  "Title" TEXT,  "Date" TEXT,  PRIMARY KEY("EventID")  )"""  )  cur.close()  conn.close() |

**A Queue Class**

A queue would be useful for processing tasks one at a time. The program could queue up all the tasks and sequentially dequeue them, possibly providing an option to send them to the back of the queue. The benefits of using a circular queue come from its efficient reuse of space which would lend itself well to this idea of sending some items to the back of the queue once you reach them.

| **Circular Queue** | |
| --- | --- |
| class CircularQueue:  def \_\_init\_\_(self, \_\_size):  self.\_\_size = \_\_size  self.\_\_queue = [None] \* \_\_size  self.\_\_front = -1  self.\_\_rear = -1  def is\_empty(self):  return self.\_\_front == -1  def is\_full(self):  return (self.\_\_rear + 1) % self.\_\_size == self.\_\_front  def enqueue(self, item):  if self.is\_full():  return  if self.is\_empty():  self.\_\_front = 0  self.\_\_rear = (self.\_\_rear + 1) % self.\_\_size  self.\_\_queue[self.\_\_rear] = item  def dequeue(self):  if self.is\_empty():  return None  item = self.\_\_queue[self.\_\_front]  if self.\_\_front == self.\_\_rear:  self.\_\_front = -1  self.\_\_rear = -1  else:  self.\_\_front = (self.\_\_front + 1) % self.\_\_size  return item | |

**A Hash Function**

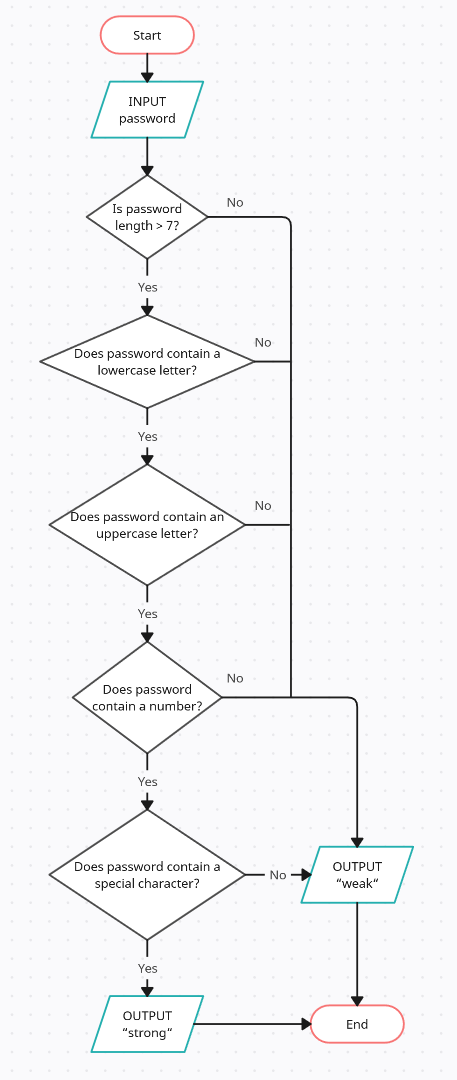
I plan to use hashing techniques to increase the security and efficiency of my program. For example, a hash function like the one below could be used to sort usernames into a hash table in order to test for uniqueness.

| **Hash Function** | |
| --- | --- |
| def hash(input):  output = 0  pos\_multiplier = 31  for i, char in enumerate(list(input)):  output += ord(char) \* (pos\_multiplier\*\*i)  return output % TABLE\_SIZE | |

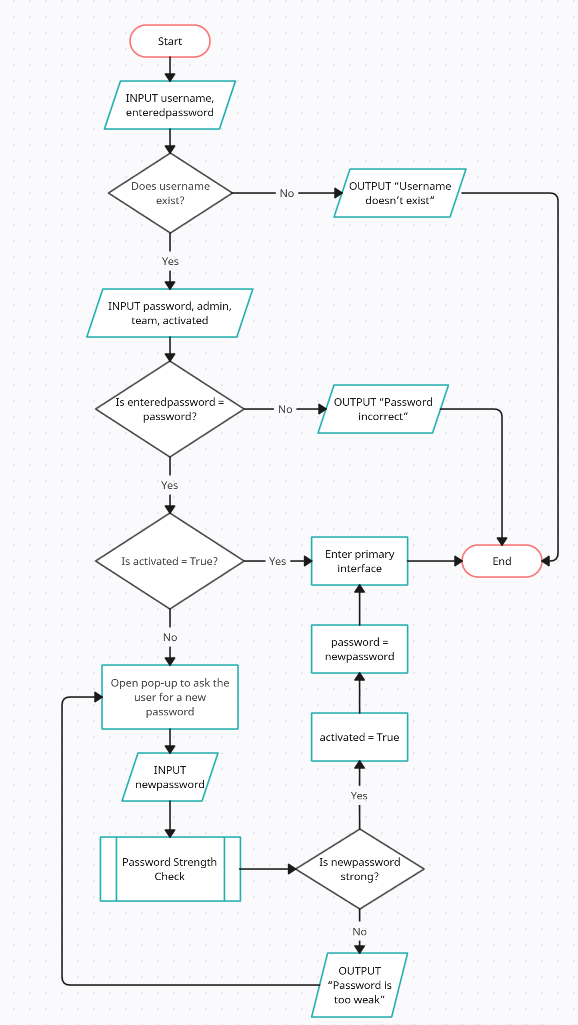
**Flowcharts For The Account System**

Below are first-draft flowchart diagrams to some of the processes that will make up the account system.

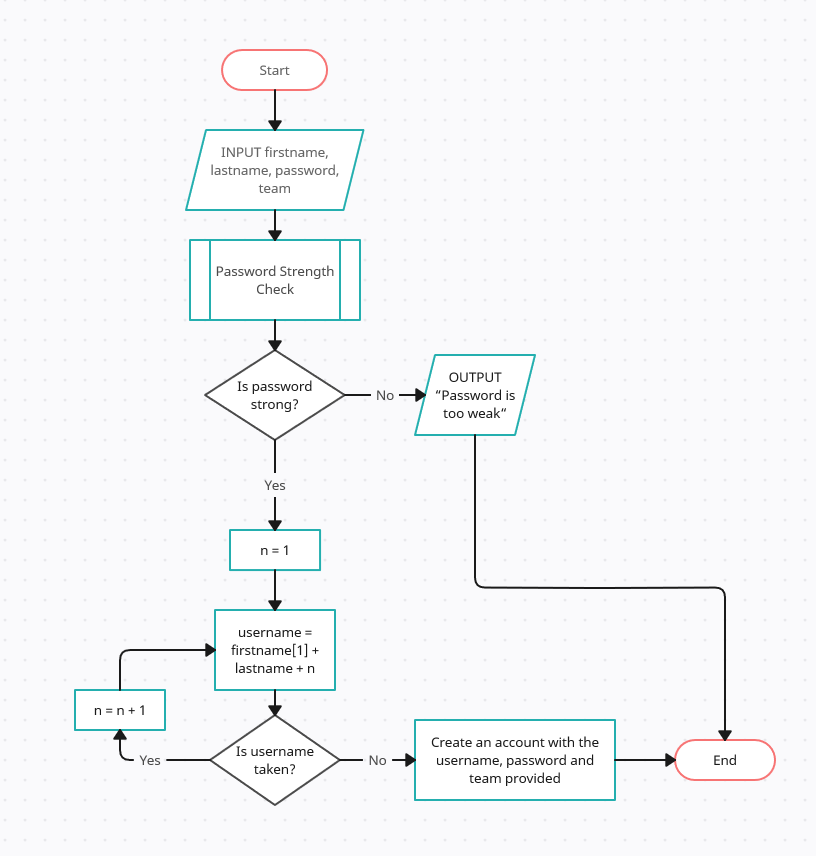
**Password Strength Check**



**Login**



**Add Account**



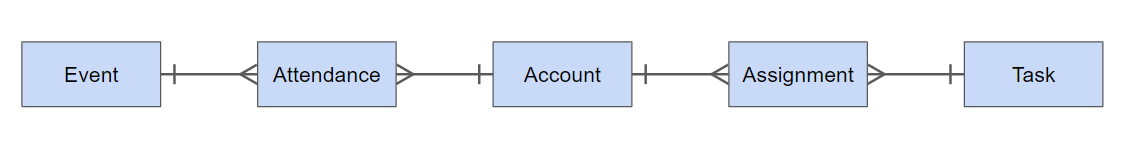
**A Database Analysis**

**Entity Relationship Diagram**

**The diagram with many-to-many relationships:**



**The diagram with the many-to-many relationships removed:**



**Entity Definitions**

**Account**

Account(Username, FirstName, LastName, Password, Team, Admin, Active)

**Task**

Task(TaskID, Title, Description, State, Deadline, Importance, Team)

**Assignment**

Assignment(Username, TaskID)

**Event**

Event(EventID, Title, Date)

**Attendance**

Attendee(Username, EventID)

**Entity Dictionaries**

**Account**

| **Field Name** | **Type Affinity** | **Key Type** | **Criteria For Validity** | **Example** |
| --- | --- | --- | --- | --- |
| Username | TEXT | Primary | Must be of the form; first letter of first name + last name + smallest available positive integer | tgeary1 |
| FirstName | TEXT | Secondary | Must be present | Greg |
| LastName | TEXT | Secondary | Must be present | Heffley |
| Password | BLOB | Secondary | Must be 8 or more characters in length, must contain a special character, must contain a number, must contain an uppercase letter, must contain a lowercase letter, must be encrypted | Mr!Man1999 (decrypted) |
| Team | TEXT | Secondary | Must be “operational”, “development”, or “both” | operational |
| Admin | INTEGER | Secondary | Must be Boolean (0 or 1) | 0 |
| Active | INTEGER | Secondary | Must be Boolean (0 or 1) | 1 |

**Task**

| **Field Name** | **Type Affinity** | **Key Type** | **Criteria For Validity** | **Example** |
| --- | --- | --- | --- | --- |
| TaskID | INTEGER | Primary | Must be one more than the greatest ID number upon creation | 184 |
| Title | TEXT | Secondary | Must be present | Buy a coffee machine |
| State | TEXT | Secondary | Must be “backlog”, “to do”, “doing”, or “done” | doing |
| Description | TEXT | Secondary | None | Ensure that the machine costs less than £200. |
| Deadline | TEXT | Secondary | Must be a future date of the form year-month-day or it must be “0-0-0” | 2026-08-11 |
| BaseImportance | INTEGER | Secondary | Must be 0, 1 (low), 2 (medium), or 3 (high) | 2 |
| Team | TEXT | Secondary | Must be “operational”, “development”, or “both” | operational |

**Assignment**

| **Field Name** | **Type Affinity** | **Key Type** | **Criteria For Validity** | **Example** |
| --- | --- | --- | --- | --- |
| Username | TEXT | Primary, Foreign (Account) | Must be of the form; first letter of first name + last name + smallest available positive integer | tgeary1 |
| TaskID | INTEGER | Primary, Foreign (Task) | Must be the smallest available positive integer | 184 |

**Event**

| **Field Name** | **Type Affinity** | **Key Type** | **Criteria For Validity** | **Example** |
| --- | --- | --- | --- | --- |
| EventID | INTEGER | Primary | Must be one more than the greatest ID number upon creation | 48 |
| Title | TEXT | Secondary | Must be present | Talent show |
| Date | TEXT | Secondary | Must be a future date of the form; YYYY-MM-DD | 2026-08-11 |

**Attendance**

| **Field Name** | **Type Affinity** | **Key Type** | **Criteria For Validity** | **Example** |
| --- | --- | --- | --- | --- |
| Username | TEXT | Primary, Foreign (Account) | Must be of the form; first letter of first name + last name + smallest available positive integer | tgeary1 |
| EventID | INTEGER | Primary, Foreign (Event) | Must be the smallest available positive integer | 48 |

**IPSO Diagrams For Adding New Items**

**Adding Tasks**

| **Input** | Title, Team, Description, Deadline |
| --- | --- |
| **Process** | Check that a title is present.  Check that a team has been selected.  Check that the deadline is valid (i.e. must be in the future, must be a date that will actually occur). |
| **Store** | Store the data in the Task table of the database. |
| **Output** | Either an error message if any of the checks come back negative, detailing which check has come back negative or a message reading “Task Added”. |

**Adding Events**

| **Input** | Title, Date, Attendees |
| --- | --- |
| **Process** | Check that a title is present.  Check that the date is valid (i.e. must be in the future, must be a date that will actually occur).  Check that at least one attendee has been selected. |
| **Store** | Store the data in the Event table of the database. |
| **Output** | Either an error message if any of the checks come back negative, detailing which check has come back negative or a message reading “Event Added”. |

**Adding Accounts**

| **Input** | First Name, Last Name, Team, Admin, Password |
| --- | --- |
| **Process** | Check that a title is present.  Check that the date is valid (i.e. must be in the future, must be a date that will actually occur). |
| **Store** | Store the data in the Account table of the database. |
| **Output** | Either an error message if any of the checks come back negative, detailing which check has come back negative or a message reading “Account Added”. |

**Design**

**The Interface**

**Format Details**

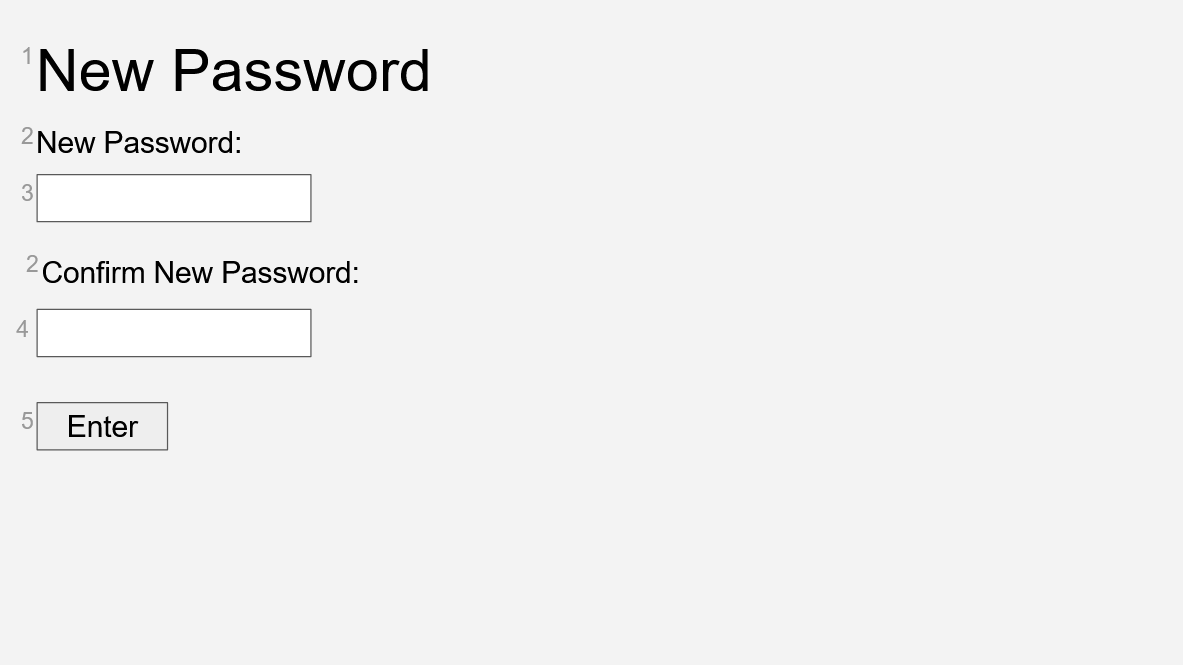
All text is black and uses the Arial font unless stated otherwise. All widgets use the default Tkinter appearance which generally consists of various shades of grey. All backgrounds are plain and light grey. ”Type” refers to the type of widget in Tkinter that will be used and the brackets that follow determine the font size and other relevant formatting information.

**Figure 1.1 - Login**



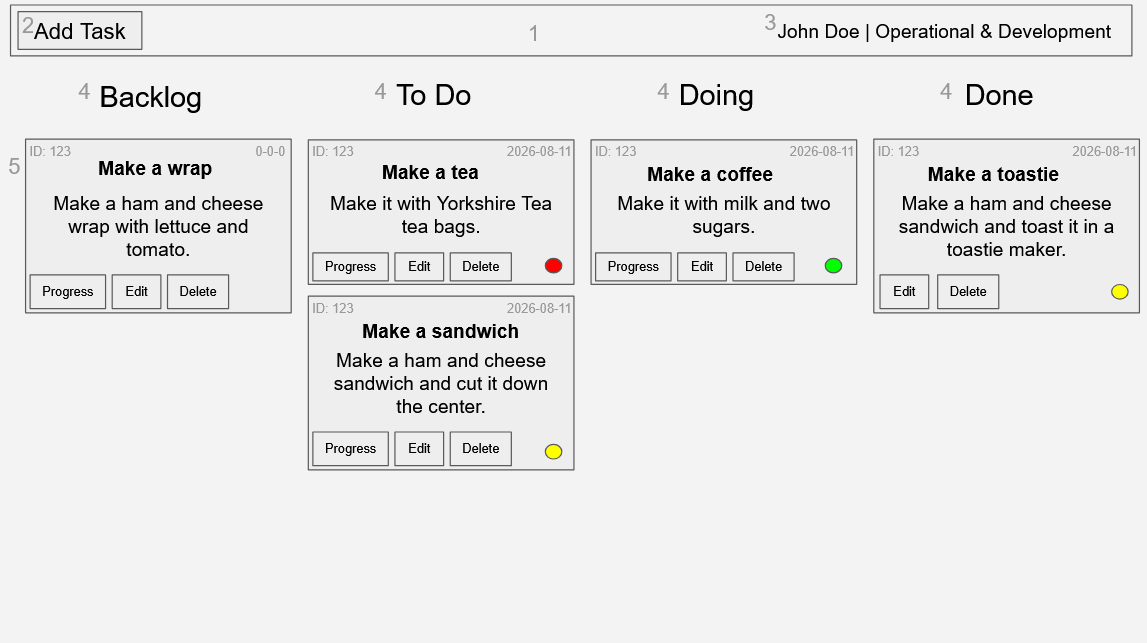
| **Index** | **Type** | **Purpose** |
| --- | --- | --- |
| 1 | Label (48) | Titles the page. |
| 2 | Label (24) | Labels the following widget. |
| 3 | Entry (18) | Allows entry of a username. |
| 4 | Entry (18) | Allows entry of a password. |
| 5 | Button (24) | Allows the user to submit the login details. |

**Figure 1.2 - New Password**



| **Index** | **Type** | **Purpose** |
| --- | --- | --- |
| 1 | Label (48) | Titles the page. |
| 2 | Label (24) | Labels the following widget. |
| 3 | Entry (18) | Allows entry of a new password. |
| 4 | Entry (18) | Allows entry of the new password again to check that the password is as intended. |
| 5 | Button (24) | Allows the user to submit the new password. |

**Figure 2.1 - Task Interface**



| **Index** | **Type** | **Purpose** |
| --- | --- | --- |
| 1 | Frame | Contains the toolbar widgets. |
| 2 | Button (12) | Allows the addition of new tasks. |
| 3 | Label (12) | Shows the user’s name and team. |
| 4 | Label (18) | Labels the task columns. |
| 5 | Task Widget (see Figure 2.1.1) | Shows a task and its details. |

**Figure 2.1.1 - Task Widget**



| **Index** | **Type** | **Purpose** |
| --- | --- | --- |
| 1 | Frame | Contains the other widgets. |
| 2 | Label (8, grey) | Shows the task’s ID number. |
| 3 | Label (8, grey) | Shows the task’s deadline. |
| 4 | Label (12, bold) | Shows the task’s title. |
| 5 | Label (12) | Shows the task’s description. |
| 6 | Button (12) | Allows progression of the task. |
| 7 | Button (12) | Allows the user to edit the task. |
| 8 | Button (12) | Deletes the task. |
| 9 | Label | Contains a dot representing the task’s importance (green - low, yellow - medium, orange - high, red - critical, purple - deadline passed). |

**Figure 2.2 - New Task**



| **Index** | **Type** | **Purpose** |
| --- | --- | --- |
| 1 | Label (48) | Titles the page. |
| 2 | Label (24) | Labels the following widget. |
| 3 | Entry (18) | Allows entry of a title for the new task. |
| 4 | ComboBox (18) | Allows selection of the team(s) responsible for the task. |
| 5 | Text (18) | Allows entry of a description for the new task. |
| 6 | Button (24) | Opens a series of widgets that allow the entry of a deadline. See Figure 2.3 (indexes 2, 5, and 6) for the resulting widgets. |
| 7 | Button (24) | Allows the user to add the task detailed above to the database. |

**Figure 2.3 - Edit Task**



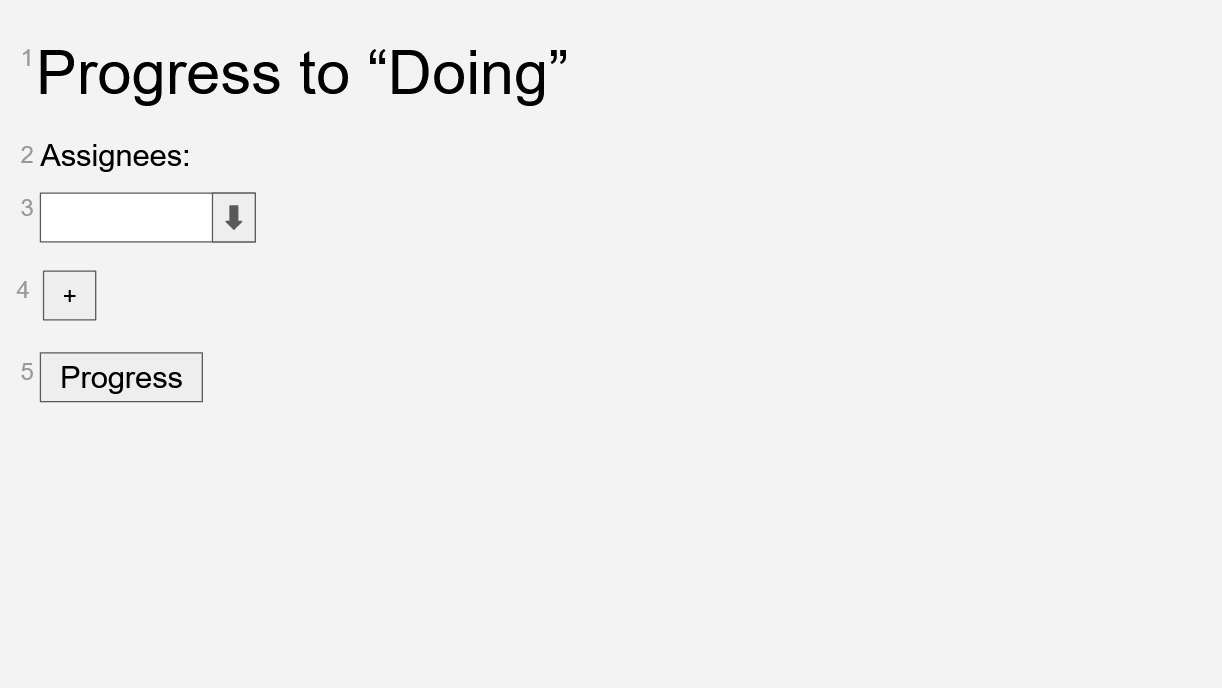
| **Index** | **Type** | **Purpose** |
| --- | --- | --- |
| 1 | Label (48) | Titles the page. |
| 2 | Label (24) | Labels the following widget. |
| 3 | ComboBox (18) | Allows selection of the team(s) responsible for the task. |
| 4 | Text (18) | Allows entry of a description for the new task. |
| 5 | Label (18) | Labels the deadline entry boxes. |
| 6 | Entry (18) | Allows entry of the deadline. |
| 7 | ComboBox (18) | Allows selection of the task’s importance. |
| 8 | Button (24) | Allows the user to save the changes detailed above. |

**Figure 2.4 - Progress to “To Do”**



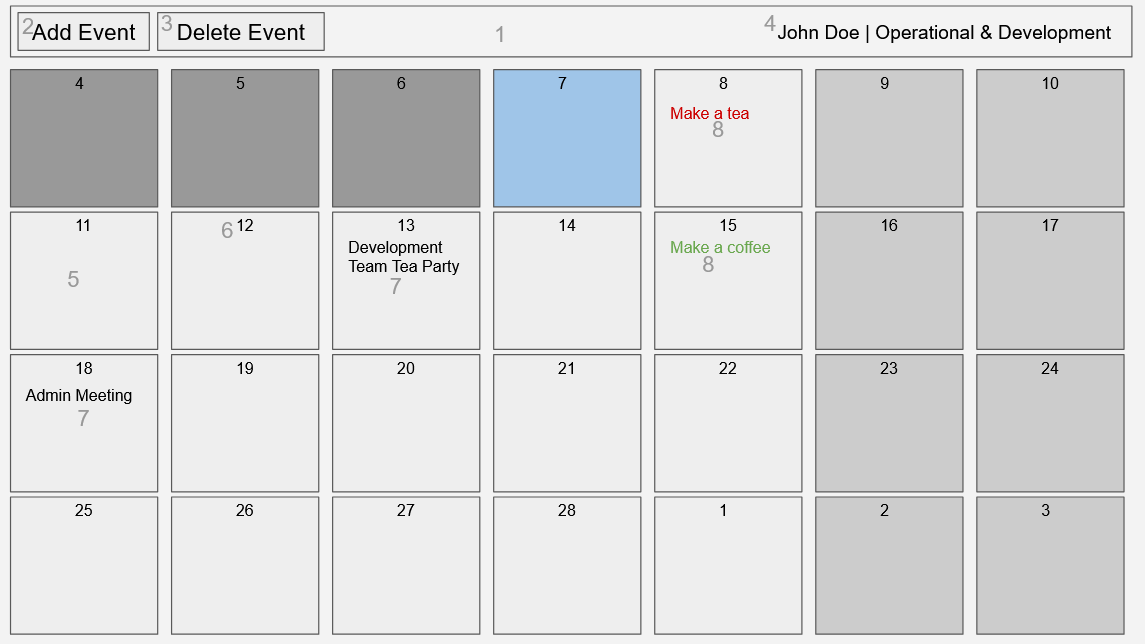
| **Index** | **Type** | **Purpose** |
| --- | --- | --- |
| 1 | Label (48) | Titles the page. |
| 2 | Label (24) | Labels the following widget. |
| 3 | ComboBox (18) | Allows selection of the task’s importance. |
| 4 | Label (18) | Labels the deadline entry boxes. |
| 5 | Entry (18) | Allows entry of the deadline. |
| 6 | Button (24) | Allows the user to progress the task to “To Do”. |

**Figure 2.5 - Progress to “Doing”**



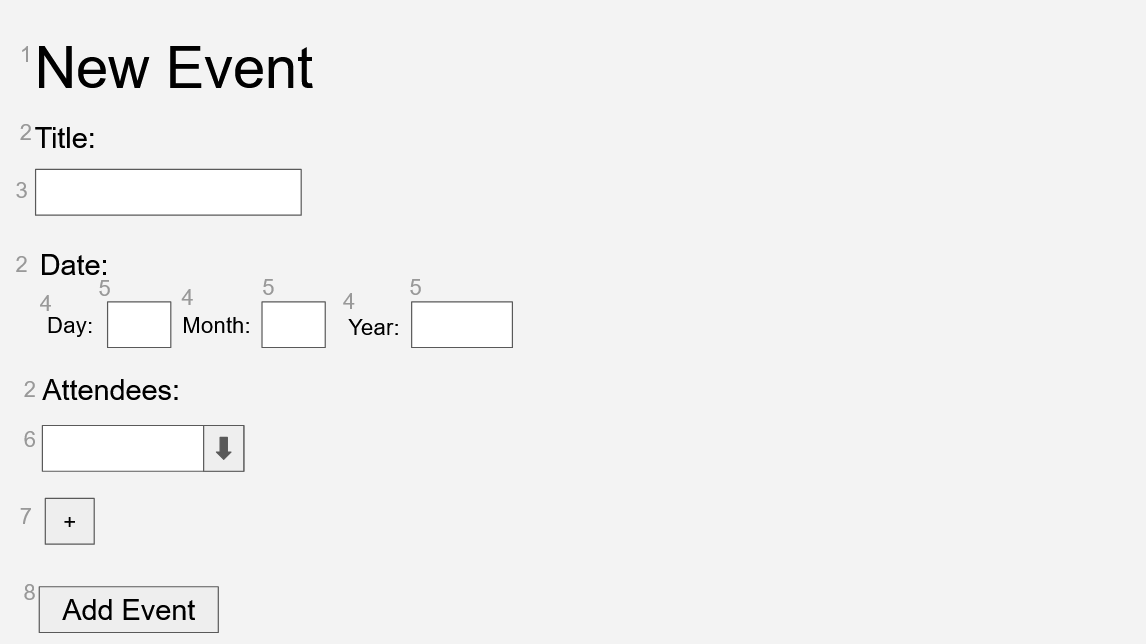
| **Index** | **Type** | **Purpose** |
| --- | --- | --- |
| 1 | Label (48) | Titles the page. |
| 2 | Label (24) | Labels the following widget. |
| 3 | ComboBox (18) | Allows selection of the employees that are doing the task. |
| 4 | Button (24) | Allows more assignees to be added by creating more ComboBox widgets. |
| 5 | Button (24) | Allows the user to progress the task to “Doing”. |

**Figure 3.1 - Timeline Interface**



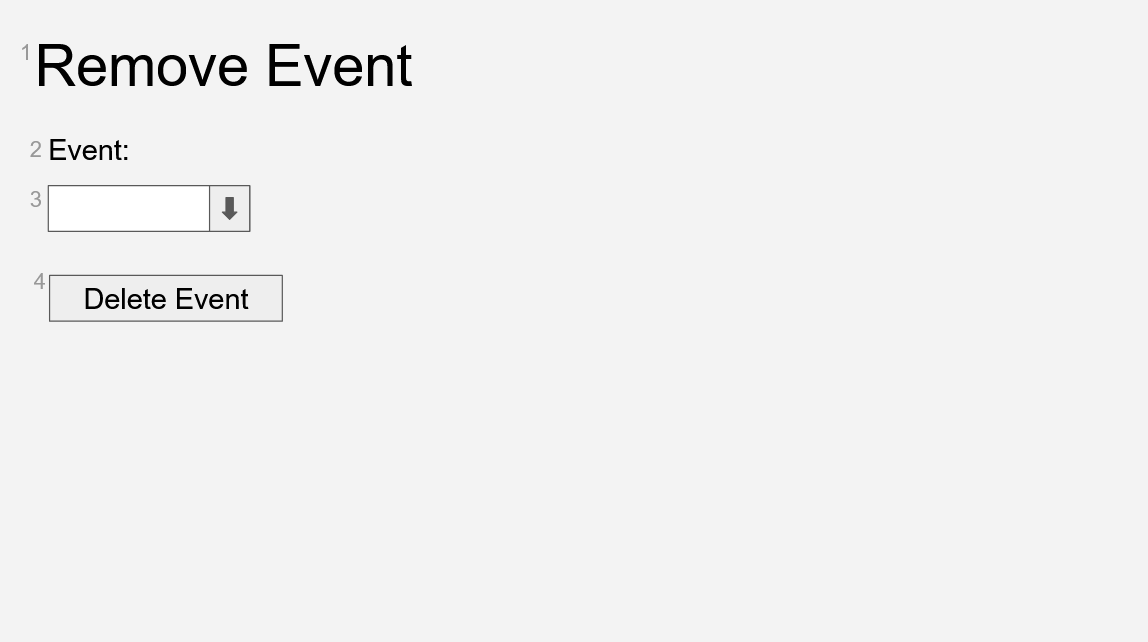
| **Index** | **Type** | **Purpose** |
| --- | --- | --- |
| 1 | Frame | Contains the toolbar widgets. |
| 2 | Button (12) | Allows the addition of new events. |
| 3 | Button (12) | Allows the deletion of events. |
| 4 | Label (12) | Shows the user’s name and team. |
| 5 | Frame | Contains the widgets relevant to a specific date with the background representing the type of day (light grey - weekday, medium grey - weekend, dark grey - in the past, light blue - today). |
| 6 | Label (8) | Shows the date represented by a cell. |
| 7 | Label (12) | Shows the title of an event. |
| 8 | Label (12) | Shows the title of a task with a deadline on that day in which its colour represents its importance. |

**Figure 3.2 - New Event**



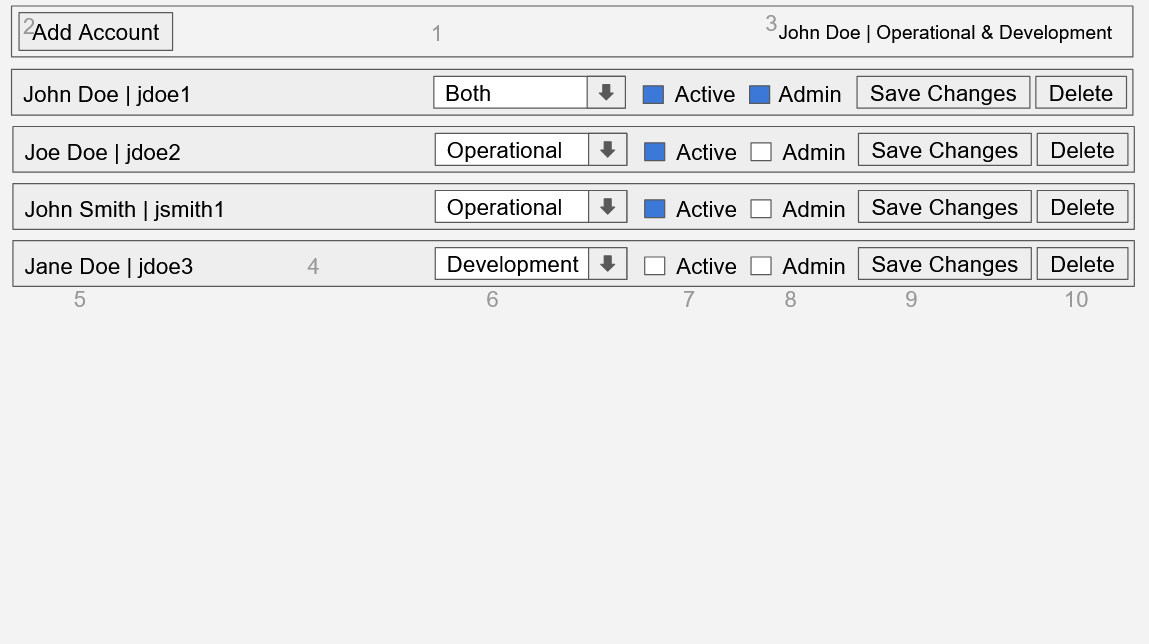
| **Index** | **Type** | **Purpose** |
| --- | --- | --- |
| 1 | Label (48) | Titles the page. |
| 2 | Label (24) | Labels the following widget. |
| 3 | Entry (18) | Allows entry of a title for the event. |
| 4 | Label (18) | Labels the date entry boxes. |
| 5 | Entry (18) | Allows entry of the event date. |
| 6 | ComboBox (18) | Allows selection of the employees that are attending the event. |
| 7 | Button (24) | Allows more attendees to be added by creating more ComboBox widgets. |
| 8 | Button (24) | Allows the user to add the event detailed above to the database. |

**Figure 3.3 - Remove Event**



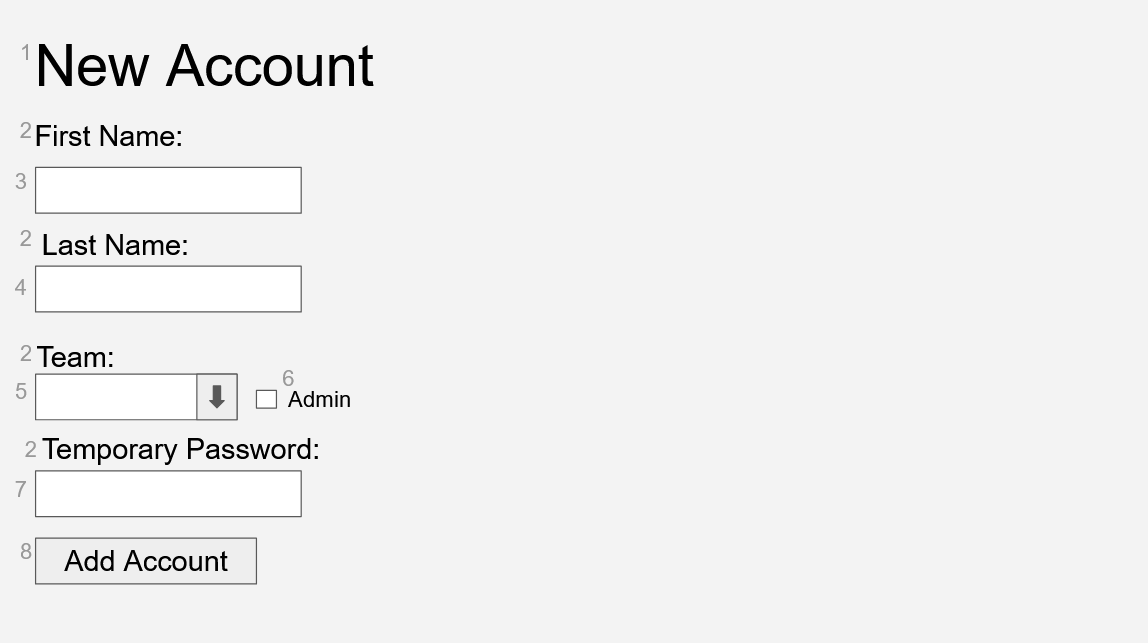
| **Index** | **Type** | **Purpose** |
| --- | --- | --- |
| 1 | Label (48) | Titles the page. |
| 2 | Label (24) | Labels the following widget. |
| 6 | ComboBox (18) | Allows selection of the event to be deleted. |
| 8 | Button (24) | Allows the user to delete the specified event from the database. |

**Figure 4.1 - Account Interface (Admins Only)**



| **Index** | **Type** | **Purpose** |
| --- | --- | --- |
| 1 | Frame | Contains the toolbar widgets. |
| 2 | Button (12) | Allows the addition of new accounts. |
| 3 | Label (12) | Shows the user’s name and team. |
| 4 | Frame | Contains each account’s corresponding widgets. |
| 5 | Label (12) | Shows the account’s name and username |
| 6 | ComboBox (18) | Allows selection of the account’s team(s). |
| 7 | Checkbutton (18) | Allows the user to toggle whether or not the account is active (i.e. whether or not the user will be required to change their password upon login). |
| 8 | Checkbutton (18) | Allows the user to toggle whether or not the account is an admin account. |
| 9 | Button (12) | Allows the user to save the changes detailed within that account’s widgets. |
| 10 | Button (12) | Allows the deletion of accounts. |

**Figure 4.2 - New Account**



| **Index** | **Type** | **Purpose** |
| --- | --- | --- |
| 1 | Label (48) | Titles the page. |
| 2 | Label (24) | Labels the following widget. |
| 3 | Entry (18) | Allows entry of a first name. |
| 4 | Entry (18) | Allows entry of a last name. |
| 5 | ComboBox (18) | Allows selection of the account’s team(s). |
| 6 | Checkbutton (18) | Allows the user to toggle whether or not the new account will be an admin account. |
| 7 | Entry (18) | Allows entry of a temporary password. |
| 8 | Button (24) | Allows the user to add the account detailed above to the database. |

**The Database**

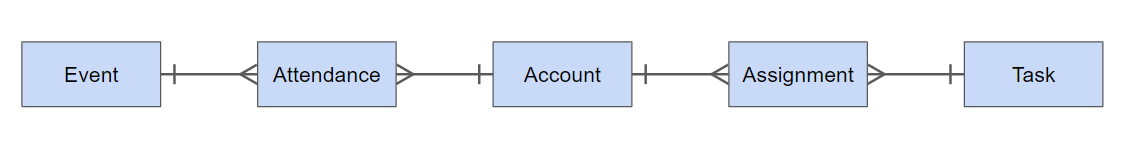
After revisiting the database analysis that I did, I’ve decided that no changes are needed as the database is already comprehensive and fully normalised. It is copied below:

**Entity Relationship Diagram**

**The diagram with many-to-many relationships:**



**The diagram with the many-to-many relationships removed:**



**Entity Definitions**

**Account**

Account(Username, FirstName, LastName, Password, Team, Admin, Active)

**Task**

Task(TaskID, Title, Description, State, Deadline, Importance, Team)

**Assignment**

Assignment(Username, TaskID)

**Event**

Event(EventID, Title, Date)

**Attendance**

Attendee(Username, EventID)

**Entity Dictionaries**

**Account**

| **Field Name** | **Type Affinity** | **Key Type** | **Criteria For Validity** | **Example** |
| --- | --- | --- | --- | --- |
| Username | TEXT | Primary | Must be of the form; first letter of first name + last name + smallest available positive integer | tgeary1 |
| FirstName | TEXT | Secondary | Must be present | Greg |
| LastName | TEXT | Secondary | Must be present | Heffley |
| Password | BLOB | Secondary | Must be 8 or more characters in length, must contain a special character, must contain a number, must contain an uppercase letter, must contain a lowercase letter, must be encrypted | Mr!Man1999 (decrypted) |
| Team | TEXT | Secondary | Must be “operational”, “development”, or “both” | operational |
| Admin | INTEGER | Secondary | Must be Boolean (0 or 1) | 0 |
| Active | INTEGER | Secondary | Must be Boolean (0 or 1) | 1 |

**Task**

| **Field Name** | **Type Affinity** | **Key Type** | **Criteria For Validity** | **Example** |
| --- | --- | --- | --- | --- |
| TaskID | INTEGER | Primary | Must be one more than the greatest ID number upon creation | 184 |
| Title | TEXT | Secondary | Must be present | Buy a coffee machine |
| State | TEXT | Secondary | Must be “backlog”, “to do”, “doing”, or “done” | doing |
| Description | TEXT | Secondary | None | Ensure that the machine costs less than £200. |
| Deadline | TEXT | Secondary | Must be a future date of the form year-month-day or it must be “0-0-0” | 2026-08-11 |
| BaseImportance | INTEGER | Secondary | Must be 0, 1 (low), 2 (medium), or 3 (high) | 2 |
| Team | TEXT | Secondary | Must be “operational”, “development”, or “both” | operational |

**Assignment**

| **Field Name** | **Type Affinity** | **Key Type** | **Criteria For Validity** | **Example** |
| --- | --- | --- | --- | --- |
| Username | TEXT | Primary, Foreign (Account) | Must be of the form; first letter of first name + last name + smallest available positive integer | tgeary1 |
| TaskID | INTEGER | Primary, Foreign (Task) | Must be the smallest available positive integer | 184 |

**Event**

| **Field Name** | **Type Affinity** | **Key Type** | **Criteria For Validity** | **Example** |
| --- | --- | --- | --- | --- |
| EventID | INTEGER | Primary | Must be one more than the greatest ID number upon creation | 48 |
| Title | TEXT | Secondary | Must be present | Talent show |
| Date | TEXT | Secondary | Must be a future date of the form; YYYY-MM-DD | 2026-08-11 |

**Attendance**

| **Field Name** | **Type Affinity** | **Key Type** | **Criteria For Validity** | **Example** |
| --- | --- | --- | --- | --- |
| Username | TEXT | Primary, Foreign (Account) | Must be of the form; first letter of first name + last name + smallest available positive integer | tgeary1 |
| EventID | INTEGER | Primary, Foreign (Event) | Must be the smallest available positive integer | 48 |

**Flowcharts**

The following flowcharts are organised into five broad groups:

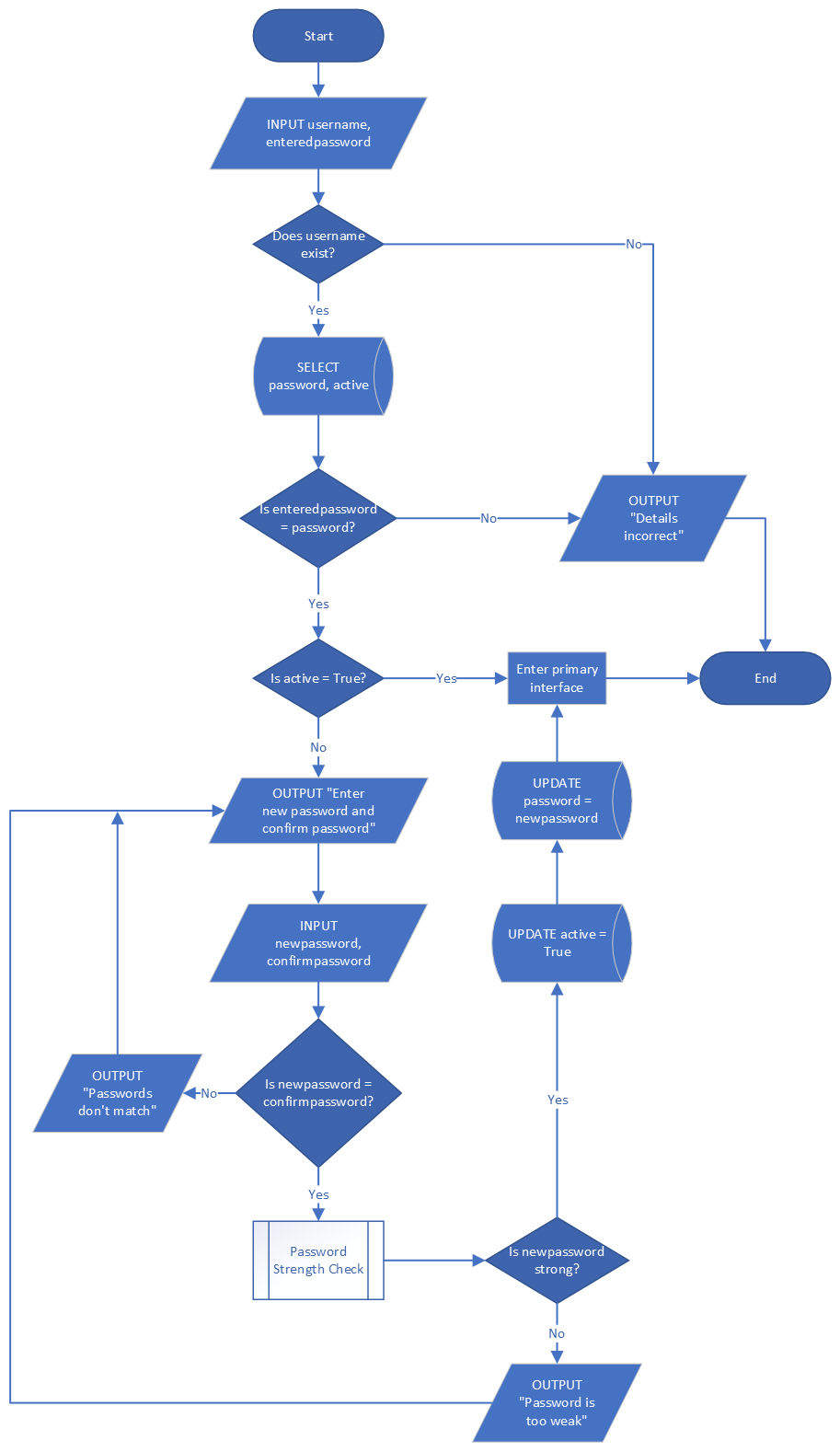
1. Login System
2. Task System
3. Event System
4. Account System
5. Miscellaneous

It is worth noting that I have used the input shape to exclusively refer to information that requires extra steps to be accessed (e.g. user inputs or accessing widget data). This means that some information being used within the flowchart is not explicitly inputted. For example, the Delete Task flowchart is for when the delete button on the task’s own widget is pressed. Within the flowchart, the database is queried to delete the task which would require the task ID, however the task ID won’t be explicitly inputted as it will already be accessible as an attribute. This is done so that I can more easily map the logic of the flowcharts into my program.

Additionally, due to the interactive nature of my program, some components have been abstracted down for the purpose of having useful flowcharts.

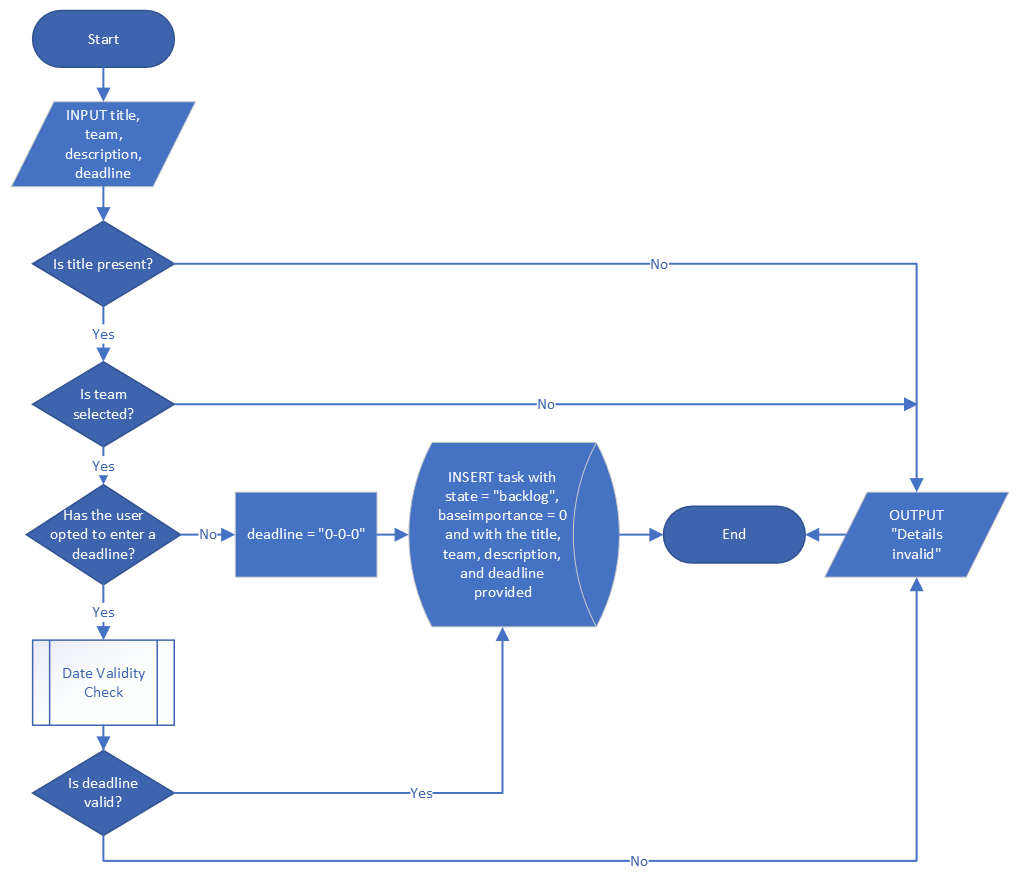
**1 - Login**

Handles user logins and password changes.



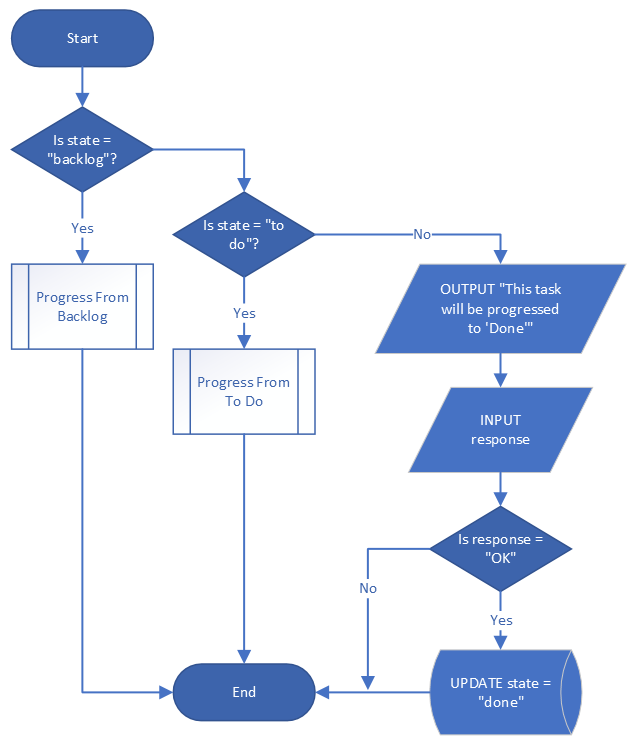
**2 - Add Task**

Allows new tasks to be created.



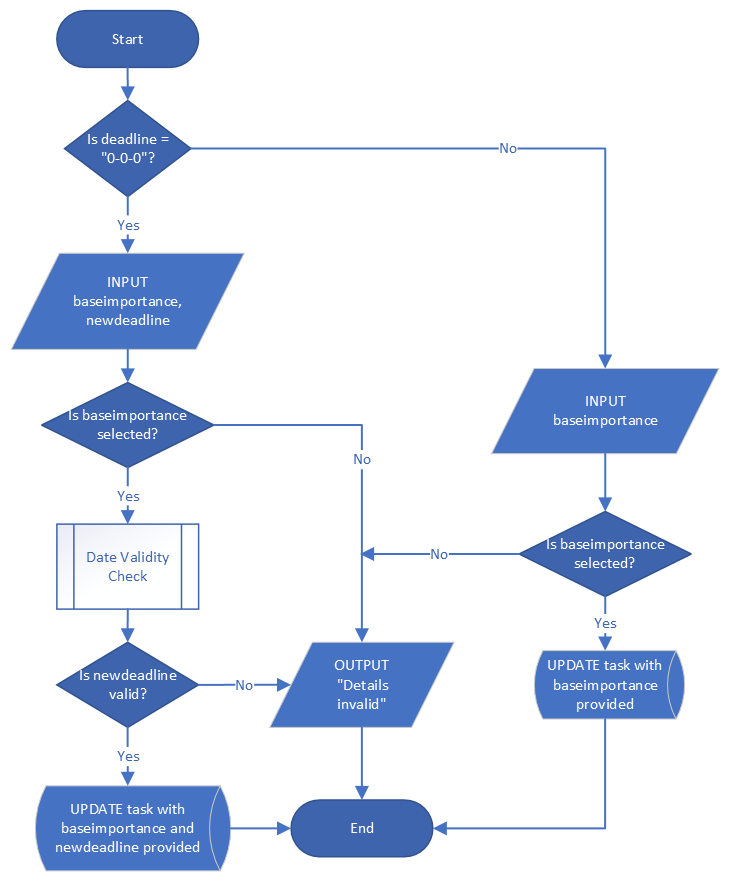
**2 - Progress Task**

Provides the general process for the progression of any task to the next state.



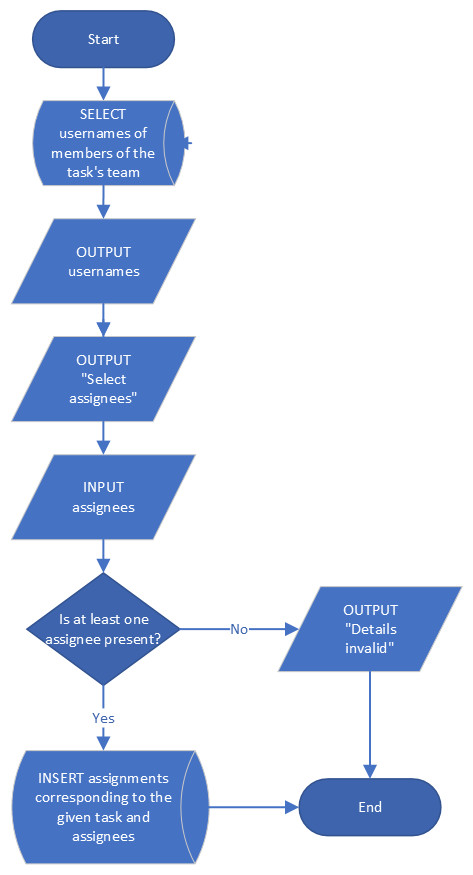
**2 - Progress From “Backlog”**

Allows tasks to be progressed from the state “Doing” to “To Do”.



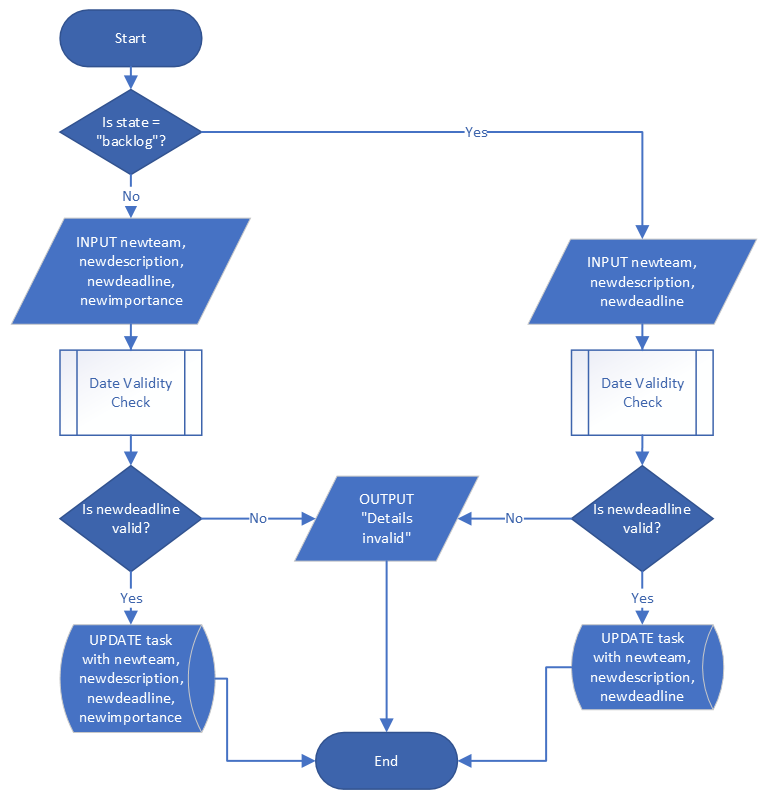
**2 - Progress From “To Do”**

Allows tasks to be progressed from the state “To Do” to “Doing”.



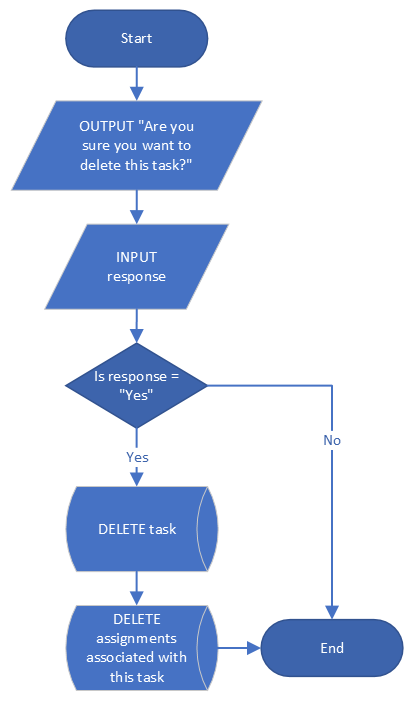
**2 - Edit Task**

Allows tasks to be edited.



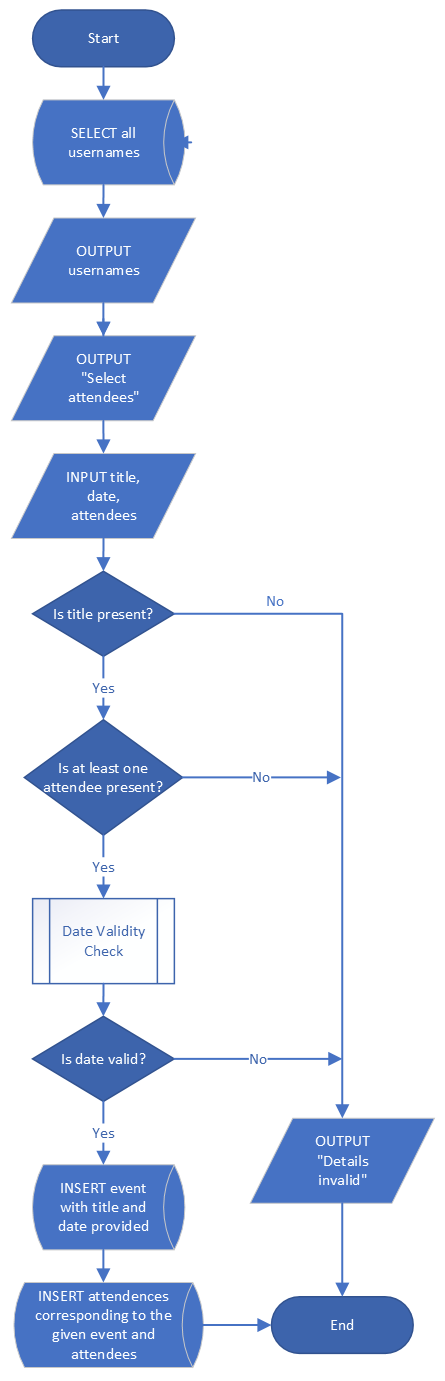
**2 - Delete Task**

Deletes tasks after a confirmation message.



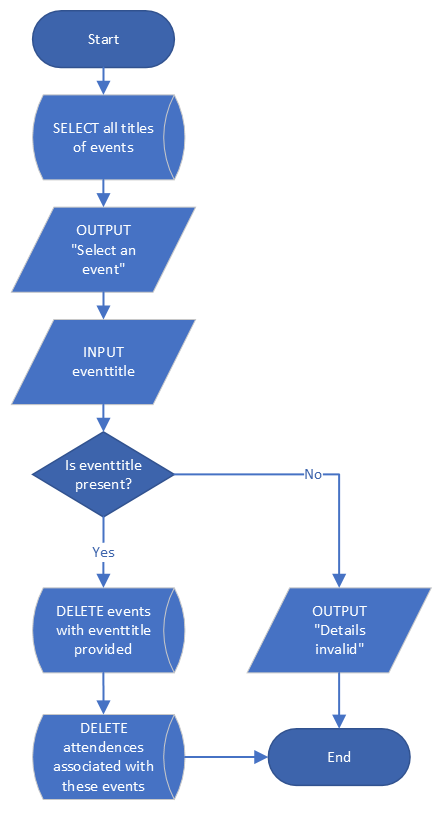
**3 - Add Event**

Allows new events to be created.



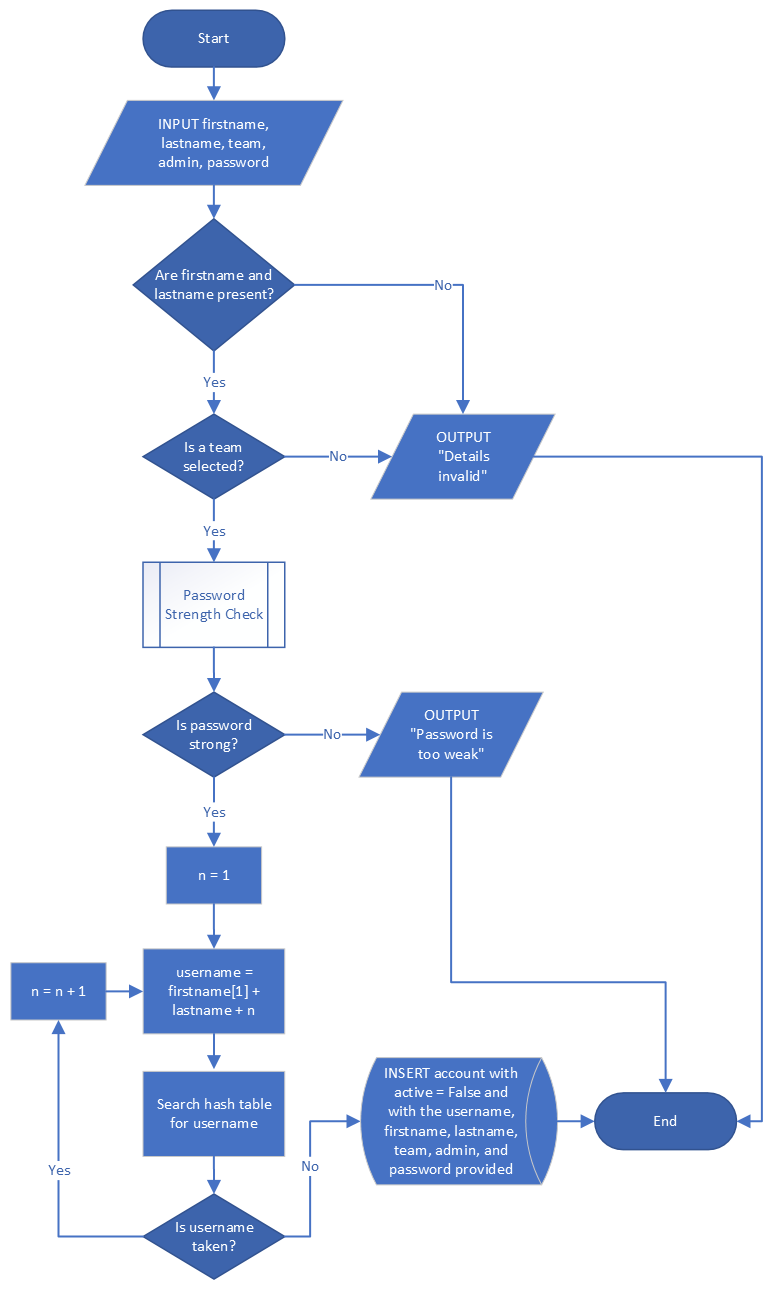
**3 - Delete Event**

Allows events to be deleted.



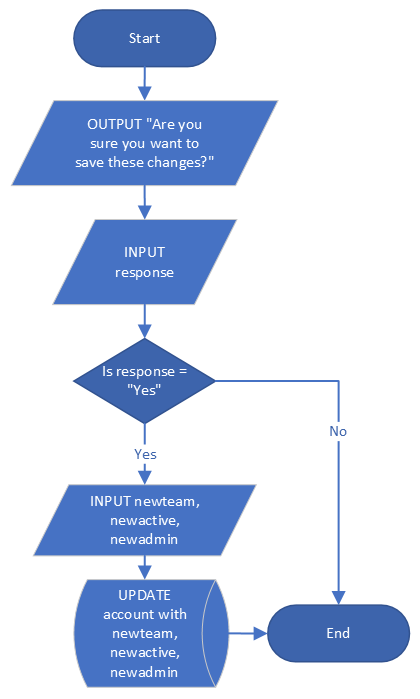
**4 - Add Account**

Allows new accounts to be created.



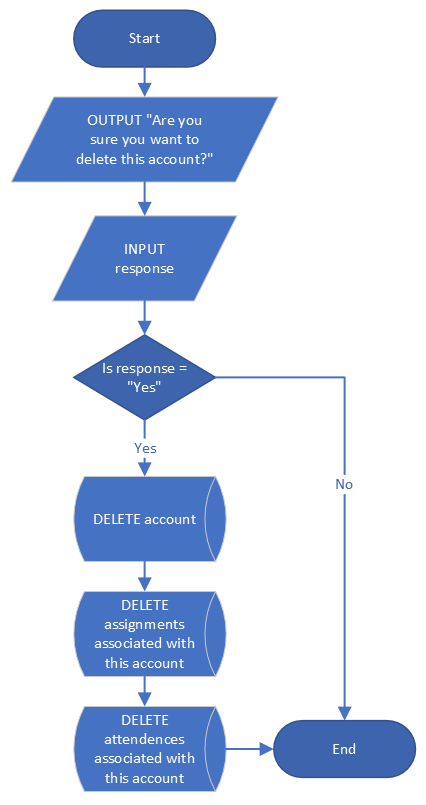
**4 - Save Account Changes**

Saves any changes made to the account details via the user interface.



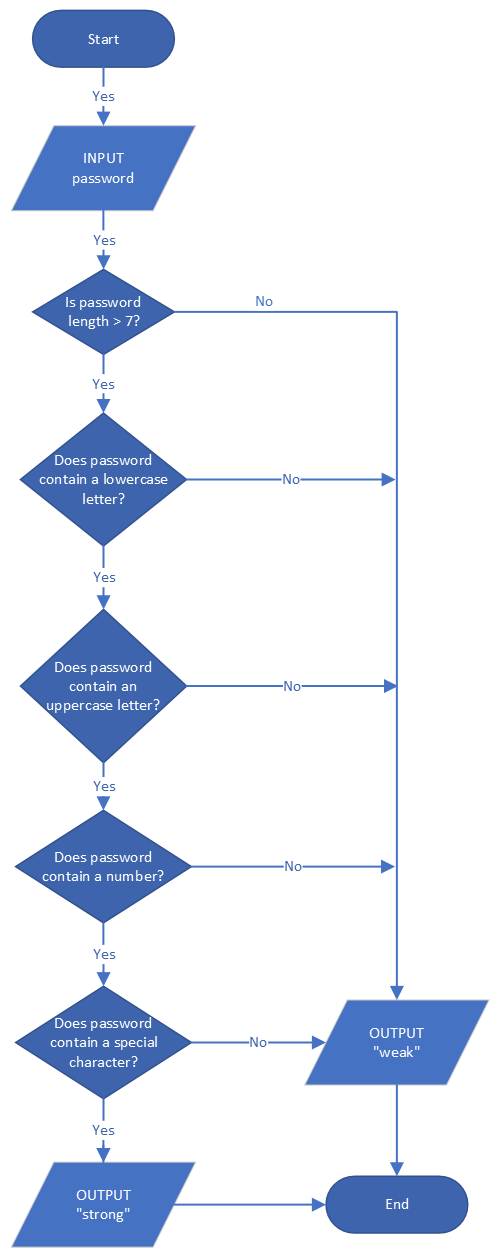
**4 - Delete Account**

Deletes accounts after a confirmation message.



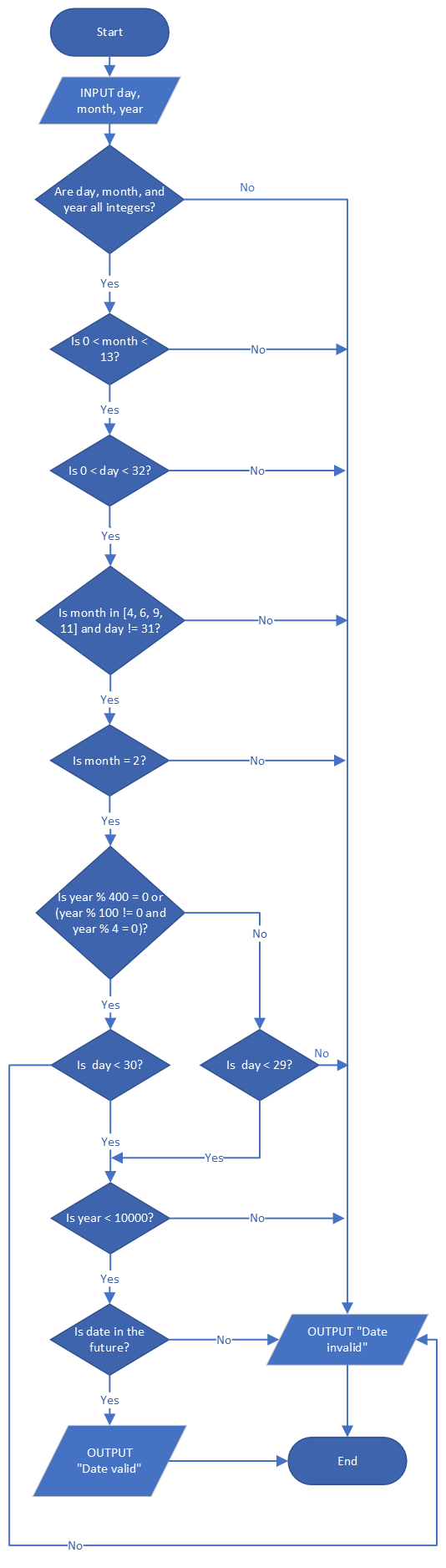
**5 - Password Strength Check**

Ensures a given password matches the criteria for validity.



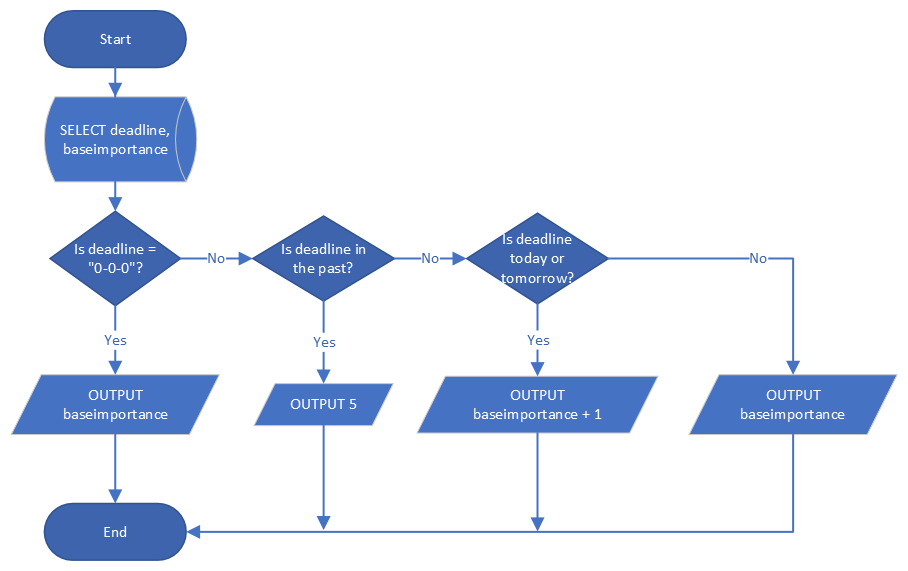
**5 - Date Validity Check**

Ensures a given date is valid and in the future.



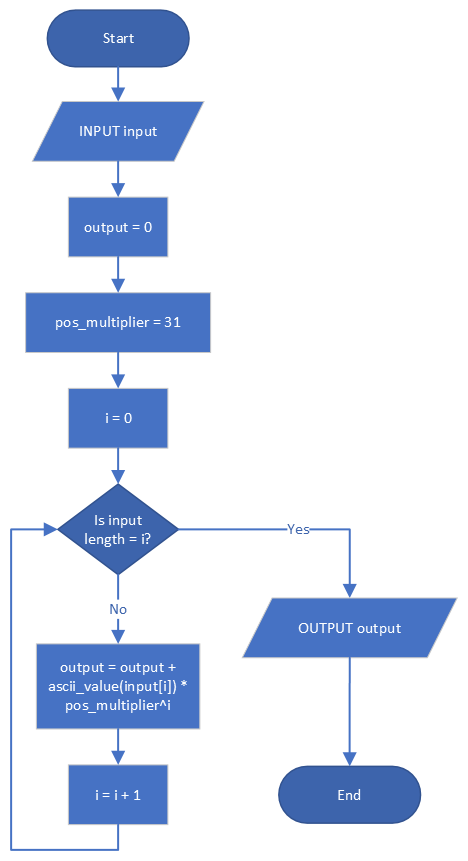
**5 - Determine Importance**

Determines what level of importance a task should be treated with.



**5 - Hash**

Determines the hashed value for a particular input.



**Implementation**

**The following programs—along with the database, a binary hash table, and a key file—compose my entire application.**

| **team\_tracker.py** |
| --- |
| import tkinter as tk  from circular\_queue import CircularQueue  from tkinter import ttk, messagebox, scrolledtext  from sql\_functions import \*  from misc\_functions import \*  FONT\_1 = ("Arial", 7)  FONT\_2 = ("Arial", 10)  FONT\_3 = ("Arial", 15)  FONT\_4 = ("Arial", 20)  FONT\_5 = ("Arial", 25)  FONT\_6 = ("Arial", 40)  ROW\_HEIGHT = 33  IMPORTANCES = ["Low", "Medium", "High"]  IMPORTANCE\_COLOURS = [  "#006400", # green  "#ea9400", # gold  "#ff5733", # salmon  "#c70039", # crimson  "#4b0082", # purple  "#00008b", # navy  ]  ### window  class Window(tk.Tk):  def \_\_init\_\_(self, minwidth, minheight):  super().\_\_init\_\_()  self.title("Team Tracker")  self.geometry(f"{minwidth}x{minheight}")  self.minsize(minwidth, minheight)  self.login()  def login(self):  Login(self)  def new\_password(self, username):  NewPassword(self, username)  def main(self, username):  Main(self, username)  ### login  class Login(tk.Frame):  def \_\_init\_\_(self, master):  super().\_\_init\_\_(master, height=11 \* ROW\_HEIGHT, width=10000)  # width = 10000 is an awkward solution to the frame not filling the  # screen width while grid\_progagate is false  self.fill\_frame()  self.grid\_propagate(False)  # this stops the frame resizing to fill its container  self.grid(row=0, column=0, padx=20, pady=20)  def fill\_frame(self):  ## create widgets  self.login\_label = tk.Label(self, text="Login", font=FONT\_6)  self.username\_label = tk.Label(self, text="Username:", font=FONT\_4)  self.username\_entry = tk.Entry(self, font=FONT\_3)  self.password\_label = tk.Label(self, text="Password:", font=FONT\_4)  self.password\_entry = tk.Entry(self, show="\*", font=FONT\_3)  self.enter\_button = tk.Button(  self, text="Enter", font=FONT\_4, command=self.enter  )  ## create grid  self.grid\_rowconfigure(  tuple([n for n in range(11)]), weight=1, uniform="a"  )  # defines 11 equally sized rows  ## grid widgets  self.login\_label.grid(row=0, column=0, rowspan=2, sticky="w")  self.username\_label.grid(row=3, column=0, sticky="w")  self.username\_entry.grid(row=4, column=0, sticky="w")  self.password\_label.grid(row=6, column=0, sticky="w")  self.password\_entry.grid(row=7, column=0, sticky="w")  self.enter\_button.grid(row=9, column=0, rowspan=2, sticky="w")  def enter(self):  username = self.username\_entry.get()  password = self.password\_entry.get()  if login\_valid(username, password):  if active(username):  self.destroy()  self.master.main(username)  else:  messagebox.showwarning("Warning", "A new password is required")  self.destroy()  self.master.new\_password(username)  else:  messagebox.showerror("Error", "Invalid username or password")  class NewPassword(tk.Frame):  def \_\_init\_\_(self, master, username):  super().\_\_init\_\_(master, height=11 \* ROW\_HEIGHT, width=10000)  self.username = username  self.fill\_frame()  self.grid\_propagate(False)  self.grid(row=0, column=0, padx=20, pady=20)  def fill\_frame(self):  ## create widgets  self.new\_password\_title\_label = tk.Label(  self, text="New Password", font=FONT\_6  )  self.new\_password\_label = tk.Label(  self, text="New Password:", font=FONT\_4  )  self.new\_password\_entry = tk.Entry(self, show="\*", font=FONT\_3)  self.confirm\_new\_password\_label = tk.Label(  self, text="Confirm New Password:", font=FONT\_4  )  self.confirm\_new\_password\_entry = tk.Entry(self, show="\*", font=FONT\_3)  self.enter\_button = tk.Button(  self, text="Enter", font=FONT\_4, command=self.enter  )  ## create grid  self.grid\_rowconfigure(  tuple([n for n in range(11)]), weight=1, uniform="a"  )  ## grid widgets  self.new\_password\_title\_label.grid(  row=0, column=0, rowspan=2, sticky="w"  )  self.new\_password\_label.grid(row=3, column=0, sticky="w")  self.new\_password\_entry.grid(row=4, column=0, sticky="w")  self.confirm\_new\_password\_label.grid(row=6, column=0, sticky="w")  self.confirm\_new\_password\_entry.grid(row=7, column=0, sticky="w")  self.enter\_button.grid(row=9, column=0, rowspan=2, sticky="w")  def enter(self):  if (  self.new\_password\_entry.get()  == self.confirm\_new\_password\_entry.get()  ):  new\_password = self.new\_password\_entry.get()  if strong(new\_password):  set\_new\_password(self.username, new\_password)  self.destroy()  self.master.main(self.username)  else:  messagebox.showerror(  "Error",  "Password not strong enough, try using a longer password or a larger variety of character types",  )  else:  messagebox.showerror("Error", "Passwords do not match")  ### main  class Main(ttk.Notebook):  def \_\_init\_\_(self, master, username):  super().\_\_init\_\_(master)  self.username = username  self.admin = admin(self.username)  self.fill\_notebook()  self.pack(fill="both", expand=True)  def fill\_notebook(self):  ## add tabs  self.task\_interface = TaskInterface(self, self.username)  self.event\_interface = EventInterface(self, self.username)  self.add(self.task\_interface, text="Tasks")  self.add(self.event\_interface, text="Events")  if self.admin:  self.account\_interface = AccountInterface(self, self.username)  self.add(self.account\_interface, text="Accounts")  ## events  self.bind("<<NotebookTabChanged>>", lambda event: self.tab\_change())  # refreshes tabs as they are changed to  # lambda used to ensure correct number of positional arguments given  def tab\_change(self):  if self.index("current") == 0:  self.task\_interface.winfo\_children()[0].destroy()  self.task\_interface.task\_menu()  elif self.index("current") == 1:  self.event\_interface.winfo\_children()[0].destroy()  self.event\_interface.calendar()  elif self.index("current") == 2:  self.account\_interface.winfo\_children()[0].destroy()  self.account\_interface.account\_menu()  ### task interface  class TaskInterface(tk.Frame):  def \_\_init\_\_(self, master, username):  super().\_\_init\_\_(master)  self.username = username  self.sort\_through\_backlog\_action = ""  self.task\_menu()  def task\_menu(self):  TaskMenu(self, self.username)  def new\_task(self):  NewTask(self)  def progress\_task(self, task\_id):  if get\_task(task\_id)[3] == "backlog":  ProgressTaskFromBacklog(self, task\_id)  else:  ProgressTaskFromToDo(self, task\_id)  def edit\_task(self, task\_id):  EditTask(self, task\_id)  def sort\_through\_backlog(self):  queue = CircularQueue(100)  for task in get\_tasks(team(self.username), 0):  queue.enqueue(task)  while not queue.is\_empty():  task = queue.dequeue()  progress\_interface = ProgressTaskFromBacklog(  self, task[0], sort=True  )  # ensures the following code only runs once the  # ProgressTaskFromBacklog window is closed  while progress\_interface.get\_running():  self.update()  if self.sort\_through\_backlog\_action == "cancel":  self.set\_sort\_through\_backlog\_action("")  break  if self.sort\_through\_backlog\_action == "send to back":  queue.enqueue(task)  self.set\_sort\_through\_backlog\_action("")  self.task\_menu()  def set\_sort\_through\_backlog\_action(self, action):  self.sort\_through\_backlog\_action = action  class TaskMenu(tk.Frame):  def \_\_init\_\_(self, master, username):  super().\_\_init\_\_(master)  self.username = username  self.scroll\_toggle = True  self.fill\_frame()  self.pack(fill="both", expand=True)  def fill\_frame(self):  ## toolbar  self.toolbar = tk.Frame(self, relief="ridge", bd=4)  self.toolbar.pack(fill="x")  self.add\_task\_button = tk.Button(  self.toolbar, text="Add Task", font=FONT\_2, command=self.new\_task  )  self.sort\_through\_backlog\_button = tk.Button(  self.toolbar,  text="Sort Through Backlog",  font=FONT\_2,  command=self.sort\_through\_backlog,  )  self.clear\_done\_button = tk.Button(  self.toolbar,  text='Clear "Done"',  font=FONT\_2,  command=self.clear\_done,  )  self.add\_task\_button.pack(side="left", padx=4, pady=4)  self.sort\_through\_backlog\_button.pack(side="left", padx=4, pady=4)  self.clear\_done\_button.pack(side="left", padx=4, pady=4)  self.stats\_button = tk.Button(  self.toolbar,  text="Stats",  font=FONT\_2,  command=lambda: messagebox.showinfo(  "Stats", stats\_string(self.username)  ),  )  self.stats\_button.pack(side="right", padx=4, pady=4)  self.account\_info\_label = tk.Label(  self.toolbar, text=account\_info\_string(self.username), font=FONT\_2  )  self.account\_info\_label.pack(side="right")  ## canvas  self.canvas = tk.Canvas(self)  self.canvas.pack(fill="both", expand=True)  ## canvas frame  self.canvas\_frame = tk.Frame(self.canvas)  self.canvas\_frame.columnconfigure(  tuple([n for n in range(4)]), weight=1, uniform="a"  )  for i in range(4):  TaskMenuColumn(self.canvas\_frame, self.username, i)  ## events  self.bind\_all("<MouseWheel>", self.scroll)  # ensures the above event is destroyed along with the canvas  self.canvas.bind(  "<Destroy>", lambda event: self.unbind\_all("<MouseWheel>")  )  # <Configure> triggers when the widget size is changed i.e. when the  # window is resized  self.canvas.bind(  "<Configure>",  lambda event: self.update\_size(event.width, event.height),  )  def update\_size(self, canvas\_width, canvas\_height):  if canvas\_height < self.canvas\_frame.winfo\_reqheight():  height = self.canvas\_frame.winfo\_reqheight()  self.scroll\_toggle = True  self.canvas.configure(  scrollregion=(  0,  0,  canvas\_width,  self.canvas\_frame.winfo\_reqheight(),  )  )  else:  height = canvas\_height  self.scroll\_toggle = False  self.canvas.create\_window(  (0, 0),  window=self.canvas\_frame,  anchor="nw",  width=canvas\_width,  height=height,  )  def scroll(self, event):  # the height aspect of update\_size() is unreliable during window  # resizing (due to text wrapping in task descriptions) so must be  # recalled before scrolling  self.update\_size(self.canvas.winfo\_width(), self.canvas.winfo\_height())  if self.scroll\_toggle:  self.canvas.yview\_scroll(int(-event.delta / 120), "units")  def new\_task(self):  self.destroy()  self.master.new\_task()  def sort\_through\_backlog(self):  self.destroy()  self.master.sort\_through\_backlog()  def clear\_done(self):  clear\_done()  self.destroy()  self.master.task\_menu()  class TaskMenuColumn(tk.Frame):  def \_\_init\_\_(self, master, username, column):  super().\_\_init\_\_(master)  self.username = username  self.column = column  if self.column == 0:  self.title = "Backlog"  elif self.column == 1:  self.title = "To Do"  elif self.column == 2:  self.title = "Doing"  elif self.column == 3:  self.title = "Done"  self.fill\_frame()  self.grid(row=0, column=self.column, sticky="nesw", pady=5)  def fill\_frame(self):  ## title  self.title\_label = tk.Label(self, text=self.title, font=FONT\_3)  self.title\_label.pack()  ## tasks  self.tasks = get\_tasks(team(self.username), self.column)  for task in self.tasks:  Task(self, task, self.username)  class Task(tk.Frame):  def \_\_init\_\_(self, master, task, username):  super().\_\_init\_\_(  master, relief="raised", bd=4, bg=get\_task\_bg(task[0], username)  )  self.task = task  self.username = username  self.bg = get\_task\_bg(self.task[0], self.username)  self.importance = importance(self.task[4], self.task[5])  self.fill\_frame()  self.pack(fill="x", padx=10, pady=5)  def fill\_frame(self):  ## create widgets  self.id\_label = tk.Label(  self,  text=f"ID:{self.task[0]}",  font=FONT\_1,  fg="grey",  bg=self.bg,  )  self.deadline\_label = tk.Label(  self,  text=self.task[4],  font=FONT\_1,  fg="grey",  bg=self.bg,  )  self.title\_label = tk.Label(  self,  text=self.task[1],  font=FONT\_2 + ("bold",),  bg=self.bg,  )  self.description\_label = tk.Label(  self,  text=self.task[2],  font=FONT\_2,  bg=self.bg,  )  ## create button frame  self.button\_frame = tk.Frame(self, bg=self.bg)  self.progress\_button = tk.Button(  self.button\_frame,  text="Progress",  font=FONT\_2,  command=self.progress,  )  self.edit\_button = tk.Button(  self.button\_frame, text="Edit", font=FONT\_2, command=self.edit  )  self.delete\_button = tk.Button(  self.button\_frame, text="Delete", font=FONT\_2, command=self.delete  )  self.importance\_dot = tk.Label(  self.button\_frame,  text="⬤",  font=FONT\_2,  fg=IMPORTANCE\_COLOURS[self.importance - 1],  bg=self.bg,  )  if self.master.column != 3:  self.progress\_button.pack(side="left", padx=4, pady=4)  self.edit\_button.pack(side="left", padx=4, pady=4)  self.delete\_button.pack(side="left", padx=4, pady=4)  if self.master.column != 0:  self.importance\_dot.pack(side="right", padx=10)  ## create grid  self.grid\_columnconfigure(0, weight=1)  # ensures contents of the task frame are centered correctly  ## grid widgets  self.id\_label.grid(row=0, column=0, sticky="w")  self.deadline\_label.grid(row=0, column=0, sticky="e")  self.title\_label.grid(row=1, column=0)  if self.task[2]:  self.description\_label.grid(row=2, column=0, sticky="ew")  self.button\_frame.grid(row=3, column=0, sticky="nesw")  ## events  self.bind(  "<Configure>",  lambda event: self.description\_label.configure(  wraplength=event.width - 10  ),  )  # ensures description\_label wraps text to fit frame width whenever it is  # resized  def progress(self):  if self.master.column == 2:  answer = messagebox.askokcancel(  'Progress to "Done"',  f'{self.task[1]} will be progressed to "Done"',  )  if answer:  update\_state(self.task[0], "done")  self.master.master.master.master.destroy()  self.master.master.master.master.master.task\_menu()  else:  self.master.master.master.master.destroy()  self.master.master.master.master.master.progress\_task(self.task[0])  def edit(self):  self.master.master.master.master.destroy()  self.master.master.master.master.master.edit\_task(self.task[0])  def delete(self):  if messagebox.askyesno(  "Delete", "Are you sure you want to delete this task?"  ):  delete\_task(self.task[0])  self.destroy()  class NewTask(tk.Frame):  def \_\_init\_\_(self, master):  super().\_\_init\_\_(master, height=18 \* ROW\_HEIGHT, width=10000)  self.deadline = False  self.fill\_frame()  self.grid\_propagate(False)  self.grid(row=0, column=0, padx=20, pady=20)  def fill\_frame(self):  ## create widgets  self.new\_task\_label = tk.Label(self, text="New Task", font=FONT\_6)  self.title\_label = tk.Label(self, text="Title:", font=FONT\_4)  self.title\_entry = tk.Entry(self, font=FONT\_3)  self.team\_label = tk.Label(self, text="Team:", font=FONT\_4)  self.description\_label = tk.Label(  self, text="Description:", font=FONT\_4  )  self.description\_scrolledtext = scrolledtext.ScrolledText(  self, font=FONT\_3, height=2  )  self.add\_deadline\_button = tk.Button(  self, text="+ Deadline", font=FONT\_4, command=self.add\_deadline  )  self.deadline\_label = tk.Label(self, text="Deadline:", font=FONT\_4)  self.deadline\_entry\_frame = tk.Frame(self)  ## create team optionmenu  self.team\_optionmenu\_value = tk.StringVar(  self, value="Select a team..."  )  self.team\_optionmenu = tk.OptionMenu(  self,  self.team\_optionmenu\_value,  "Operational",  "Development",  "Both",  )  self.team\_optionmenu.configure(font=FONT\_3)  self.team\_optionmenu["menu"].configure(font=FONT\_2)  ## fill deadline entry frame  self.day\_label = tk.Label(  self.deadline\_entry\_frame, text="Day:", font=FONT\_4  )  self.day\_entry = tk.Entry(self.deadline\_entry\_frame, font=FONT\_3)  self.month\_label = tk.Label(  self.deadline\_entry\_frame, text="Month:", font=FONT\_4  )  self.month\_entry = tk.Entry(self.deadline\_entry\_frame, font=FONT\_3)  self.year\_label = tk.Label(  self.deadline\_entry\_frame, text="Year:", font=FONT\_4  )  self.year\_entry = tk.Entry(self.deadline\_entry\_frame, font=FONT\_3)  self.day\_label.pack(side="left")  self.day\_entry.pack(side="left")  self.month\_label.pack(side="left")  self.month\_entry.pack(side="left")  self.year\_label.pack(side="left")  self.year\_entry.pack(side="left")  ## create button frame  self.button\_frame = tk.Frame(self)  self.add\_task\_button = tk.Button(  self.button\_frame,  text="Add Task",  font=FONT\_4,  command=self.add\_task,  )  self.cancel\_button = tk.Button(  self.button\_frame,  text="Cancel",  font=FONT\_4,  command=self.reload\_task\_menu,  )  self.add\_task\_button.pack(side="left")  self.cancel\_button.pack(side="left", padx=8)  ## create grid  self.grid\_rowconfigure(  tuple([n for n in range(18)]), weight=1, uniform="a"  )  ## grid widgets  self.new\_task\_label.grid(row=0, column=0, rowspan=2, sticky="w")  self.title\_label.grid(row=3, column=0, sticky="w")  self.title\_entry.grid(row=4, column=0, sticky="w")  self.team\_label.grid(row=6, column=0, sticky="w")  self.team\_optionmenu.grid(row=7, column=0, sticky="w")  self.description\_label.grid(row=9, column=0, sticky="w")  self.description\_scrolledtext.grid(  row=10, column=0, rowspan=2, sticky="w"  )  self.add\_deadline\_button.grid(row=13, column=0, sticky="w")  self.button\_frame.grid(row=15, column=0, rowspan=2, sticky="w")  def add\_deadline(self):  # deadlines are optional here as sometimes deadlines will be concrete  # and known when the task is first created but sometimes a reasonable  # deadline will have to be assigned when the task is progressed to "to  # do"  self.add\_deadline\_button.grid\_forget()  self.deadline\_label.grid(row=13, column=0, sticky="w")  self.deadline\_entry\_frame.grid(row=14, column=0, sticky="w")  self.button\_frame.grid(row=16, column=0, rowspan=2, sticky="w")  self.deadline = True  def details\_valid(self):  if not self.title\_entry.get():  return False  if self.team\_optionmenu\_value.get() == "Select a team...":  return False  if self.deadline and not date\_valid(  self.day\_entry.get(), self.month\_entry.get(), self.year\_entry.get()  ):  return False  return True  def add\_task(self):  if self.details\_valid():  if self.deadline:  deadline = f"{int(self.year\_entry.get())}-{int(self.month\_entry.get())}-{int(self.day\_entry.get())}"  else:  deadline = "0-0-0"  add\_task(  self.title\_entry.get(),  self.description\_scrolledtext.get("1.0", "end-1c"),  "backlog",  deadline,  0,  self.team\_optionmenu\_value.get().lower(),  )  self.reload\_task\_menu()  else:  messagebox.showerror(  "Error",  "The details you've entered do not meet the criteria for validity",  )  def reload\_task\_menu(self):  self.destroy()  self.master.task\_menu()  class ProgressTaskFromBacklog(tk.Frame):  def \_\_init\_\_(self, master, task\_id, sort=False):  super().\_\_init\_\_(master, height=17 \* ROW\_HEIGHT, width=10000)  self.task = get\_task(task\_id)  self.sort = sort  self.running = True  self.fill\_frame()  self.grid\_propagate(False)  self.grid(row=0, column=0, padx=20, pady=20)  def fill\_frame(self):  sort\_shift = 6 if self.sort else 0  ## create widgets  self.progress\_task\_label = tk.Label(  self, text='Progress to "To Do"', font=FONT\_6  )  if self.sort:  self.title = tk.Label(self, text=self.task[1], font=FONT\_5)  self.description\_label = tk.Label(  self, text="Description:", font=FONT\_4  )  self.description\_scrolledtext = scrolledtext.ScrolledText(  self, font=FONT\_3, height=2  )  self.description\_scrolledtext.insert("end", self.task[2])  self.description\_scrolledtext.configure(state="disabled")  ## create importance widgets  self.importance\_label = tk.Label(self, text="Importance:", font=FONT\_4)  self.importance\_optionmenu\_value = tk.StringVar(  self, value="Select an importance..."  )  self.importance\_optionmenu = tk.OptionMenu(  self, self.importance\_optionmenu\_value, "Low", "Medium", "High"  )  self.importance\_optionmenu.configure(font=FONT\_3)  self.importance\_optionmenu["menu"].configure(font=FONT\_2)  ## create button frame  self.button\_frame = tk.Frame(self)  self.progress\_task\_button = tk.Button(  self.button\_frame,  text="Progress Task",  font=FONT\_4,  command=self.progress\_task,  )  if self.sort:  self.send\_to\_back\_button = tk.Button(  self.button\_frame,  text="Send to Back",  font=FONT\_4,  command=lambda: self.exit\_interface(action="send to back"),  )  self.skip\_button = tk.Button(  self.button\_frame,  text="Skip",  font=FONT\_4,  command=self.exit\_interface,  )  self.cancel\_button = tk.Button(  self.button\_frame,  text="Cancel",  font=FONT\_4,  command=lambda: self.exit\_interface(action="cancel"),  )  # this padding arrangement ensures the progress button is as far left as  # possible  self.progress\_task\_button.pack(side="left")  if self.sort:  self.send\_to\_back\_button.pack(side="left", padx=8)  self.skip\_button.pack(side="left")  self.cancel\_button.pack(side="left", padx=8)  ## create grid  self.grid\_rowconfigure(  tuple([n for n in range(17)]), weight=1, uniform="a"  )  ## grid widgets  self.progress\_task\_label.grid(row=0, column=0, rowspan=2, sticky="w")  if self.sort:  self.title.grid(row=3, column=0, sticky="w")  self.description\_label.grid(row=5, column=0, sticky="w")  self.description\_scrolledtext.grid(  row=6, column=0, rowspan=2, sticky="w"  )  self.importance\_label.grid(row=3 + sort\_shift, column=0, sticky="w")  self.importance\_optionmenu.grid(  row=4 + sort\_shift, column=0, sticky="w"  )  # only adds deadline widgets if the task has no deadline yet  if not deadline\_present(self.task[0]):  ## create deadline widgets  self.deadline\_label = tk.Label(self, text="Deadline:", font=FONT\_4)  self.deadline\_entry\_frame = tk.Frame(self)  self.day\_label = tk.Label(  self.deadline\_entry\_frame, text="Day:", font=FONT\_4  )  self.day\_entry = tk.Entry(self.deadline\_entry\_frame, font=FONT\_3)  self.month\_label = tk.Label(  self.deadline\_entry\_frame, text="Month:", font=FONT\_4  )  self.month\_entry = tk.Entry(self.deadline\_entry\_frame, font=FONT\_3)  self.year\_label = tk.Label(  self.deadline\_entry\_frame, text="Year:", font=FONT\_4  )  self.year\_entry = tk.Entry(self.deadline\_entry\_frame, font=FONT\_3)  self.day\_label.pack(side="left")  self.day\_entry.pack(side="left")  self.month\_label.pack(side="left")  self.month\_entry.pack(side="left")  self.year\_label.pack(side="left")  self.year\_entry.pack(side="left")  ## grid remaining widgets  self.deadline\_label.grid(row=6 + sort\_shift, column=0, sticky="w")  self.deadline\_entry\_frame.grid(  row=7 + sort\_shift, column=0, sticky="w"  )  self.button\_frame.grid(  row=9 + sort\_shift, column=0, rowspan=2, sticky="w"  )  else:  self.button\_frame.grid(  row=6 + sort\_shift, column=0, rowspan=2, sticky="w"  )  def details\_valid(self):  if self.importance\_optionmenu\_value.get() == "Select an importance...":  return False  if not deadline\_present(self.task[0]):  if not date\_valid(  self.day\_entry.get(),  self.month\_entry.get(),  self.year\_entry.get(),  ):  return False  return True  def progress\_task(self):  if self.details\_valid():  if not deadline\_present(self.task[0]):  update\_deadline(  self.task[0],  f"{int(self.year\_entry.get())}-{int(self.month\_entry.get())}-{int(self.day\_entry.get())}",  )  update\_importance(  self.task[0],  IMPORTANCES.index(self.importance\_optionmenu\_value.get()) + 1,  )  update\_state(self.task[0], "to do")  self.exit\_interface()  else:  messagebox.showerror(  "Error",  "The details you've entered do not meet the criteria for validity",  )  def get\_running(self):  return self.running  def exit\_interface(self, action=""):  self.destroy()  if not self.sort:  self.master.task\_menu()  elif action:  self.master.set\_sort\_through\_backlog\_action(action)  self.running = False  class ProgressTaskFromToDo(tk.Frame):  def \_\_init\_\_(self, master, task\_id):  super().\_\_init\_\_(master, height=10 \* ROW\_HEIGHT, width=10000)  self.task = get\_task(task\_id)  self.assignee\_count = 0  self.fill\_frame()  self.grid\_propagate(False)  self.grid(row=0, column=0, padx=20, pady=20)  def fill\_frame(self):  ## create widgets  self.progress\_task\_label = tk.Label(  self, text='Progress to "Doing"', font=FONT\_6  )  ## create assignee widgets  self.assignee\_label = tk.Label(self, text="Assignee(s):", font=FONT\_4)  self.assignee\_optionmenu\_value\_list = [  tk.StringVar(self, value="Select an assignee...") for i in range(9)  ]  self.assignee\_optionmenu\_list = [  tk.OptionMenu(  self,  self.assignee\_optionmenu\_value\_list[i],  \*get\_usernames\_from\_team(self.task[6]),  )  for i in range(9)  ]  for optionmenu in self.assignee\_optionmenu\_list:  optionmenu.configure(font=FONT\_3)  optionmenu["menu"].configure(font=FONT\_2)  self.new\_assignee\_button = tk.Button(  self, text="+", font=FONT\_4, command=self.add\_new\_assignee  )  ## create button frame  self.button\_frame = tk.Frame(self)  self.progress\_task\_button = tk.Button(  self.button\_frame,  text="Progress Task",  font=FONT\_4,  command=self.progress\_task,  )  self.cancel\_button = tk.Button(  self.button\_frame,  text="Cancel",  font=FONT\_4,  command=self.reload\_task\_menu,  )  self.progress\_task\_button.pack(side="left")  self.cancel\_button.pack(side="left", padx=8)  ## create grid  self.grid\_rowconfigure(  tuple([n for n in range(10)]), weight=1, uniform="a"  )  self.grid\_columnconfigure(tuple([n for n in range(3)]), minsize=300)  ## grid widgets  self.progress\_task\_label.grid(  row=0, column=0, rowspan=2, columnspan=3, sticky="w"  )  self.assignee\_label.grid(row=3, column=0, columnspan=3, sticky="w")  self.add\_new\_assignee()  self.button\_frame.grid(  row=6, column=0, rowspan=2, columnspan=3, sticky="w"  )  # columnspan=3 ensures the assignee widgets grid themselves with minimum  # column width  def add\_new\_assignee(self):  self.assignee\_count += 1  self.assignee\_optionmenu\_list[self.assignee\_count - 1].grid(  row=4 + (self.assignee\_count - 1) // 3,  column=(self.assignee\_count - 1) % 3,  sticky="w",  )  if self.assignee\_count == 9:  self.new\_assignee\_button.grid\_forget()  else:  self.new\_assignee\_button.grid(  row=4 + self.assignee\_count // 3,  column=self.assignee\_count % 3,  sticky="w",  )  if self.assignee\_count == 3 or self.assignee\_count == 6:  self.button\_frame.grid(  row=6 + self.assignee\_count // 3,  column=0,  rowspan=2,  columnspan=3,  sticky="w",  )  def get\_assignees(self):  assignees = []  for i in range(self.assignee\_count):  if (  not self.assignee\_optionmenu\_value\_list[i].get()  == "Select an assignee..."  ):  if self.assignee\_optionmenu\_value\_list[i].get() in assignees:  return []  else:  assignees.append(  self.assignee\_optionmenu\_value\_list[i].get()  )  return assignees  def progress\_task(self):  if self.get\_assignees():  for assignee in self.get\_assignees():  add\_assignment(assignee, self.task[0])  update\_state(self.task[0], "doing")  self.reload\_task\_menu()  else:  messagebox.showerror(  "Error",  "The details you've entered do not meet the criteria for validity",  )  def reload\_task\_menu(self):  self.destroy()  self.master.task\_menu()  class EditTask(tk.Frame):  def \_\_init\_\_(self, master, task\_id):  super().\_\_init\_\_(master, height=18 \* ROW\_HEIGHT, width=10000)  self.task = get\_task(task\_id)  self.fill\_frame()  self.grid\_propagate(False)  self.grid(row=0, column=0, padx=20, pady=20)  def fill\_frame(self):  ## create widgets  self.edit\_task\_label = tk.Label(self, text="Edit Task", font=FONT\_6)  self.description\_label = tk.Label(  self, text="Description:", font=FONT\_4  )  self.description\_scrolledtext = scrolledtext.ScrolledText(  self, font=FONT\_3, height=2  )  self.description\_scrolledtext.insert("end", self.task[2])  ## create team widgets  self.team\_label = tk.Label(self, text="Team:", font=FONT\_4)  self.team\_optionmenu\_value = tk.StringVar(  self, value=self.task[6][0].upper() + self.task[6][1:]  )  self.team\_optionmenu = tk.OptionMenu(  self,  self.team\_optionmenu\_value,  "Operational",  "Development",  "Both",  )  self.team\_optionmenu.configure(font=FONT\_3)  self.team\_optionmenu["menu"].configure(font=FONT\_2)  ## create deadline widgets  self.deadline\_label = tk.Label(self, text="Deadline:", font=FONT\_4)  self.deadline\_entry\_frame = tk.Frame(self)  self.day\_label = tk.Label(  self.deadline\_entry\_frame, text="Day:", font=FONT\_4  )  self.day\_entry = tk.Entry(self.deadline\_entry\_frame, font=FONT\_3)  self.month\_label = tk.Label(  self.deadline\_entry\_frame, text="Month:", font=FONT\_4  )  self.month\_entry = tk.Entry(self.deadline\_entry\_frame, font=FONT\_3)  self.year\_label = tk.Label(  self.deadline\_entry\_frame, text="Year:", font=FONT\_4  )  self.year\_entry = tk.Entry(self.deadline\_entry\_frame, font=FONT\_3)  self.day\_entry.insert("end", self.task[4].split("-")[2])  self.month\_entry.insert("end", self.task[4].split("-")[1])  self.year\_entry.insert("end", self.task[4].split("-")[0])  self.day\_label.pack(side="left")  self.day\_entry.pack(side="left")  self.month\_label.pack(side="left")  self.month\_entry.pack(side="left")  self.year\_label.pack(side="left")  self.year\_entry.pack(side="left")  ## create importance widgets  shift = 0  if self.task[3] != "backlog":  self.importance\_label = tk.Label(  self, text="Importance:", font=FONT\_4  )  self.importance\_optionmenu\_value = tk.StringVar(  self, value=IMPORTANCES[int(self.task[5]) - 1]  )  self.importance\_optionmenu = tk.OptionMenu(  self, self.importance\_optionmenu\_value, "Low", "Medium", "High"  )  self.importance\_optionmenu.configure(font=FONT\_3)  self.importance\_optionmenu["menu"].configure(font=FONT\_2)  self.importance\_label.grid(row=13, column=0, sticky="w")  self.importance\_optionmenu.grid(row=14, column=0, sticky="w")  shift = 3  ## create button frame  self.button\_frame = tk.Frame(self)  self.save\_edits\_button = tk.Button(  self.button\_frame,  text="Save Edits",  font=FONT\_4,  command=self.save\_edits,  )  self.cancel\_button = tk.Button(  self.button\_frame,  text="Cancel",  font=FONT\_4,  command=self.reload\_task\_menu,  )  self.save\_edits\_button.pack(side="left")  self.cancel\_button.pack(side="left", padx=8)  ## create grid  self.grid\_rowconfigure(  tuple([n for n in range(18)]), weight=1, uniform="a"  )  ## grid widgets  self.edit\_task\_label.grid(row=0, column=0, rowspan=2, sticky="w")  self.team\_label.grid(row=3, column=0, sticky="w")  self.team\_optionmenu.grid(row=4, column=0, sticky="w")  self.description\_label.grid(row=6, column=0, sticky="w")  self.description\_scrolledtext.grid(  row=7, column=0, rowspan=2, sticky="w"  )  self.deadline\_label.grid(row=10, column=0, sticky="w")  self.deadline\_entry\_frame.grid(row=11, column=0, sticky="w")  self.button\_frame.grid(row=13 + shift, column=0, rowspan=2, sticky="w")  def details\_valid(self):  try:  day = self.day\_entry.get()  month = self.month\_entry.get()  year = self.year\_entry.get()  # check for the special backlog case (0-0-0)  is\_backlog\_zero\_date = (  int(year) == 0  and int(month) == 0  and int(day) == 0  and self.task[3] == "backlog"  )  return date\_valid(day, month, year) or is\_backlog\_zero\_date  except ValueError:  return False  def save\_edits(self):  if self.details\_valid():  if self.task[3] == "backlog":  importance = 0  else:  importance = (  IMPORTANCES.index(self.importance\_optionmenu\_value.get())  + 1  )  edit\_task(  self.task[0],  self.team\_optionmenu\_value.get().lower(),  self.description\_scrolledtext.get("1.0", "end-1c"),  f"{int(self.year\_entry.get())}-{int(self.month\_entry.get())}-{int(self.day\_entry.get())}",  importance,  )  self.reload\_task\_menu()  else:  messagebox.showerror(  "Error",  "The details you've entered do not meet the criteria for validity",  )  def reload\_task\_menu(self):  self.destroy()  self.master.task\_menu()  ### event interface  class EventInterface(tk.Frame):  def \_\_init\_\_(self, master, username):  super().\_\_init\_\_(master)  self.username = username  self.calendar()  def calendar(self):  Calendar(self, self.username)  def new\_event(self):  NewEvent(self)  def new\_team\_event(self):  NewTeamEvent(self)  def remove\_event(self):  RemoveEvent(self)  class Calendar(tk.Frame):  def \_\_init\_\_(self, master, username):  super().\_\_init\_\_(master)  self.username = username  self.cells = []  self.fill\_frame()  self.pack(fill="both", expand=True)  def fill\_frame(self):  ## toolbar  self.toolbar = tk.Frame(self, relief="ridge", bd=4)  self.toolbar.pack(fill="x")  self.add\_event\_button = tk.Button(  self.toolbar, text="Add Event", font=FONT\_2, command=self.new\_event  )  self.add\_event\_button.pack(side="left", padx=4, pady=4)  self.add\_team\_event\_button = tk.Button(  self.toolbar,  text="Add Team Event",  font=FONT\_2,  command=self.new\_team\_event,  )  self.add\_team\_event\_button.pack(side="left", padx=4, pady=4)  self.delete\_event\_button = tk.Button(  self.toolbar,  text="Delete Event",  font=FONT\_2,  command=self.remove\_event,  )  self.delete\_event\_button.pack(side="left", padx=4, pady=4)  self.clear\_past\_events\_button = tk.Button(  self.toolbar,  text="Clear Past Events",  font=FONT\_2,  command=self.clear\_past\_events,  )  self.clear\_past\_events\_button.pack(side="left", padx=4, pady=4)  self.stats\_button = tk.Button(  self.toolbar,  text="Stats",  font=FONT\_2,  command=lambda: messagebox.showinfo(  "Stats", stats\_string(self.username)  ),  )  self.stats\_button.pack(side="right", padx=4, pady=4)  self.account\_info\_label = tk.Label(  self.toolbar, text=account\_info\_string(self.username), font=FONT\_2  )  self.account\_info\_label.pack(side="right")  ## calendar  self.calendar\_frame = tk.Frame(self)  self.calendar\_frame.pack(fill="both", expand=True)  self.calendar\_frame.grid\_rowconfigure(  tuple([n for n in range(4)]), weight=1, uniform="a"  )  self.calendar\_frame.grid\_columnconfigure(  tuple([n for n in range(7)]), weight=1, uniform="a"  )  for i in range(28):  cell = tk.Frame(  self.calendar\_frame, relief="raised", bd=4, bg=get\_cell\_bg(i)  )  cell.grid(row=i // 7, column=i % 7, sticky="nsew")  cell\_day\_label = tk.Label(  cell, text=get\_cell\_day(i), font=FONT\_1, bg=get\_cell\_bg(i)  )  cell\_day\_label.pack()  cell\_list = [cell, cell\_day\_label, []]  for event in get\_events(get\_cell\_date\_string(i), self.username):  cell\_list[2].append(  tk.Label(cell, text=event, font=FONT\_2, bg=get\_cell\_bg(i))  )  cell\_list[2][-1].pack()  for deadline in get\_deadlines(  get\_cell\_date\_string(i), self.username  ):  cell\_list[2].append(  tk.Label(  cell,  text=deadline[0],  font=FONT\_2,  bg=get\_cell\_bg(i),  fg=IMPORTANCE\_COLOURS[  importance(get\_cell\_date\_string(i), deadline[1]) - 1  ],  )  )  cell\_list[2][-1].pack()  self.cells.append(cell\_list)  ## events  cell.bind(  "<Configure>",  lambda event: self.update\_wraplengths(event),  )  # having this out of the for loop ensures the binding is attached to the  # last cell only, so that wraplengths are only updated once per resize  # and they are updated all at once  def update\_wraplengths(self, event):  for cell in self.cells:  for label in cell[2]:  label.configure(wraplength=event.width - 10)  def new\_event(self):  self.destroy()  self.master.new\_event()  def new\_team\_event(self):  self.destroy()  self.master.new\_team\_event()  def remove\_event(self):  self.destroy()  self.master.remove\_event()  def clear\_past\_events(self):  clear\_past\_events()  self.destroy()  self.master.calendar()  class NewEvent(tk.Frame):  def \_\_init\_\_(self, master):  super().\_\_init\_\_(master, height=18 \* ROW\_HEIGHT, width=10000)  self.attendee\_count = 0  self.fill\_frame()  self.grid\_propagate(False)  self.grid(row=0, column=0, padx=20, pady=20)  def fill\_frame(self):  ## create widgets  self.new\_event\_label = tk.Label(self, text="New Event", font=FONT\_6)  self.title\_label = tk.Label(self, text="Title:", font=FONT\_4)  self.title\_entry = tk.Entry(self, font=FONT\_3)  self.date\_label = tk.Label(self, text="Date:", font=FONT\_4)  self.date\_entry\_frame = tk.Frame(self)  ## create attendee widgets  self.attendee\_label = tk.Label(self, text="Attendee(s):", font=FONT\_4)  self.attendee\_optionmenu\_value\_list = [  tk.StringVar(self, value="Select an attendee...") for i in range(15)  ]  self.attendee\_optionmenu\_list = [  tk.OptionMenu(  self,  self.attendee\_optionmenu\_value\_list[i],  \*get\_all\_usernames(),  )  for i in range(15)  ]  for optionmenu in self.attendee\_optionmenu\_list:  optionmenu.configure(font=FONT\_3)  optionmenu["menu"].configure(font=FONT\_2)  self.new\_attendee\_button = tk.Button(  self, text="+", font=FONT\_4, command=self.add\_new\_attendee  )  ## create button frame  self.button\_frame = tk.Frame(self)  self.add\_event\_button = tk.Button(  self.button\_frame,  text="Add Event",  font=FONT\_4,  command=self.add\_event,  )  self.cancel\_button = tk.Button(  self.button\_frame,  text="Cancel",  font=FONT\_4,  command=self.reload\_calendar,  )  self.add\_event\_button.pack(side="left")  self.cancel\_button.pack(side="left", padx=8)  ## fill date entry frame  self.day\_label = tk.Label(  self.date\_entry\_frame, text="Day:", font=FONT\_4  )  self.day\_entry = tk.Entry(self.date\_entry\_frame, font=FONT\_3)  self.month\_label = tk.Label(  self.date\_entry\_frame, text="Month:", font=FONT\_4  )  self.month\_entry = tk.Entry(self.date\_entry\_frame, font=FONT\_3)  self.year\_label = tk.Label(  self.date\_entry\_frame, text="Year:", font=FONT\_4  )  self.year\_entry = tk.Entry(self.date\_entry\_frame, font=FONT\_3)  self.day\_label.pack(side="left")  self.day\_entry.pack(side="left")  self.month\_label.pack(side="left")  self.month\_entry.pack(side="left")  self.year\_label.pack(side="left")  self.year\_entry.pack(side="left")  ## create grid  self.grid\_rowconfigure(  tuple([n for n in range(18)]), weight=1, uniform="a"  )  self.grid\_columnconfigure(tuple([n for n in range(3)]), minsize=300)  ## grid widgets  self.new\_event\_label.grid(  row=0, column=0, rowspan=2, columnspan=3, sticky="w"  )  self.title\_label.grid(row=3, column=0, columnspan=3, sticky="w")  self.title\_entry.grid(row=4, column=0, columnspan=3, sticky="w")  self.date\_label.grid(row=6, column=0, columnspan=3, sticky="w")  self.date\_entry\_frame.grid(row=7, column=0, columnspan=3, sticky="w")  self.attendee\_label.grid(row=9, column=0, columnspan=3, sticky="w")  self.add\_new\_attendee()  self.button\_frame.grid(  row=12, column=0, rowspan=2, columnspan=3, sticky="w"  )  def add\_new\_attendee(self):  self.attendee\_count += 1  self.attendee\_optionmenu\_list[self.attendee\_count - 1].grid(  row=10 + (self.attendee\_count - 1) // 3,  column=(self.attendee\_count - 1) % 3,  sticky="w",  )  if self.attendee\_count == 15:  self.new\_attendee\_button.grid\_forget()  else:  self.new\_attendee\_button.grid(  row=10 + self.attendee\_count // 3,  column=self.attendee\_count % 3,  sticky="w",  )  if self.attendee\_count in [3, 6, 9, 12]:  self.button\_frame.grid(  row=12 + self.attendee\_count // 3,  column=0,  rowspan=2,  columnspan=3,  sticky="w",  )  def get\_attendees(self):  attendees = []  for i in range(self.attendee\_count):  if (  not self.attendee\_optionmenu\_value\_list[i].get()  == "Select an attendee..."  ):  if self.attendee\_optionmenu\_value\_list[i].get() in attendees:  return []  else:  attendees.append(  self.attendee\_optionmenu\_value\_list[i].get()  )  return attendees  def details\_valid(self):  if not self.title\_entry.get():  return False  if not date\_valid(  self.day\_entry.get(), self.month\_entry.get(), self.year\_entry.get()  ):  return False  if not self.get\_attendees():  return False  return True  def add\_event(self):  if self.details\_valid():  date = f"{int(self.year\_entry.get())}-{int(self.month\_entry.get())}-{int(self.day\_entry.get())}"  add\_event(  self.title\_entry.get(),  date,  self.get\_attendees(),  )  self.reload\_calendar()  else:  messagebox.showerror(  "Error",  "The details you've entered do not meet the criteria for validity",  )  def reload\_calendar(self):  self.destroy()  self.master.calendar()  class NewTeamEvent(tk.Frame):  def \_\_init\_\_(self, master):  super().\_\_init\_\_(master, height=14 \* ROW\_HEIGHT, width=10000)  self.attendee\_count = 0  self.fill\_frame()  self.grid\_propagate(False)  self.grid(row=0, column=0, padx=20, pady=20)  def fill\_frame(self):  ## create widgets  self.new\_team\_event\_label = tk.Label(  self, text="New Team Event", font=FONT\_6  )  self.title\_label = tk.Label(self, text="Title:", font=FONT\_4)  self.title\_entry = tk.Entry(self, font=FONT\_3)  self.date\_label = tk.Label(self, text="Date:", font=FONT\_4)  self.date\_entry\_frame = tk.Frame(self)  self.team\_label = tk.Label(self, text="Team:", font=FONT\_4)  ## create team optionmenu  self.team\_optionmenu\_value = tk.StringVar(  self, value="Select a team..."  )  self.team\_optionmenu = tk.OptionMenu(  self,  self.team\_optionmenu\_value,  "Operational",  "Development",  "Both",  )  self.team\_optionmenu.configure(font=FONT\_3)  self.team\_optionmenu["menu"].configure(font=FONT\_2)  ## create button frame  self.button\_frame = tk.Frame(self)  self.add\_event\_button = tk.Button(  self.button\_frame,  text="Add Event",  font=FONT\_4,  command=self.add\_event,  )  self.cancel\_button = tk.Button(  self.button\_frame,  text="Cancel",  font=FONT\_4,  command=self.reload\_calendar,  )  self.add\_event\_button.pack(side="left")  self.cancel\_button.pack(side="left", padx=8)  ## fill date entry frame  self.day\_label = tk.Label(  self.date\_entry\_frame, text="Day:", font=FONT\_4  )  self.day\_entry = tk.Entry(self.date\_entry\_frame, font=FONT\_3)  self.month\_label = tk.Label(  self.date\_entry\_frame, text="Month:", font=FONT\_4  )  self.month\_entry = tk.Entry(self.date\_entry\_frame, font=FONT\_3)  self.year\_label = tk.Label(  self.date\_entry\_frame, text="Year:", font=FONT\_4  )  self.year\_entry = tk.Entry(self.date\_entry\_frame, font=FONT\_3)  self.day\_label.pack(side="left")  self.day\_entry.pack(side="left")  self.month\_label.pack(side="left")  self.month\_entry.pack(side="left")  self.year\_label.pack(side="left")  self.year\_entry.pack(side="left")  ## create grid  self.grid\_rowconfigure(  tuple([n for n in range(14)]), weight=1, uniform="a"  )  self.grid\_columnconfigure(tuple([n for n in range(3)]), minsize=300)  ## grid widgets  self.new\_team\_event\_label.grid(  row=0, column=0, rowspan=2, columnspan=3, sticky="w"  )  self.title\_label.grid(row=3, column=0, columnspan=3, sticky="w")  self.title\_entry.grid(row=4, column=0, columnspan=3, sticky="w")  self.date\_label.grid(row=6, column=0, columnspan=3, sticky="w")  self.date\_entry\_frame.grid(row=7, column=0, columnspan=3, sticky="w")  self.team\_label.grid(row=9, column=0, sticky="w")  self.team\_optionmenu.grid(row=10, column=0, sticky="w")  self.button\_frame.grid(  row=12, column=0, rowspan=2, columnspan=3, sticky="w"  )  def details\_valid(self):  if not self.title\_entry.get():  return False  if not date\_valid(  self.day\_entry.get(), self.month\_entry.get(), self.year\_entry.get()  ):  return False  if self.team\_optionmenu\_value.get() == "Select a team...":  return False  return True  def add\_event(self):  if self.details\_valid():  date = f"{int(self.year\_entry.get())}-{int(self.month\_entry.get())}-{int(self.day\_entry.get())}"  add\_event(  self.title\_entry.get(),  date,  get\_usernames\_from\_team(  self.team\_optionmenu\_value.get().lower()  ),  )  self.reload\_calendar()  else:  messagebox.showerror(  "Error",  "The details you've entered do not meet the criteria for validity",  )  def reload\_calendar(self):  self.destroy()  self.master.calendar()  class RemoveEvent(tk.Frame):  def \_\_init\_\_(self, master):  super().\_\_init\_\_(master, height=8 \* ROW\_HEIGHT, width=10000)  self.fill\_frame()  self.grid\_propagate(False)  self.grid(row=0, column=0, padx=20, pady=20)  def fill\_frame(self):  ## create widgets  self.remove\_event\_label = tk.Label(  self, text="Remove Event", font=FONT\_6  )  self.event\_title\_label = tk.Label(self, text="Event:", font=FONT\_4)  self.event\_optionmenu\_value = tk.StringVar(  self, value="Select an event to delete..."  )  self.event\_optionmenu = tk.OptionMenu(  self, self.event\_optionmenu\_value, \*get\_all\_events()  )  self.event\_optionmenu.configure(font=FONT\_3)  self.event\_optionmenu["menu"].configure(font=FONT\_2)  ## create button frame  self.button\_frame = tk.Frame(self)  self.delete\_event\_button = tk.Button(  self.button\_frame,  text="Delete Event",  font=FONT\_4,  command=self.delete,  )  self.cancel\_button = tk.Button(  self.button\_frame,  text="Cancel",  font=FONT\_4,  command=self.reload\_calendar,  )  self.delete\_event\_button.pack(side="left")  self.cancel\_button.pack(side="left", padx=8)  ## create grid  self.grid\_rowconfigure(  tuple([n for n in range(8)]), weight=1, uniform="a"  )  ## grid widgets  self.remove\_event\_label.grid(row=0, column=0, rowspan=2, sticky="w")  self.event\_title\_label.grid(row=3, column=0, sticky="w")  self.event\_optionmenu.grid(row=4, column=0, sticky="w")  self.button\_frame.grid(row=6, column=0, rowspan=2, sticky="w")  def delete(self):  if self.event\_optionmenu\_value.get() != "Select an event to delete...":  delete\_event(self.event\_optionmenu\_value.get())  self.reload\_calendar()  else:  messagebox.showerror(  "Error",  "The details you've entered do not meet the criteria for validity",  )  def reload\_calendar(self):  self.destroy()  self.master.calendar()  ### accounts interface  class AccountInterface(tk.Frame):  def \_\_init\_\_(self, master, username):  super().\_\_init\_\_(master)  self.username = username  self.account\_menu()  def account\_menu(self):  AccountMenu(self, self.username)  def new\_account(self):  NewAccount(self)  class AccountMenu(tk.Frame):  def \_\_init\_\_(self, master, username):  super().\_\_init\_\_(master)  self.username = username  self.scroll\_toggle = True  self.fill\_frame()  self.pack(fill="both", expand=True)  def fill\_frame(self):  ## toolbar  self.toolbar = tk.Frame(self, relief="ridge", bd=4)  self.toolbar.pack(fill="x")  self.add\_account\_button = tk.Button(  self.toolbar,  text="Add Account",  font=FONT\_2,  command=self.new\_account,  )  self.add\_account\_button.pack(side="left", padx=4, pady=4)  self.stats\_button = tk.Button(  self.toolbar,  text="Stats",  font=FONT\_2,  command=lambda: messagebox.showinfo(  "Stats", stats\_string(self.username)  ),  )  self.stats\_button.pack(side="right", padx=4, pady=4)  self.account\_info\_label = tk.Label(  self.toolbar, text=account\_info\_string(self.username), font=FONT\_2  )  self.account\_info\_label.pack(side="right")  ## canvas  self.canvas = tk.Canvas(self)  self.canvas.pack(fill="both", expand=True)  ## canvas frame  self.canvas\_frame = tk.Frame(self.canvas)  for username in merge\_sort(  get\_all\_usernames(), key=lambda username: get\_name(username)  ):  # sorts by name  Account(self.canvas\_frame, username)  ## events  self.bind\_all("<MouseWheel>", self.scroll)  self.canvas.bind(  "<Destroy>", lambda event: self.unbind\_all("<MouseWheel>")  )  self.canvas.bind(  "<Configure>",  lambda event: self.update\_size(event.width, event.height),  )  def update\_size(self, canvas\_width, canvas\_height):  if canvas\_height < self.canvas\_frame.winfo\_reqheight():  height = self.canvas\_frame.winfo\_reqheight()  self.scroll\_toggle = True  self.canvas.configure(  scrollregion=(  0,  0,  canvas\_width,  self.canvas\_frame.winfo\_reqheight(),  )  )  else:  height = canvas\_height  self.scroll\_toggle = False  self.canvas.create\_window(  (0, 0),  window=self.canvas\_frame,  anchor="nw",  width=canvas\_width,  height=height,  )  def scroll(self, event):  self.update\_size(self.canvas.winfo\_width(), self.canvas.winfo\_height())  if self.scroll\_toggle:  self.canvas.yview\_scroll(int(-event.delta / 120), "units")  def new\_account(self):  self.destroy()  self.master.new\_account()  class Account(tk.Frame):  def \_\_init\_\_(self, master, username):  super().\_\_init\_\_(master, relief="raised", bd=4)  self.username = username  self.fill\_frame()  self.pack(fill="x", padx=8, pady=4)  def fill\_frame(self):  ## create widgets  self.name\_label = tk.Label(  self,  text=f"{get\_name(self.username)} | {self.username}",  font=FONT\_2,  )  self.delete\_button = tk.Button(  self, text="Delete", font=FONT\_2, command=self.delete  )  self.save\_changes\_button = tk.Button(  self, text="Save Changes", font=FONT\_2, command=self.save\_changes  )  self.admin\_checkbox\_value = tk.BooleanVar(  self, value=admin(self.username)  )  self.admin\_checkbox = tk.Checkbutton(  self,  text="Admin",  font=FONT\_2,  variable=self.admin\_checkbox\_value,  onvalue=True,  offvalue=False,  )  self.password\_change\_scheduled\_checkbox\_value = tk.BooleanVar(  self, value=not active(self.username)  )  self.password\_change\_scheduled\_checkbox = tk.Checkbutton(  self,  text="Password Change Scheduled",  font=FONT\_2,  variable=self.password\_change\_scheduled\_checkbox\_value,  onvalue=True,  offvalue=False,  )  ## create team optionmenu  self.team\_optionmenu\_value = tk.StringVar(  self, value=team(self.username)[0].upper() + team(self.username)[1:]  )  self.team\_optionmenu = tk.OptionMenu(  self,  self.team\_optionmenu\_value,  "Operational",  "Development",  "Both",  )  self.team\_optionmenu.configure(font=FONT\_2)  self.team\_optionmenu["menu"].configure(font=FONT\_2)  ## grid widgets  self.name\_label.pack(side="left")  self.delete\_button.pack(side="right", padx=4, pady=4)  self.save\_changes\_button.pack(side="right", padx=4, pady=4)  self.admin\_checkbox.pack(side="right", padx=4, pady=4)  self.password\_change\_scheduled\_checkbox.pack(  side="right", padx=4, pady=4  )  self.team\_optionmenu.pack(side="right", padx=4, pady=4)  def delete(self):  if messagebox.askyesno(  "Delete", "Are you sure you want to delete this account?"  ):  delete\_account(self.username)  self.destroy()  def save\_changes(self):  if messagebox.askyesno(  "Save Changes", "Are you sure you want to save these changes?"  ):  save\_changes(  self.username,  self.team\_optionmenu\_value.get().lower(),  not self.password\_change\_scheduled\_checkbox\_value.get(),  self.admin\_checkbox\_value.get(),  )  self.master.master.master.destroy()  self.master.master.master.master.account\_menu()  class NewAccount(tk.Frame):  def \_\_init\_\_(self, master):  super().\_\_init\_\_(master, height=17 \* ROW\_HEIGHT, width=10000)  self.fill\_frame()  self.grid\_propagate(False)  self.grid(row=0, column=0, padx=20, pady=20)  def fill\_frame(self):  ## create widgets  self.new\_account\_label = tk.Label(self, text="New Account", font=FONT\_6)  self.first\_name\_label = tk.Label(self, text="First Name:", font=FONT\_4)  self.first\_name\_entry = tk.Entry(self, font=FONT\_3)  self.last\_name\_label = tk.Label(self, text="Last Name:", font=FONT\_4)  self.last\_name\_entry = tk.Entry(self, font=FONT\_3)  self.temporary\_password\_label = tk.Label(  self, text="Temporary Password:", font=FONT\_4  )  self.temporary\_password\_entry = tk.Entry(self, font=FONT\_3)  ## create account type widgets  self.account\_type\_label = tk.Label(self, text="Team:", font=FONT\_4)  self.account\_type\_frame = tk.Frame(self)  self.team\_optionmenu\_value = tk.StringVar(  self.account\_type\_frame, value="Select a team..."  )  self.team\_optionmenu = tk.OptionMenu(  self.account\_type\_frame,  self.team\_optionmenu\_value,  "Operational",  "Development",  "Both",  )  self.team\_optionmenu.configure(font=FONT\_3)  self.team\_optionmenu["menu"].configure(font=FONT\_2)  self.admin\_checkbox\_value = tk.BooleanVar(  self.account\_type\_frame, value=False  )  self.admin\_checkbox = tk.Checkbutton(  self.account\_type\_frame,  text="Admin",  font=FONT\_3,  variable=self.admin\_checkbox\_value,  onvalue=True,  offvalue=False,  )  self.team\_optionmenu.pack(side="left")  self.admin\_checkbox.pack(side="left", padx=8)  ## create button frame  self.button\_frame = tk.Frame(self)  self.add\_account\_button = tk.Button(  self.button\_frame,  text="Add Account",  font=FONT\_4,  command=self.add\_account,  )  self.cancel\_button = tk.Button(  self.button\_frame,  text="Cancel",  font=FONT\_4,  command=self.reload\_account\_menu,  )  self.add\_account\_button.pack(side="left")  self.cancel\_button.pack(side="left", padx=8)  ## create grid  self.grid\_rowconfigure(  tuple([n for n in range(17)]), weight=1, uniform="a"  )  ## grid widgets  self.new\_account\_label.grid(row=0, column=0, rowspan=2, sticky="w")  self.first\_name\_label.grid(row=3, column=0, sticky="w")  self.first\_name\_entry.grid(row=4, column=0, sticky="w")  self.last\_name\_label.grid(row=6, column=0, sticky="w")  self.last\_name\_entry.grid(row=7, column=0, sticky="w")  self.account\_type\_label.grid(row=9, column=0, sticky="w")  self.account\_type\_frame.grid(row=10, column=0, sticky="w")  self.temporary\_password\_label.grid(row=12, column=0, sticky="w")  self.temporary\_password\_entry.grid(row=13, column=0, sticky="w")  self.button\_frame.grid(row=15, column=0, rowspan=2, sticky="w")  def details\_valid(self):  if not self.first\_name\_entry.get():  return False  if not self.last\_name\_entry.get():  return False  if self.team\_optionmenu\_value.get() == "Select a team...":  return False  return True  def add\_account(self):  if self.details\_valid():  firstname = self.first\_name\_entry.get()  lastname = self.last\_name\_entry.get()  password = self.temporary\_password\_entry.get()  if strong(password):  add\_account(  create\_username(firstname, lastname),  firstname,  lastname,  password,  self.team\_optionmenu\_value.get().lower(),  self.admin\_checkbox\_value.get(),  )  self.reload\_account\_menu()  else:  messagebox.showerror(  "Error",  "Password not strong enough, try using a longer password or a larger variety of character types",  )  else:  messagebox.showerror(  "Error",  "The details you've entered do not meet the criteria for validity",  )  def reload\_account\_menu(self):  self.destroy()  self.master.account\_menu()  if \_\_name\_\_ == "\_\_main\_\_":  Window(1100, 670).mainloop() |

| **sql\_functions.py** |
| --- |
| import sqlite3 as sq  import hash\_table\_functions as ht  from cryptography.fernet import Fernet  from misc\_functions import \*  def \_get\_fernet\_object():  with open("key.key", "rb") as key\_file:  fernet = Fernet(key\_file.read())  return fernet  # SELECT functions  def login\_valid(username, entered\_password):  fernet = \_get\_fernet\_object()  conn = sq.connect("team\_tracker.db")  cur = conn.cursor()  cur.execute(  """SELECT Password FROM Account WHERE Username = ?""", (username,)  )  stored\_password = cur.fetchall()  cur.close()  conn.close()  if stored\_password:  if entered\_password == fernet.decrypt(stored\_password[0][0]).decode():  return True  return False  def active(username):  conn = sq.connect("team\_tracker.db")  cur = conn.cursor()  cur.execute(  """SELECT Active FROM Account WHERE Username = ?""", (username,)  )  active = cur.fetchall()[0][0]  cur.close()  conn.close()  return bool(active)  def admin(username):  conn = sq.connect("team\_tracker.db")  cur = conn.cursor()  cur.execute("""SELECT Admin FROM Account WHERE Username = ?""", (username,))  admin = cur.fetchall()[0][0]  cur.close()  conn.close()  return bool(admin)  def deadline\_present(task\_id):  conn = sq.connect("team\_tracker.db")  cur = conn.cursor()  cur.execute("""SELECT Deadline FROM Task WHERE TaskID = ?""", (task\_id,))  deadline = cur.fetchall()[0][0]  cur.close()  conn.close()  if deadline == "0-0-0":  return False  return True  def account\_info\_string(username):  conn = sq.connect("team\_tracker.db")  cur = conn.cursor()  cur.execute("""SELECT Team FROM Account WHERE Username = ?""", (username,))  account\_info = cur.fetchall()[0][0]  cur.close()  conn.close()  if account\_info == "operational":  return f"{get\_name(username)} | Operational"  elif account\_info == "development":  return f"{get\_name(username)} | Development"  elif account\_info == "both":  return f"{get\_name(username)} | Operational & Development"  def stats\_string(username):  conn = sq.connect("team\_tracker.db")  cur = conn.cursor()  cur.execute(  """SELECT COUNT(Task.TaskID)  FROM Task, Assignment, Account  WHERE Task.TaskID = Assignment.TaskID  AND Assignment.Username = Account.Username  AND Account.Username = ?  AND Task.State = 'doing'""",  (username,),  )  doing = cur.fetchall()[0][0]  cur.execute(  """SELECT COUNT(Task.TaskID)  FROM Task, Assignment, Account  WHERE Task.TaskID = Assignment.TaskID  AND Assignment.Username = Account.Username  AND Account.Username = ?  AND Task.State = 'done'""",  (username,),  )  done = cur.fetchall()[0][0]  cur.execute(  """SELECT COUNT(Event.EventID)  FROM Event, Attendance, Account  WHERE Event.EventID = Attendance.EventID  AND Attendance.Username = Account.Username  AND Account.Username = ?""",  (username,),  )  events = cur.fetchall()[0][0]  cur.close()  conn.close()  return f"""You have {doing+done} tasks assigned to you, {done} of which are completed.\nYou have {events} events in your calendar."""  def team(username):  conn = sq.connect("team\_tracker.db")  cur = conn.cursor()  cur.execute("""SELECT Team FROM Account WHERE Username = ?""", (username,))  team = cur.fetchall()[0][0]  cur.close()  conn.close()  return team  def get\_name(username):  conn = sq.connect("team\_tracker.db")  cur = conn.cursor()  cur.execute(  """SELECT FirstName, LastName FROM Account WHERE Username = ?""",  (username,),  )  name = cur.fetchall()[0]  cur.close()  conn.close()  return f"{name[0]} {name[1]}"  def get\_tasks(team, column):  if team == "both":  team1 = "operational"  team2 = "development"  else:  team1 = team2 = team  conn = sq.connect("team\_tracker.db")  cur = conn.cursor()  if column == 0:  cur.execute(  """SELECT \* FROM Task  WHERE (Team = ? OR Team = ? OR Team = 'both')  AND State = 'backlog'""",  (team1, team2),  )  elif column == 1:  cur.execute(  """SELECT \* FROM Task  WHERE (Team = ? OR Team = ? OR Team = 'both')  AND State = 'to do'""",  (team1, team2),  )  elif column == 2:  cur.execute(  """SELECT \* FROM Task  WHERE (Team = ? OR Team = ? OR Team = 'both')  AND State = 'doing'""",  (team1, team2),  )  elif column == 3:  cur.execute(  """SELECT \* FROM Task  WHERE (Team = ? OR Team = ? OR Team = 'both')  AND State = 'done'""",  (team1, team2),  )  tasks = cur.fetchall()  cur.close()  conn.close()  return merge\_sort(tasks, key=lambda task: importance(task[4], task[5]))[  ::-1  ]  def get\_task(task\_id):  conn = sq.connect("team\_tracker.db")  cur = conn.cursor()  cur.execute("""SELECT \* FROM Task WHERE TaskID = ?""", (task\_id,))  task = cur.fetchall()[0]  cur.close()  conn.close()  return task  def get\_task\_bg(task\_id, username):  conn = sq.connect("team\_tracker.db")  cur = conn.cursor()  cur.execute(  """SELECT \* FROM Assignment  WHERE TaskID = ? AND Username = ?""",  (task\_id, username),  )  assigned = cur.fetchall()  cur.close()  conn.close()  return "lightblue" if bool(assigned) else "SystemButtonFace"  def get\_usernames\_from\_team(team):  if team == "both":  conn = sq.connect("team\_tracker.db")  cur = conn.cursor()  cur.execute("""SELECT Username FROM Account""")  usernames = cur.fetchall()  cur.close()  conn.close()  else:  conn = sq.connect("team\_tracker.db")  cur = conn.cursor()  cur.execute(  """SELECT Username FROM Account  WHERE Team = ? OR Team = 'both'  ORDER BY Username""",  (team,),  )  usernames = cur.fetchall()  cur.close()  conn.close()  return [i[0] for i in usernames]  def get\_deadlines(date, username):  conn = sq.connect("team\_tracker.db")  cur = conn.cursor()  userteam = team(username)  if userteam == "both":  cur.execute(  """SELECT Task.Title, Task.BaseImportance  FROM Task, Assignment  WHERE Task.Deadline = ?  AND ((Task.State = 'backlog' OR Task.State = 'to do')  OR (Task.TaskID = Assignment.TaskID  AND Assignment.Username = ?))  ORDER BY Task.Title""",  (date, username),  )  else:  cur.execute(  """SELECT Task.Title, Task.BaseImportance  FROM Task, Assignment  WHERE Task.Deadline = ?  AND (((Task.State = 'backlog' OR Task.State = 'to do')  AND Task.Team = ?)  OR (Task.TaskID = Assignment.TaskID  AND Assignment.Username = ?))  ORDER BY Task.Title""",  (date, userteam, username),  )  data = cur.fetchall()  cur.close()  conn.close()  deadlines = []  for i in data:  if i[0] not in [j[0] for j in deadlines]:  deadlines.append(i)  return deadlines  def get\_events(date, username):  conn = sq.connect("team\_tracker.db")  cur = conn.cursor()  cur.execute(  """SELECT Event.Title  FROM Event, Attendance  WHERE Event.EventID = Attendance.EventID  AND Event.Date = ?  AND Attendance.Username = ?  ORDER BY Event.Title""",  (date, username),  )  events = cur.fetchall()  cur.close()  conn.close()  return [i[0] for i in events]  def get\_all\_usernames():  conn = sq.connect("team\_tracker.db")  cur = conn.cursor()  cur.execute("""SELECT Username FROM Account ORDER BY Username""")  usernames = cur.fetchall()  cur.close()  conn.close()  return [i[0] for i in usernames]  def get\_all\_events():  conn = sq.connect("team\_tracker.db")  cur = conn.cursor()  cur.execute("""SELECT Title FROM Event ORDER BY Title""")  events = cur.fetchall()  cur.close()  conn.close()  return list(dict.fromkeys([i[0] for i in events]))  # UPDATE functions  def set\_new\_password(username, new\_password):  fernet = \_get\_fernet\_object()  conn = sq.connect("team\_tracker.db")  cur = conn.cursor()  cur.execute(  """UPDATE Account  SET Password = ?, Active = 1  WHERE Username = ?""",  (fernet.encrypt(new\_password.encode()), username),  )  conn.commit()  cur.close()  conn.close()  def update\_state(task\_id, new\_state):  conn = sq.connect("team\_tracker.db")  cur = conn.cursor()  cur.execute(  """UPDATE Task SET State = ? WHERE TaskID = ?""", (new\_state, task\_id)  )  conn.commit()  cur.close()  conn.close()  def update\_deadline(task\_id, new\_deadline):  conn = sq.connect("team\_tracker.db")  cur = conn.cursor()  cur.execute(  """UPDATE Task SET Deadline = ? WHERE TaskID = ?""",  (new\_deadline, task\_id),  )  conn.commit()  cur.close()  conn.close()  def update\_importance(task\_id, new\_importance):  conn = sq.connect("team\_tracker.db")  cur = conn.cursor()  cur.execute(  """UPDATE Task SET BaseImportance = ? WHERE TaskID = ?""",  (new\_importance, task\_id),  )  conn.commit()  cur.close()  conn.close()  def edit\_task(task\_id, team, description, deadline, baseimportance):  conn = sq.connect("team\_tracker.db")  cur = conn.cursor()  cur.execute(  """UPDATE Task  SET Description = ?, Deadline = ?,  BaseImportance = ?, Team = ?  WHERE TaskID = ?""",  (description, deadline, baseimportance, team, task\_id),  )  conn.commit()  cur.close()  conn.close()  def save\_changes(username, team, active, admin):  conn = sq.connect("team\_tracker.db")  cur = conn.cursor()  cur.execute(  """UPDATE Account  SET Team = ?, Active = ?, Admin = ?  WHERE Username = ?""",  (team, active, admin, username),  )  conn.commit()  cur.close()  conn.close()  # DELETE functions  def delete\_task(task\_id):  conn = sq.connect("team\_tracker.db")  cur = conn.cursor()  cur.execute("""DELETE FROM Assignment WHERE TaskID = ?""", (task\_id,))  cur.execute("""DELETE FROM Task WHERE TaskID = ?""", (task\_id,))  conn.commit()  cur.close()  conn.close()  def delete\_event(title):  conn = sq.connect("team\_tracker.db")  cur = conn.cursor()  cur.execute(  """DELETE FROM Attendance  WHERE EventID IN (SELECT EventID FROM Event WHERE Title = ?)""",  (title,),  )  cur.execute("""DELETE FROM Event WHERE Title = ?""", (title,))  conn.commit()  cur.close()  conn.close()  def delete\_account(username):  conn = sq.connect("team\_tracker.db")  cur = conn.cursor()  cur.execute("""DELETE FROM Assignment WHERE Username = ?""", (username,))  cur.execute("""DELETE FROM Attendance WHERE Username = ?""", (username,))  cur.execute("""DELETE FROM Account WHERE Username = ?""", (username,))  conn.commit()  cur.close()  conn.close()  ht.delete(username)  def clear\_done():  conn = sq.connect("team\_tracker.db")  cur = conn.cursor()  cur.execute("""SELECT TaskID FROM Task WHERE State = 'done'""")  for task\_id in cur.fetchall():  delete\_task(task\_id[0])  conn.commit()  cur.close()  conn.close()  def clear\_past\_events():  conn = sq.connect("team\_tracker.db")  cur = conn.cursor()  cur.execute("""SELECT EventID, Date FROM Event""")  for event\_id, date\_str in cur.fetchall():  event\_date = dt.datetime.strptime(date\_str, "%Y-%m-%d").date()  if event\_date < dt.datetime.now().date():  cur.execute(  """DELETE FROM Attendance WHERE EventID = ?""", (event\_id,)  )  cur.execute("""DELETE FROM Event WHERE EventID = ?""", (event\_id,))  conn.commit()  cur.close()  conn.close()  # INSERT functions  def add\_task(title, description, state, deadline, baseimportance, team):  conn = sq.connect("team\_tracker.db")  cur = conn.cursor()  cur.execute(  """INSERT INTO Task  (Title, Description, State, Deadline, BaseImportance, Team)  VALUES (?, ?, ?, ?, ?, ?)""",  (title, description, state, deadline, baseimportance, team),  )  conn.commit()  cur.close()  conn.close()  def add\_assignment(username, task\_id):  conn = sq.connect("team\_tracker.db")  cur = conn.cursor()  cur.execute(  """INSERT INTO Assignment (Username, TaskID) VALUES (?, ?)""",  (username, task\_id),  )  conn.commit()  cur.close()  conn.close()  def add\_event(title, date, attendees):  conn = sq.connect("team\_tracker.db")  cur = conn.cursor()  cur.execute(  """INSERT INTO Event (Title, Date) VALUES (?, ?)""", (title, date)  )  event\_id = cur.lastrowid  for attendee in attendees:  cur.execute(  """INSERT INTO Attendance (Username, EventID) VALUES (?, ?)""",  (attendee, event\_id),  )  conn.commit()  cur.close()  conn.close()  def add\_account(username, firstname, lastname, password, team, admin):  fernet = \_get\_fernet\_object()  conn = sq.connect("team\_tracker.db")  cur = conn.cursor()  cur.execute(  """INSERT INTO Account  (Username, FirstName, LastName, Password, Team, Admin, Active)  VALUES (?, ?, ?, ?, ?, ?, 0)""",  (  username,  firstname,  lastname,  fernet.encrypt(password.encode()),  team,  admin,  ),  )  conn.commit()  cur.close()  conn.close() |

| **hash\_table\_functions.py** |
| --- |
| FILE\_NAME = "username\_hash\_table.bin"  TABLE\_SIZE = 64  RECORD\_LENGTH = 32  EMPTY\_BYTE = b"\x00"  OCCUPIED\_FLAG = b"\x01"  TOMBSTONE\_FLAG = b"\x02"  def hash(input):  output = 0  pos\_multiplier = 31  for i, char in enumerate(list(input)):  output += ord(char) \* (pos\_multiplier\*\*i)  return output % TABLE\_SIZE  def pad(item):  return OCCUPIED\_FLAG + item.encode("utf-8").ljust(  RECORD\_LENGTH - 1, EMPTY\_BYTE  )  def unpad(raw\_bytes):  return raw\_bytes[1:].rstrip(EMPTY\_BYTE).decode("utf-8")  def insert(item):  index = hash(item)  first\_tombstone\_offset = -1  for i in range(TABLE\_SIZE):  probe\_index = (index + i) % TABLE\_SIZE  offset = probe\_index \* RECORD\_LENGTH  with open(FILE\_NAME, "r+b") as f:  f.seek(offset)  raw = f.read(RECORD\_LENGTH)  flag = raw[0:1]  data = raw[1:]  existing\_item = data.rstrip(EMPTY\_BYTE).decode("utf-8")  if flag == EMPTY\_BYTE:  f.seek(  first\_tombstone\_offset  if first\_tombstone\_offset != -1  else offset  )  f.write(pad(item))  return True  # sets first tombstone offset  elif flag == TOMBSTONE\_FLAG and first\_tombstone\_offset == -1:  first\_tombstone\_offset = offset  # checks if the item already exists in the table  elif flag == OCCUPIED\_FLAG and existing\_item == item:  return False  # no empty spaces but a tombstone is available  if first\_tombstone\_offset is not None:  with open(FILE\_NAME, "r+b") as f:  f.seek(first\_tombstone\_offset)  f.write(pad(item))  return True  return False  def search(item):  index = hash(item)  for i in range(TABLE\_SIZE):  probe\_index = (index + i) % TABLE\_SIZE  offset = probe\_index \* RECORD\_LENGTH  with open(FILE\_NAME, "rb") as f:  f.seek(offset)  raw = f.read(RECORD\_LENGTH)  flag = raw[0:1]  if flag == EMPTY\_BYTE:  return False  elif flag == OCCUPIED\_FLAG and unpad(raw) == item:  return True  return False  def delete(item):  index = hash(item)  for i in range(TABLE\_SIZE):  probe\_index = (index + i) % TABLE\_SIZE  offset = probe\_index \* RECORD\_LENGTH  with open(FILE\_NAME, "r+b") as f:  f.seek(offset)  raw = f.read(RECORD\_LENGTH)  flag = raw[0:1]  if flag == EMPTY\_BYTE:  return False  elif flag == OCCUPIED\_FLAG and unpad(raw) == item:  f.seek(offset)  f.write(TOMBSTONE\_FLAG + EMPTY\_BYTE \* (RECORD\_LENGTH - 1))  return True  return False |

| **circular\_queue.py** |
| --- |
| from abc import ABC, abstractmethod  # abstract class used as the implementation of these methods may be updated at  # some point  class AbstractQueue(ABC):  def \_\_init\_\_(self, \_\_size):  self.\_\_size = \_\_size  self.\_\_queue = [None] \* \_\_size  self.\_\_front = -1  self.\_\_rear = -1  @abstractmethod  def is\_empty(self):  pass  @abstractmethod  def is\_full(self):  pass  @abstractmethod  def enqueue(self, item):  pass  @abstractmethod  def dequeue(self):  pass  class CircularQueue(AbstractQueue):  def \_\_init\_\_(self, \_\_size):  super().\_\_init\_\_(\_\_size)  def is\_empty(self):  return self.\_\_front == -1  def is\_full(self):  return (self.\_\_rear + 1) % self.\_\_size == self.\_\_front  def enqueue(self, item):  if self.is\_full():  return  # the front is only modified to indicate that the queue is no longer  # empty  if self.is\_empty():  self.\_\_front = 0  self.\_\_rear = (self.\_\_rear + 1) % self.\_\_size  self.\_\_queue[self.\_\_rear] = item  def dequeue(self):  if self.is\_empty():  return None  item = self.\_\_queue[self.\_\_front]  if self.\_\_front == self.\_\_rear:  # essentially resets the queue if it is empty  self.\_\_front = -1  self.\_\_rear = -1  else:  self.\_\_front = (self.\_\_front + 1) % self.\_\_size  return item |

| **misc\_functions.py** |
| --- |
| import datetime as dt  import hash\_table\_functions as ht  # strength check  def strong(password):  if len(password) < 8:  return False  if not any(char.islower() for char in password):  return False  if not any(char.isupper() for char in password):  return False  if not any(char.isdigit() for char in password):  return False  if not any(char in "!@#$%^&\*()-\_=+" for char in password):  return False  return True  # importance functions  def deadline\_missed(datestr):  return dt.datetime.strptime(datestr, "%Y-%m-%d").date() < dt.date.today()  def deadline\_approaching(datestr):  return dt.datetime.strptime(datestr, "%Y-%m-%d").date() in [  dt.date.today(),  dt.date.today() + dt.timedelta(1),  ]  def importance(datestr, baseimportance):  if baseimportance == 0:  return 6  if datestr == "0-0-0":  return baseimportance  if deadline\_missed(datestr):  return 5  if deadline\_approaching(datestr):  return baseimportance + 1  return baseimportance  # create username  def create\_username(firstname, lastname):  i = 1  while True:  username = f"{firstname[0].lower()}{lastname.lower()}{i}"  if not ht.search(username):  ht.insert(username)  return username  i += 1  # date validity check  def date\_valid(day, month, year):  if not (day.isdigit() and month.isdigit() and year.isdigit()):  return False  day = int(day)  month = int(month)  year = int(year)  if month > 12 or month < 1:  return False  if day > 31 or day < 1:  return False  if month in [4, 6, 9, 11] and day > 30:  return False  if month == 2:  if (year % 400 == 0) or (year % 100 != 0 and year % 4 == 0):  max\_days = 29  else:  max\_days = 28  if day > max\_days:  return False  if year > 9999:  return False  if dt.date(year, month, day) < dt.date.today():  return False  return True  # calendar cell functions  def get\_cell\_day(i):  return (  dt.date.today() + dt.timedelta(days=i - dt.date.today().weekday())  ).day  def get\_cell\_date\_string(i):  cell\_day = dt.date.today() + dt.timedelta(  days=i - dt.date.today().weekday()  )  return f"{cell\_day.year}-{cell\_day.month}-{cell\_day.day}"  def get\_cell\_bg(i):  cell\_day = dt.date.today() + dt.timedelta(  days=i - dt.date.today().weekday()  )  if cell\_day < dt.date.today():  return "darkgrey"  if cell\_day == dt.date.today():  return "lightblue"  if cell\_day.weekday() in [5, 6]:  return "lightgrey"  return "white"  # sort  def merge\_sort(data, key=lambda x: x):  if len(data) <= 1:  return data  mid = len(data) // 2  left = merge\_sort(data[:mid], key)  right = merge\_sort(data[mid:], key)  return \_merge(left, right, key)  def \_merge(left, right, key):  # underscore signals the subroutine is for use internally  result = []  i = j = 0  # i is left index, j is right index  while i < len(left) and j < len(right):  if key(left[i]) < key(right[j]):  result.append(left[i])  i += 1  else:  result.append(right[j])  j += 1  while i < len(left):  result.append(left[i])  i += 1  while j < len(right):  result.append(right[j])  j += 1  return result |

**These final three programs create the database, the binary hash table, and the key file.**

| **database\_table\_creator.py** |
| --- |
| import sqlite3 as sq  def account():  conn = sq.connect("team\_tracker.db")  cur = conn.cursor()  cur.execute(  """CREATE TABLE "Account" (  "Username" TEXT,  "FirstName" TEXT,  "LastName" TEXT,  "Password" BLOB,  "Team" TEXT,  "Admin" INTEGER,  "Active" INTEGER,  PRIMARY KEY("Username")  )"""  )  cur.close()  conn.close()  def task():  conn = sq.connect("team\_tracker.db")  cur = conn.cursor()  cur.execute(  """CREATE TABLE "Task" (  "TaskID" INTEGER,  "Title" TEXT,  "Description" TEXT,  "State" TEXT,  "Deadline" TEXT,  "BaseImportance" INTEGER,  "Team" TEXT,  PRIMARY KEY("TaskID")  )"""  )  cur.close()  conn.close()  def event():  conn = sq.connect("team\_tracker.db")  cur = conn.cursor()  cur.execute(  """CREATE TABLE "Event" (  "EventID" INTEGER,  "Title" TEXT,  "Date" TEXT,  PRIMARY KEY("EventID")  )"""  )  cur.close()  conn.close()  def assignment():  conn = sq.connect("team\_tracker.db")  cur = conn.cursor()  cur.execute(  """CREATE TABLE "Assignment" (  "Username" TEXT,  "TaskID" INTEGER,  PRIMARY KEY("Username","TaskID"),  FOREIGN KEY("TaskID") REFERENCES "Task"("TaskID"),  FOREIGN KEY("Username") REFERENCES "Account"("Username")  )"""  )  cur.close()  conn.close()  def attendance():  conn = sq.connect("team\_tracker.db")  cur = conn.cursor()  cur.execute(  """CREATE TABLE "Attendance" (  "Username" TEXT,  "EventID" INTEGER,  PRIMARY KEY("Username","EventID"),  FOREIGN KEY("EventID") REFERENCES "Event"("EventID"),  FOREIGN KEY("Username") REFERENCES "Account"("Username")  )"""  )  cur.close()  conn.close()  def all():  account()  task()  event()  assignment()  attendance() |

| **hash\_table\_creator.py** |
| --- |
| from hash\_table\_functions import \*  from sql\_functions import get\_all\_usernames  def reset\_hash\_table():  with open(FILE\_NAME, "wb") as f:  f.write(EMPTY\_BYTE \* TABLE\_SIZE \* RECORD\_LENGTH)  def refill\_hash\_table():  for username in get\_all\_usernames():  insert(username) |

| **key\_creator.py** |
| --- |
| from cryptography.fernet import Fernet  key = Fernet.generate\_key()  with open('key.key', 'wb') as key\_file:  key\_file.write(key) |

**Testing**

**Notes**

Each table and video corresponds to the fewest number of objectives as is reasonable. Some objectives are so closely related that it would be hard to separate them. Additionally, some objectives may be shown to be achieved across multiple tests (e.g. objective 1.2.2—“The password is only accepted if it is strong enough”—will be demonstrated across the many tests in which we verify that the password strength check system works as intended). These objectives will be mentioned in brackets next to the objective actually being tested. Letter indexing (e.g. “Test 1a”, “Test 1b”, …) is used to show multiple different types of tests being used to test the validity of the same objective(s).

**Testing Videos**

<https://drive.google.com/drive/folders/17DLf-OdeeOYi9YdL7uvhsCyt_tcbj06g?usp=sharing>

**The Login System**

**Test 1**

| Objective | 1.1, 1.1.1 |
| --- | --- |
| Test Type | N/A |
| Test | N/A |
| Expected Outcome | The user can enter a username and password with the password appearing as “\*”s |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test1.mkv](https://drive.google.com/open?id=1p1fFl2yxd0uk9YzhoAJu_e7IZe4BOE4O&usp=drive_copy) |
| Improvements | N/A |

**Test 2**

| Objective | 1.1.2.1 (1.1.2) |
| --- | --- |
| Test Type | Valid |
| Test | Username “gdriver1” and password “!1Atesting” |
| Expected Outcome | The main program loads up |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test2.mkv](https://drive.google.com/open?id=1J5t6Wge1Qicfv1JsKcFz-UxJTz3WC5-A&usp=drive_copy) |
| Improvements | N/A |

**Test 3a**

| Objective | 1.1.2.2 (1.1.2) |
| --- | --- |
| Test Type | Invalid |
| Test | Valid username “gdriver1” and invalid password “test” |
| Expected Outcome | A pop-up notifies the user that the details are incorrect |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test3a.mkv](https://drive.google.com/open?id=1cM5VHpdzGy3O9O34Q469xqQfBn9z1k6K&usp=drive_copy) |
| Improvements | N/A |

**Test 3b**

| Objective | 1.1.2.2 (1.1.2) |
| --- | --- |
| Test Type | Invalid |
| Test | Invalid username “gdriver12” and valid password “!1Atesting” |
| Expected Outcome | A pop-up notifies the user that the details are incorrect |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test3b.mkv](https://drive.google.com/open?id=1nPX5bWmFdn6GoBvS0THtT-jMKg-cjFyj&usp=drive_copy) |
| Improvements | N/A |

**Test 4**

| Objective | 1.2 |
| --- | --- |
| Test Type | N/A |
| Test | A user logs into a newly-created account |
| Expected Outcome | The user is prompted to change their password |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test4.mkv](https://drive.google.com/open?id=1Z8pZxkJwJLzZU9t0JKf50ancICCBlHv_&usp=drive_copy) |
| Improvements | N/A |

**Test 5**

| Objective | 1.2.1, 1.2.1.1 |
| --- | --- |
| Test Type | N/A |
| Test | N/A |
| Expected Outcome | The user can enter a password with the characters appearing as “\*”s |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test5.mkv](https://drive.google.com/open?id=1fO4wiLstxNow1GDqisYmt7b5hNFU8dl9&usp=drive_copy) |
| Improvements | N/A |

**Test 6**

| Objective | 1.2.1.2, 1.2.2.1-5 (1.2.2) |
| --- | --- |
| Test Type | Valid |
| Test | Both passwords are “!1Aabcdefg” |
| Expected Outcome | The main program loads up |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test6.mkv](https://drive.google.com/open?id=1ad8y5ouybqvfv4eLcrW3c4h2URiPK-Xk&usp=drive_copy) |
| Improvements | N/A |

**Test 7**

| Objective | 1.2.1.2 |
| --- | --- |
| Test Type | Invalid |
| Test | The first password is “!1Aabcdefg” and the second password is “!2Aabcdefg” |
| Expected Outcome | A pop-up notifies the user that the passwords are different |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test7.mkv](https://drive.google.com/open?id=1y1NzZXK0gLksQlu21CSy1IGBoqOZTolp&usp=drive_copy) |
| Improvements | N/A |

**Test 8a**

| Objective | 1.2.2.1 (1.2.2) |
| --- | --- |
| Test Type | Valid Boundary |
| Test | Both passwords are “!1Aabcde” |
| Expected Outcome | The main program is loaded up |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test8a.mkv](https://drive.google.com/open?id=1x5_RO8ebVmbETNaLy7Ph2R4JiDOTUVka&usp=drive_copy) |
| Improvements | N/A |

**Test 8b**

| Objective | 1.2.2.1 (1.2.3) |
| --- | --- |
| Test Type | Invalid Boundary |
| Test | Both passwords are “!1Aabcd” |
| Expected Outcome | A pop-up notifies the user that the password isn’t strong enough |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test8b.mkv](https://drive.google.com/open?id=1wUTRx7FhJPFf7DRGRVLd4nlWwJinTRQO&usp=drive_copy) |
| Improvements | N/A |

**Test 9**

| Objective | 1.2.2.2 (1.2.3) |
| --- | --- |
| Test Type | Invalid |
| Test | Both passwords are “1Aabcdefg” |
| Expected Outcome | A pop-up notifies the user that the password isn’t strong enough |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test9.mkv](https://drive.google.com/open?id=1J-LP_7HOhlXI_qnAPDCkHK-uDmqQh7J5&usp=drive_copy) |
| Improvements | N/A |

**Test 10**

| Objective | 1.2.2.3 (1.2.3) |
| --- | --- |
| Test Type | Invalid |
| Test | Both passwords are “!Aabcdefg” |
| Expected Outcome | A pop-up notifies the user that the password isn’t strong enough |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test10.mkv](https://drive.google.com/open?id=14yfRcpKxmuiaTvul104GzFG8MesvUn7t&usp=drive_copy) |
| Improvements | N/A |

**Test 11**

| Objective | 1.2.2.4 (1.2.3) |
| --- | --- |
| Test Type | Invalid |
| Test | Both passwords are “!1abcdefg” |
| Expected Outcome | A pop-up notifies the user that the password isn’t strong enough |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test11.mkv](https://drive.google.com/open?id=1pNCSLJwrlFecg73SFfpbPjiEy6ldvbNr&usp=drive_copy) |
| Improvements | N/A |

**Test 12**

| Objective | 1.2.2.5 (1.2.3) |
| --- | --- |
| Test Type | Invalid |
| Test | Both passwords are “!1ABCDEFG” |
| Expected Outcome | A pop-up notifies the user that the password isn’t strong enough |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test12.mkv](https://drive.google.com/open?id=1lNdQWgkSE6YLAo0W2sQUh8oqXAv41K4Z&usp=drive_copy) |
| Improvements | N/A |

NOTE: As the system for checking password strength is the same across the program, this will be the only time I do a comprehensive test of this functionality.

**Test 13**

| Objective | 1.2.4, 1.2.4.1 |
| --- | --- |
| Test Type | N/A |
| Test | Password changes from “!1Aabcdefg” to “!1Atesting” |
| Expected Outcome | The hash in the database changes value |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test13.mkv](https://drive.google.com/open?id=11fY0lVGObx4EIkG3CbG11zEhw99A8B7m&usp=drive_copy) |
| Improvements | N/A |

**Test 14a**

| Objective | 1.3.1 (1.3) |
| --- | --- |
| Test Type | N/A |
| Test | Login with admin account “gdriver1” |
| Expected Outcome | The accounts interface can be accessed |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test14a.mkv](https://drive.google.com/open?id=1YwQv2RwLVqwieHHEVbe5cG6HPT6eDppj&usp=drive_copy) |
| Improvements | N/A |

**Test 14b**

| Objective | 1.3.1 (1.3) |
| --- | --- |
| Test Type | N/A |
| Test | Login with non-admin account “gdriver2” |
| Expected Outcome | The accounts interface cannot be accessed |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test14b.mkv](https://drive.google.com/open?id=1dXQXnaX9ZTXtpXtSH4uhgk4eOjooKwRu&usp=drive_copy) |
| Improvements | N/A |

**Test 15a**

| Objective | 1.3.2 (1.3) |
| --- | --- |
| Test Type | N/A |
| Test | Login with account “jdoe1” which is assigned to the development team |
| Expected Outcome | We see “Development” next to the account name in the top right corner |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test15a.mkv](https://drive.google.com/open?id=1Mh6tbS1vpO-pUgHIk7-hnsjmNt9oYlQ0&usp=drive_copy) |
| Improvements | N/A |

**Test 15b**

| Objective | 1.3.2 (1.3) |
| --- | --- |
| Test Type | N/A |
| Test | Login with account “gdriver1” which is assigned to both teams |
| Expected Outcome | We see “Operational & Development” next to the account name in the top right corner |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test15b.mkv](https://drive.google.com/open?id=183fHjAB_D7r5ExsTtoGWvG4RiVxfs5iA&usp=drive_copy) |
| Improvements | N/A |

**Test 16**

| Objective | 1.3.3, 1.3.3.1 |
| --- | --- |
| Test Type | N/A |
| Test | Switch from and back to the tasks tab while having the “New Task” interface open |
| Expected Outcome | The tab should revert to the main task interface |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test16.mkv](https://drive.google.com/open?id=1QMxgXkqLPyR47wC7MllaMMnw0usdp3Jl&usp=drive_copy) |
| Improvements | N/A |

**The Task Interface**

**Test 17**

| Objective | 2.1, 2.1.1, 2.1.2-5, 2.2, 2.2.1-6 |
| --- | --- |
| Test Type | N/A |
| Test | Select the task interface tab on the “jdoe1” account |
| Expected Outcome | The correct tasks (i.e. “Development” team tasks) are visible with their IDs, titles, descriptions, deadlines, and importances. They are organised into columns (“Backlog”, “To Do”, “Doing”, “Done”) that can be scrolled down and that are internally ordered by importance. |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test17.mkv](https://drive.google.com/open?id=1-CPbGUEpnOjCVE4A_eNMINyEF_jYstZ5&usp=drive_copy) |
| Improvements | N/A |

**Test 18a**

| Objective | 2.1.5.1 |
| --- | --- |
| Test Type | Boundary |
| Test | The task “Bake a cheesecake” (with importance “Low”) has a deadline “2025-4-30” (in two days, as of testing) |
| Expected Outcome | The task’s importance is shown to be green |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test18a.mkv](https://drive.google.com/open?id=1nKldWCRnd4Seu8-4qyn6fBBLXmT9ZJOj&usp=drive_copy) |
| Improvements | N/A |

**Test 18b**

| Objective | 2.1.5.1 |
| --- | --- |
| Test Type | Boundary |
| Test | The task “Bake a muffin” (with importance “High”) has a deadline “2025-4-29” (tomorrow, as of testing) |
| Expected Outcome | The task’s importance is shown to be red as opposed to orange |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test18b.mkv](https://drive.google.com/open?id=1lSXLYFIyyYwCn6TjklkxHJMtFSi-Q_NR&usp=drive_copy) |
| Improvements | N/A |

**Test 18c**

| Objective | 2.1.5.1 |
| --- | --- |
| Test Type | Boundary |
| Test | The task “Buy some eggs” has a deadline “2025-4-27” (yesterday, as of testing) |
| Expected Outcome | The task’s importance is shown to be purple |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test18c.mkv](https://drive.google.com/open?id=14J2pHvrNp0GZsGwV-tbGy6jMM3EBMZ_w&usp=drive_copy) |
| Improvements | N/A |

**Test 18d**

| Objective | 2.1.5.1 |
| --- | --- |
| Test Type | Boundary |
| Test | The task “Buy some cheese” (with importance “Low”) has a deadline “2025-4-28” (today, as of testing) |
| Expected Outcome | The task’s importance is shown to be yellow as opposed to green |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test18d.mkv](https://drive.google.com/open?id=1ERI0fKCOBelUgU1CTnBI-Y6ALKdRHG3D&usp=drive_copy) |
| Improvements | N/A |

**Test 19**

| Objective | 2.3.1, 2.3.2, 2.3.2.1-4, 2.3.2.4.1, 2.3.3 |
| --- | --- |
| Test Type | N/A |
| Test | Select the “Add Task” button |
| Expected Outcome | The “New Task” interface opens in which you can enter a title, team, description, and deadline (requiring a button to reach the entry widgets). A button to create the task is present. |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test19.mkv](https://drive.google.com/open?id=1uFuwsx63szCn9oNB3x0gDiVGJU2AnEJW&usp=drive_copy) |
| Improvements | N/A |

**Test 20**

| Objective | 2.3.3.1.1-3 (2.3.3.1, 2.3.4, 2.3) |
| --- | --- |
| Test Type | Valid |
| Test | Title “Bake a cake”, team “Operational”, and deadline “2025-9-2” |
| Expected Outcome | The main task interface should reload up with the new task |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test20.mkv](https://drive.google.com/open?id=1qXR0Uccd5pBmV0D6YfJ-YM7674Gzr4Rs&usp=drive_copy) |
| Improvements | N/A |

**Test 21**

| Objective | 2.3.3.1.1 (2.3.3.2) |
| --- | --- |
| Test Type | Invalid |
| Test | Title “”, team “Operational”, and deadline “2025-9-2” |
| Expected Outcome | A pop-up notifies the user that the details aren’t valid |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test21.mkv](https://drive.google.com/open?id=1yYfLIPunXaL0N9lY_GsjFIrwE1Nc0N7c&usp=drive_copy) |
| Improvements | N/A |

**Test 22**

| Objective | 2.3.3.1.2 (2.3.3.2) |
| --- | --- |
| Test Type | Invalid |
| Test | Title “Bake a cake”, team “Select a team…”, and deadline “2025-9-2” |
| Expected Outcome | A pop-up notifies the user that the details aren’t valid |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test22.mkv](https://drive.google.com/open?id=11U0IDtmharmZsuhVlMHRVEusA70BVW-r&usp=drive_copy) |
| Improvements | N/A |

**Test 23a**

| Objective | 2.3.3.1.3 (2.3.3.1, 2.3.4, 2.3) |
| --- | --- |
| Test Type | Valid Boundary |
| Test | Title “Bake a cake”, team “Operational”, and deadline “2028-2-29” |
| Expected Outcome | The main task interface should reload up with the new task |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test23a.mkv](https://drive.google.com/open?id=1BYm1ZRGjBdzjo_KVDtmbu6EzJ4zgHQnA&usp=drive_copy) |
| Improvements | N/A |

**Test 23b**

| Objective | 2.3.3.1.3 (2.3.3.1, 2.3.4, 2.3) |
| --- | --- |
| Test Type | Valid Boundary |
| Test | Title “Bake a cake”, team “Operational”, and deadline “2025-4-28” (today, as of testing) |
| Expected Outcome | The main task interface should reload up with the new task |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test23b.mkv](https://drive.google.com/open?id=1OYUAUX0Bwxq0tX1zFYnmqiaf_aCL6h_A&usp=drive_copy) |
| Improvements | N/A |

**Test 23c**

| Objective | 2.3.3.1.3 (2.3.3.2) |
| --- | --- |
| Test Type | Invalid Boundary |
| Test | Title “Bake a cake”, team “Operational”, and deadline “2026-2-29” |
| Expected Outcome | A pop-up notifies the user that the details aren’t valid |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test23c.mkv](https://drive.google.com/open?id=14mb-28Zk6n0m0FLvWrJgu_A4HHYnLyxQ&usp=drive_copy) |
| Improvements | N/A |

**Test 23d**

| Objective | 2.3.3.1.3 (2.3.3.2) |
| --- | --- |
| Test Type | Invalid Boundary |
| Test | Title “Bake a cake”, team “Operational”, and deadline “2025-4-27” (yesterday, as of testing) |
| Expected Outcome | A pop-up notifies the user that the details aren’t valid |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test23d.mkv](https://drive.google.com/open?id=1Ka07FKvYl57XswMNJqzVxOJ0DJSPA3lC&usp=drive_copy) |
| Improvements | N/A |

**Test 23e**

| Objective | 2.3.3.1.3 (2.3.3.2) |
| --- | --- |
| Test Type | Invalid Boundary |
| Test | Title “Bake a cake”, team “Operational”, and deadline “2025-13-23” |
| Expected Outcome | A pop-up notifies the user that the details aren’t valid |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test23e.mkv](https://drive.google.com/open?id=1eoYsrH9scdJl285j7dEu9JmCH0EMPLYm&usp=drive_copy) |
| Improvements | N/A |

NOTE: As the system for checking date validity is the same across the program, this will be the only time I do a comprehensive test of this functionality.

**Test 24**

| Objective | 2.4.1, 2.4.2, 2.4.2.1-4, 2.4.3 |
| --- | --- |
| Test Type | N/A |
| Test | Select the “Edit Task” button |
| Expected Outcome | The “Edit Task” interface opens in which you can edit the team, description, deadline, and importance. A button to save the changes is present. |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test24.mkv](https://drive.google.com/open?id=12KiHaCd2SZ_D2wtiTdFSgObId-JUDjfl&usp=drive_copy) |
| Improvements | N/A |

**Test 25**

| Objective | 2.4.3.1.1 (2.4.3.1, 2.4.4, 2.4) |
| --- | --- |
| Test Type | Valid |
| Test | Change all values in the “Bake a cake” task. Change the importance from “Low” to “High”. Change the description from “” to “Victoria sponge”. Change the deadline from “2025-5-5” to “2026-6-6”. Change the team from “Operational” to “Both”. |
| Expected Outcome | The main task interface should reload up with the changes visible |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test25.mkv](https://drive.google.com/open?id=1WWuacIIbjxtkGob9wsxgXFSuvIaQLtJ6&usp=drive_copy) |
| Improvements | N/A |

**Test 26**

| Objective | 2.4.3.1.2 (2.4.3.2) |
| --- | --- |
| Test Type | Invalid |
| Test | Change the deadline for the “Bake a cake” task from “2025-9-2” to “1000-0-0” |
| Expected Outcome | A pop-up notifies the user that the details aren’t valid |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test26.mkv](https://drive.google.com/open?id=1Qqa0NWzfTlE_zrV0225nz_qIkEskgp_i&usp=drive_copy) |
| Improvements | N/A |

**Test 27**

| Objective | 2.5.1, 2.5.1.1 |
| --- | --- |
| Test Type | N/A |
| Test | Select the “Delete” button |
| Expected Outcome | A pop-up should appear asking the user to confirm the deletion of this task |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test27.mkv](https://drive.google.com/open?id=1pk7tGSi5Rbq3bJAgRZ8MhBW56xaV8Etn&usp=drive_copy) |
| Improvements | N/A |

**Test 28**

| Objective | 2.5.1.2, 2.5.1.3, 2.5 |
| --- | --- |
| Test Type | N/A |
| Test | Confirm the deletion |
| Expected Outcome | The task should be deleted from the interface and should stay so even after switching to and from tabs |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test28.mkv](https://drive.google.com/open?id=1mObi7rQ9KDf5IayJu0hhR-OuYKJd711s&usp=drive_copy) |
| Improvements | N/A |

**Test 29**

| Objective | 2.6.1.1.1.1 (2.6.1, 2.6.1.1, 2.6.1.1.1, 2.6.1.1.2) |
| --- | --- |
| Test Type | N/A |
| Test | Select the “Progress” button on a task in the “Backlog” column |
| Expected Outcome | The “Progress to “To Do”” interface opens in which you can enter an importance and a deadline. A button to progress the task is present. |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test29.mkv](https://drive.google.com/open?id=1-tfdOOQUgclMkcnQC1vcqELgMhmJ3t-B&usp=drive_copy) |
| Improvements | N/A |

**Test 30**

| Objective | 2.6.1.1.1.2 (2.6.1, 2.6.1.1, 2.6.1.1.1, 2.6.1.1.2) |
| --- | --- |
| Test Type | N/A |
| Test | Select the “Progress” button on a task in the “To Do” column |
| Expected Outcome | The “Progress to “Doing”” interface opens in which you can select assignees. A button to progress the task is present. |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test30.mkv](https://drive.google.com/open?id=1MQK0zPAVctAu13YYt6HF_hOPp3mG1mfH&usp=drive_copy) |
| Improvements | N/A |

**Test 31a**

| Objective | 2.6.1.1.1.1.1 |
| --- | --- |
| Test Type | N/A |
| Test | The “Progress” button is selected on a task without a deadline present in the database |
| Expected Outcome | A deadline can be entered |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test31a.mkv](https://drive.google.com/open?id=14Ii_KOYdRxhQDK9SZrUd90uLBmEsM9JU&usp=drive_copy) |
| Improvements | N/A |

**Test 31b**

| Objective | 2.6.1.1.1.1.1 |
| --- | --- |
| Test Type | N/A |
| Test | The “Progress” button is selected on a task with a deadline present in the database |
| Expected Outcome | A deadline can’t be entered |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test31b.mkv](https://drive.google.com/open?id=1isLafo_k0IZiSsk8LpYa7C9ENfXQEY__&usp=drive_copy) |
| Improvements | N/A |

**Test 32**

| Objective | 2.6.1.1.2.1.1, 2.6.1.1.2.1.2 (2.6.1.1.2.1, 2.6.1.1.3, 2.6) |
| --- | --- |
| Test Type | Valid |
| Test | For a task without a deadline; importance “Low”, deadline “2025-9-4” |
| Expected Outcome | The main task interface should reload up with the task progressed |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test32.mkv](https://drive.google.com/open?id=1GaTHlSk5n6g9IvXcOconpk3-8fFEMfDh&usp=drive_copy) |
| Improvements | N/A |

**Test 33**

| Objective | 2.6.1.1.2.1.1 (2.6.1.1.2.1, 2.6.1.1.3, 2.6) |
| --- | --- |
| Test Type | Valid |
| Test | For a task with a deadline; importance “Low” |
| Expected Outcome | The main task interface should reload up with the task progressed |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test33.mkv](https://drive.google.com/open?id=1r2-pAbLkCEXNp4QOnHgtDUXLA1_ZF0HX&usp=drive_copy) |
| Improvements | N/A |

**Test 34**

| Objective | 2.6.1.1.2.1.1 (2.6.1.1.3) |
| --- | --- |
| Test Type | Invalid |
| Test | No importance selected, deadline “2025-9-4” |
| Expected Outcome | A pop-up notifies the user that the details aren’t valid |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test34.mkv](https://drive.google.com/open?id=1kchT57Fhe76lobUnUTZmtcmI0AuuzHyU&usp=drive_copy) |
| Improvements | N/A |

**Test 35**

| Objective | 2.6.1.1.2.1.2 (2.6.1.1.3) |
| --- | --- |
| Test Type | Invalid |
| Test | Importance “Low”, deadline “2021-9-4” |
| Expected Outcome | A pop-up notifies the user that the details aren’t valid |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test35.mkv](https://drive.google.com/open?id=1fLGXvUdE404OLS1VxyGiktW2Wz4wXlgZ&usp=drive_copy) |
| Improvements | N/A |

**Test 36a**

| Objective | 2.6.1.1.2.1.3 (2.6.1.1.2.1, 2.6.1.1.3, 2.6) |
| --- | --- |
| Test Type | Valid |
| Test | Assignees “gdriver1” |
| Expected Outcome | The main task interface should reload up with the task progressed |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test36a.mkv](https://drive.google.com/open?id=180voyBW9-4UEGsbQLouDccP8NSFd2yuH&usp=drive_copy) |
| Improvements | N/A |

**Test 36b**

| Objective | 2.6.1.1.2.1.3 (2.6.1.1.2.1, 2.6.1.1.3, 2.6) |
| --- | --- |
| Test Type | Valid |
| Test | Assignees “gdriver1” and “jdoe2” |
| Expected Outcome | The main task interface should reload up with the task progressed |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test36b.mkv](https://drive.google.com/open?id=1NPq4GWSLKZmHCI5S2X9cHeLp506oZ3tB&usp=drive_copy) |
| Improvements | N/A |

**Test 36c**

| Objective | 2.6.1.1.2.1.3 (2.6.1.1.2.1, 2.6.1.1.3, 2.6) |
| --- | --- |
| Test Type | Valid |
| Test | Assignees “gdriver1” and an unselected assignee |
| Expected Outcome | The main task interface should reload up with the task progressed |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test36c.mkv](https://drive.google.com/open?id=1YEFwd-Yezb_dAkEPdm_ZI9SABJXtOOna&usp=drive_copy) |
| Improvements | N/A |

**Test 37**

| Objective | 2.6.1.1.2.1.3 (2.6.1.1.2.2) |
| --- | --- |
| Test Type | Invalid |
| Test | No assignees selected |
| Expected Outcome | A pop-up notifies the user that the details aren’t valid |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test37.mkv](https://drive.google.com/open?id=10EVmZ62XX16DOwdtk4y2NaAA-sAZIIRZ&usp=drive_copy) |
| Improvements | N/A |

**Test 38**

| Objective | 2.6.1.2 |
| --- | --- |
| Test Type | N/A |
| Test | Select the “Progress” button on a task in the “Doing” column |
| Expected Outcome | A pop-up should appear asking the user to confirm the progression of this task to the “Done” column |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test38.mkv](https://drive.google.com/open?id=1RJW77aFGJLXqtj1Pc37e6F4JG1kSmU1o&usp=drive_copy) |
| Improvements | N/A |

**Test 39**

| Objective | 2.6.1.2.1, 2.6.1.2.2 (2.6) |
| --- | --- |
| Test Type | N/A |
| Test | Confirm the progression |
| Expected Outcome | The task should be progressed to the “Done” column |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test39.mkv](https://drive.google.com/open?id=1Yn5lG965n4Uv61kN4_IzqH4Pm6LcnXKh&usp=drive_copy) |
| Improvements | N/A |

**The Event Interface**

**Test 40**

| Objective | 3.1, 3.1.1.3, 3.2, 3.2.1, 3.2.1.1, 3.2.1.2 |
| --- | --- |
| Test Type | N/A |
| Test | Select the event interface tab on the “jdoe1” account |
| Expected Outcome | The correct events (i.e. events assigned to him) and deadlines (i.e. “Development” team deadlines) are visible with their titles and importances. They are organised in a 7x4 grid showing 4 weeks of dates in which the grid is coloured to make it clear which days have passed and which days are weekends. |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test40.mkv](https://drive.google.com/open?id=1_3IcOx0K_L2EX0tLToBet6qthhqTI02g&usp=drive_copy) |
| Improvements | N/A |

**Test 41**

| Objective | 3.3.1, 3.3.2, 3.3.2.1, 3.3.2.2, 3.3.3 |
| --- | --- |
| Test Type | N/A |
| Test | Select the “Add Event” button |
| Expected Outcome | The “New Event” interface opens in which you can enter a title and date. A button to create the event is present. |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test41.mkv](https://drive.google.com/open?id=1E9x3w7fOCG2-atAW1uiMsm1UW4mS3JLg&usp=drive_copy) |
| Improvements | N/A |

**Test 42**

| Objective | 3.3.3.1.1, 3.3.3.1.2 (3.3.3.1, 3.3.4, 3.3) |
| --- | --- |
| Test Type | Valid |
| Test | Title “The cake festival”, date “2025-5-16”, attendees “jdoe1” |
| Expected Outcome | The main event interface should reload up with the new event |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test42.mkv](https://drive.google.com/open?id=1fiDVPPeO46evxoMAr4jmmRy_75GOM2gM&usp=drive_copy) |
| Improvements | N/A |

**Test 43**

| Objective | 3.3.3.1.1 (3.3.3.2) |
| --- | --- |
| Test Type | Invalid |
| Test | Title “”, date “2025-5-16”, attendees “jdoe1” |
| Expected Outcome | A pop-up notifies the user that the details aren’t valid |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test43.mkv](https://drive.google.com/open?id=1wVX8avypjNsvlDQe-HoWKGpQnzhB996n&usp=drive_copy) |
| Improvements | N/A |

**Test 44**

| Objective | 3.3.3.1.2 (3.3.3.2) |
| --- | --- |
| Test Type | Invalid |
| Test | Title “The cake festival”, date “2021-7-7”, attendees “jdoe1” |
| Expected Outcome | A pop-up notifies the user that the details aren’t valid |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test44.mkv](https://drive.google.com/open?id=1fh5rgjFJMRNXJ0Y-EzvUvWNvw_peWdDz&usp=drive_copy) |
| Improvements | N/A |

NOTE: The attendee system will be tested further down as it was not a part of my objectives.

**Test 45**

| Objective | 3.4.1, 3.4.2, 3.4.3 |
| --- | --- |
| Test Type | N/A |
| Test | Select the “Delete Event” button |
| Expected Outcome | The “Remove Event” interface opens in which you can select an event to delete. A button to delete the event is present. |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test45.mkv](https://drive.google.com/open?id=1RfZvs_EXKhBL1hg1uYhfMQSMKkuBVv36&usp=drive_copy) |
| Improvements | N/A |

**Test 46**

| Objective | 3.4.3.1, 3.4.4, 3.4 |
| --- | --- |
| Test Type | Valid |
| Test | Event “The cake festival” |
| Expected Outcome | The main event interface should reload up without the deleted event |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test46.mkv](https://drive.google.com/open?id=1irv27b-0FF4UDy4mzlUYzIr5aPHXbWde&usp=drive_copy) |
| Improvements | N/A |

**Test 47**

| Objective | 3.4.3.2 |
| --- | --- |
| Test Type | Invalid |
| Test | No event selected |
| Expected Outcome | A pop-up notifies the user that the details aren’t valid |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test47.mkv](https://drive.google.com/open?id=1oA9hyJ3C3wjw8UQkH315Q95ES3OhuEZ7&usp=drive_copy) |
| Improvements | N/A |

**The Account Interface**

**Test 48**

| Objective | 4.1, 4.1.1-5, 4.2, 4.2.1, 4.2.2, 4.4.1, 4.4.1.1-4, 4.4.2 |
| --- | --- |
| Test Type | N/A |
| Test | Select the account interface tab |
| Expected Outcome | All accounts are visible with their full names, usernames, and teams along with whether they are an admin account or whether they have a password change scheduled (whether they are active or not). They are organised alphabetically by full name in a list that can be scrolled down. |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test48.mkv](https://drive.google.com/open?id=10lpIycGCyOUdKN0iphDTShEnWA7-6252&usp=drive_copy) |
| Improvements | N/A |

**Test 49**

| Objective | 4.3.1, 4.3.2, 4.3.2.1-4, 4.3.3 |
| --- | --- |
| Test Type | N/A |
| Test | Select the “Add Account” button |
| Expected Outcome | The “New Account” interface opens in which you can enter a first name, last name, team, whether or not the account should be an admin account, and a temporary password. A button to create the account is present. |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test49.mkv](https://drive.google.com/open?id=12N9E5pmYsBuzQj4MGDOn_q0FNCvKbNYo&usp=drive_copy) |
| Improvements | N/A |

**Test 50**

| Objective | 4.3.3.1.1-4, 4.3.3.1.4.1 (4.3.3.1, 4.3.4, 4.3) |
| --- | --- |
| Test Type | Valid |
| Test | First name “John”, last name “Roe”, team “Both”, admin unchecked, temporary password “!1Aabcdefg” |
| Expected Outcome | The main account interface should reload up with the new account with a username “jroe1” |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test50.mkv](https://drive.google.com/open?id=1R8iM-mAvnmiIHmfo3SkoM9Hs3LWcyRi4&usp=drive_copy) |
| Improvements | N/A |

**Test 51a**

| Objective | 4.3.3.1.1 (4.3.3.2) |
| --- | --- |
| Test Type | Invalid |
| Test | First name “”, last name “Roe”, team “Both”, admin unchecked, temporary password “!1Aabcdefg” |
| Expected Outcome | A pop-up notifies the user that the details aren’t valid |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test51a.mkv](https://drive.google.com/open?id=1Poug6a_6L9bZXemvMBf_dlX8zEd8Ld4i&usp=drive_copy) |
| Improvements | N/A |

**Test 51b**

| Objective | 4.3.3.1.1 (4.3.3.2) |
| --- | --- |
| Test Type | Invalid |
| Test | First name “John”, last name “”, team “Both”, admin unchecked, temporary password “!1Aabcdefg” |
| Expected Outcome | A pop-up notifies the user that the details aren’t valid |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test51b.mkv](https://drive.google.com/open?id=1OI1Btu8ce3aD9Tlk9WJ-Eqi47sNT6qbz&usp=drive_copy) |
| Improvements | N/A |

**Test 52**

| Objective | 4.3.3.1.2 (4.3.3.2) |
| --- | --- |
| Test Type | Invalid |
| Test | First name “John”, last name “Roe”, no team selected, admin unchecked, temporary password “!1Aabcdefg” |
| Expected Outcome | A pop-up notifies the user that the details aren’t valid |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test52.mkv](https://drive.google.com/open?id=1LLloy_jeAKvK0fz0md69hiTABmk7qlEc&usp=drive_copy) |
| Improvements | N/A |

**Test 53**

| Objective | 4.3.3.1.3 (4.3.3.2) |
| --- | --- |
| Test Type | Invalid |
| Test | First name “John”, last name “Roe”, team “Both”, admin unchecked, temporary password “!1abcdefg” |
| Expected Outcome | A pop-up notifies the user that the password isn’t strong enough |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test53.mkv](https://drive.google.com/open?id=1LjtghU5Isr2Y50bZfiPcHeUNNZQpEcyq&usp=drive_copy) |
| Improvements | N/A |

**Test 54**

| Objective | 4.3.3.1.4.1 |
| --- | --- |
| Test Type | N/A |
| Test | First name “Jane”, last name “Roe”, team “Both”, admin checked, temporary password “!1Aabcdefg” |
| Expected Outcome | The main account interface should reload up with the new account with a username “jroe2” |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test54.mkv](https://drive.google.com/open?id=1QvTKn5sme2M1BOtri0GRwt362_KenfwJ&usp=drive_copy) |
| Improvements | N/A |

**Test 55**

| Objective | 4.4.2.1 |
| --- | --- |
| Test Type | N/A |
| Test | Select the “Delete” button |
| Expected Outcome | A pop-up should appear asking the user to confirm the deletion of this account |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test55.mkv](https://drive.google.com/open?id=1XM6zpjCsZRbzSEZ2xk-DNV_PIGn14mOV&usp=drive_copy) |
| Improvements | N/A |

**Test 56**

| Objective | 4.4.2.2, 4.4.2.3 |
| --- | --- |
| Test Type | N/A |
| Test | Confirm the deletion |
| Expected Outcome | The account should be deleted from the interface and should stay so even after switching to and from tabs |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test56.mkv](https://drive.google.com/open?id=16erKVpmftiE_ek1jOCX8SCBzvB7nN7Oz&usp=drive_copy) |
| Improvements | N/A |

**Additional Tests**

Below I have tested various features that weren’t required by my objectives.

**Test 57a**

| Feature | The “Clear “Done”” button |
| --- | --- |
| Test Type | N/A |
| Test | Select the “Clear “Done”” button with multiple tasks in the “Done” column |
| Expected Outcome | The tasks in this column should be deleted from the interface and should stay so even after switching to and from tabs |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test57a.mkv](https://drive.google.com/open?id=1DjiCAACTQdACrlMP8ivycJQmmGzHUhUD&usp=drive_copy) |
| Improvements | N/A |

**Test 57b**

| Feature | The “Clear “Done”” button |
| --- | --- |
| Test Type | N/A |
| Test | Select the “Clear “Done”” button without any tasks in the “Done” column |
| Expected Outcome | Nothing should happen |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test57b.mkv](https://drive.google.com/open?id=1X1VFk5n_nv4HmUhdJLdlwFUTUfSR0htW&usp=drive_copy) |
| Improvements | N/A |

**Test 58a**

| Feature | The “Clear Past Events” button |
| --- | --- |
| Test Type | Boundary |
| Test | Select the “Clear Past Events” button with multiple events yesterday and an event today |
| Expected Outcome | The events from yesterday should be deleted from the interface and should stay so even after switching to and from tabs, while the event from today is unchanged |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test58a.mkv](https://drive.google.com/open?id=1EdE7b9s4SLatpe-esjela-bv4MyGhXd4&usp=drive_copy) |
| Improvements | N/A |

**Test 58b**

| Feature | The “Clear Past Events” button |
| --- | --- |
| Test Type | N/A |
| Test | Select the “Clear Past Events” button without any past events |
| Expected Outcome | Nothing should happen |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test58b.mkv](https://drive.google.com/open?id=1rqjCLbR92CpM7oFuROZ-xmVodEG-npxE&usp=drive_copy) |
| Improvements | N/A |

**Test 59**

| Feature | The description widget scroll bar |
| --- | --- |
| Test Type | N/A |
| Test | Click “Edit” on a task with a long description and scroll through it |
| Expected Outcome | The text can be scrolled through |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test59.mkv](https://drive.google.com/open?id=1iJqg_vf4tKsWrME2t0vYBPsfZdvjsAPb&usp=drive_copy) |
| Improvements | N/A |

**Test 60a**

| Feature | The event attendee system |
| --- | --- |
| Test Type | N/A |
| Test | Add an event with the only attendee being you |
| Expected Outcome | The event will appear on your calendar |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test60a.mkv](https://drive.google.com/open?id=101vSufdsqrSWEaJI_sbHDH9tDCb_7Gb8&usp=drive_copy) |
| Improvements | N/A |

**Test 60b**

| Feature | The event attendee system |
| --- | --- |
| Test Type | N/A |
| Test | Add an event with the only attendee being another user |
| Expected Outcome | The event won’t appear on your calendar |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test60b.mkv](https://drive.google.com/open?id=1FMzSPvYW56QJvPfJokkm--Hy8FhUQ4uD&usp=drive_copy) |
| Improvements | N/A |

**Test 60c**

| Feature | The event attendee system |
| --- | --- |
| Test Type | N/A |
| Test | Add an event with the only attendees being you and one other user |
| Expected Outcome | The event will appear on your calendar |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test60c.mkv](https://drive.google.com/open?id=1T4K2ChVIumxUN_6kp8znJ2jqUJxLBJrV&usp=drive_copy) |
| Improvements | N/A |

**Test 60d**

| Feature | The event attendee system |
| --- | --- |
| Test Type | N/A |
| Test | From an “Development” account, add a team event with the attending team being “Both” |
| Expected Outcome | The event will appear on your calendar |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test60d.mkv](https://drive.google.com/open?id=1fsiBYNcscj-uHRlCmKQjLkU6nVlMRFVW&usp=drive_copy) |
| Improvements | N/A |

**Test 60e**

| Feature | The event attendee system |
| --- | --- |
| Test Type | N/A |
| Test | From an “Development” account, add a team event with the attending team being “Operational” |
| Expected Outcome | The event won’t appear on your calendar |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test60e.mkv](https://drive.google.com/open?id=14WOYJdjVud_dUfpkSOgf2v9P3rZ67jSy&usp=drive_copy) |
| Improvements | N/A |

**Test 60f**

| Feature | The event attendee system |
| --- | --- |
| Test Type | N/A |
| Test | From a “Both” account, add a team event with the attending team being “Development” |
| Expected Outcome | The event will appear on your calendar |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test60f.mkv](https://drive.google.com/open?id=1yr7p5YgmzrjfjXIZxpYN5woM3CJ43Igz&usp=drive_copy) |
| Improvements | N/A |

**Test 61a**

| Feature | The “Sort Through Backlog” button |
| --- | --- |
| Test Type | N/A |
| Test | Progress a task and then select “Cancel” |
| Expected Outcome | The task is progressed from “Backlog” |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test61a.mkv](https://drive.google.com/open?id=19YzYgvpH7cLSrVFzBQ1X2ERKJG4l53L1&usp=drive_copy) |
| Improvements | N/A |

**Test 61b**

| Feature | The “Sort Through Backlog” button |
| --- | --- |
| Test Type | N/A |
| Test | Repeatedly press the “Send to Back” button |
| Expected Outcome | The tasks are repeatedly looped over |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test61b.mkv](https://drive.google.com/open?id=1JjlZi8UTxYubp2CkH7757r70FJIUMNKq&usp=drive_copy) |
| Improvements | N/A |

**Test 61c**

| Feature | The “Sort Through Backlog” button |
| --- | --- |
| Test Type | N/A |
| Test | Repeatedly press the “Skip” button |
| Expected Outcome | You continue through all the backlogged tasks before returning to the main interface |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test61c.mkv](https://drive.google.com/open?id=1f0MIdQoirvwTSAMXXjRPpTeDStz2SGCw&usp=drive_copy) |
| Improvements | N/A |

**Test 61d**

| Feature | The “Sort Through Backlog” button |
| --- | --- |
| Test Type | N/A |
| Test | Select “Progress” without selecting an importance |
| Expected Outcome | A pop-up notifies the user that the details aren’t valid |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test61d.mkv](https://drive.google.com/open?id=1qlimNpR9PaB0FEPWN4o6wZYtTRQV2xAo&usp=drive_copy) |
| Improvements | N/A |

**Test 61e**

| Feature | The “Sort Through Backlog” button |
| --- | --- |
| Test Type | N/A |
| Test | Select “Progress” with importance “Low” and deadline “2021-9-4” |
| Expected Outcome | A pop-up notifies the user that the details aren’t valid |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test61e.mkv](https://drive.google.com/open?id=1_YGTCNRt_SiWMiZ01i05TWhoxnbKV3Zi&usp=drive_copy) |
| Improvements | N/A |

**Test 62a**

| Feature | The “Stats” button |
| --- | --- |
| Test Type | N/A |
| Test | Select the “Stats” button while having one uncompleted assigned task, one completed assigned task, and one event on your calendar |
| Expected Outcome | The stats will read “You have 2 tasks assigned to you, 1 of which are completed. You have 1 events in your calendar.” |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test62a.mkv](https://drive.google.com/open?id=1HKa2oDh_7jZh0aBhFePoOng1oRkh3P-4&usp=drive_copy) |
| Improvements | N/A |

**Test 62b**

| Feature | The “Stats” button |
| --- | --- |
| Test Type | Boundary |
| Test | Select the “Stats” button while having no assigned tasks and no events on your calendar |
| Expected Outcome | The stats will read “You have 0 tasks assigned to you, 0 of which are completed. You have 0 events in your calendar.” |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test62b.mkv](https://drive.google.com/open?id=1gN9g_cO8u55AbhDe9EtUK7tYl-gv5iwS&usp=drive_copy) |
| Improvements | N/A |

**Test 63a**

| Feature | Tasks assigned to you appear light blue |
| --- | --- |
| Test Type | N/A |
| Test | Add a task and assign it to yourself only |
| Expected Outcome | The task widget appears light blue |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test63a.mkv](https://drive.google.com/open?id=1K2z9yVWmaB_rz7KvyHlb6gB71Emhgr0R&usp=drive_copy) |
| Improvements | N/A |

**Test 63b**

| Feature | Tasks assigned to you appear light blue |
| --- | --- |
| Test Type | N/A |
| Test | Add a task and assign it to another user only |
| Expected Outcome | The task widget appears light grey |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test63b.mkv](https://drive.google.com/open?id=1fr0ikrmv1fmPKTiOw3DF0_CdrXt-K90V&usp=drive_copy) |
| Improvements | N/A |

**Test 64**

| Feature | Many screens have a “Cancel” button |
| --- | --- |
| Test Type | N/A |
| Test | Test each cancel button, inputting data on each screen apart from the first before selecting “Cancel” |
| Expected Outcome | The user should return to the main interface after each “Cancel” without any changes to the data displayed |
| Actual Outcome | [Copy of Driver\_George\_Nea\_Test64.mkv](https://drive.google.com/open?id=1qw_Z5sTKKOI333hJA3Nx3WESVwjyuqjb&usp=drive_copy) |
| Improvements | N/A |

**Test 65**

This test shows the general functioning of my program, demonstrating all of the application’s most important features, ensuring it all works together as intended.

Video: [Copy of Driver\_George\_Nea\_Test65.mkv](https://drive.google.com/open?id=1PrdqO72ioF4HB5jcls0U7WWzkfggAaHS&usp=drive_copy)

**Evaluation**

**Objectives Reflection**

Throughout the process of designing my program, I found the objectives to be a very useful checklist for features I needed to add. I believe they were well-formulated, covering the necessary components of the program comprehensively and providing an effective framework to base my testing off of. However, I also identified and implemented a number of useful extra features that weren’t detailed by the objectives. Below are a list of these extra features:

* The “Clear “Done”” button on the task interface is used to delete all tasks that have been completed in one go, to avoid having to delete them separately.
* The “Clear Past Events” button on the event interface is used to delete all past events in one go, to avoid having to delete them separately.
* The description widgets have a scrolling function to more easily edit different parts of the description as it is being written.
* The event attendee system allows users to either select individual accounts as attendees to an event or select entire teams as attendees. Additionally, only events that a user plans to attend will be shown on the calendar.
* The “Sort Through Backlog” button on the task interface allows users to sequentially sort through tasks in the backlog, allowing for efficient progression of these tasks.
* The “Stats” button in the toolbar provides some basic stats on the amount of work set and the amount of work completed, as well as telling the user how many events they have coming up.
* Tasks that have been assigned to you will appear light blue in the task interface.
* Many screens now include a “Cancel” button to allow you to return to the main interface.

Although these features aren’t necessary to the program's functionality (look at the statistics feature for example), some of these features are quite important for the effectiveness of my application (like the “Cancel” button or the event attendee system). It is these features—the ones that aren’t necessary but are important nonetheless—that should have been included in my objectives.

Despite this oversight, my testing gives me confidence that I have identified any and all gaps in my objectives, and overall, I think my objectives were a success, proving a useful tool in the subsequent planning, implementation, and testing of my project.

**Testing Reflection**

Throughout my testing, I determined that all of my objectives had been met with no need for further improvements. This is reflective of the approach I took to programming my application in which I thoroughly tested features as I implemented them, helping me to catch any issues before even starting my final testing.

Despite this, during my testing I did notice a flaw in my program, unrelated to any particular test. Events in the past weren’t automatically deleted. This wasn’t a problem in and of itself because these events could be deleted manually, but since there was no method to reliably and easily tell whether an event was in the past or not, it would likely require the team to delete the events as they passed; a solution which seemed unnecessarily tedious. As a result, I implemented the “Clear Past Events” button which would delete any past events upon being pressed and which I later validated in the “Additional Tests” section.

I think the above situation shows the importance of considering and trying to simulate real user interaction throughout testing, as it can not only help to locate bugs but it can highlight gaps in usability.

**The Evaluation Interview**

**The Interview With Mr. Jon Driver**

**1 - “What do you think of the application as a whole?”**

I think it looks great. I love the way it's organised and enjoyed testing the various features, all of which seemed like they’d be a great help.

**2 - “What aspects of the program do you think could be improved?”**

I noticed that the calendar was only able to show you four weeks worth of dates. I think it would be helpful for my team’s planning if they were able to view subsequent weeks.

**3 - “Did you find the interface intuitive and easy to navigate?”**

Very, I liked the use of tabs to navigate between events and tasks and thought the toolbar was well implemented.

**4 - “Did you encounter any bugs or issues in your use?”**

No.

**5 - “Did you find the application to be quick and responsive? Did you notice any performance issues?”**

Everything seemed to run relatively smoothly. I did notice that the scrolling looked a little bit flickery, but other than that, everything was great.

**6 - “Do you think this program is well-suited to the kind of work input you get?”**

Yes, I think the long scrollable description box provides plenty of space to enter information for any type of task and I think the way in which you add certain information as the task moves to the next stage will work well for our team.

**7 - “Do you think this solution is more suitable than other existing applications?”**

Yes, the program seems to have everything we need and is effectively tailored to both the structure of my team and the kind of work we do.

**8 - "Would this application actually save your team time and effort? Why or why not?"**

Definitely, the system for organising tasks seems very efficient and intuitive. The use of colours to show which tasks are a priority is simple but will definitely be a big help to my team. Our system at the moment is a mess so I’m sure this program will be an immense time saver.

**Insights From The Interview**

**1** - Overall, the program is well-organised and helpful showing that the approach taken throughout this project was successful and effective.

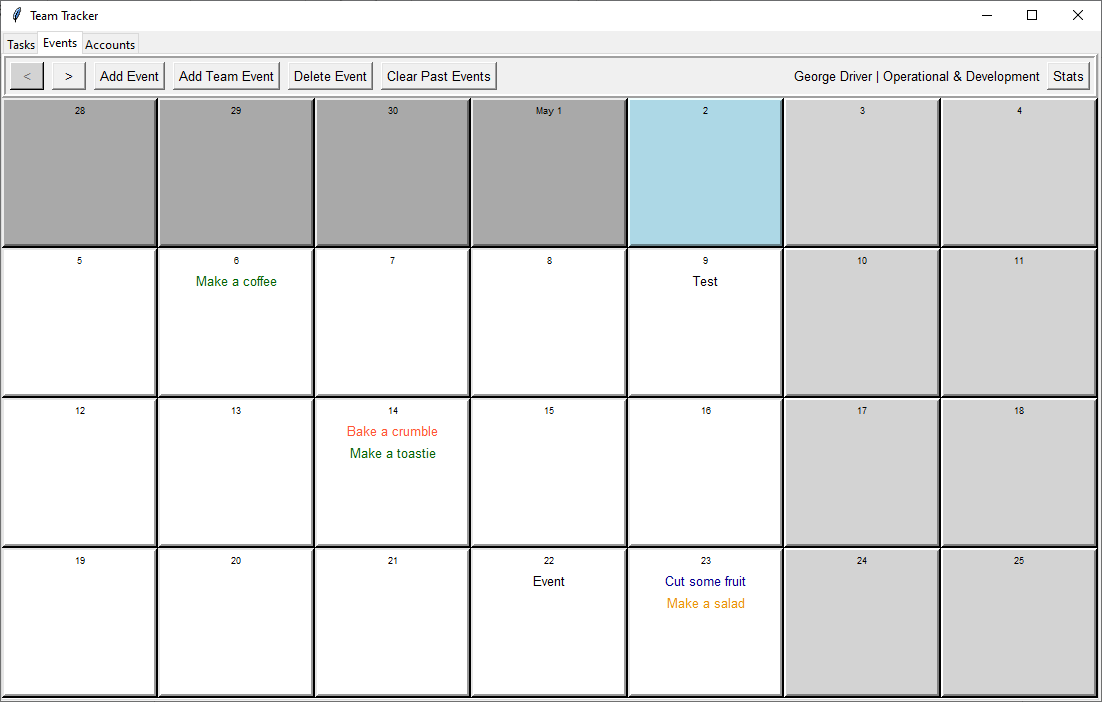
**2** - This answer brings up a very valid critique with the current calendar system. Originally, I’d assumed that viewing deadlines and events for the next four weeks would be enough. Although I believe this is the case for the vast majority of cases, it would be beneficial to see further ahead, especially for things like making sure planned events don’t clash and scheduling components of a long-term project.

To implement this, I added arrows in the toolbar of the events tab which allow you to cycle through four week chunks. The changed code required for this feature is below (shown in red).

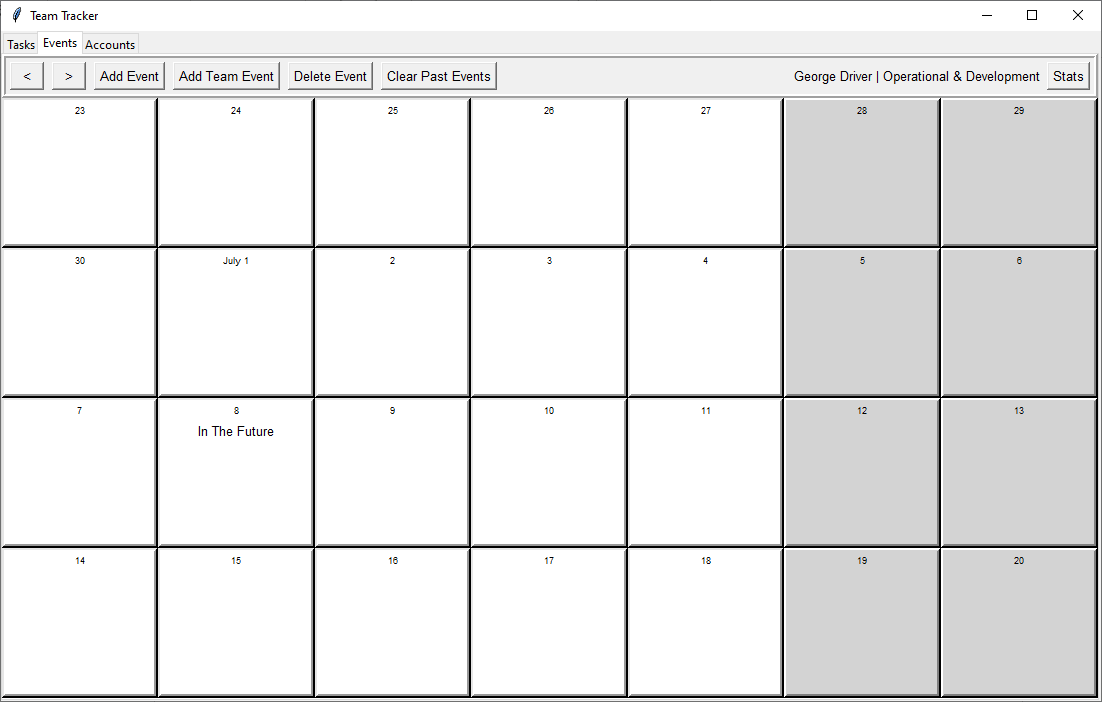
| **Changed parts of the EventInterface and Calendar classes from team\_tracker.py** |
| --- |
| class EventInterface(tk.Frame):  def \_\_init\_\_(self, master, username):  ...  def calendar(self, page=0):  Calendar(self, self.username, page)  def new\_event(self):  ...  def new\_team\_event(self):  ...  def remove\_event(self):  ...  class Calendar(tk.Frame):  def \_\_init\_\_(self, master, username, page):  super().\_\_init\_\_(master)  self.username = username  self.page = page  self.cells = []  self.fill\_frame()  self.pack(fill="both", expand=True)  def fill\_frame(self):  ## toolbar  self.toolbar = tk.Frame(self, relief="ridge", bd=4)  self.toolbar.pack(side="top", fill="x")  self.left\_arrow = tk.Button(  self.toolbar,  text="<",  font=FONT\_2,  command=lambda: self.change\_page(self.page - 1),  width=3,  )  self.left\_arrow.pack(side="left", padx=4, pady=4)  self.right\_arrow = tk.Button(  self.toolbar,  text=">",  font=FONT\_2,  command=lambda: self.change\_page(self.page + 1),  width=3,  )  self.right\_arrow.pack(side="left", padx=4, pady=4)  if self.page == 0:  self.left\_arrow.config(bg="lightgrey")  self.left\_arrow.config(state="disabled")  if self.page == 100:  self.right\_arrow.config(bg="lightgrey")  self.right\_arrow.config(state="disabled")  self.add\_event\_button = tk.Button(  self.toolbar, text="Add Event", font=FONT\_2, command=self.new\_event  )  self.add\_event\_button.pack(side="left", padx=4, pady=4)  self.add\_team\_event\_button = tk.Button(  self.toolbar,  text="Add Team Event",  font=FONT\_2,  command=self.new\_team\_event,  )  self.add\_team\_event\_button.pack(side="left", padx=4, pady=4)  self.delete\_event\_button = tk.Button(  self.toolbar,  text="Delete Event",  font=FONT\_2,  command=self.remove\_event,  )  self.delete\_event\_button.pack(side="left", padx=4, pady=4)  self.clear\_past\_events\_button = tk.Button(  self.toolbar,  text="Clear Past Events",  font=FONT\_2,  command=self.clear\_past\_events,  )  self.clear\_past\_events\_button.pack(side="left", padx=4, pady=4)  self.stats\_button = tk.Button(  self.toolbar,  text="Stats",  font=FONT\_2,  command=lambda: messagebox.showinfo(  "Stats", stats\_string(self.username)  ),  )  self.stats\_button.pack(side="right", padx=4, pady=4)  self.account\_info\_label = tk.Label(  self.toolbar, text=account\_info\_string(self.username), font=FONT\_2  )  self.account\_info\_label.pack(side="right")  ## calendar  self.calendar\_frame = tk.Frame(self)  self.calendar\_frame.pack(fill="both", expand=True)  self.calendar\_frame.grid\_rowconfigure(  tuple([n for n in range(4)]), weight=1, uniform="a"  )  self.calendar\_frame.grid\_columnconfigure(  tuple([n for n in range(7)]), weight=1, uniform="a"  )  for i in range(self.page \* 28, (self.page + 1) \* 28):  cell = tk.Frame(  self.calendar\_frame, relief="raised", bd=4, bg=get\_cell\_bg(i)  )  cell.grid(row=i // 7 - 4 \* self.page, column=i % 7, sticky="nsew")  cell\_day\_label = tk.Label(  cell, text=get\_cell\_day(i), font=FONT\_1, bg=get\_cell\_bg(i)  )  cell\_day\_label.pack()  cell\_list = [cell, cell\_day\_label, []]  for event in get\_events(get\_cell\_date\_string(i), self.username):  cell\_list[2].append(  tk.Label(cell, text=event, font=FONT\_2, bg=get\_cell\_bg(i))  )  cell\_list[2][-1].pack()  for deadline in get\_deadlines(  get\_cell\_date\_string(i), self.username  ):  cell\_list[2].append(  tk.Label(  cell,  text=deadline[0],  font=FONT\_2,  bg=get\_cell\_bg(i),  fg=IMPORTANCE\_COLOURS[  importance(get\_cell\_date\_string(i), deadline[1]) - 1  ],  )  )  cell\_list[2][-1].pack()  self.cells.append(cell\_list)  ## events  cell.bind(  "<Configure>",  lambda event: self.update\_wraplengths(event),  )  # having this out of the for loop ensures the binding is attached to the  # last cell only, so that wraplengths are only updated once per resize  # and they are updated all at once  def update\_wraplengths(self, event):  ...  def change\_page(self, newpage):  self.destroy()  self.master.calendar(page=newpage)  def new\_event(self):  ...  def new\_team\_event(self):  ...  def remove\_event(self):  ...  def clear\_past\_events(self):  ... |

Additionally, I edited the get\_cell\_day subroutine to return both the day and month (as opposed to just the day) on the first day of each month, helping users work out when they are as they navigate the calendar. The new subroutine is copied below.

| **The new get\_cell\_day subroutine from misc\_functions.py** |
| --- |
| def get\_cell\_day(i):  months = [  "January",  "February",  "March",  "April",  "May",  "June",  "July",  "August",  "September",  "October",  "November",  "December",  ]  date = dt.date.today() + dt.timedelta(days=i - dt.date.today().weekday())  return (  (months[date.month - 1] + " " + str(date.day))  if date.day == 1  else date.day  ) |



*A screenshot showing the new calendar navigation arrows, with the left arrow greyed-out as it goes into the past.*



*A screenshot of the calendar after pressing the right arrow twice, now showing an event in July.*

**3** - The layout of the interface is effective and intuitive.

**4** - Mr. Driver identified no further bugs, further demonstrating the rigor and completeness of the tests done in the testing phase.

**5** - The program was generally quick and responsive. The issue Mr. Driver rises about the scrolling feature seems to be an unavoidable consequence of using the Tkinter scrolling feature in the way that I have. Although I’m sure it is fixable through some other means, I suspect it would involve changing an unreasonable amount of the program.

**6** - The program has been effectively tailored to the work input that Mr. Driver mentioned in question 8 of the analysis interview.

**7** - This solution is more effective than current existing solutions due to it being tailored and specialised to the issues faced by Mr. Driver’s team.

**8** - This answer reaffirms the application's effectiveness as a solution to Mr. Driver’s problem, particularly noting the method I used to make priority tasks clear.

**Next Steps**

**Deployment**

Although the program functions well, it is not quite ready for use. Currently, the program has only been implemented and tested for use locally. For the program to be of use to Mr. Driver’s employees, each device at their office must have the program installed and each instance of the program should communicate with a single centralized database, like the one I’ve used to test the program locally.

Unfortunately, this level of implementation will add complexities which will need to be dealt with accordingly. For example, issues could arise if multiple people are attempting to edit the same entity simultaneously as they might overwrite each other causing a loss of crucial information. Additionally, if a change occurs in the database before the program reacts to it, the program may end up trying to edit non-existent entities, raising an error.

**Scaling Up**

Although there is no immediate need to scale up the use of this program, a need to support a greater number of users, tasks and/or events is foreseeable. Implementations like the merge sort or the hash table to check for username uniqueness have helped prepare the application for scalability, providing efficient algorithms which will become more and more essential as the user base grows. Additionally, features such as the “Clear “Done”” button, the “Clear Past Events”, and the “Sort Through Backlog” button help users deal with large numbers of events and tasks effectively.