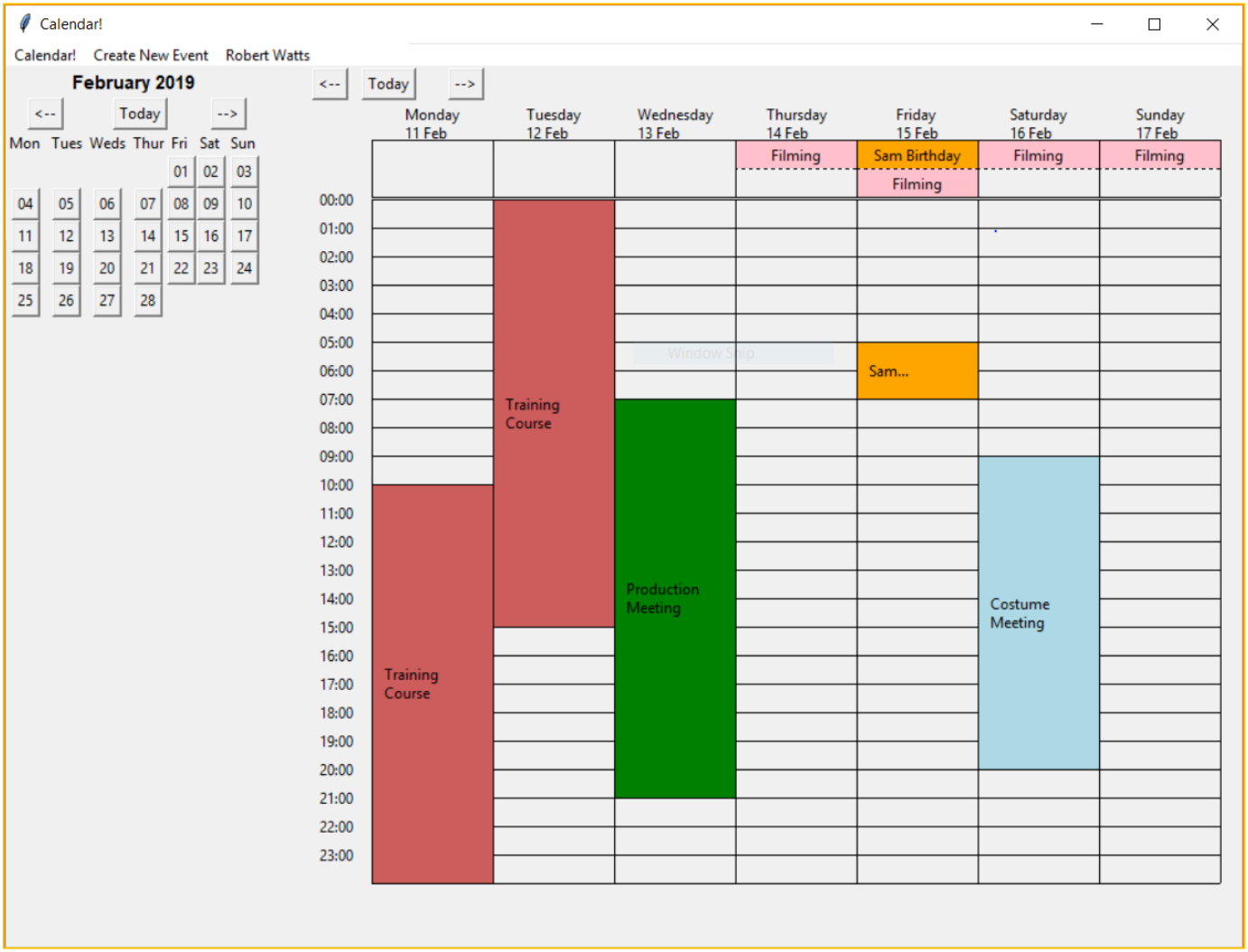
2019

Calendar

AN A-LEVEL COMPUTING PROJECT

ROBERT WATTS

Centre Number: 62105
Candidate Number:2702



Contents

Analysis

Design Brief

I am aiming to make a Calendar Program, which will allow people to manage their time, very similar to Google Calendar, or Outlook Calendar. This will involve allowing users to create, edit, and delete events. The user will then need to be able to view events by day, week or month, to see what events are coming up. The program should be simple and intuitive to use.

To complete this, I am going to need to design and create different forms, as well as having windows that are scalable depending on the user's screen size. I will need to create a mini calendar from which the user can select dates. I will also need a database for the program to read from and save data to.

Stake Holders

Everyone needs some way of structuring their life and events, and all calendars are focused at a particular group of people. The app will be on a computer, so it needs to be quick and easy to use the program and get the information required, as it will take time to turn on the computer to check, which will take time out of people's day. The program could easily be adapted to be on smartphone, which is how most users would interact with it. Because I am doing it on a computer not a smartphone, I am limiting my target market. This means that my program needs to be easily extendable to make it sync with a phone.

With this in mind, I am going to have my target audience as a business person who is aged between 20-30.

User Profile

A business person aged between 20 and 30 (Millennials) is someone who has used technology their entire life. All they have really known are computers and smart phones for if not their entire education, their entire working life. Paper calendars are something in the past. This group of people are interested in technology, and tend to keep up with modern trends. This means that my product will be competing with already established players in the industry. Business people more than most need a way to remember events, and doing this quickly and efficiently is important to these people as time is money.

I have selected Ryan Ellis to personify my target audience onto. He is an 18 year old student who is in my computing class. This is a little younger than my target market, however I will have regular contact with him, which will allow me to run idea's by him a lot quicker.

How The Problem Can Be Solved by Computational Methods?

For most people it makes sense to have their calendar on a computer. When people are out, they do not always have a paper calendar with them, but they are never far from a computer in the form of a phone or laptop. Below are some examples to explain how the computer fits within the problem to help me solve it.

Thinking Abstractly

In the calendar the user will be inputting real world information. A computer will not be able to handle everything about the real world, so my calendar program will have to use an abstraction of reality – simplifying the real world. Some of the simplifications I will make are:

- The user doesn't need their calendar down to seconds and milliseconds. For this reason, I am only going to have hours and minutes for events.
- The user will just see a calendar; they will not know if it is the Gregorian, Roman or Julian calendar. In reality, I am going to be using the Gregorian calendar in this project.
- I will not include bank holidays or religious holidays as business people may not need to know these. Holidays also vary depending on the country, so a business person will know what holidays are going on the country they are working in.

Thinking Ahead

Thinking ahead will help me plan what needs to be done during this project. I am going to use python as my programming language, because it is high level it means that it will be easier to code as the built in commands will go further. I am then going to use a built in module called TKinter to create the window that the code will be developed in. The reason for using TKinter is because of its cross platform support, meaning my program will work without much customising for Windows, Mac OS X, and Unix. This will help to increase the reach of my program.

The inputs of my program will be a keyboard and mouse. The user will have different buttons to modify, add or remove events. The outputs will be a window with all of the events placed on it in time order, depending on how the user requests to see it. Some calendars also have a sound output to remind the user of upcoming events, this is something that may be added to my calendar.

In the future I do not know how the program will be developed. As previously mentioned, most users will want their calendar on their phone, so I need the database to be made so that it can easily be hosted online. This means that the users events can be shared across platforms. The program could also be easily modified to read and modify a hosted database, rather than a local one. This will mean an app could easily sync with my software.

Thinking Logically

I am going to plan out my code. This will help me to have the most logical and efficient code possible.

I am also going to plan how the windows will look. This will allow me to make sure that the interface is usable quickly and easily. Currently, I am planning to have two main parts within the window.

The first part is an events viewer, which will display the calendar. There will need to be some sort of iteration, checking the databases, to make sure that there are no updates that need displaying. If there are, it will need to cause a branch and refresh this part of the window. For example, if a new event is created the viewer will need to display this.

The other part of the window is a navigator. This will need to have all the tools for a user to navigate through the software. The user will need to be able to navigate through the weeks, months and years. I could also have the arrow keys (left and right) to move along time. This would mean there will have to be a constant iteration checking to see if the keys have been pressed. There will also need to be buttons in this area to create new events. I could add the key 'n' to do this too.

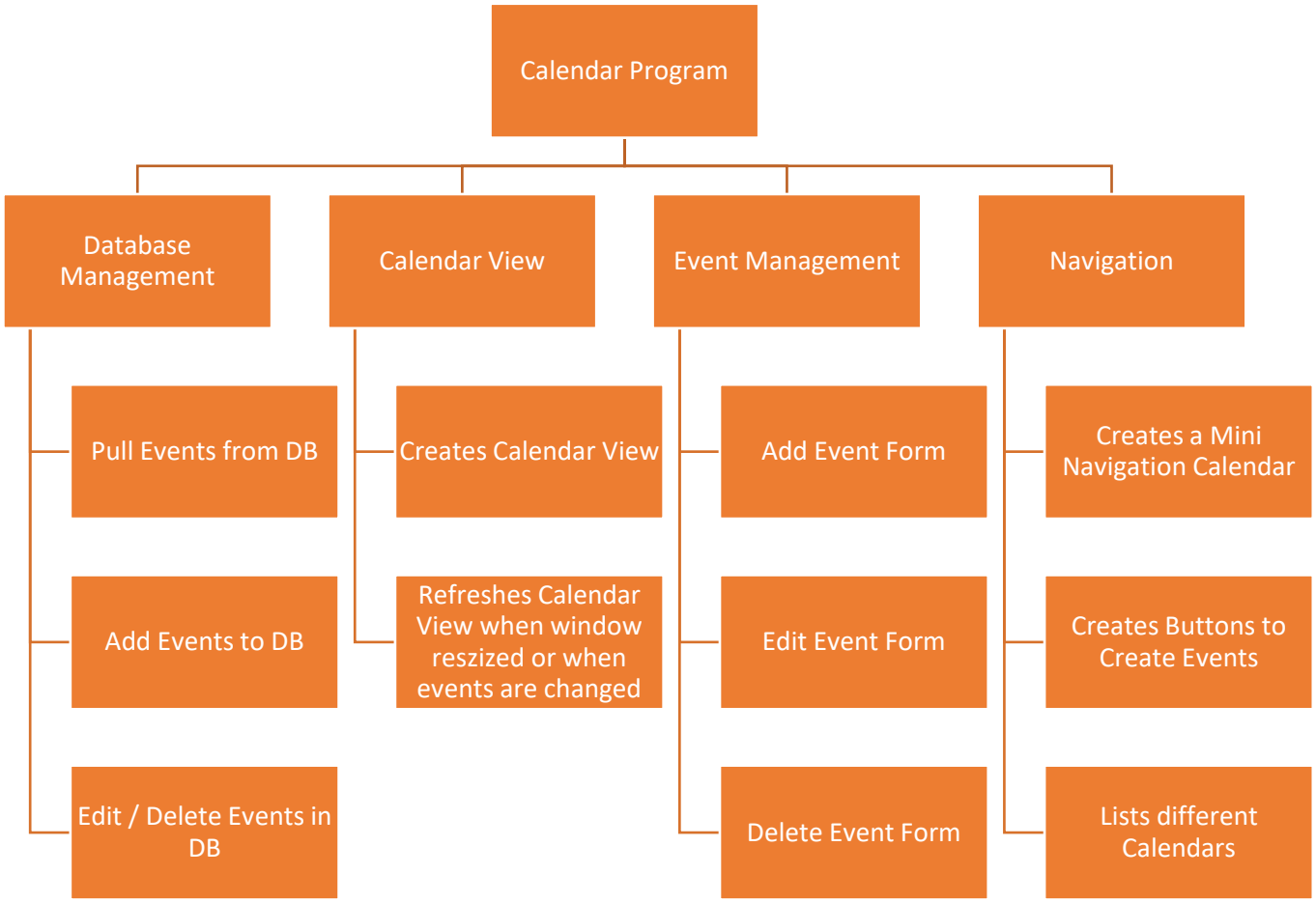
Thinking Concurrently

Thinking concurrently will help me to make the most efficient solution, by constantly thinking about how all the different parts of the programs will interrelate, and run simultaneously to produce a seamless program. For example, I am going to need to create a date picker for the program. If I build this by thinking concurrently, I will be able to reuse this in several different parts of the program. Another example of this is that the program will need be displaying different events while listening out for clicks of buttons or existing events.

Thinking Procedurally

Thinking procedurally will help to break down the problem to it's basic parts, to help me understand what needs to happen and how to achieve each part. This will help me to make the program more efficient as I already know how different parts will interrelate.

I have created a chart below to break down the problem:



By thinking procedurally, I have broken the problem down into four main parts, which if completed will provide a successful provide a basic program.

Database Management

This part will manage the database, this will need to have commands for each part of the database that will need to be modified. Each part will need to take an input, validate it, create an SQL statement depending on what needs doing, then execute it on the Database.

Calendar View

This part of the program will be used to select a way for the user to view all the events on the calendar. The view needs to be scaleable, so that if the user changes the size of there window, the size of the calendar changes. It will need to update whenever events are modified.

Event Management

The user will need a form to create or edit events. This will need a date selector and textboxes.

Navigation

These are for the user to interact with the calendar, allowing them to scroll to or find different dates quickly and easily. Included in this, the user should be able to show or hide calendars.

Thinking Legally

This application needs to be compliant with relevant laws. The first law I am going to look at is the data protection act of 1998. I have listed the 8 principles and how I am going to keep to them.

Principles	What I am going to do to abide by this part of the law
1. Personal data should be obtained and processed fairly and lawfully	The user will not be forced to create an account, they will have the option to use other services like Google Calendar if they do not want to give the program their information.
2. Personal data can be held only for specified and lawful purposes	The data will only be stored in the database and should not be sold or used without the users consent.
3. Personal data should be adequate, relevant and not excessive for the required purpose	I am only going to store certain information about a user. For example, as a calendar application I do not need to know the users bank details, so I will not ask.
4. Personal data should be accurate and kept up-to-date	The user will have the option to edit their account details.
5. Personal data should not be kept for longer than is necessary	The users details should be easy to remove from the storage method.

6. Data must be processed in accordance with the rights of the data subject	The user has the right to see there information, which they will be able to from within the program.
7. Appropriate security measures must be taken against unauthorised access	I could encrypt the database as well as have user control, so the user accessing the database does not have full read/write access on it. I could also hash the password.
8. Personal data cannot be transferred to countries outside the EU unless the country has similar legislation to the Data Protection Act	The data should be stored on the users computer. In the future if the database is hosted then it should hosted within the European Union.

So that the user complies with the computer misuses act I could create some sort of user licence agreement, so they know how and where the software can be used. I could use a premade one such as creative commons, which would allow the user to distribute the program as much as they like. This Licence can be seen here: https://creativecommons.org/licenses/by-nd/4.0/ .

The final law I need to comply by is the Copywrite and Patent Act of 1988. To comply with this law I need to make sure that I do not out right copy other pieces of software. I also need to make sure that I credit any external modules used in the code.

Conclusion

In this section, I have shown that a calendar is useful to have on a computer. I have shown way in which I intend to go about designing and implementing my program. In doing this I have given the problem some structure to help me solve it and help me remain legal while solving it.

I will know that I have solved the design brief correctly when the stake holder is able to quickly manage events within the calendar. This will include viewing, editing and creating events. Hopefully this program will make my stakeholders more efficient with their time, helping them to make more money.

Research

Research Plan

Once I have completed my research, I hope to be able to answer the following questions:

- What features are important in a calendar?
- What should the GUI look like?
- How should the program be controlled?
- What are the limitations of my design brief, and what can be done about this?
- What hardware and software should be used?

Initial Interview

Plan

This interview will be constructed so I know what to focus my research around. This will in turn give the direction my calendar will take. It will be conducted with Ryan Ellis.

How do you use a calendar?

- Do you use one for home, school, work, or something else?
- Should a digital calendar be on your phone or on the computer?
- Would you want to login to protect your calendar, or would you prefer that the calendar just shows you events?

What features do you like about existing calendars?

- What basic features should a calendar have?
- Should a calendar have lots of colour, or be fairly greyscale?
- Should you be able to have multiple calendars, that you can overlay?
- Are there any features that you believe existing digital calendars lack?

How should the calendar look and feel?

- Would you prefer a calendar that looks good, or is functional?
- Should the calendar make sounds when an event is about to start?
- Would you want all you all of your calendars overlaid, or keep them in separate views?
- Would you want visualisations with the calendar, for example if you put in that you have football practice, a picture of a football will appear on the calendar with the event?

Interview Script

How do you use a calendar?

- **Do you use one for home, school, work, or something else?**
  - I use a computer for home and school
- **Should a digital calendar be on your phone or on the computer?**
  - I prefer the phone, because I have that with me all the time, however I am on a computer enough that I would probably be fine with it on the computer
- **Would you want to login to protect your calendar, or would you prefer that the calendar just shows you events?**
  - I would want to have a login page protecting my data because I might have private events in my calendar that I do not want other people knowing about.

What features do you like about existing calendars?

- **What basic features should a calendar have?**
  - Create, Edit and view events. It is important to be able to see all the events I have in one day all at once.
- **Should a calendar have lots of colour, or be fairly greyscale?**
  - Colourful. Obviously.
- **Should you be able to have multiple calendars, that you can over lay?**
  - It can be useful to have multiple calendars as then I can separate out school and home.
- **Are there any features that you believe existing digital calendars lack?**
  - I use google calendar, and I cannot think of anything that could improve it

How should the calendar look and feel?

- **Would you prefer a calendar that looks good, or is functional?**
  - Function is more important than looks, that being said if it is not fluid, by that I mean easy to use, I won't use it.
- **Should the calendar make sounds when an event is about to start?**
  - I find that annoying.
- **Would you want all you all of your calendars overlaid, or keep them in separate views?**
  - Overlaid
- **Would you want visualisations with the calendar, for example if you put in that you have football practice, a picture of a football will appear on the calendar with the event?**
  - That could be nice

Analysis of Interview

My aim of the interview was to determine what the stake holder wants from the product. Generally, I now believe that function is more important that looks. However, Ryan said he preferred colour rather than grey scale. I think that this is important as it can make the calendar feel nicer to use.

Ryan said that security of the calendar was important. This will mean that there will need to be some sort of lock function to make sure that no one can access the calendar without permission. This will involve having a username and password for each user.

As the stakeholder is using the computer, I believe that the program should be controlled with a mouse and keyboard. This will keep it easy as this is how most computers are used.

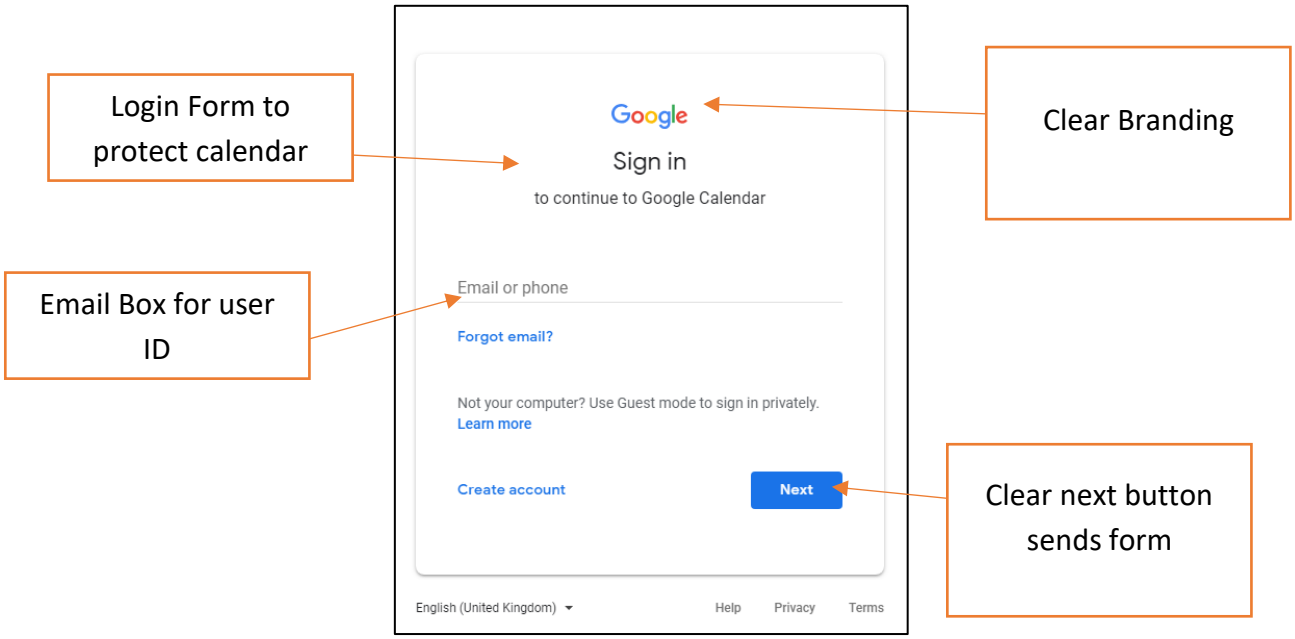
One limitation of my design brief is that that many users may wish to access their calendar away from home. This is something that Ryan said during the interview, and is a valid point. I am going to make the program easily extendable so that is can easily be turned into a program that can sync with a phone.

The interview was useful to get an idea of what to look for during the rest of my research.

Existing Product Evaluation: Google Calendar

Google Calendar is an existing calendar application available on web, Android, and IOS. Over 100 million people use it daily, with over 1.2 Billion people having google accounts. (https://www.calendar.com/google-calendar-guide/) It allows you to create calendars, then add, edit or modify those events. These events sync over the internet so that it is available on all devices. You can then share your calendars with anyone who needs it.

The first thing you do with the calendar is you have to login. This helps to protect your calendar from unauthorized people.



Once you log in, there you come to a main page as show below.

Easy navigation to Today.

Navigation buttons, to move week by week.

Events Scale to fit the page. This is important as you don't know what screen sizes are going to be used.

Google calendar is in a Web Browser meaning it can be accessed from any computer or phone easily.

Calendar thumbnail view for quick navigation between months. Speeds up use.

User Settings. This allows you to change password, and settings.

Greyed out after events done. This means you know when events are done.

Days along X axis. Much like in a paper calendar. This helps it to feel less like you are using a computer.

Different Calendars allows user to split events.

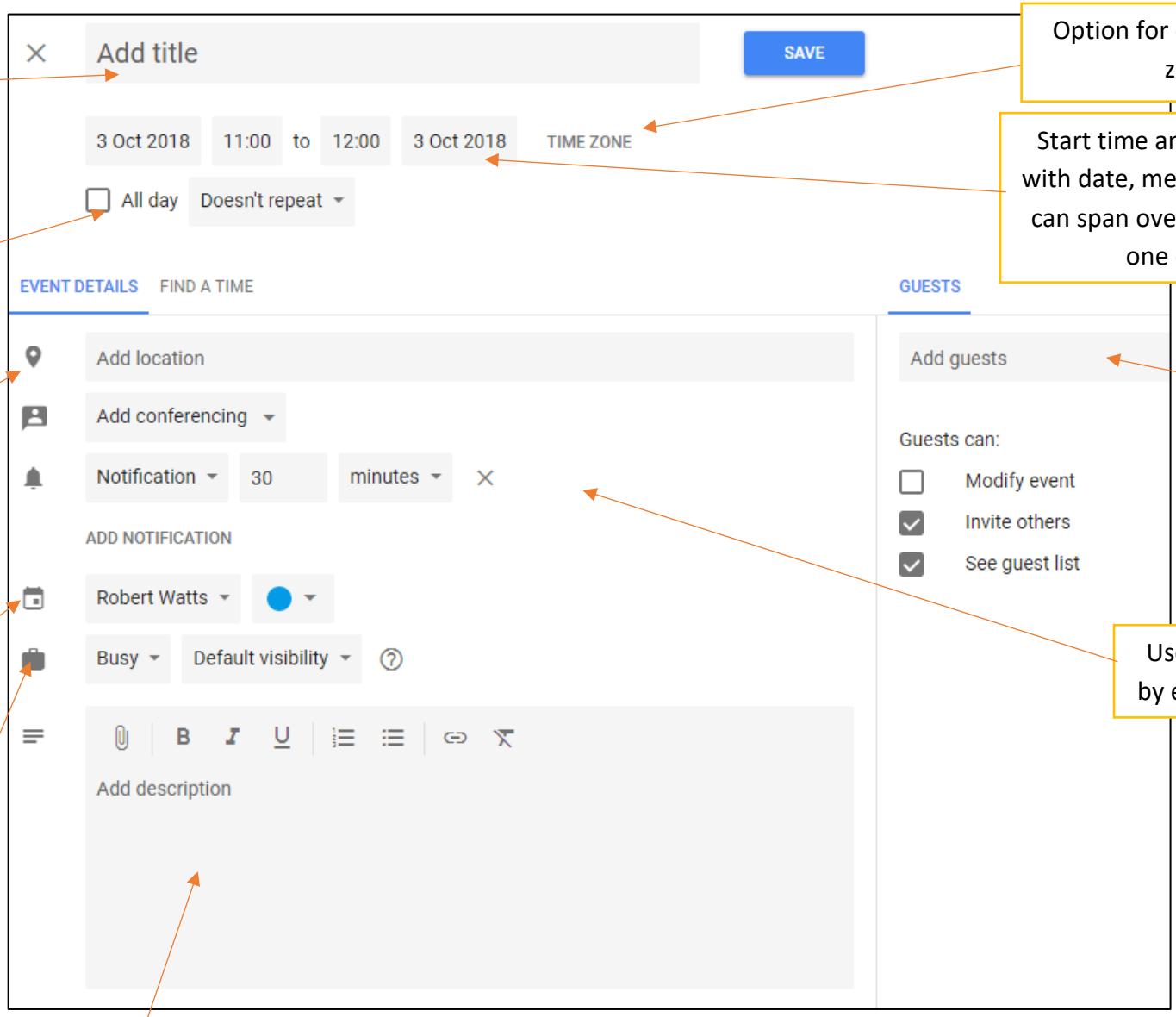
Makes a ping sound as reminders. These can be customised, for example you can change the sound, or how far before

Calendars can be shown or hidden depending on what user wants

Time along Y Axis

Create New Events button is clear. When clicked a new form comes up, allowing the user to create new events.

Events from different calendars are different colours



Title of event is bigger than the rest of the page.

Option for different time zone

Start time and end time with date, meaning events can span over more than one day

On the main page, all day events are put into different boxes. When this is checked, you cannot input a time, only days.

You can add locations of events, and it can then tell you how long it will take to get from your previous event to this one.

You can invite guests to an event. This sends them an email and adds it to their calendar.

User can get reminders by email and notification

You can select the colour of the event and which calendar it is put into. Different calendars have different default colours.

This can automatically tell other people if you are busy

Description Box

Candidate Name: Robert Watts Candidate Number: 2702 Centre Number: 62105

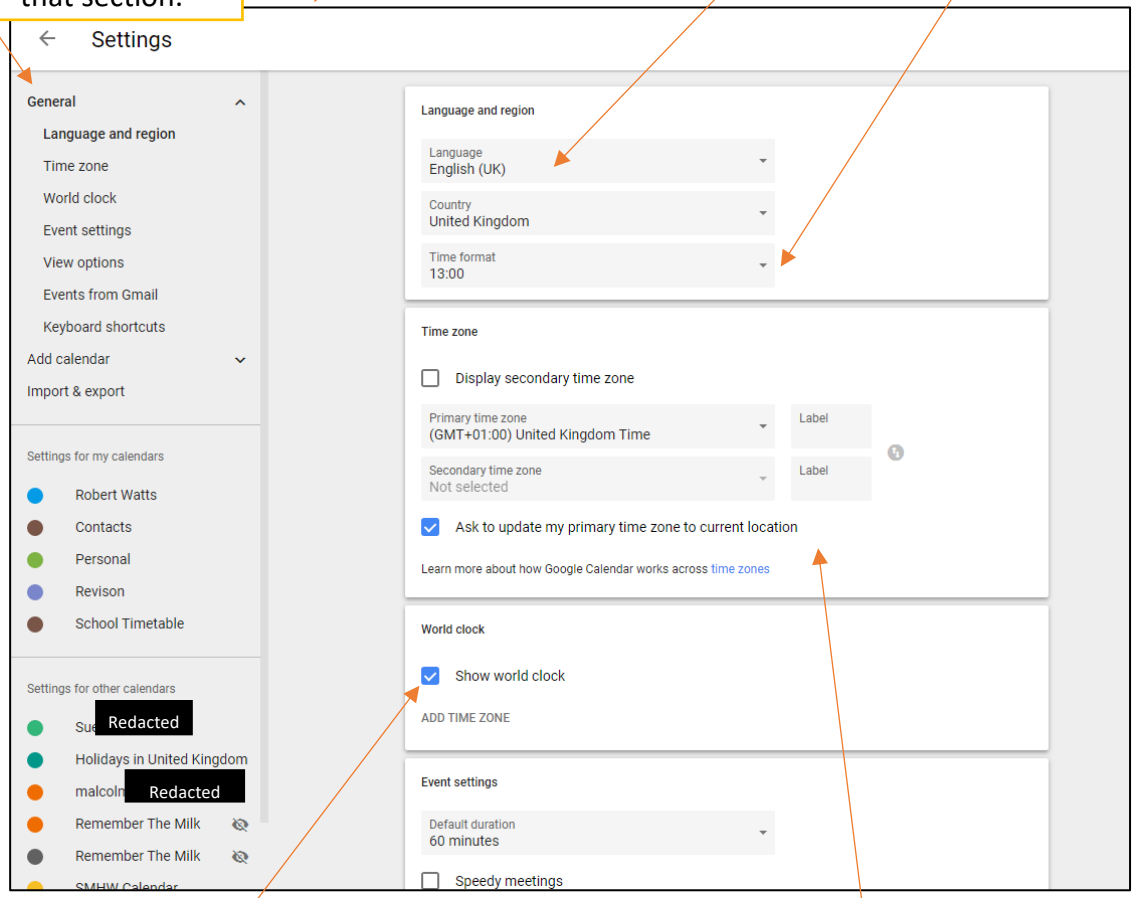
Easy to use menu, click and it brings the settings for that section.

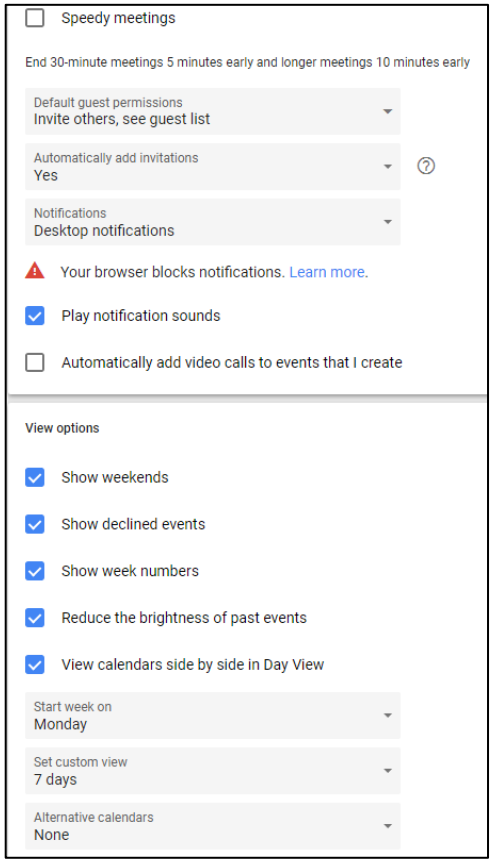
Settings page, to allow the user to customise the calendar.

User can choose language and location

Time Format is customisable

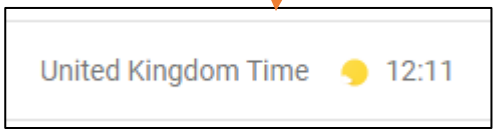
More Settings (scrolled down)



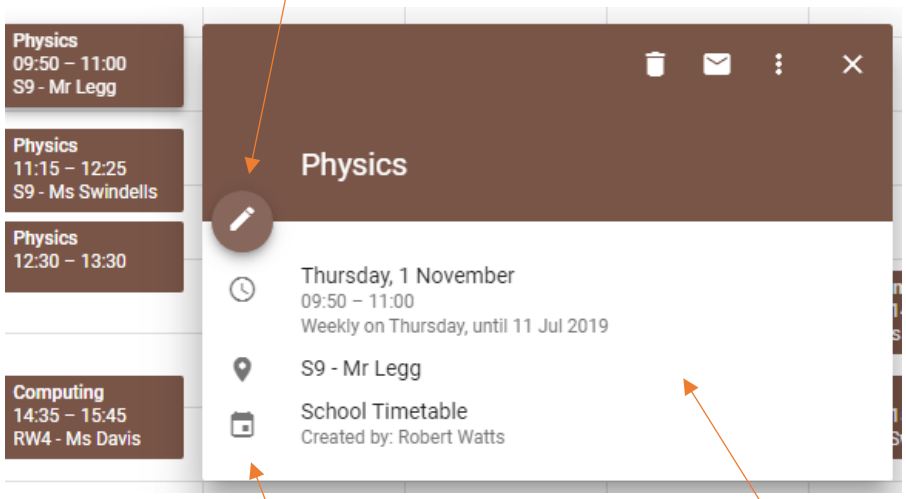


Displays clock and weather for a selected timezone.

User can select default time zone



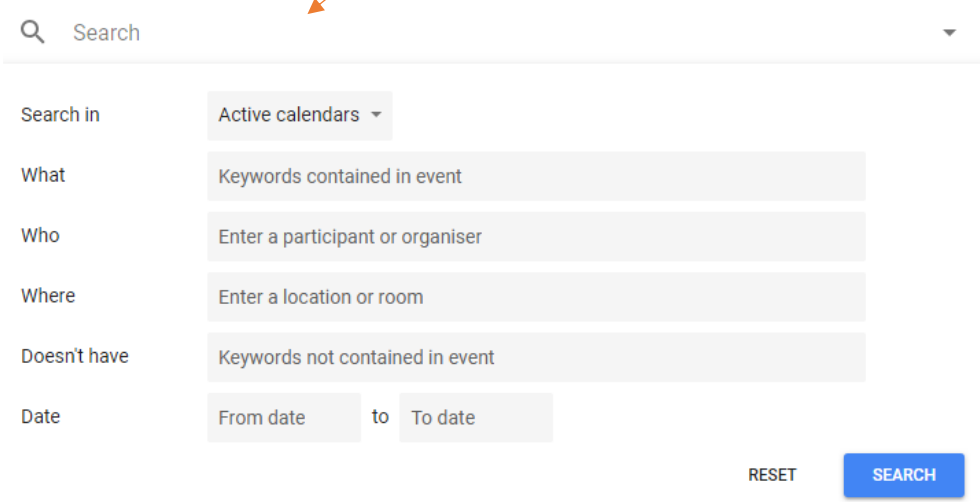
Icons make it easy to see what you are looking at



Icons make it easy to see what you are looking at

When you click on an event you get all the information about an event

Can search for events to make it easy to find things



Can search for events to make it easy to find things

How do the features of 'Google Calendar' effect the usability of the product?

Login

This helps to protect the users calendar. Without this, I do not believe that anyone would properly use the calendar because it is not secure.

Multiple Calendars

This helps the user to split their events into different categories, allowing you to easily see what events are for different things for example work or home. This helps the user to understand how much time they are spending at each.

Main View

The main view is the frame with all of the events on it. This helps the user to see at a glance if they are free at a particular point. This is scalable, meaning it can adjust to different sizes easily. This is useful as not every screen is the same size.

Menu and Controls

The calendar is customisable from within the menus. This helps the user make the calendar their own. The controls are very intuitive – for example the button to create a new event is big and clear. This allows you to get more information about an event quickly

New / Edit Events Form

This is quick, and you can fill as many of the options out as you need – the event doesn't even need a name! This helps to make the calendar quick to use and edit, which is important because you don't need to spend hours putting events in.

Reminders

To remind user of events it can make sound, send an email and send a notification to your phone. This helps to keep the user on track with what they should be doing.

Overall looking at google calendar has given me a good idea as to what a good calendar should look like.

Features that will be used in the product

Second Interview Plan

From this interview I intend to get an idea of more specific features that will be required in my calendar. I am going to split my interview into several sections:

User Interface

- Where on the page would you like the new event button to be?
- When you click on an event would you rather have a popup or a completely new page display the information about that event?
- Other than a title and description, is there any other information that would be useful to store about events?
- What colours should events be?
- Should events that are marked “All Day” go at the top of that day, or be shown through the whole day?
- There will be a button that says “Today” which will take the calendar to that days date. Where on the page should that be?

Usability

- How many calendars would you like?
- What size should the window be?
- What day of the week should be first – Sunday or Monday?
- Should there be a mini calendar for quick navigation?
- Should the Date of the day you are looking at be big or should it be smaller, with the day name being more important?
- Should you be able to show and hide calendars?

Personal Data

- To protect your calendar, you will need to login. When creating an account would you mind giving your….
    - Name
    - Email
    - Age
    - Gender
    - A Picture of you
        - If no to the picture, could we use your email to pull a picture of you from services like Gravatar?
- Would you prefer a username to obscure your identity?

- Would it be useful to be able to output all of your events as an ICAL file to import into other pieces of software?

Interview Script

User Interface

- **Where on the page would you like the new event button to be?**
  - Top Right is where it is on the calendar I currently use, so it would be nice to have it there
- **When you click on an event would you rather have a popup or a completely new page display the information about that event?**
  - Popup
- **Other than a title and description, is there any other information that would be useful to store about events?**
  - Location
- **What colours should events be?**
  - You should have an option of 4 colours
- **Should events that are marked "All Day" go at the top of that day, or be shown through the whole day?**
  - At the top
- **There will be a button that says "Today" which will take the calendar to that days date. Where on the page should that be?**
  - Above the calendar, it doesn't really matter as long as it is clear

Usability

- **How many calendars would you like?**
  - If I can choose the colour of an events one colour would be fine as I could change the colour depending on what I want.
- **What size should the window be?**
  - big
- **What day of the week should be first – Sunday or Monday?**
  - Monday
- **Should there be a mini calendar for quick navigation?**
  - Yes
- **Should the Date of the day you are looking at be big or should it be smaller, with the day name being more important?**
  - Date
- **Should you be able to show and hide calendars?**
  - Yes

Personal Data

- **To protect your calendar, you will need to login. When creating an account would you mind giving your….**
    - **Name**
        - Yes
    - **Email**
        - Yes
    - **Age**
        - Yes
    - **Gender**
        - No
    - **A Picture of you**
        - No
        - **If no, could we use your email to pull a picture of you from services like Gravatar?**
            - It would be a bit weird
- **Would you prefer a username to obscure your identity?**
    - Yes
- **Would it be useful to be able to output all of your events as an ICAL file to import into other pieces of software?**
    - Its not something I would use on a daily basis, but it might be useful to back up my calendar this way.

Review of the Interview

Through this interview the some of main features and layouts of the calendar have been established.

User Interface

- I want a menu at the top of the page where I can have buttons such as new event.
- When clicking on an event it will display a popup with all the information about an event.
- In the database the user will be able to store the title, description, and location of an event, as well as the date and time. It will also store what colour the user wants the event to appear on the calendar.
- There will be a separate section of the window to display events marked as all day.
- There will be easy controls to navigate through the calendar, at the top of the page.

Usability

- It would be good to be able to split events into calendars, however this may be a time issue, as I don't have long to create the product.
- The window should be dynamic.
- The calendar will show Monday as the first day of the week.
- Making sure that you know which week you are looking at is important to the usability of the product.

- You should be able to hide multiple calendars.

Personal Data

- I have learnt that some data is off limits. From my interview I learnt that Ryan does not want certain information to be stored about him. This understandable, so I have to be careful about what data I store about a user.
- I am going to store the name, username, email, and password about a user.
- The information about a user's events is also sensitive. I do not need to know exact information about an event. For this reason, I am not going to force a user to input things such as location or description about an event.

Features of Proposed Solution

Below are the main proposed features for the game.

Login	This is to protect the users calendar to give the user peace of mind that their events are safe.
Create Account	This is to allow a new user to create an account.
Database	This means the project can easily be extended to an app. It also helps to keep the data more secure. Having a database also makes the project scalable to more users.
Menu Bar	This gives the user control options such as adding events and editing user accounts.
New Events Page	This allows the user to create new events. It will also allow them to add extra information about the event such as the name and location of the event.
Edit events Page	This allows the user to edit events. It will also allow them to edit extra information about the event such as the name and location of the event.
User Account	This will allow the user to edit their details. This helps to keep the account data up to date and relevant.
Navigation Buttons	Allows the user to move forward a week or go to the previous from the current view
Navigation Calendar	A small calendar that will allow you to easily skip to a date far in the future.
Event Viewer	The user will be able to see the whole week in one easy view, allowing them to see what is going on that week
Event Details	A user will be able to click on an event to display its details, and have options to edit or delete it.
Background change colour	The background should change colour when the user clicks on an event to the colour of that event.
Mouse Controls	This is the most appropriate control as it will allow a user to easily click on the window, allowing easy interaction of the program.
Keyboard	This makes it really easy for the user to add more details about an event.

I will not have long to produce this project, about 4 months and I am doing 2 other subjects along with computer science. Because of this I have provided a second set of features that could be completed if the original completed and I have time left over.

Loading Page	So the user knows when things are loading.
Scalable to different page sizes	Not all users have a screen of a fixed size so making it scalable will allow users of all devices to see their calendar.

Multiple calendars per user	A user can create, edit and delete calendars. This allows the user to split their events and just display certain calendars with the events they want to look at
Pictures with events	The calendar detects key words in the event title and adds little pictures with them.
Recurring events	A lot of events happen more than once, and this will allow the user to automatically put events into calendars based on rules
ICAL Support	Allows the user to put calendars from other places into the calendar

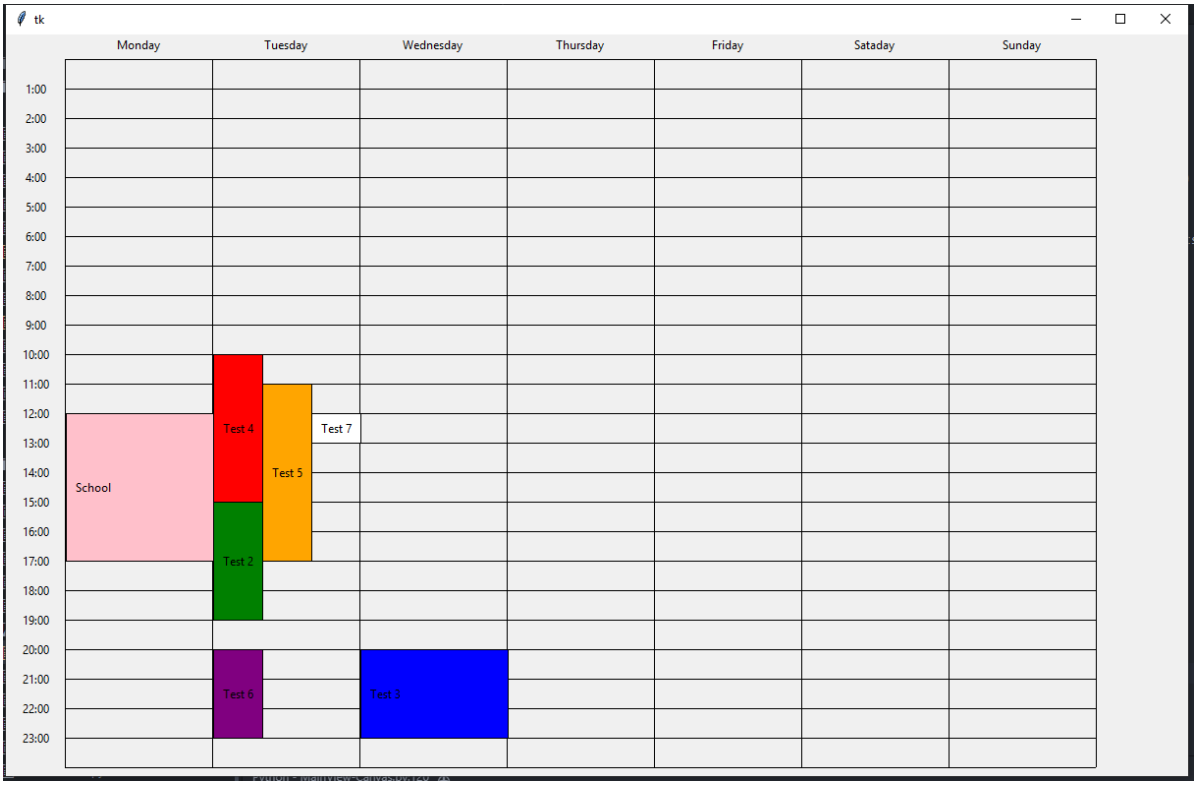
Prototype

There are a few things that I am not sure about when it comes to creating the project. Before I start creating the pseudocode, I want to test out a few things. The first thing that I tested was the view that will show all of the events. I created the following code to get do this.

```
from tkinter import *

#Define Grid Height
WIDTH = 150
HEIGHT = 30
GRID_START_CORDINATE_X = 60
GRID_START_CORDINATE_Y = 25

#Create Window
window = Tk()
can = Canvas(window, height=(HEIGHT * 25), width=(WIDTH * 8))

#Create Vertical Grid Lines
verticalLength = (HEIGHT * 24 )
x = GRID_START_CORDINATE_X
y = GRID_START_CORDINATE_Y
for line in range(0, 8):
    can.create_line(x,y,x,y+verticalLength)
    x += WIDTH

#Add Horizontal Grid Lines
horizontalLength = (WIDTH * 7)
x = GRID_START_CORDINATE_X
y = GRID_START_CORDINATE_Y
for line in range(0, 25):
    can.create_line(x, y, x+horizontalLength, y)
    y +=  HEIGHT

#Add Day Labels
x = GRID_START_CORDINATE_X
y = 10
days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Sataday", "Sunday"]
for day in days:
    can.create_window(x+(WIDTH/2),y, width=WIDTH, window=Label(text=day))
    x += WIDTH

#Add Time Labels
x = 30
y = GRID_START_CORDINATE_Y + HEIGHT

for time in range(1,24):
    text=str(time) + ":00"
    can.create_window(x, y, window=Label(text=text))
    y +=HEIGHT




events = [ {"title": "School", "day": 1, "start": 1700, "finish": 1200},
           {"title": "Test 2", "day": 2, "start": 1500, "finish": 1900},
           {"title": "Test 3", "day": 3, "start": 2000, "finish": 2300},
           {"title": "Test 4", "day": 2, "start": 1000, "finish": 1500},
           {"title": "Test 5", "day": 2, "start": 1100, "finish": 1700},
           {"title": "Test 6", "day": 2, "start": 2000, "finish": 2300},
           {"title": "Test 7", "day": 2, "start": 1200, "finish": 1300},
]


#Create Colours
color = ["Pink", "Green", "Blue", "Red", "Orange", "purple", "white"]
colorID = 0

#Work out Max Depth for each Day
dayMaxEventWidth = [1,1,1,1,1,1,1]
for eventID in range(0,len(events)):
    items = list(range(0,len(events)))
    items.remove(eventID)
    itemsThatOverlap = [eventID]
    for eventToCheckID in items:
        if events[eventToCheckID]["day"] == events[eventID]["day"]:
```

```
            if events[eventID]["start"]  <=  events[eventToCheckID]["start"] < events[eventID]["finish"] :
                itemsThatOverlap.append(eventToCheckID)
    if len(itemsThatOverlap) > 0:
          day = events[eventID]["day"] - 1
          if dayMaxEventWidth[day] < len(itemsThatOverlap):
              dayMaxEventWidth[day] = len(itemsThatOverlap)

#Create Matrix Grid
MATRIX = []
for dayNum in range(0,7):
    Day = []
    for MinNum in range(0,1440):
        min = []
        for positionVector in range(0,dayMaxEventWidth[dayNum]):
            min.append(None)
        Day.append(min)
    MATRIX.append(Day)


#Layout
for eventID in range(0,len(events)):
    day = events[eventID]["day"] - 1
    width = dayMaxEventWidth[day]

    eventStartDigets = [str(i) for i in str(events[eventID]["start"])]
    eventStart = (int(eventStartDigets[0] + eventStartDigets[1]) * 60) + int(eventStartDigets[2] + eventStartDigets[3])

    eventFinishDigets = [str(i) for i in str(events[eventID]["finish"])]
    eventFinish = (int(eventFinishDigets[0] + eventFinishDigets[1]) * 60) + int(eventFinishDigets[2] + eventFinishDigets[3])
    #print(eventStart, eventFinish)
    column = 0
    stop = False
    while stop == False:
        spaceFree  = True
        for min in range(eventStart, eventFinish):
            #print(min)
            if MATRIX[day][min][column] != None:
                spaceFree = False
        if spaceFree == True:
            for min in range(eventStart, eventFinish):
                MATRIX[day][min][column] = eventID

            x1 = GRID_START_CORDINATE_X + ((events[eventID]["day"]-1)*WIDTH) + ((WIDTH / width)*column +1)
            y1 = GRID_START_CORDINATE_Y + ((events[eventID]["start"]/100)*HEIGHT)
            x2 = x1 + (WIDTH / width)
            y2 = GRID_START_CORDINATE_Y + ((events[eventID]["finish"]/100)*HEIGHT)
            can.create_rectangle(x1, y1, x2, y2, fill=color[colorID])
            can.create_text((x1+10, 0.5*(y2-y1) + y1), text=events[eventID]["title"], anchor="w", width=(x2-x1-15))
            colorID += 1
            print(colorID, eventID)
            stop = True
        column +=1

can.pack()


window.mainloop()
```

I tested this and got the following output:

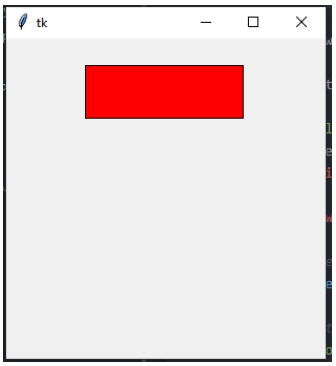


This works, which shows that I am able to produce the events on a grid. While doing this I struggled with getting the events to scale. This took me quite a lot of time. The easiest way to do it that I found was have an array with every minute in a day. For every time an event is covers that minute a time is incremented by one. I can then run through and see the maximum number along the time of that event. That allows me to calculate the width that each event needs to be. When I create this for real, I am going to turn it into a class with different functions.

The next thing that I want to prototype is clicking on the canvas. This is so that I can click on an event to show its details. I created the following code to test this. I used this website as research: https://effbot.org/tkinterbook/tkinter-events-and-bindings.htm

```python
from tkinter import Tk, Canvas

window = Tk()

#Create Canvas
c = Canvas(window, width=300, height=300)

def clicked(event,*args):
    #Click Event
    print("You clicked play!")
    print(dir(event))
    id = event.widget.find_closest(event.x, event.y)
    print(id)
    print(event.widget.gettags(id[0]))

#Create Rectangle
button = c.create_rectangle(75, 25, 225, 75, fill="red",tags="button 7")

#Bind click event
c.tag_bind("button","<Button-1>", clicked)

c.pack()

window.mainloop()

```

I tested this and go the following screen displayed:



When I clicked on the red square I got the following output:

```
You clicked play!
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le_
(1,)
('button', '7', 'current')
```

This meant that I could register click events and it would allow me to click on the events in the event viewer. I am happy with the prototypes that I have produce. I hope to use these to develop my solution.

Hardware and Software Requirements

Hardware Requirements

Hardware	Justification
Monitor	To display the calendar to the user
Mouse	To allow the user to interact with the program, by clicking buttons
Speakers	If the user wants notifications of when events are about to start then a speaker would be required to play these
5MB of secondary storage	The program will be very small so will not take up much storage.
500MB of RAM	Tkinter and python can run fine on computers such as the 'raspberry pi' which have very small amounts of RAM. This program will be a bit more RAM intensive, so I have increased it to 500MB

Software Requirements

Software	Justification
Python	This is the programming language.
Tkinter Module	This is used for the GUI
Datetime Module	This is used to do maths on dates and time with the timestamp
Time	This is used to get todays date and time
If Windows is Used:	
C	Python passes all commands to C
Tk Widgets	C calls this module to create each of the widgets that will be displayed on the screen.
Tk	implement the final mapping of widgets and pass them to xlib to be displayed
Xlib	Xlib library to draw graphics on the screen
If Mac is used	
C	Python passes all commands to C
Aqua Cocoa Tk	There are 3 variants for Tk on macOS. Any of them will work for what I am doing. C will pass the Tk commands to one these to then display what I need.
Aqua Carbon Tk	
X11 Tk	

Success Criteria

No.	Requirements	Justification	Completed?	Reference
1	Login Screen	To protect the users events from unauthorised access		• Google Calendar • Interview 1
2	When the user click login it checks that they have access from a database	This stops unauthorized access to the calendar. By using a database it make easy to change data about the user.		• Google Calendar
3	Create Account	A new user will need to create an account		• Google Calendar
4	Checks user does not already have an account	This is to stop duplication of data. This also helps to keep data relevant about a user		• Data Protection Act
5	If an account already exists then an error message will be shown	This is so the user knows that there is a problem		• Google Calendar
6	A mini calendar for quick navigation	To make it really easy for a user to find the dates that they want.		• Google Calendar
7	A view to see all the events that week	That allows the user to easily see what the is happening in their life that week		• Google Calendar
8	The week's events should be split into days and hours in the view on the main page	This is to easily see when events start		• Google Calendar
9	When viewing events on the main page, the user should see the colour of the event and the name.	This helps the user to easily see different types of events with colour, and can see basic information about that event from the name.		• Google Calendar • Interview 1
10	If the event name is too long to display on the main page, then it should cut the name so that it fits	This helps to keep the name presentable because you aren't trying to fit really long text in a small area.		• Google Calendar
11	When an event is clicked on it should display information about it	More details should be hidden initially to stop clutter. When clicking on it should display the info about that event		• Google Calendar

12	When the user clicks on an event the colour of that event should become the background	This helps to keep the project colourful and exciting. It also helps the user easily identify the event they are viewing		• Interview 1
13	A menu bar at the top detailing different options	Navigation should be easy, keeping everything at the top is where the user will expect it to be.		• Interview 2 • Interview 1
14	An all-day event is shown at the top of that day	So the user can see at a glance what is going on all day.		• Google Calendar
15	Navigation Buttons at top of page	To make it easy to navigate between weeks		• Google Calendar
16	A “Today” Button to skip to today’s date	Quick way to find the today, so you can see what you should be doing now		• Interview 2
17	Create event page	A form is needed to input data about new events.		• Google Calendar
18	An event must have a name and start date at a minimum, otherwise an error message is shown.	This is to prevent events ending before they start, and so that every event has a title.		• Google Calendar
19	Delete events	The user needs to be able to delete events		• Google Calendar
20	Edit Events page	A form is needed to change data about existing events.		• Google Calendar
21	When an event is edited it should go through the same validation as when creating an event	This is to prevent events ending before they start, and so that every event has a title.		• Google Calendar
22	A database to store user data, events etc	This allows the program to be expandable. The database could be linked to other programs like a phone app in the future.		• Google Calendar
23	The personal data about a user is email, name, username and password	I don’t need more data; this keeps me compliant with the Data Protection Act.		• Interview 2 • Data protection act
24	A user can change the information stored about them	The Data Protection Act says data should be kept up to date and relevant.		• Data Protection Act • Interview 2
25	When a user changes the information about them if	This is so that two different users cannot		• Data Protection Act

	goes through the same validation as when they create a new account	have the same name and email		• Interview 2
26	Store title, description, and location of an event. As well as start and end times	This is the data that is needed in the form.		• Interview 2 • Interview 1
27	User can change colour of event	This helps the user to easily see different events with colours		• Interview 1
28	There are at least 5 colours that a user can choose from for an event	This helps the user to split their calendar into different colour to easily see what is going on.		• Interview 2 • Google Calendar
29	The colour of an event defaults to the colour of the calendar	This means that a calendar can have a colour which all events default to		• Google Calendar
30	An event can be set to all day	Some events do not have times but you still need reminding		• Google Calendar • Interview 2
31	The title of the window should say “Calendar !”	So the user knows what the window is and the window purpose is clear to the user		

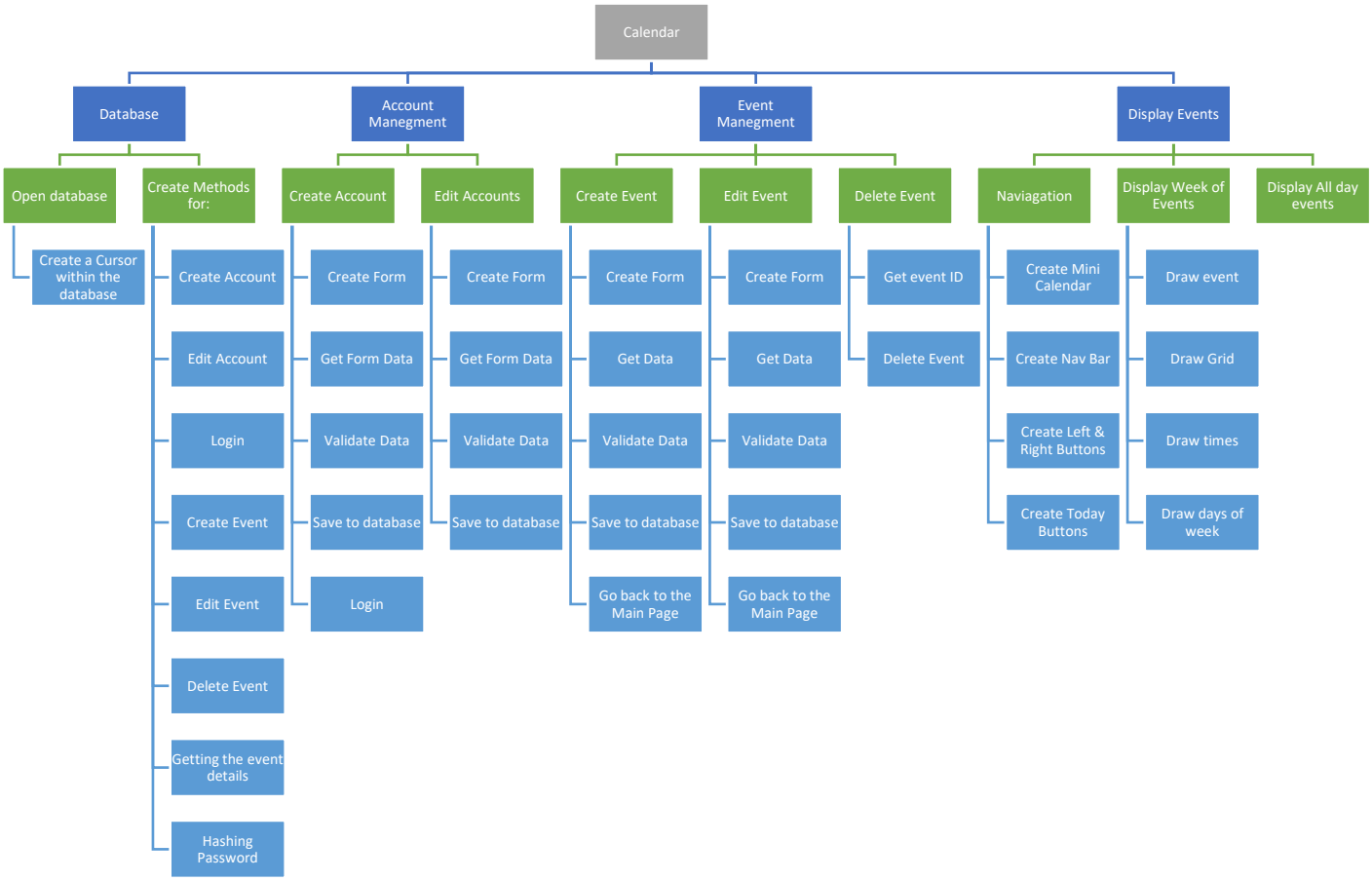
The following success criteria are for if I complete the others early and have time left over.

No.	Requirements	Justification	Completed?	Reference
32	Loading Page	So the user knows the page is loading		• Interview 2
33	Scalable Forms	All of the forms adjust their size to the size of the window that they are in		• Interview 2
34	Scalable Main Page	The main page will be scalable		• Interview 2
35	Scalable Event Viewer	The event viewer will change width and height. The height can be changed by adding a scroll bar, and the width can be changed by reducing the number of days shown		• Interview 2
36	Create New Calendars	Creates a form for the user to create a new calendar.		• Interview 2 • Google Calendar
37	Edit Calendars	Allows the user to change the name and default colour of a calendar		• Interview 2 • Google Calendar

38	Delete Calendars	Allows the user to remove a calendar and all events associated with it		• Interview 2 • Google Calendar
39	Pictures to do with events	Add a small picture when viewing event details from looking at event names		• Interview 2 • Google Calendar
40	ICAL Output	Output all of the users events in an ICAL file for them to upload elsewhere		• Google Calendar
41	ICAL Import	Allow the user to import an ICAL file from another calendar application.		• Google Calendar

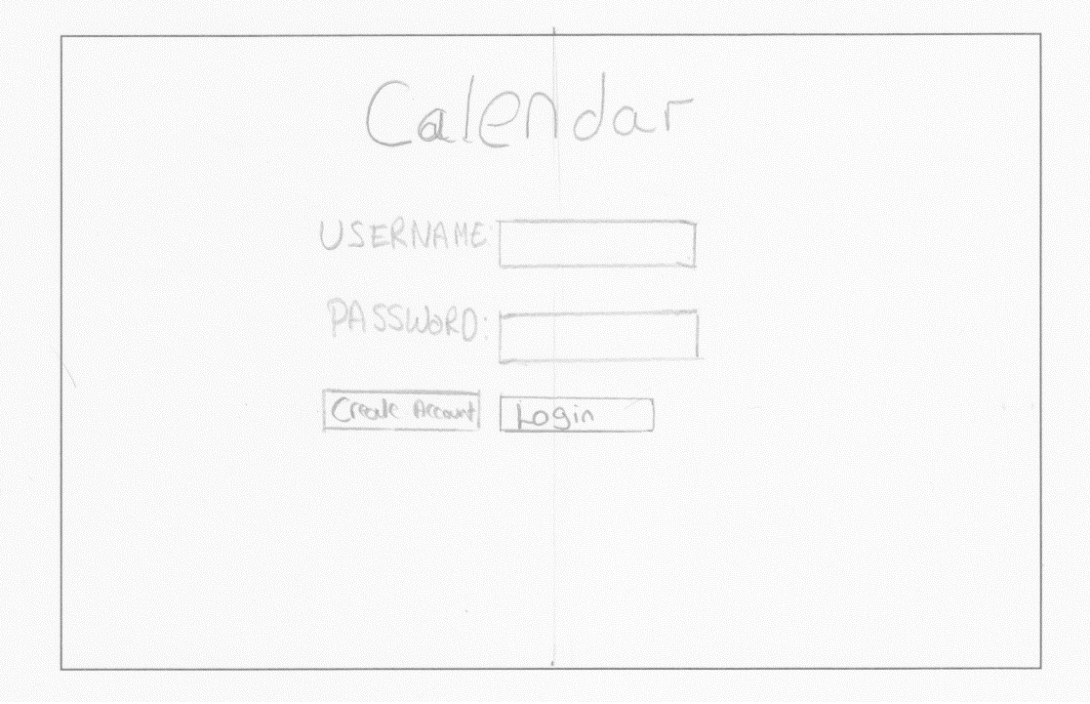
Design

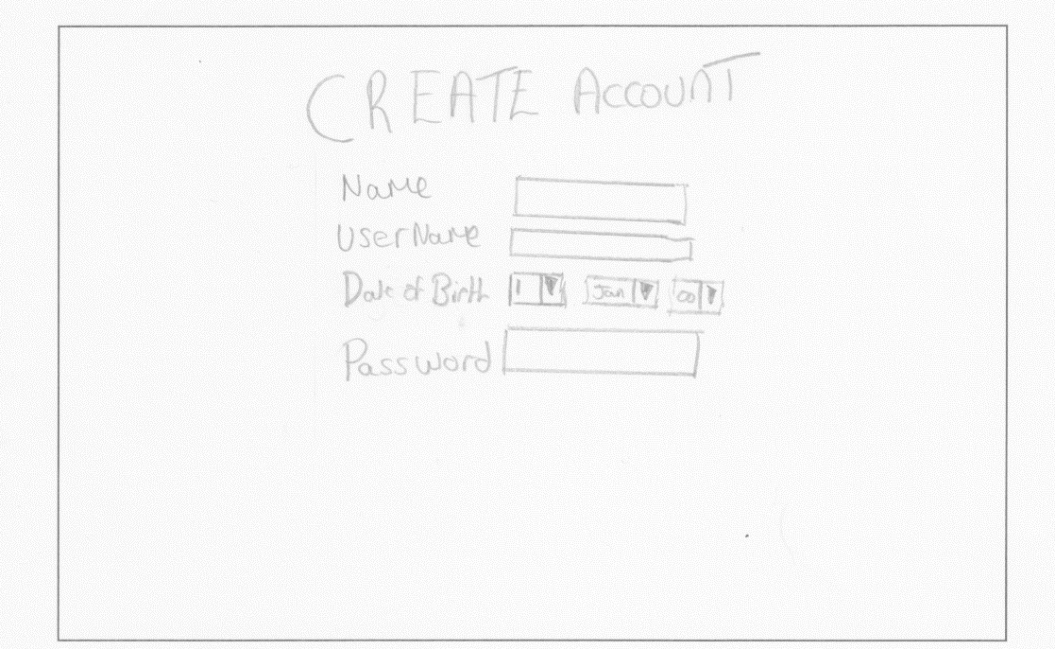
Systems Diagrams



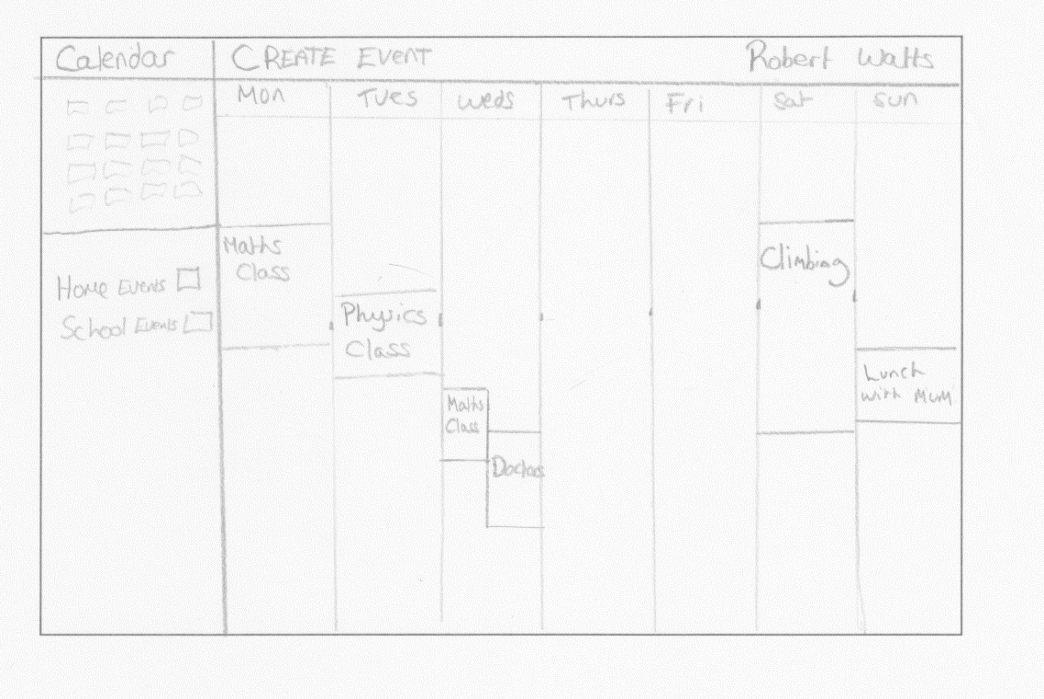
GUI Diagrams

I have created some GUI Diagrams. This will has helped me to take the success criteria and turn it into what the product should look like.

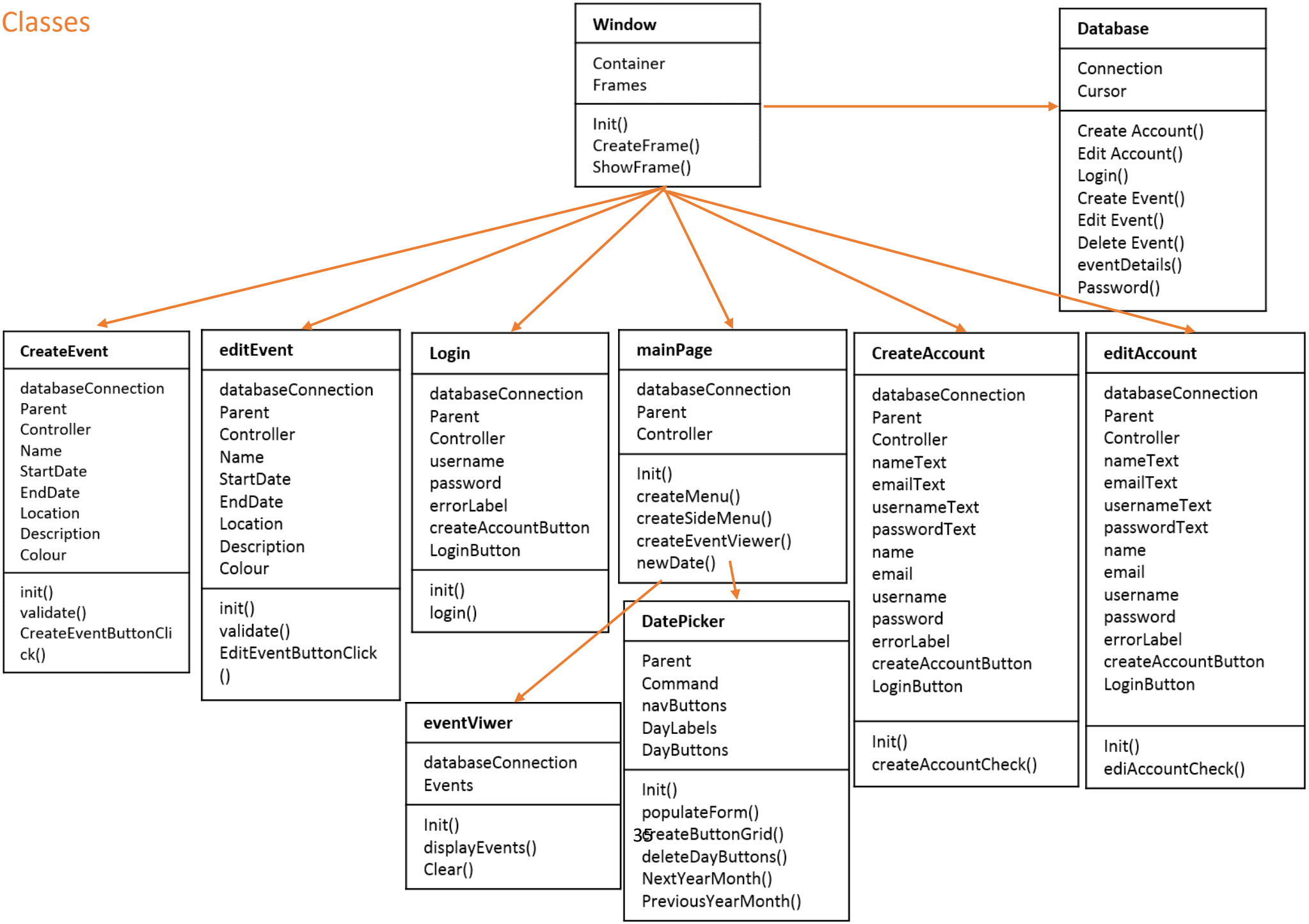




Since creating the create account page I have since realised that the fields I am storing are wrong. As per the success criteria the form elements should be email, age, name, username and password.


Calendar
CREATE EVENT
Robert Watts
Mon
Tues
Weds
Thurs
Fri
Sat
Sun
Home Events
School Events
Maths Class
Physics Class
Maths Class
Doctor
Climbing
Lunch with Mum

Classes


Window
Container
Frames
Init()
CreateFrame()
ShowFrame()
Database
Connection
Cursor
Create Account()
Edit Account()
Login()
Create Event()
Edit Event()
Delete Event()
eventDetails()
Password()
CreateEvent
databaseConnection
Parent
Controller
Name
StartDate
EndDate
Location
Description
Colour
init()
validate()
CreateEventButtonClick()
editEvent
databaseConnection
Parent
Controller
Name
StartDate
EndDate
Location
Description
Colour
init()
validate()
EditEventButtonClick()
Login
databaseConnection
Parent
Controller
username
password
errorLabel
createAccountButton
LoginButton
init()
login()
mainPage
databaseConnection
Parent
Controller
Init()
createMenu()
createSideMenu()
createEventViewer()
newDate()
CreateAccount
databaseConnection
Parent
Controller
nameText
emailText
usernameText
passwordText
name
email
username
password
errorLabel
createAccountButton
LoginButton
Init()
createAccountCheck()
editAccount
databaseConnection
Parent
Controller
nameText
emailText
usernameText
passwordText
name
email
username
password
errorLabel
createAccountButton
LoginButton
Init()
ediAccountCheck()
eventViwer
databaseConnection
Events
Init()
displayEvents()
Clear()
DatePicker
Parent
Command
navButtons
DayLabels
DayButtons
Init()
populateForm()
CreateButtonGrid()
deleteDayButtons()
NextYearMonth()
PreviousYearMonth()

Algorithms

In this section is my pseudocode. This is not all of my planning, however this is enough to get the basic functionality working. Once this is completed I will plan the next steps.

Class - Window

```
INCLUDE tkinter

CLASS Window:

    FUNCTION __init__ (self, *args, **kwargs):
        TKINTER INITALISE
        self.container = TKINTER FRAME

        self.frames = {}

    END FUNCTION

    FUCNTION createFrame(self, newFrame)
      frame = newFrame(self.container, self)
      self.frames[newFrame] = frame
      frame.TKINTER GRID (row=0, column=0, sticky="nsew")

    END FUNCTION

    FUNCTION showFrame(self, FrametoShow):
        frame = self.frames[cont]
        frame.TKINTER RAISE()

END CLASS
```

Class – Login

```
INCLUDE tkinter

INCLUDE database


CLASS login (TYPE = TKINTER FRAME):

    FUNCTION __init__ (self, parent, controller):

         #Create  labels
         usernameText = TKINTER LABEL (text="Username")
         passwordText = TKINTER LABEL (text="Password")

         self.username = TKINTER ENTRY

         #Create password entry
         self.password = TKINTER ENTRY
         self.password.TKINTER CONFIG (show="*")

         #Create error message label
         self.errorLabel = TKINTER LABEL
```

```
        #Create Login Button + Create account button
        createAccountButton = TKINTER BUTTON ( text="Create Account")

        LoginButton = TKINTER BUTTON ( text="Login", command = self.Login)

    END FUNCTION

    FUNCTION Login:
        usernameHolder = TKINTER GET TEXT FROM ENTRY self.username
        passwordHolder = TKINTER GET TEXT FROM ENTRY self.password

        IF DATABASE.checkLogin(usernameHolder,passwordHolder) EQUALS TRUE:
            self.controller.show_frame(Loading Page)

        ELSE:
            self.errorLabel.TEXT = "Username or password inccorect!"

    END FUNCTION
END CLASS
```

Class - Create Account

```
INCLUDE tkinter
INCLUDE database

CLASS createAccount (TYPE = TKINTER FRAME):

    FUNCTION __init__ (self, parent, controller):

        #Create  labels
        nameText = TKINTER LABEL (text="Name")
        emailText = TKINTER LABEL (text="Email")
        usernameText = TKINTER LABEL (text="Username")
        passwordText = TKINTER LABEL (text="Password")

        self.name = TKINTER ENTRY
        self.email = TKINTER ENTRY
        self.username = TKINTER ENTRY


        #Create password entry
        self.password = TKINTER ENTRY
        self.password.TKINTER CONFIG (show="*")

        #Create error message label
        self.errorLabel = TKINTER LABEL

        #Create Login Button + Create account button
        createAccountButton = TKINTER BUTTON ( text="Create Account", command =
self.createAccountCheck)

        LoginButton = TKINTER BUTTON ( text="Login")

    END FUNCTION

    FUNCTION createAccountCheck(self):

        nameHolder = TKINTER GET TEXT FROM ENTRY self.name
```

```
        emailHolder = TKINTER GET TEXT FROM ENTRY self.email
        usernameHolder = TKINTER GET TEXT FROM ENTRY self.username
        passwordHolder = TKINTER GET TEXT FROM ENTRY self.password

        result = DATABASE.createAccount(nameHolder, emailHolder,
usernameHolder, passwordHolder)

        IF result EQUALS TRUE:
            self.controller.show_frame(Loading Page)

        ELSE:
            self.errorLabel.TEXT = result

    END FUNCTION
END CLASS
```

Class – Main Page

```
INCLUDE tkinter
INCLUDE database
INCLUDE DatePicker
INCLUDE EventViewer

CLASS mainPage (TYPE = TKINTER FRAME):

    FUNCTION __init__ (self, parent, controller):
        self.createMenu()

    END FUNCTION

    FUNCTION createMenu(self):
        menubar = TKINTER MENU
        menubar TKINTER MENU ADD TEXT "Calendar"
        menubar TKINTER MENU ADD BUTTON "Create New Event"
        menubar TKINTER MENU ADD BUTTON "Settings"
    END FUNCTION

    FUNCTION createSideMenu(self):
        self.dateWheel = DATEPICKER
    END FUNCTION

    FUNCTION createEventViewer(self):
        self.eventViewer = EVENTVIEWER
    END FUNCTION

    FUNCTION newDate(self, day, month, year):
        dayOfWeek = DAY OF WEEK IN NUM
        firstDayToShow = day-dayOfWeek

        events = []
        FOR x IN RANGE firstDayToShow TO firstDayToShow + 7:
            events APPEND DATABASE getEvents(day, month, year)
        END FOR LOOP
```

```
        self.eventViewer UPDATE WITH events

    END FUNCTION

 END Class
```

Class – Loading Page

```
INCLUDE tkinter


CLASS loadingPage (TYPE = TKINTER FRAME):
    FUNCTION __init__ (self, parent, controller):
        TKINTER FRAME INITALISE

        loadingImageLocation = IMAGE LOCATION

        imageHolder = TKINTER PHOTOIMAGE (file = loadingImageLocation)

        image = TKINTER LABEL (image = imageHolder)

        image.TKINTER PACK

    END FUNCTION

END CLASS
```

Class – Date Picker

```
INCLUDE tkinter, clanedar, datetime

CLASS DatePicker (tpye = TKINTER Frame):
    FUNCTION __init__ (self, parent, command):

        self.command = command

        TKINTER INITALISE FRAME
        self.populateForm()
        self.navButtonClick("t")

    END FUNCTION

    FUNCTION populateForm(self):
        self.title = TKINTER LABEL

        self.navButtons = []

        self.navButtons APPEND  TKINTER Button (text="<--"
command=self.navButtonClick("l"))
        self.navButtons APPEND  TKINTER Button (text="-->"
command=self.navButtonClick("r"))
        self.navButtons APPEND  TKINTER Button (text="Today"
command=self.navButtonClick("t"))
```

```
        days = ["Mon ", "Tues", "Weds", "Thur", "Fri ", "Sat ", "Sun "]
        self.DayLabels = []

        FOR EACH day IN days:
            self.DayLabels  APPEND TKINTER LABEL(text=day)

    END FUNCTION

    FUNCTION createButtonGrid(self, month, year):
        self.DayButtons = []

        # Number of days in month
        monthDetails = CALENDAR monthrange(year, month)

        FOR x IN range(0, monthDetails[NUMBER OF DAYS]):
          self.DayButtons APPEND TKINTER Button

        END FOR

    END FUNCTION

    FUNCTION deleteDayButtons(self):

        FOR x IN self.DayButtons:
              x.destroy()

    END FUNCTION

    FUNCTION navButtonClick(self,command):
        IF command EQUALS "l":
                self.Year, self.Month = self.PreviousYearMonth(self.Year,
self.Month)
        ELSE IF command EQUALS "r":
            self.Year, self.Month, = self.NextYearMonth(self.Year, self.Month)
        ELSE IF command EQUALS "t":
            now = DATETIME TIME NOW
            self.Year = now.CURRENT YEAR
            self.Month = now.CURRENT MONTH

        self.DisplayMonth(self.Year,self.Month)

    END FUNCTION

    FUNCTION NextYearMonth(self, year, month):
        IF month EQUALS 12:
            RETURN year + 1, 1
        ELSE:
            RETURN year, month+1
    END FUNCTION

    FUNCTION PreviousYearMonth(self, year, month):
        IF month EQUALS 1:
            RETURN year - 1, 12
        ELSE:
            RETURN year, month-1
    END FUNCTION
```

Key Variables and Data Structures

Method	Name	Data Type	Explanation	Justification
Window				
Procedure	createFrame	N/A	This is used to add a frame to the 'frames' variable.	This stops direct access to the variable
Procedure	showFrame	N/A	This hides the frame that is currently displayed and shows a new one	This keeps the program from constantly deleting and re-creating a new container, which is better for the operating system
Variable	container	Object	This it the Tkinter Window	This is required for the program to run. It needs to be passed to all the frames that I display.
Variable	frames	Dictionary	This will hold all of the 'pages' that the program can go to.	This makes it easy to go find the pages because they can have a text name because it is a dictionary.
Login				
Procedure	Login	N/A	Validates form data then checks against database	Need to make sure that the user is authenticated
Variable	username	String	The username that the user has entered	Allows easy access of the username when checking against the database.
Variable	password	String	The password the user has entered	Allows easy access of the password when checking against the database.
Create Account				
Procedure	createAccountCheck	N/A	This validates the data from the form and then passes it to the database to be added	All the data needs to be valid before letting a user entering it into a database. This then also logs the user into their new account.
Variable	username	String	The username that the user has entered	Allows easy access of the username when checking validating the data.
Variable	password	String	The password that the user has entered	Allows easy access of the password when checking validating the data.
Variable	email	String	The email that the user has entered	Allows easy access of the email when checking validating the data.
Variable	name	String	The name that the user has entered	Allows easy access of the name when checking validating the data.

Main Page				
Procedure	createMenu	N/A	Creates the menu along the top	This allows easy access of certain functions. By having it as a procedure allows it to be easily extended in the future if that is required.
Procedure	createSideMenu	N/A	Creates the menu on the side	By having it as a procedure allows it to be easily extended in the future if that is required. For instance if I have multiple calendars per user I can easily add things to this menu.
Procedure	createEventViewer	N/A	Creates the event viewer object	By having this as a separate procedure allows a future programmer to expand it easily
Procedure	newDate	N/A	Takes an input of day, month and year and changes the view to that week	Allows any part of the program to easily change the date that is being displayed.
Variable	database	Object	The database class	This allows me to get and pass information to the database without each function running different SQL statements, they are all within the database class.
Date Picker				
Procedure	populateForm	N/A	This creates the navigation buttons and the labels for each day at the top	These are needed to make it easy to use, and to label everything. By having it as a separate function makes it easy to add features in the future to this part of the program.
Procedure	navButtonClick	N/A	This is called when one of the navigation buttons are clicked	By having one function for all three navigation buttons it makes it easy to see where all the code for dealing with these presses is. I also have to have a function to call as this is what the buttons require.
Function	NextYearMonth	N/A	This takes an input of year and month, and calculate the next month.	This is required because I cannot just add 1 to the month for the next one, I may have to add one to the year, and then loop back to 1 for the month. This function handles this.
Function	PreviousYearMonth	N/A	This takes an input of year and month, and calculates the previous month.	This is required because I cannot just subtract 1 from the month for the previous one, I may have to subtract one from the year, and

				then loop back to 12 for the month. This function handles this.
Procedure	deleteDayButtons	N/A	This deletes all of the buttons for the month that is currently being displayed	This is so that when a new month is needed to be displayed, I can clear the current one.
Procedure	createButtonGrid	N/A	Create a button for every day within month with the correct date on each button	This is so that I can display a months worth of dates, with clickable buttons so that I can get an output from the class when the buttons are pressed.
Variable	navButtons	List	This is a list of the navigation buttons on the form	This is so that I can access all the navigation easily buttons if I need to.
Variable	DayLabels	List	This is a list of all the labels for the days	This is so that I can access all the labels easily if I need to.
Variable	DayButtons	List	This is a list of all the 30 or so individual buttons for each day	This is used so that I can loop through and delete all of the buttons when the user changes the month that is currently being displayed.
Variable	title	String	This is the title at the top so that the user know which month is being displayed.	This needs to be accessible from different functions because every time a new month is displayed this needs to change.
Variable	command	Object	This is the function that is called when a user presses on a day	This is so that there can be an output from the class, and date picker can affect different classes and functions.
Variable	Month	Integer	This is the month that is currently being displayed	So that it can be called upon when changing date and outputting a selected date.
Variable	Year	Integer	This is the year that is currently being displayed	So that it can be called upon when changing date, and outputting a selected date.
Event Viewer				
Procedure	createGrid	N/A	Draws all of the lines on the canvas	This makes it easy to add extra lines and when looking down the file it is easy to see where all the lines are drawn
Procedure	createLabels	N/A	Draws all of the labels – times and days – on the canvas	This makes it easy to add extra labels and when looking down the file it is easy to see where all the labels are drawn

Procedure	displayDate	N/A	Displays a date that is passed to it	This runs all the code dealing with this.
Procedure	clearCanvas	N/A	Clears the canvas and resets all variables	So that the canvas can be cleared and reset when changing week.
Variable	Height	Integer	Hold the height of the event viewer	Makes the class extendable because I can easily change the height of the canvas, meaning if the window size changes I could change this
Variable	Width	Integer	Hold the width of the event viewer	Makes the class extendable because I can easily change the width of the canvas, meaning if the window size changes I could change this
Variable	Labels	List	Holds all of the labels displayed on the canvas	Makes it really easy to go through and delete all of them
Variable	events	List	A list of all the events to display on the calendar	I can then easily run through every event and display it. It also allows me to easily get the details about an event without needing to go back to the database
Create Event				
Procedure	createEvent	N/A	When the user click a create event button it will run this function	This needs to validate the data and then run add the event to the database
Procedure	clearForm	N/A	Clears the form	This makes it easy to reset the form elements and all of their associated variables.
Variable	Name	Object	The name of the event that the user has entered	Allows easy access of the name when checking and validating the data.
Variable	location	Object	The location of the event that the user has entered	Allows easy access of the location when adding to the database.
Variable	allDay	Object	Holds whether or not the event is all day and will be displayed at the top of the page	Allows easy access of whether the event is all day when adding to the database.
Variable	colour	Object	The colour of the event that the user has entered	Allows easy access of the colour when adding to the database.

Variable	description	Object	The description of the event that the user has entered	Allows easy access of the description when adding to the database.
Variable	startTime	List	The start time of the event that the user has entered	Allows easy access of the start time when checking validating the data.
Variable	endTime	List	The end time of the event that the user has entered	Allows easy access of the end time when checking validating the data.
Database				
Procedure	CreateAccount	N/A	Takes the information the user provides, validates it, and adds the data to the database	I need to make sure that all the user information I require is there and validated, so I have this function to do this and add it to the database.
Procedure	EditAccount	N/A	Edits a user's details in the database by user ID	This makes sure that all of the user data is validated and that it the information is Up To Date in the database.
Procedure	Login	N/A	Checks the users details against the database to make sure they are a valid user. Returns the users ID	I need to make sure that the username and password is valid to make sure that the user details and events are secure.
Procedure	CreateEvent	N/A	Takes all of the information for a new event and adds it to the database	This creates a new event in the database. It takes input of all of the info that is required for the event and then it handles all the SQL. It also validates the information about the event.
Procedure	EditEvent	N/A	Takes all of the information for an existing event and changes it in the database	This edits a particular event from in the database. It takes input of all of the info that is required for the event and then it handles all the SQL. It also validates the information about the event.
Procedure	DeleteEvent	N/A	Takes an event ID and deletes the event with that ID	This deletes a particular event from the database. This handles all the SQL and cursor etc.
Procedure	EventDetails	N/A	Takes an event ID and gets the info about it.	This gets all of the details about a particular event from the database. This handles all of the SQL and cursor etc.
File	DatabaseFile	.db	Holds the database	This is the file that holds the whole database, and is required.

Variable	Connection	Object	Opens the database file	Handles the database file. Helps to keep the database consistent by locking it when in use.
Variable	Cursor	Object	Runs commands on the database file	Allows me to run SQL commands on the database and get the output.

Test Data

In this section I will detail all the tests I will run on all of the different parts of the program to make sure they all work.

User Accounts

Login Form Validation

Test Data	Type
All form elements filled in	**Valid**
One form element not filled in	**Invalid**

Login Form Complete

Test Data	Type
Left click on login button	**Valid**
Right Click on event	**Invalid**
“enter key”	**Invalid**

Email Validation

Test Data	Type
bob@gmail.com	**Valid**
bob@gmail.co.uk	**Valid**
Bob	**Invalid**
Bob@	**Invalid**
@gmail.com	**Invalid**
bob@gmail.	**Invalid**
.com	**Invalid**

Create Account Form Validation

Test Data	Type
All form elements filled in	**Valid**
One form element not filled in	**Invalid**

Test Data	Type
Username or email not already attributed to another user	**Valid**
Username attributed to another user	**Invalid**
Email attributed to another user	**Invalid**

Create Account Form Complete

Test Data	Type
Left Click on create account Button	**Valid**
Right Click on event	**Invalid**
"Enter" key	**Invalid**

Edit Account Form Validation

Test Data	Type
All form elements filled in	**Valid**
One form element not filled in	**Invalid**

Test Data	Type
Username or email not already attributed to another user	**Valid**
Username attributed to another user	**Invalid**
Email attributed to another user	**Invalid**

Edit Account Form Complete

Test Data	Type
Left Click on create account Button	**Valid**
Right Click on event	**Invalid**
"Enter" key	**Invalid**
Left Click on 'Back' Button	**Invalid**

Navigation

Navigation in the Mini Calendar

Test Data	Type
Navigate to the 15th March	**Valid**
Navigate to the 32nd March	**Invalid**

Selecting an date in the Mini Calendar

Test Data	Type
Left click on date	**Valid**

Right click on date	**Invalid**
Dragging several Dates	**Invalid**

Changing date using the left button

Test Data	Type
Left click on button	**Valid**
Right click on button	**Invalid**
Click outside button	**Invalid**

Changing date using the right button

Test Data	Type
Left click on button	**Valid**
Right click on button	**Invalid**
Click outside button	**Invalid**

Changing date using the today button

Test Data	Type
Left click on button	**Valid**
Right click on button	**Invalid**
Click outside button	**Invalid**

Create new event button

Test Data	Type
Left click on button	**Valid**
Right click on button	**Invalid**
Click outside button	**Invalid**

Account details button

Test Data	Type
Left click on button	**Valid**
Right click on button	**Invalid**
Click outside button	**Invalid**

Home buttons pressed

Test Data	Type
Left click on button	**Valid**
Right click on button	**Invalid**
Click outside button	**Invalid**

Events

Event Viewer to show event details

Test Data	Type
Left Click on event	**Valid**
Right Click on event	**Invalid**
Press “Enter” key on event	**Invalid**

Event Clicked on to show more details

Test Data	Type
Left click on button	**Valid**
Right click on button	**Invalid**
Click outside event	**Invalid**

Edit Button

Test Data	Type
Left click on button	**Valid**
Right click on button	**Invalid**
Click outside event	**Invalid**

Delete Button

Test Data	Type
Left click on button	**Valid**
Right click on button	**Invalid**
Click outside event	**Invalid**

Create New Event Validation

Test Data	Type
Name, start time filled in and all day checked	**Valid**
Name, start time, and end time filled in	**Valid**
Name not filled in	**Invalid**
Start time not filled in	**Invalid**

Test Data	Type
Name: “Lunch 1” Start time: “1/1/2019 @ 15:00” End Time: “1/1/2019 @ 17:00”	**Valid**

Name: “Lunch 2” Start time: “1/1/2019 @ 15:00” End Time: “2/1/2019 @ 17:00”	**Valid**
Name: “School 1” Start time: “1/1/2019 @ 15:00” All Day: Checked	**Valid**
Name: “Lunch 3” Start time: “1/1/2019 @ 15:00” End Time: “1/1/2019 @ 12:00”	**Invalid**
Name: “School 2” Start time: “1/1/2019 @ 15:00” All Day: Unchecked	**Invalid**

Create New Event Form Complete

Test Data	Type
Left Click on create account Button	**Valid**
Right Click on event	**Invalid**
“Enter” key	**Invalid**
Left Click on ‘Back’ Button	**Invalid**

Edit Event Validation

Test Data	Type
Name, start time filled in, and all day checked	**Valid**
Name, start time, and end time filled in	**Valid**
Name not filled in	**Invalid**
Start time not filled in	**Invalid**

Test Data	Type
Name: “Lunch 1” Start time: “1/1/2019 @ 15:00” End Time: “1/1/2019 @ 17:00”	**Valid**
Name: “Lunch 2” Start time: “1/1/2019 @ 15:00” End Time: “2/1/2019 @ 17:00”	**Valid**
Name: “School 1” Start time: “1/1/2019 @ 15:00” All Day: Checked	**Valid**
Name: “Lunch 3” Start time: “1/1/2019 @ 15:00” End Time: “1/1/2019 @ 12:00”	**Invalid**
Name: “School 2” Start time: “1/1/2019 @ 15:00” All Day: Unchecked	**Invalid**

Edit Event Form Complete

Test Data	Type
Left Click on create account Button	**Valid**
Right Click on event	**Invalid**
“Enter” key	**Invalid**
Left Click on ‘Back’ Button	**Invalid**

Acceptance Testing

In this section I am going to detail the test that I will complete to know whether each success criteria point has been completed.

No.	Requirements	Input	Expected Outcome
1	Login Screen	Open App	There is a form that allows the user to login
2	When the user clicks login it checks that they have access from a database	**Valid**: A left mouse click on a login button checks with username and password filled in **Invalid**: A left mouse click where either the username or password are not filled it	**Valid**: Check the username and password against the database, and give an error message if not correct **Invalid**: An error message telling the user to fill in the username and password
3	Create Account	Clicks Create Account button	There is a form that allows the user to create an account
4	Checks user does not already have an account	**Valid**: The username, email, password, and name filled out, and a left click on the create account button **Invalid**: At least one of the following not filled out: username, email, password, and name. Then a left click on the create account button	**Valid**: The inputs are validated and checked against the database **Invalid**: An error message is shown to get the user to fill in the “Create Account” form.
5	If an account already exists then an error message will be shown	**Valid**: An username or email that is in the database **Invalid**: A username or email that is not in the database	**Valid**: An error message something like “Account already exists in the database” **Invalid**: No error message is shown to the user.
6	A mini calendar for quick navigation	**N/A**	There is a mini calendar in the top left corner so that the user can quickly move between dates.

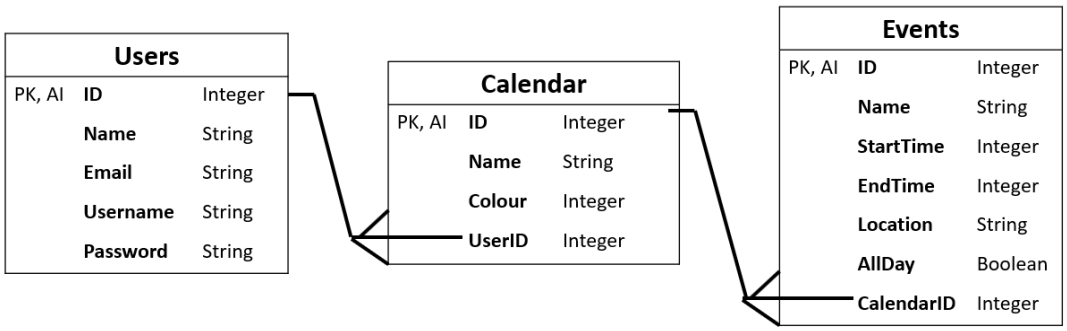
7	A view to see all the events that week	**N/A**	The user should be able to see all their events within a week on the screen in chronological order.
8	The week's events should be split into days and hours in the view on the main page	**N/A**	The user should be able to see the events on a grid with each day split into hours when logged in and navigating around the program.
9	When viewing events on the main page, the user should see the colour of the event and the name.	**N/A**	The events should be coloured when viewing them in the main view and the text should be black.
10	If the event name is too long to display on the main page, then it should trim the name so that it fits	**Valid**: Have an event called "This is a really long name for an event" and then navigate to it in the event viewer **Invalid**: Have a event with a short name, called "Lunch" and navigate to it in the event viewer	**Valid**: The name of the event is trimmed depending on the width of the event for example it might be "This is a…." **Invalid**: The event name is not cut and is displayed in full.
11	When an event is clicked on it should display information about it	**Valid**: Left click on an event **Invalid**: Not clicking on an event or using a right click	**Valid**: The information about the event clicked on is displayed. **Invalid**: Nothing should happen
12	When the user clicks on an event the colour of that event should become the background	**Valid**: Left click on an orange event **Invalid**: Not clicking on an event or using a right click	**Valid**: The background of the window should become orange **Invalid**: Nothing should happen
13	A menu bar at the top detailing different options	**N/A**	A menu at the top with the different options for a user to choose from.
14	An all-day event is shown at the top of that day	**Valid**: A event that is marked as all day in the database **Invalid**: A event that is not marked as all day in the database	**Valid**: The event is displayed at the top of the day, before midnight. **Invalid**: The event is not displayed at the top of the day but within the rest of the grid.
15	Navigation Buttons at top of page	**N/A**	A left and right button to move the event viewer should be at the top of the window.
16	A "Today" Button to skip to today's date	**Valid**: Left click on "Today" Button	**Valid**: The event viewer is moved to that days date **Invalid**: The event viewer does not go to today's date.
17	Create event page	**N/A**	There is a form that a user can use to create a new event
18	An event must have a name and start date at a	**Valid**: The event has a start time, end time and a name.	**Valid**: A event is created in the database.

	minimum, otherwise an error message is shown.	The end time is after the start time. **Invalid**: The event is missing either a start time, end time or a name. The start time may also be after the end time. **Boundary:** The event has a start time, end time and a name. The end time is equal to the start time.	**Invalid**: Display an error message because it is either missing data or it is starting before it ends, and an event cannot do this. **Boundary:** Display an error message because an event cannot have a length of 0
19	Delete events	**N/A**	A user can delete an event when viewing more details about the event.
20	Edit Events page	**N/A**	There is a page that allows the user to edit the information about the event.
21	When an event is edited it should go through the same validation as when creating an event	**Valid**: The event has a start time, end time and a name. The end time is after the start time. **Invalid**: The event is missing either a start time, end time or a name. The start time may also be after the end time. **Boundary:** The event has a start time, end time and a name. The end time is equal to the start time.	**Valid**: An event is created in the database. **Invalid**: Display an error message because it is either missing data or it is starting before it ends and an event cannot do this. **Boundary:** Display an error message because an event cannot have a length of 0.
22	A database to store user data, events etc	**N/A**	There is a database file to store all the information about a user and their events
23	The personal data about a user is email, name, username and password	**Valid**: The information stored about a user is: email, name, username and password.	**Valid**: This is the information about a user **Invalid**: There should be no more information collected about the user when they create the account nor at any other point.
24	A user can change the information stored about them	**Valid**: A user can change their email, name, username and password.	**Valid**: There is a form that allows the user to change all the information about them
25	When a user changes the information about them it goes through the same validation as when they create a new account	**Valid**: A valid email and a username that is not attributed to another user. **Invalid**: A email address that is not valid or a username	**Valid**: The account information is edited in the database **Invalid**: An error is displayed to the user.

		that is attributed to another user.	
26	Store title, description, and location of an event. As well as start and end times	**Valid**: The information stored about an event is: title, description, location, start and end times.	**Valid**: This is the information about an event
27	User can change colour of event	**Valid**: The user can change the colour of the event	**Valid**: When creating or editing an event there is a dropdown allowing the user to edit the event colour **Invalid**: The user cannot change the colour of an event
28	There are at least 5 colours that a user can choose from for an event	**N/A**	**Valid**: There are at least 5 colours in the dropdown **Invalid**: There are less than 5 colours
29	The colour of an event defaults to the colour of the calendar	**N/A**	When creating an event the default colour in the dropdown is the same colour as the calendar colour.
30	An event can be set to all day	**N/A**	There is a checkbox when editing or deleting an event to set it to all day
31	The title of the window should say “Calendar !”	**N/A**	The top of the window should say “Calendar!”

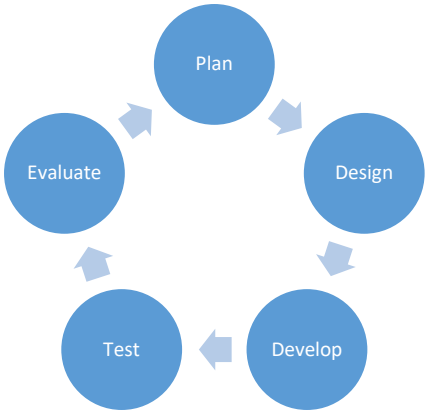
Database Design

I have created a database design, labelling the Primary Keys (PK), fields that need to auto increment (AI), and one-many relationships.



The Development Story

I now feel ready to start developing my solution. While developing my solution I am going to try to stick to an Agile Development methodology. This means that I will loop through the 5 steps:



By using this, I hope this methodology will give me flexibility over other type's methodologies. For example, if I want to go back and improve sections of the program, this methodology will allow me to, whereas something like waterfall is more rigid, so will not allow me to go back. It is also the best methodology given that I do not know how some sections will work, so I am given time to plan and design these before I create them.

Session 1 – 20/01/2019

The first thing that I did today was take my database design and turn it into SQL. This will create my database. Since designing the database I have realised that I need an extra column in the Events table. I need an "all-day" column which will be a Boolean. This will mean that All Day events can be displayed separately. All of this will help to satisfy the following success criteria points:

No.	Requirements	Justification
22	A database to store user data, events etc	This allows the program to be expandable. The database could be linked to other programs like a phone app in the future.
23	The personal data about a user is email, name, username and password	I don't need more data; this keeps me compliant with the Data Protection Act.
26	Store title, description, and location of an event. As well as start and end times	This is the data that is needed in the form.

The SQL I created is below:

```
/*Create Users Table - Holds User Data*/
CREATE TABLE Users (
        ID int NOT NULL AUTO_INCREMENT,
        Name varchar(255) NOT NULL,
        Email varchar(255) NOT NULL,
        Username varchar(255) NOT NULL,
        Password varchar(255) NOT NULL,

        PRIMARY KEY (ID)
);

/*Create Calendars Table - Holds data about all the calendars that exist */
CREATE TABLE Calendars (
        ID int NOT NULL AUTO_INCREMENT,
        Name varchar(255) NOT NULL,
        Colour int,
        UserID int NOT NULL,

        PRIMARY KEY (ID),
        FOREIGN KEY (UserID) REFERENCES Users(ID)
);

/*Create Calendars Table - Holds data about all the events that exist */
CREATE TABLE Events(
        ID int NOT NULL AUTO_INCREMENT,
        Name varchar(255),
        StartTimestamp int NOT NULL,
        EndTimestamp int NOT NULL,
        Location varchar(255),
        AllDay bollean
        CalendarID int NOT NULL,

        PRIMARY KEY (ID),
        FOREIGN KEY (CalendarID) REFERENCES Calendars(ID)
);
```

I then created a basic Python program to create the database for me. There is a variable 'SQL' is just a string of the SQL shown above.

```
import sqlite3

#Open Database
conn = sqlite3.connect('Calendar.db')
c = conn.cursor()

#Run SQL and Close Database
c.execute(SQL)
conn.commit()
conn.close()
```

It did not initially work when I ran it because I had made some mistakes in the SQL. The first mistake I made is that SQLite, the database engine that I am using, does not support the command *AUTO_INCREMENT*. The primary key is auto incremented without the need for an extra command. My new ID columns look like `ID int,` They all will still auto increment, I just don't need the command. The second mistake I made was that I missed a comma on *Allday* column in Events. It now looks like, `AllDay bollean,` . When I corrected the mistakes and I created the database it worked and I had a working database.

I then created the window file as I had in the Pseudocode.

```
from tkinter import *
class Window(Tk):
    def __init__(self, *args, **kwargs):
        Tk.__init__(self, *args, **kwargs)
        self.container = Frame(self)
        self.frames = {}


    def createFrame(self,newFrame):
        frame = newFrame(self.container, self)
        self.frames[newFrame] = frame
        frame.grid(row=0, column=0, sticky="nsew")


    def show_frame(self, FrametoShow):
        frame = self.frames[FrametoShow]
        frame.tkraise()
```

What I have realised while creating this class is that I have forgotten about certain key elements of Tkinter code in my pseudocode. For example, I have forgotten to pack or grid any form elements that I have created. I am going to add these in as I see fit. This piece of code now looks like:

```
from tkinter import *
from Login import *
from CreateAccount import createAccount
class Window(Tk):
    def __init__(self, *args, **kwargs):
        Tk.__init__(self, *args, **kwargs)

        self.container = Frame(self)

        self.container.pack(side="top", fill="both", expand = True)

        self.container.grid_rowconfigure(0, weight=1)
        self.container.grid_columnconfigure(0, weight=1)


        self.frames = {}

        for F in [login, createAccount]:
            self.createFrame(F)

        self.show_frame(login)


    def createFrame(self,newFrame):

        frame = newFrame(self.container, self)
        self.frames[newFrame] = frame
        frame.grid(row=0, column=0, sticky="nsew")


    def show_frame(self, FrametoShow):
        frame = self.frames[FrametoShow]
        frame.tkraise()
```

In this version I also created empty files of login and create account, just so that I can import them.

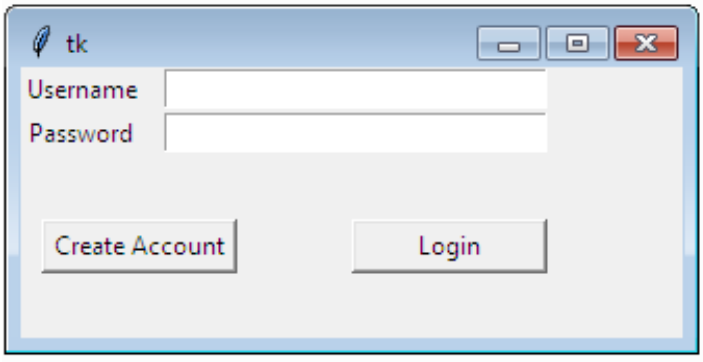
I have now created the login screen to test this. This is to satisfy the following success criteria:

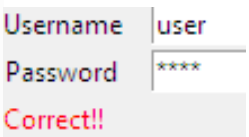
No.	Requirements	Justification
1	Login Screen	To protect the user's events from unauthorised access

I created this from my pseudocode in the design section. Again I had to add the packing of all of the elements as this was not in my plan.

```
class login(Frame):
    def  __init__ (self, parent, controller):
        Frame.__init__(self,parent)

        #Create  labels
        usernameText = Label(self, text="Username")
        passwordText = Label(self, text="Password")
        usernameText.grid(column=0, row=1)
        passwordText.grid(column=0, row=2)

        #Create Username entry
        self.username = Entry(self,width=30)
        self.username.grid(column=1, row=1, padx=10)

        #Create password entry
        self.password = Entry(self,width=30)
        self.password.grid(column=1, row=2)
        self.password.config(show="*")

        #Create error message label
        self.errorLabel = Label(self, text="")
        self.errorLabel.grid(column=0, row=3, sticky=W, columnspan=2)
        self.errorLabel.config(fg='red')

        #Create Login Button + Create account button
        createAccountButton = Button(self, text="Create Account", width=12, command=lambda: controller.show_frame(createAccount))
        createAccountButton.grid(column=0, row=4, columnspan=2, sticky=W, padx=10, pady=10)
        LoginButton = Button(self, text="Login", width=12, command=self.Login)
        LoginButton.grid(column=1, row=4, padx=10, pady=10, sticky=E)

    def Login(self):
        usernameHolder = self.username.get()
        passwordHolder = self.password.get()
        message = "Incorect"
        if usernameHolder == "user" and passwordHolder == "pass":
            message = "Correct!!"

        self.errorLabel.config(text=message)
```

When I ran Window.py, it successfully opened this screen, with a login page. I also tested a basic login system. This is not yet linked to the database, which is something I will look at in my next session.





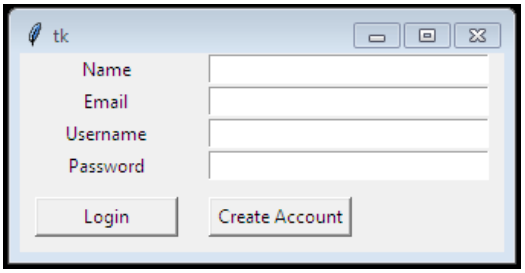
Session 2 – 23/01/2019

My plan for today is to get the create account form working. From that I will begin to get the database backend working, as I will have an easy way to enter data.

I have created a basic create account page from my pseudocode. I did not originally have any grid positions in the pseudocode. Because of this I have created a 2 by 4 grid in the page to add everything into.

```
class createAccount(Frame):
    def  __init__ (self, parent, controller):
        Frame.__init__(self,parent)

        nameText = Label(self, text="Name")
        emailText = Label(self, text="Email")
        usernameText = Label(self, text="Username")
        passwordText = Label(self, text="Password")

        nameText.grid(column=0, row=0)
        emailText.grid(column=0, row=1)
        usernameText.grid(column=0, row=2)
        passwordText.grid(column=0, row=3)

        #Create entry
        self.name = Entry(self,width=30)
        self.name.grid(column=1, row=0, padx=10)

        self.email = Entry(self,width=30)
        self.email.grid(column=1, row=1, padx=10)

        self.username = Entry(self,width=30)
        self.username.grid(column=1, row=2, padx=10)

        #Create password entry
        self.password = Entry(self,width=30)
        self.password.grid(column=1, row=3)
        self.password.config(show="*")

        #Create Login Button + Create account button
        createAccountButton = Button(self, text="Create Account", width=12)
        createAccountButton.grid(column=1, row=4, columnspan=2, sticky=W, padx=10, pady=10)
        LoginButton = Button(self, text="Login", width=12, command=lambda: controller.show_frame(Login.login))
        LoginButton.grid(column=0, row=4, padx=10, pady=10, sticky=E)
```

I tested this and it looked like:



This worked as I expected it to. When I pressed the 'login' it went back to the login page as I expected. This is using my show frame class in the window, which will allow all my windows to switch to other windows, without the program being reopened.

I am now going to create my *database.py* file. This will hold a class called '*database'* which will interface with the database file. Today I am going to create the basic functions of creating and reading users for the login system. This will also include validation of all the fields. This is all to satisfy the following success criteria:

No.	Requirements	Justification
2	When the user click login it checks that they have access from a database	This stops unauthorized access to the calendar. By using a database it make easy to change data about the user.

I will first create pseudocode for these functions.

```
INCLUDE sqlite3
INCLUDE regular Expressions

CLASS Database:
    FUNCTION __init__(self):

        databaseLocation = DATABASE LOCATION
        connection = SQLITE CONNECT (databaseLocation)

        self.cur = SQLITE CONNECTION CURSOR

    END FUNCTION

    FUNCTION checkLogin(self, username, password):
        sql = "SELECT ID,Password FROM users WHERE Username= " + username

        self.cur.EXECUTE sql

        rows = self.cur.FETCH ALL

        FOR users IN rows:
            IF password EQUALS row[LOCATION OF PASSWORD FROM DATABASE]:
                RETURN TRUE and user ID

        RETURN FALSE

    END FUNCTION

    FUNCTION createAccountCheck(self, name, email, username, password):
        IF self.validateEmail(email) EQUALS FALSE:
            RETURN "Email Not Valid"

        IF name EMPTY OR email EMPTY OR password EMPTY:
            RETURN "Fill In Missing Values"

        sql = "INSERT INTO Users VALUES " + name, email, username, password
```

```
        self.cur.EXECUTE sql
        self.cur.COMMIT DATABASE

        RETURN TRUE

    END FUNCTION

    FUNCTION validateEmail(self, email):
        regx = "[^@]+@[^@]+\.[^@]+"

        IF REGULAR EXPRESSIONS MATCH (regx, email) EQUALS TRUE:
            RETURN TRUE
        ELSE:
            RETURN FALSE

    END FUNCTION

END CLASS
```

I started off by creating the *init* of the function.

```
import sqlite3 as lite
import os

class Database:
    def __init__(self):
        #Create Database connection
        databaseLocation = "Calendar.db"
        print(databaseLocation)
        self.con = lite.connect(databaseLocation)
        self.cur = self.con.cursor()
```

This worked however, it could not reliably find the database file, because if the working directory is different to the location of the database then it is not there. To rectify this, I am going to find the location of the "database.py" file and then add the file name of the database. The file now looks like:

```
import sqlite3 as lite
import os

class Database:
    def __init__(self):
        #Find location of database
        dir_path = os.path.dirname(os.path.realpath(__file__))
        databaseLocation =  dir_path + "/Calendar.db"

        #Create Database connection
        self.con = lite.connect(databaseLocation)
        self.cur = self.con.cursor()
```

This added line and variable *dir_path* allows me to find the location of the database.py file. The next line, database location then adds the file name.

I have now created createAccount class. I also temperedly created a test email variable that always returns true. I did have to change from my pseudocode slightly as I used the values function meaning I do not have to manually add them to the SQL statement.

```
import sqlite3 as lite
import os

class Database:
    def __init__(self):
        #Find location of database
        dir_path = os.path.dirname(os.path.realpath(__file__))
        databaseLocation =  dir_path + "/Calendar.db"

        #Create Database connection
        self.con = lite.connect(databaseLocation)
        self.cur = self.con.cursor()


    def createAccount(self, name, email, username, password):
        if self.validateEmail(email) == False:
            return "Email Not Valid"
        if name == "" or username == "" or password == "":
            return "Fill In Missing Values"

        values = (name, email, username, password)
        sql = "INSERT INTO Users ('Name','Email', 'Username', 'Password') "
        self.cur.execute(sql,values)
        self.con.commit()

        return True


    def validateEmail(self, email):
        return True
```

I tested the create account algorithm with the code:

```
x= Database()
x.createAccount("Robert","test@test.com", "rob", "password")
```

This worked as I expected however the ID was not added automatically as it should have been. This is essential as currently there is not a primary key in that table which will allow me to reference the data. I am going to come back to this later, as I don't have any database tools at school.

I created the validate email function. The first thing that I realised that I had missed is that I need to compile the regex expression. I have added this in with the holder *p*. The code looks like:

```
def validateEmail(self, email):
    regx = "[^@]+@[^@]+\.[^@]+"
    p = re.compile(regx)

    if p.match(email):
        return True
    else:
        return False
```

In my initial testing the regular expression did not work properly, it was letting through emails that it should not. Having done some research, there is a standard called 'RFC 5322'. This standard details what an email address should look like. My regular expression was not following this paragraph, so I have found a regular expression that does (SOURCE: https://emailregex.com/). My new regex looks like `regx = "(^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$)"`. I am now going to test this function.

Test Num	Input Data	Reason for test	Expected Output	Actual Output	Pass/Fail
1	test@test.com	Normal Use case	True	True	Pass
2	example@example.co.uk	Normal Use case	True	True	Pass
3	test@test.	Missing part of domain	False	False	Pass
4	test@	Missing whole domain	False	False	Pass
5	@test.com	Missing username	False	False	Pass
6	test@.com	Missing part of domain	False	False	Pass
7	test@.	Missing domain text	False	False	Pass

This algorithm passed, meaning that all email address are going to be safely inputted into the database.

There are two things that I have forgotten to consider with my database: Password Hashing and SQL Injection. By hashing the password when they are put into the database, it will mean that if for what ever reason the database is stolen, the hacker will not immediately be able to access the users data through the program, as the password will appear to be a random string. This is a lot better than storing the users password in plain text in the database.

I am going to create an algorithm to hash the passwords. I am going to use SHA2 which will provide me a hash that is secure and quick to hash. I am going to use a python module called *hashlib* to do the hashing.

```
FUNCTION hashPassword(self, string):
    Hash = hashlib.sha224(string.encode()).hexdigest()
END FUNCTION
```

I then created this function and tested it with the input '*password*' and the hashing function worked, returning a hash.

```
def hashPassword(self,password):
    #Hash the string.
    return hashlib.sha224(password.encode()).hexdigest()
```

```
Python - Database.py:53 ✔

d63dc919e201d7bc4c825630d2cf25fdc93d4b2f0d46706d29038d01
[Finished in 0.148s]
```

I have changed the create account function to hash the password before it inputs it into the database:

```
values = (name, email, username, self.hashPassword(password))
```

The other problem that I had not foreseen was SQL injections. SQL injection is where a user inputs SQL statements into normal fields. If the program does not protect against this, then it could output the data in the database. This would be a privacy issue. Having done some research into the problem, to prevent SQL injection I have to strip the unwanted characters. With the SQL module I am using in python, as long as I use question marks instead of the data (like this: `"INSERT INTO Users ('Name','Email', 'Username', 'Password') VALUES (?,?,?,?);"`) and the execute the statement with the line `self.cur.execute(sql,values)` it automatically makes it safe for me. I will continue to do this from all inputs to the database.

I am now going to create the `checkLogin()` function in database.py. I have already done the pseudocode above, however I am going to change my code to keep it safe from SQL injection.

```
def checkLogin(self, username, password):
    #See if a username and password is valid
    sql = "SELECT ID,Password FROM users WHERE Username=?"
    self.cur.execute(sql, (username))
    rows = self.cur.fetchall()
    for user in rows:
        if self.hashPassword(password) == user[1]:
            return True, user[0]
    return False, None
```

The final thing I am going to do today is change the title at the top of the window. This is to satisfy success criteria number 31:

No.	Requirements	Justification
31	The title of the window should say “Calendar !”	So the user knows what the window is and the window purpose is clear to the user

I have added the following line to the *__init__* function of the *window* class:

```
self.title("Calendar!")
```

I tested this and got the following output:

This worked. While I was doing this I realised that I had forgotten to add comments in the *Window* class. I have added them in:

```
ss Window(Tk):
 def __init__(self, *args, **kwargs):
     #Define Tkinter Window
     Tk.__init__(self, *args, **kwargs)

     #Define Database Connection
     self.DB = Database()

     #Create Container in window
     self.container = Frame(self)
     self.container.pack(side="top", fill="both", expand = True)
     self.container.grid_rowconfigure(0, weight=1)
     self.container.grid_columnconfigure(0, weight=1)
```

```
     #User Id when logged in
     self.USERID = None

     self.frames = {}

     #Create Long and create account frames
     for F in [login, createAccount]:
         self.createFrame(F)

     #Display login page
     self.show_frame("login")


def createFrame(self,newFrame):
     #create a new frame
     frame = newFrame(self.container, self, self.DB)
     self.frames[newFrame.__name__] = frame
     frame.grid(row=0, column=0, sticky="nsew")


def show_frame(self, FrametoShow):
     #Display a frame
     frame = self.frames[FrametoShow]
     frame.tkraise()
```

In the next session I am going to fix the database primary key problem, test the *check login* function, and then get both *checkLogin*, and *createAccount* functions working with the GUI Form.

Session 3 – 30/01/2019

The first thing that I am going to look at is the database. It was not adding and auto incrementing the primary key. I used a DBMS to help me see what was wrong with my database. For some reason when I ran the original SQL, It had not set auto increment and primary key. I have used the DBMS to change the database:

Name	Type	NN	PK	AI	U
ID	INTEGER	☑	☑	☑	☑

This produced the new SQL as shown below:

```
CREATE TABLE "Users" (
    "ID"    INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
    "Name"  varchar(255) NOT NULL,
    "Email" varchar(255) NOT NULL,
    "Username"  varchar(255) NOT NULL,
    "Password"  varchar(255) NOT NULL
);

/*Create Calendars Table - Holds data about all the calendars that exist */
CREATE TABLE Calendars (
    "ID"    INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
    "Name" varchar(255) NOT NULL,
    "Colour" int,
    "UserID" int NOT NULL,

    PRIMARY KEY (ID),
    FOREIGN KEY (UserID) REFERENCES Users(ID)
);

/*Create Calendars Table - Holds data about all the events that exist */
CREATE TABLE Events(
    "ID"    INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
    "Name" varchar(255),
    "StartTimestamp" int NOT NULL,
    "EndTimestamp" int NOT NULL,
    "Location" varchar(255),
    "AllDay" bollean,
    "CalendarID" integer,

    PRIMARY KEY (ID),
    FOREIGN KEY (CalendarID) REFERENCES Calendars(ID)
);"""
```

I am now going to test the *check login* function that I created at the end of last times session. I have created this code to help me test it. I then created a testing table and completed it.

```
x= Database()
user = "rob"
password = "password"
print(x.checkLogin(user,password))
```

Test Num	Input Data	Reason for test	Expected Output	Actual Output	Pass/Fail
1	Username: “rob” Password: “password”	Normal Case	(True, 6)	(True, 6)	Pass
2	Username: “bill” Password: “password”	Normal Case	(True, 7)	(True, 7)	Pass
3	Username: “rob” Password: “password1”	Incorrect Password	(False, None)	(False, None)	Pass
4	Username: “rob1” Password: “password”	Incorrect Username	(False, None)	(False, None)	Pass

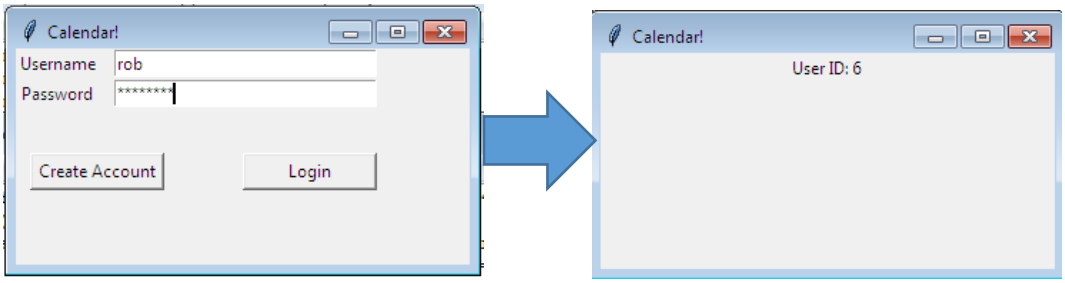
This algorithm works. I am now going to make it work with the GUI. I have to make the login function so that when the login button is pressed it checks the database and then shows the mainpage. It is doing this by using the *checkLogin* function, and seeing if it returned “true”. If it did then it saves the user ID to the controller, so that it is accesiable by other frames. This will be useful futher on, as I will be able to get the users ID.

```
def Login(self):
    #Get Data from form
    usernameHolder = self.username.get()
    passwordHolder = self.password.get()

    #Check if user Exists and run
    loginCheck = self.DB.checkLogin(usernameHolder,passwordHolder)
    if loginCheck[0]:
        #Save the users ID
        self.controller.USERID = loginCheck[1]
        #Add Frame and Open
        self.controller.createFrame(mainPage)
        self.controller.show_frame("mainPage")

    else:
        #Display Error Message
        self.errorLabel.config(text="Username or Password Incorect")
```

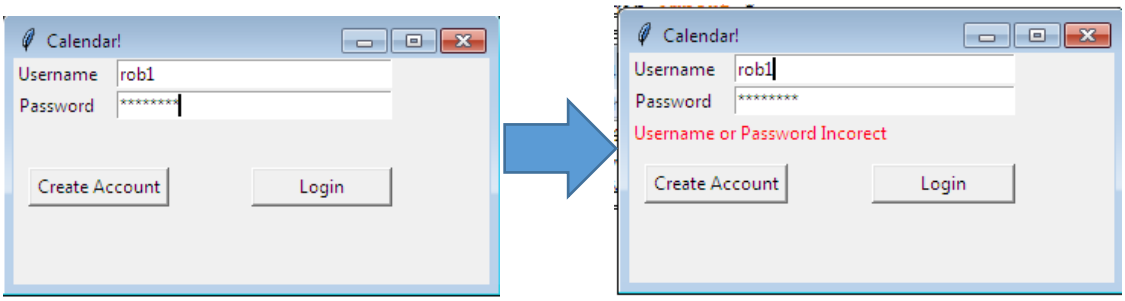
To test this I have created a basic mainPage function to test this. If the login is sucessful it should show the users ID (that it got from the controller), otherwise it should show an error message.

```
class mainPage(Frame):
    def __init__ (self, parent, controller):
        Frame.__init__(self,parent)
        self.DB = Database()

        w = Label(self, text="User ID: " + str(controller.USERID))
        w.pack()
```

I tested it with a correct username and password first. This worked as I expected.



I then tested it with an incorrect username and this worked as expected – showing me an error message.



You can see this part working in **Video 1** in the supporting documents.

The next thing that I am going to do is to get the GUI create account form, working with the database. I have created the create account function. This means that when the user clicks create account, this function will now run. All of this is to satisfy the following success criteria:

No.	Requirements	Justification
3	Create Account	A new user will need to create an account
4	Checks user does not already have an account	This is to stop duplication of data. This also helps to keep data relevant about a user
5	If an account already exists then an error message will be shown	This is so the user knows that there is a problem

The first thing this function does is to get all of the data from the form. It then uses the database function *createAccount* to validate and save the data, making sure the email is valid and no fields are empty, and make sure it is safe for the database. This stops SQL injection and keeps the data relevant. Once this has happened it then checks if there was an error. If there was, it displays the error message. If there is not an error then it uses the database function *checkLogin* to log the user in and get the users ID. It then creates and shows the frame *mainPage*. This will be the *mainPage* that is used during the use of the program.

I have taken a screen shot of this algorithm below.

```
def createAccount(self):
    #Get Data from form
    usernameHolder = self.username.get()
    passwordHolder = self.password.get()
    nameHolder = self.name.get()
    emailHolder = self.email.get()

    #Check if user Exists and run
    accountCheck = self.DB.createAccount(nameHolder,usernameHolder,emailHolder,passwordHolder)
    if accountCheck == True:
        #Log user in
        loginCheck = self.DB.checkLogin(usernameHolder,passwordHolder)
        self.controller.USERID = loginCheck[1]
        #Add Frame and Open
        self.controller.createFrame(mainPage)
        self.controller.show_frame("mainPage")

    else:
        #Display Error Message
        self.errorLabel.config(text=accountCheck)
```

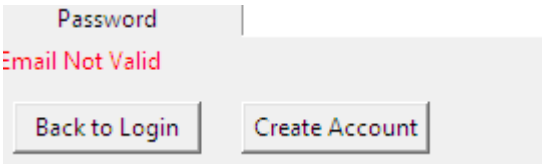
While testing this I was getting this error:

```
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python34\lib\tkinter\__init__.py", line 1487, in __call__
    return self.func(*args)
  File "N:\Work\Computing\Year 13\CO3\Download\2019-02-11\Computing-CA-master\C
de\CreateAccount.py", line 66, in createAccount
    self.errorLabel.config(text=accountCheck)
AttributeError: 'createAccount' object has no attribute 'errorLabel'
```

I have realised that I also need to add the error text to the Create Account form. This means when it was trying to add an error message, it was causing this problem. I have added this section of code when the create account form is created at the beginning.

```
#Create error message label
self.errorLabel = Label(self, text="")
self.errorLabel.grid(column=0, row=4, sticky=W, columnspan=2)
self.errorLabel.config(fg='red')
```

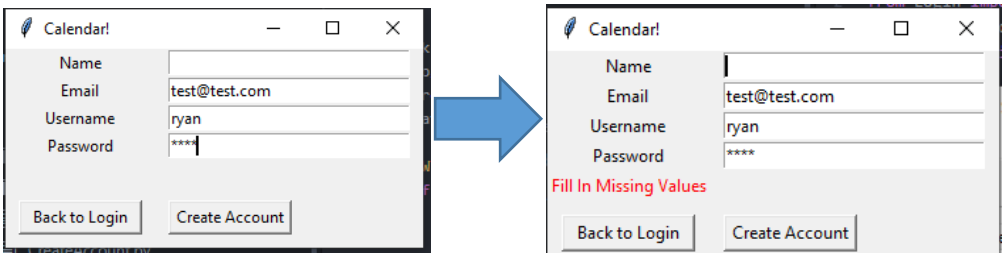
This problem has now stopped. The next problem I had was that regardless of whether the email was valid the program was always outputting the error message “Email Not Valid”:



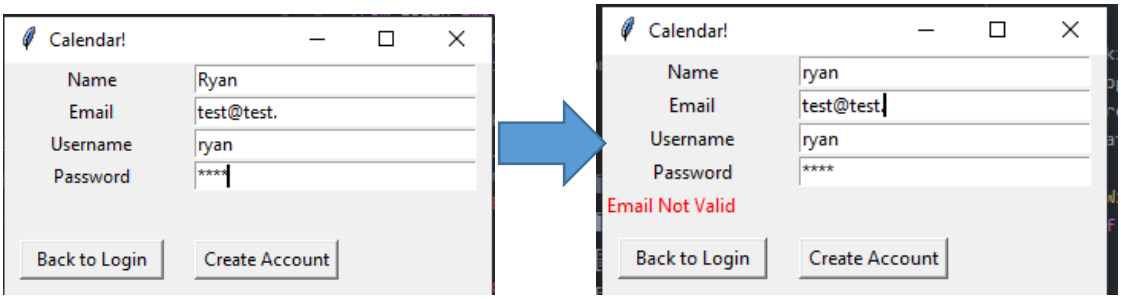
I know that the *checkEmail* function in database.py works, because I tested it, and did the testing table earlier. By using this fact, I have identified that the error is that I had the email and username the wrong way around when passing it to the createAccount function. This means that it was trying to validate the username as an email. I have switched them around, so that line now looks like:

```
#Check if user exists and run
accountCheck = self.DB.createAccount(nameHolder,emailHolder,usernameHolder,passwordHolder)
```

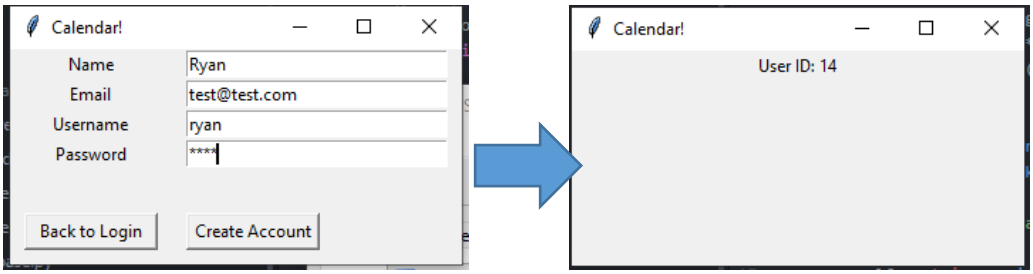
I then retested it, as shown below. The first thing I tested was missing a value.



This worked and displayed an error message as I expected. I then tested an invalid email.



This worked as I expected. When I pressed the create account button, it displayed an error message. The final test was a normal use, where all the values are correct.



This worked as expected. I then used a DBMS to check the database to make sure that the data was correctly inputted into the database. As you can see, all the data was inputted correctly, and the primary key is working.

11	Robert	test@test.com	rob	d63dc919e20...
12	Sue	sue@test.com	sue	885cbf16ccf5...
13	Bob Smith	bob@bob.com	bob	d63dc919e20...
14	Ryan	test@test.com	ryan	ccc9c73a3765...

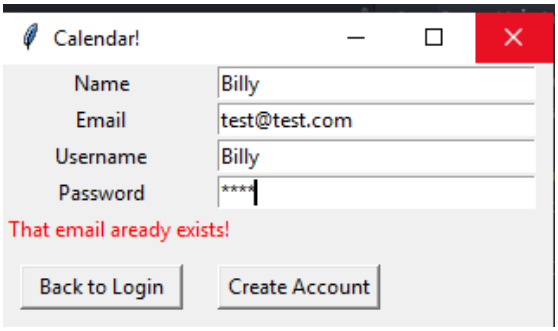
An issue that I need to fix is the fact that both Robert and Ryan (in the screen shot above) have the same email. I need to check whether the user already exists and whether the email or username is already in the database. To do the I need to pull the information from the *users* table where the email or username is the same. I have created an SQL statement to do this:

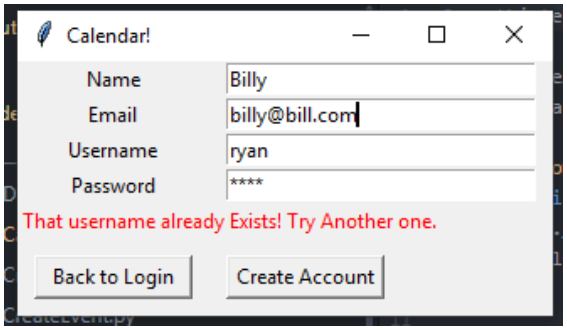
```
SELECT Email,Username FROM users WHERE Email=? OR Username=?
```

I used this to create the following code as part of the validation in the create account function (database.py). It sees if there is an email or username that matches the one the user puts in. If there is, it returns a relevant error message.

```
#See if a user already exists
sql = "SELECT Email,Username FROM users WHERE Email=? OR Username=?"
self.cur.execute(sql, (email,username))
rows = self.cur.fetchall()
for user in rows:
    if user[0] == email:
        return "That email aready exists!"
    else:
        return "That username already Exists! Try Another one."
```

I then tested this with an email and username that I knew was in the database. It worked and I got the correct error messages.





This worked, I can not have multiple users with the same username or email.

I am going to use the test data that I created earlier to test both the login and create account forms.

Login Form Validation

Test Data	Type	Actual
All form elements filled in	**Valid**	**Valid**
One form element not filled in	**Invalid**	**Invalid**

Login Form Complete

Test Data	Type	Actual
Left click on login button	**Valid**	**Valid**
Right Click on event	**Invalid**	**Invalid**
“Enter” key	**Invalid**	**Invalid**

Email Validation

Test Data	Type	Actual
bob@gmail.com	**Valid**	**Valid**
bob@gmail.co.uk	**Valid**	**Valid**
Bob	**Invalid**	**Invalid**
Bob@	**Invalid**	**Invalid**
@gmail.com	**Invalid**	**Invalid**
bob@gmail.	**Invalid**	**Invalid**
.com	**Invalid**	**Invalid**

Create Account Form Validation

Test Data	Type	Actual
All form elements filled in	**Valid**	**Valid**
One form element not filled in	**Invalid**	**Invalid**

Test Data	Type	Actual
Username or email not already attributed to another user	**Valid**	**Valid**
Username attributed to another user	**Invalid**	**Invalid**
Email attributed to another user	**Invalid**	**Invalid**

Create Account Form Complete

Test Data	Type	Actual
Left Click on create account Button	**Valid**	**Valid**
Right Click on event	**Invalid**	**Invalid**
“Enter” key	**Invalid**	**Invalid**

All of these tests worked which means that I am now happy to speak to my stakeholder about what he thinks about these parts of the program. I have spoken to Ryan about everything that exists so far. He said “I think that the login system is really easy to use. It’s really simple to create an account or login, which is a good thing because you want this to be as fast as possible. I do think that it could look a little nicer, as it is not very exciting, however the functionality is there. I think that it is a good idea that it checks to see if a

user already exists in the database, however there are some families that share an email address, and in those cases they won't be able to have multiple accounts."

From this I think I am going to spend some time on the appearance, I am going to focus on the core functionality first, and then if I have time at the end of this project I will make it more stylish. As for some families sharing multiple emails, I think that the majority of users will have their own email. Furthermore, my target market was business people; Most companies issue their employees with a business email. This means that they will have a second email that they can use, which make the problem of families sharing an email less relevant.

Session 4 – 6/02/2019

The first thing I am going to do today is create the menu at the top of the calendar. I need to do this to satisfy the following success criteria point:

No.	Requirements	Justification
13	A menu bar at the top detailing different options	Navigation should be easy, keeping everything at the top is where the user will expect it to be.

I am going to base this from my pseudocode that I did in the planning stages. I started off by creating this function:

```
def createMenu(self):
    #Create Menu
    self.menubar = Menu(self)

    #Create Menu Buttons
    self.menubar.add_command(label="Calendar!")
    self.menubar.add_command(label="Create New Event")
    self.menubar.add_command(label="Robert Watts")

    #Pack Menu
    self.config(menu=self.menubar)
```

I then changed the *__init__* function in the *mainPage* class to run this. I have also made it resize the window now the user has logged in.

```
class mainPage(Frame):
    def  __init__ (self, parent, controller, database):
        Frame.__init__(self,parent)
        self.DB = database
        self.controller = controller


        self.controller.geometry("1000x600")

        self.createMenu()
```

I tested this, and it threw up an error. I have realised that I need to affix the menu bar to the parent window. In the __init__ function I have added this line so that I can access the parent frame.

```
self.parent = parent
```

I then changed the createMenu function so the menu is packed with this line:

```
self.parent.config(menu=self.menubar)
```

. This also created an error as shown below:

```
C:\Program Files (x86)\Python3\python.exe
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Program Files (x86)\Python3\lib\tkinter\__init__.py", line 1699, in __call__
    return self.func(*args)
  File "C:\Users\rober\Documents\Homework\Year 13\Computing\Computing-CA\Code\Login.py", line 46, in Login
    self.controller.createFrame(mainPage)
  File "C:\Users\rober\Documents\Homework\Year 13\Computing\Computing-CA\Code\Window.py", line 31, in createFrame
    frame = newFrame(self.container, self, self.DB)
  File "C:\Users\rober\Documents\Homework\Year 13\Computing\Computing-CA\Code\mainPage.py", line 9, in __init__
    self.createMenu()
  File "C:\Users\rober\Documents\Homework\Year 13\Computing\Computing-CA\Code\mainPage.py", line 18, in createMenu
    self.master.tk.call(self.parent, "config", "-menu", self.menubar)
_tkinter.TclError: unknown option "-menu"

Process returned 0 (0x0)        execution time : 7.605 s
Press any key to continue . . .
```

Having done some research, the only place that a menu bar can be attached to is the window. This means that I want to add it to the controller. I have changed that line to `self.controller.config(menu=self.menubar)`

I tested this, and the menu had been successfully created, and the window has successfully been resized.

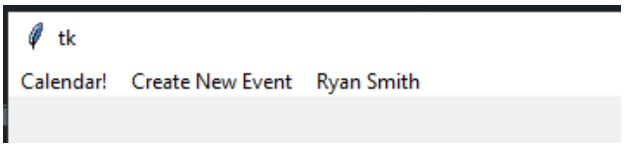
Currently I have hard coded the place holder name “Robert Watts” in to the menu. I am going to now change this to get the users details. In the *database* class I have created the following function:

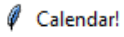
```
def getUserData(self,userID):
    #Get the details from a particular user
    sql = "SELECT Name,Email,Username FROM users WHERE ID=?"
    self.cur.execute(sql, (userID,))
    rows = self.cur.fetchall()
    for user in rows:
            return user
```

This gets the name, email and username for the user with a given ID. I have then added the following to the *mainpage* class. It saves the user details to the controller (So other frames can access the data) and then saves the same data locally (to make it easier to manipulate further on).

```
#Get User Details
self.controller.USER_DETAILS = self.DB.getUserData(self.controller.USERID)
self.USER_DETAILS = self.controller.USER_DETAILS
```

I have then changed the text in the menu bar to `self.menubar.add_command(label=self.USER_DETAILS[0])`. This should add a menu item with the users name. I tested this with an account name of "Ryan Smith". As you can see this worked:



This means that I now have easy access to all of the user's data. I have noticed that the text at the top of the window just says "tk". This is not very nice so I have added the line into the *__init__* function of the *window* class. `self.title("Calendar!")`. This worked, so the top now looks like 

The next thing that I am going to do is create a date picker. The reason this needs to be created it to stratify the following success criteria point:

No.	Requirements	Justification
6	A mini calendar for quick navigation	To make it really easy for a user to find the dates that they want.

This mini calendar will be used in two ways: allowing the user to pick a date when creating an event, and allow quick navigation through the calendar. I have already created a date picker similar to this in JavaScript & HTML for a previous project and I used this to help me create the pseudocode in the planning stage; However like the other pseudocode I created, I need to make sure I pack the elements when coding as this is not in the pseudocode. I have started of by creating the basics of the class:

```
class datePicker(Frame):
        def __init__(self, master=None, command=None, ** kw):
            self.command = command
            Frame.__init__(self,master,**kw)

        def populateForm(self):
            pass
```

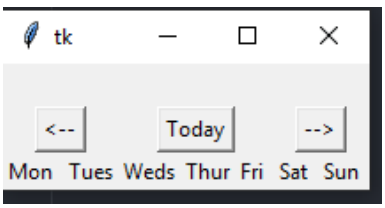
I have also created a test script, to test the date picker in isolation without needing to login to the program every time.

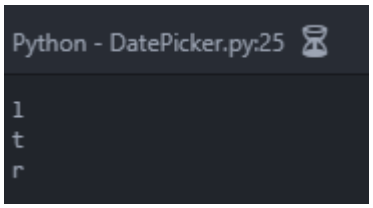
```
if __name__ == "__main__":
    x = Tk()
    y = datePicker(x)
    y.pack()

    x.mainloop()
```

I created the *populateForm* method:

```
def populateForm(self):
    #Create Title
    self.title = Label(self, font='comic-sans-ms 11 bold')
    self.title.grid(in_=self, column=0, row=0, columnspan=7)

    #Create Nav DayButtons
    self.navButtons = []

    self.navButtons.append(Button(self, text="<--", command=lambda: self.navButtonClick("l")))
    self.navButtons[-1].grid(in_=self, column=0, row=1, columnspan=2)

    self.navButtons.append(Button(self, text="-->", command=lambda: self.navButtonClick("r")))
    self.navButtons[-1].grid(in_=self, column=5, row=1, columnspan=2)

    self.navButtons.append(Button(self, text="Today", command=lambda: self.navButtonClick("t")))
    self.navButtons[-1].grid(in_=self, column=2, row=1, columnspan=3)

    #Create Labels
    days = ["Mon ", "Tues", "Weds", "Thur", "Fri ", "Sat ", "Sun "]
    self.DayLabels = []
    ColumnCount = 0
    for day in days:
        self.DayLabels.append(Label(self,text=day))
        self.DayLabels[-1].grid(in_=self, column=ColumnCount, row=2)
        ColumnCount += 1
```

The first thing it does is create an empty label for a title. After that it creates 3 nav buttons: next month, previous month and today. I then created the labels for the days. I then tested this code and initially it did not work. I realised I had not called the function, so I added the line `self.populateForm()` to the *__init__* function. I then ran it and it displayed as I expected and when I pressed the three buttons it printed as I expected (I created the *navButtonClick* function just to print the command at the moment for testing):





I am now going to create the *NextYearMonth* and *PreviousYearMonth* functions. These two functions take a year and month as an input and then calculates the next or previous month. If the year also needs incrementing or decreasing then this needs to happen. I have created both of these functions:

```
def NextYearMonth(self, year, month):
    if month == 12:
        return year + 1, 1
    else:
        return year, month+1
```

```
def PreviousYearMonth(self, year, month):
    if month == 1:
        return year - 1, 12
    else:
        return year, month-1
```

I have done a testing tables, to test both functions:

Test Num	Input Data	Reason for test	Expected Output	Actual Output	Pass/Fail
Next Year Month					
1	2019, 1	Boundary of year	2019, 2	`(2019, 2)`	Pass
2	2019,12	Boundary of year	2020, 1	`(2020, 1)`	Pass
3	2019, 3	Mid year	2019, 4	`(2019, 4)`	Pass
Previous Year Month					
4	2019, 1	Boundary of year	2018, 12	`(2018, 12)`	Pass
5	2019,12	Boundary of year	2019,11	`(2019, 11)`	Pass
6	2019, 3	Mid Year	2019, 2	`(2019, 2)`	Pass

Both functions work. The next function that I am going to create is the *navButtonClick*. Currently for testing purposes it looks like:

```
def navButtonClick(self, command):
    print(command)
```

This function needs to take a command and then work out which month to show. I have used my pseudocode to create the following code:

```
def navButtonClick(self, command):
    if command == "l":
        self.Year, self.Month = self.PreviousYearMonth(self.Year, self.Month)
    elif command == "r":
        self.Year, self.Month, = self.NextYearMonth(self.Year, self.Month)
    elif command == "t":
        now = datetime.datetime.now()
        self.Year = now.year
        self.Month = now.month

    self.DisplayMonth(self.Year,self.Month)
```

Testing:

```
2019 2
2019 2
2019 3
2019 4
2019 5
2019 4
```

To test this function I created the *DisplayMonth* function, and got it to print the inputs. I ran the program and now when I pressed the buttons, they output the month that should be displayed (shown above). This shows me that this algorithm is working.

I am now going to look at the *DisplayMonth* function; this needs to delete the month that is currently being viewed, change the title text and create the new month.

```
def DisplayMonth(self, year, month):
    #Delete Month
    for x in self.DayButtons:
        x.destroy()

    #Change Title Text
    text = month_name[month] + " " + str(year)
    self.title['text'] = text

    #Create new buttons
    self.DayButtons = []

    #Get details about the month in question:
    monthDetails = monthrange(year, month)
    gridPosition = [monthDetails[0], 3]

    for x in range(0, monthDetails[1]):
        #Work out position in grid
        if gridPosition[0] % 7 == 0:
            gridPosition[0] = 0
            gridPosition[1] += 1
        gridPosition[0] += 1
        text = x+1
        if text <= 9:
            text = "0" + str(text)

        #Create and pack da button
        self.DayButtons.append(Button(self, text=text))
        self.DayButtons[-1].grid(in_=self, column=gridPosition[0]-1, row=gridPosition[1])
```

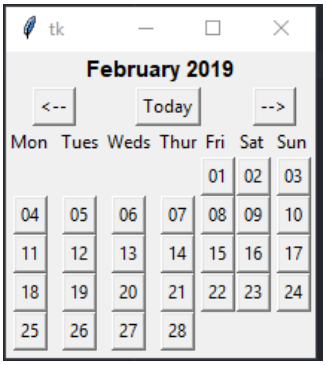
This function does everything I need it to, however when I tested it, I got a traceback error:

```
Traceback (most recent call last):
  File "C:\Users\rober\Documents\Computing CA\Code\DatePicker.py", line 98, in <module>
    y = datePicker()
  File "C:\Users\rober\Documents\Computing CA\Code\DatePicker.py", line 12, in __init__
    self.navButtonClick("t")
  File "C:\Users\rober\Documents\Computing CA\Code\DatePicker.py", line 50, in navButtonClick
    self.DisplayMonth(self.Year,self.Month)
  File "C:\Users\rober\Documents\Computing CA\Code\DatePicker.py", line 54, in DisplayMonth
    for x in self.DayButtons:
AttributeError: 'datePicker' object has no attribute 'DayButtons'
```

I realised that when I first open the date picker, I am trying to populate the form, however when I trace the algorithm I am deleting the buttons before creating them. This means there is nothing to delete, so it throws up an error. I have put that bit of the function in a try except statement. This means when it creates the date picker this error will not come up.

```
try:
    for x in self.DayButtons:
        x.destroy()
except:
    pass
```

Once I had done this, the date picker worked:



As you can see in **Video 2** in the supporting documents, this worked. I can move the date picker forwards and backwards, and when I press the today button, it skips to this month. The Title changed as I expected as well.

I used the test that I had created in my design process to make sure that everything was working as expected:

Selecting an date in the Mini Calendar

Test Data	Type	Actual
Left click on date	**Valid**	**Valid**
Right click on date	**Invalid**	**Invalid**
Dragging several Dates	**Invalid**	**Invalid**

This part of the project works well. I am now going to integrate it into the *mainpage* class. As my GUI diagrams show from my planning, it is going to be in the top left corner of the window.

The first thing I did was import the date picker, with the line `from DatePicker import datePicker`. The next thing I did was to create a function that will create the side bar:

```
def createSideBar(self):
    datePicker = datePicker(self)
    datePicker.grid(row=0,column=0)
```

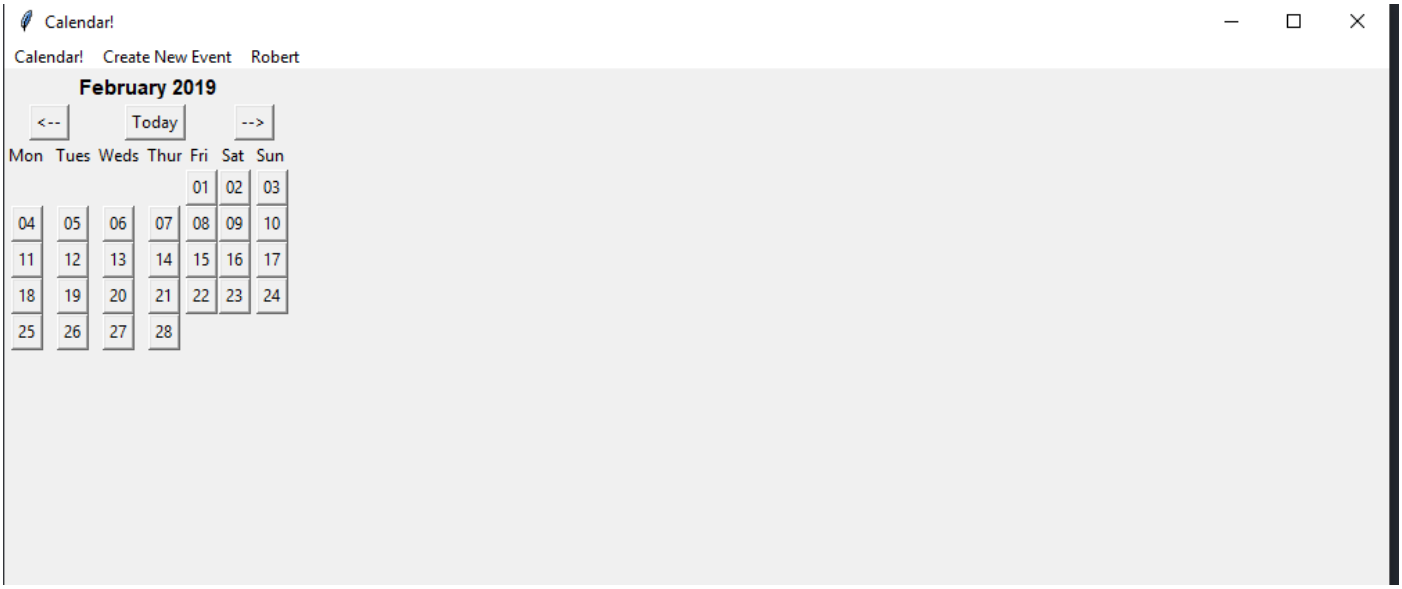
I also needed to call this in the *__init__* function, so I have added the line `self.createSideBar()`. I then tested the function, and it did not work, throwing an *'unboundLocalError'.*

```
  File "C:\Users\rober\Documents\Computing CA\Code\mainPage.py", line 37, in createSideBar
    datePicker = datePicker(self)
UnboundLocalError: local variable 'datePicker' referenced before assignment
```

What I have realised is that my variable *dataPicker* has the same name as the class *datePicker*. I have renamed the variable, so the function now looks like :

```
def createSideBar(self):
    datePickerSide = datePicker(self)
    datePickerSide.grid(row=0,column=0)
```

This change worked so the window now looks like this:



The next thing that I am going to create is the 'new event' form. Before I do this I want to check that both the create event and home buttons work. I used the tests that I produced in the planning stage to test them:

Create new event button

Test Data	Type	Actual
Left click on button	**Valid**	**Valid**
Right click on button	**Invalid**	**Invalid**
Click outside button	**Invalid**	**Invalid**

Home buttons pressed

Test Data	Type	Actual
Left click on button	**Valid**	**Valid**
Right click on button	**Invalid**	**Invalid**
Click outside button	**Invalid**	**Invalid**

These worked, so I am now going to start looking at the form. The 'new event' form needs to accept the following data about an event: title, description, location of an event, colour and start and end times. This is so that the program follows the following success criteria points:

No.	Requirements	Justification
17	Create event page	A form is needed to input data about new events.
18	An event must have a name and start date at a minimum, otherwise an error message is shown.	This is to prevent events ending before they start, and so that every event has a title.

27	User can change colour of event	This helps the user to easily see different events with colours
28	There are at least 5 colours that a user can choose from for an event	This helps the user to split their calendar into different colour to easily see what is going on.
29	The colour of an event defaults to the colour of the calendar	This means that a calendar can have a colour which all events default to
30	An event can be set to all day	Some events do not have times but you still need reminding

I have created some pseudocode for this form:

```
INCLUDE tkinter
INCLUDE database


CLASS CreateEvent (TYPE = TKINTER FRAME):

    FUNCTION __init__ (self, parent, controller):
        #Create  labels
        nameText = TKINTER LABEL (text="Name")
        startTimeText = TKINTER LABEL (text="Start Time")
        endTimeText = TKINTER LABEL (text="End Time")
        allDayText = TKINTER LABEL (text="All Day")
        DiscriptionText = TKINTER LABEL (text="Description")
        locationText = TKINTER LABEL (text="Location")
        colourText = TKINTER LABEL (text="Colour")

        #Create Text Boxes
        self.name = TKINTER ENTRY
        self.location = TKINTER ENTRY

        #Create Check Box
        self.allDay = TKINTER CHECK BOX

        #Create Dropdown
        self.colour = TKINTER DROPDOWN WITH OPTIONS (Red, Blue, Yellow,
Pink, Orange)

        #Create Text Area
        self.description = TKINTER TEXT AREA


        ClearForm = TKINTER BUTTON ( text="Back")
        ClearForm = TKINTER BUTTON ( text="Login", command =
self.ClearForm)
        CreateEventButton = TKINTER BUTTON ( text="Login", command =
self.createEvent)
```

```
    END FUNCTION

    FUNCTION createEvent(self):
        createEventCheck = Database.CreateEvent(Get Data from Form):
        IF createEventCheck == TRUE:
            self.CLEAR FORM
            GO BACK TO MAIN PAGE
        ElSE:
            DISPLAY ERROR MESSAGE
        END IF

    END FUNCTION

    FUNCTION clearForm(self):
        FOR element IN FORM ELEMENTS:
            element = ""
        END FOR

END CLASS
```

I have realised when I created the database, I needed to include columns for Description and Colour. I have changed the SQL of the events table to include these:

```
`Description`    TEXT,
`Colour`     INTEGER,
```

I then used a DBMS to recreate the table. These columns are now added I have now created the basics of the *createEvent* class. This code should just create all of the labels. It worked as shown below.

```
class createEvent(Frame):
    def  __init__ (self, parent, controller, database):
        Frame.__init__(self,parent)
        self.DB = database
        self.controller = controller

        #Create and Pack Labels
        nameText = Label(self, text="Name")
        nameText.grid(column=0, row=1)
        startTimeText = Label(self, text="Start Time")
        startTimeText.grid(column=0, row=2)
        endTimeText = Label(self, text="End Time")
        endTimeText.grid(column=0, row=3)
        allDayText = Label(self, text="All Day?")
        allDayText.grid(column=0, row=4)
        DescriptionText = Label(self, text="Description")
        DescriptionText.grid(column=0, row=5)
        locationText = Label(self, text="Location")
        locationText.grid(column=0, row=6)
        colourText = Label(self, text="Colour")
        colourText.grid(column=0, row=7)
```

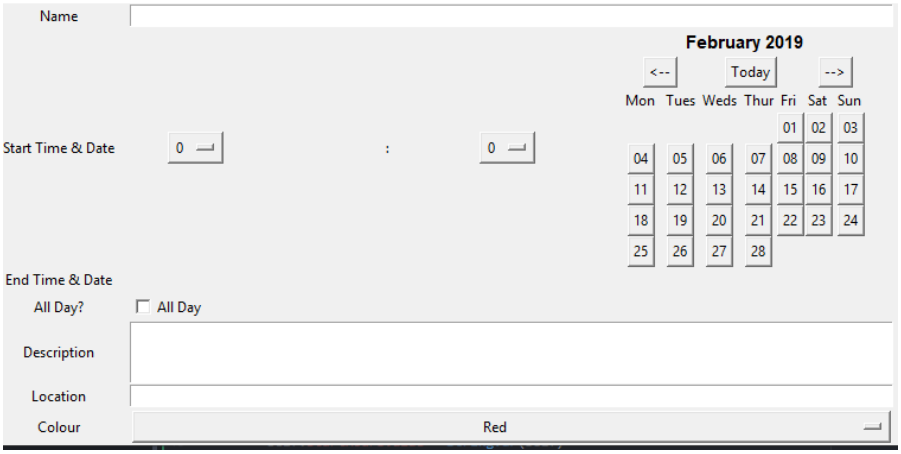
This is not a very efficient way of creating all the labels. I have rewritten the code so that it is less repetitive. I then tested it and it produced the same result. This way uses a “for loop” to run through all of the labels and then adds them to a list called *labels.*

```
#Create and Pack Labels
labelsText = ["Name","Start Time","End Time","All Day?","Description","Location","Colour"]
labels = []
row = 1
for text in labelsText:
    labels.append(Label(self, text=text))
    labels[-1].grid(column=0, row=row)
    row += 1
```

The next thing that I am going to look at is getting the fields in the form. I have created the code exactly as I had in the pseudocode:

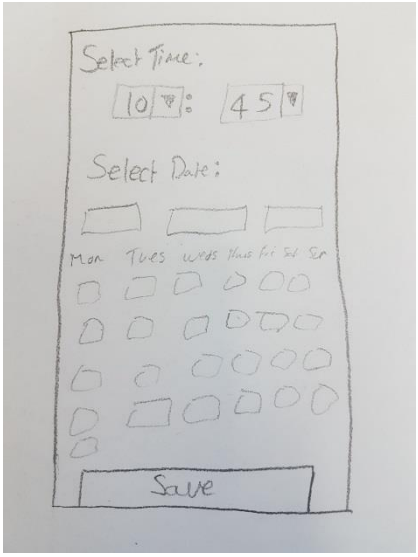
```
#Create Text Boxes
self.name = Entry(self)
self.name.grid(column=1, row=1, padx=10,sticky="ew", columnspan=5)
self.location = Entry(self)
self.location.grid(column=1, row=6, padx=10,sticky="ew", columnspan=5)

#Create Check Box
self.allDayValue = IntVar()
self.allDay = Checkbutton(self, text="All Day",variable=self.allDayValue)
self.allDay.grid(column=1, row=4, padx=10,sticky="w", columnspan=5)

#Create Text Area
self.description = Text(self,height=3)
self.description.grid(column=1, row=5, padx=10, sticky="ew", columnspan=5)

#Create Dropdown
colours = ["Red","Blue","Orange","Pink","Green"]
self.colourValue = StringVar(self)
self.colourValue.set(colours[0]) # initial value

self.colour = OptionMenu(self, self.colourValue, *colours)
self.colour.grid(column=1, row=7, padx=10, sticky="ew", columnspan=5)


#Start Time:
hours = list(range(0, 24))
mins = list(range(0, 60))

self.startHoursValue = StringVar(self)
self.startHoursValue.set(hours[0]) # initial value
self.startHours = OptionMenu(self, self.startHoursValue, *hours)
self.startHours.grid(column=1, row=2, padx=10)

colon = Label(self, text=":")
colon.grid(column=3, row=2)

self.startMinsValue = StringVar(self)
self.startMinsValue.set(mins[0]) # initial value
self.startMins = OptionMenu(self, self.startMinsValue, *hours)
self.startMins.grid(column=4, row=2, padx=10)

self.startDatePicker = datePicker(self)
self.startDatePicker.grid(column=5, row=2)
```

That code produced the following result:



I do not really like the way that the date time selector looks. I think that there is a better way of creating this using a popup window. This code also needs to be reused several time through the project (for example in the edit event form), so hard coding it in each form seems inefficient.

I have asked my stakeholder Ryan what he would prefer, the date time picker like it is at the moment or to have a popup window. He said "I think a popup window would be better because it keeps the page clean making it easier to see what dates I have chosen at a glance." I think I agree with Ryan with this. I then created a GUI Diagram for what the date picker will look like.



I asked Ryan what he thought of this design for the popup and he said "That looks really good, simple to use and easy to see what you are choosing. Much better than the other one." It is good that he and I both like it, so I am going to create it now.

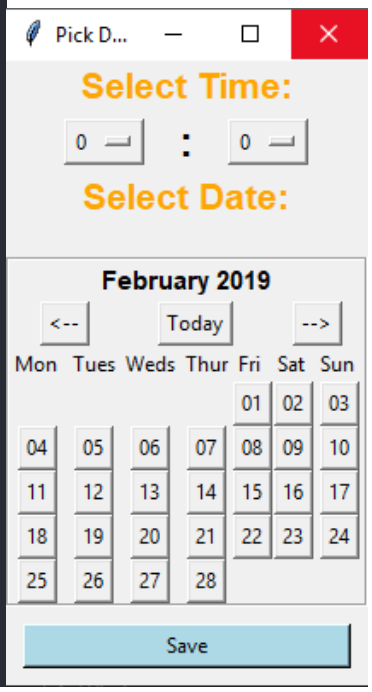
I do not know exactly how to create popups in TKinter so I am going to prototype this and see if I like it rather than properly plan the code like I have done with other modules. I am going to create a new class called *DateTimePopup.* This will have the following methods: *init*, *datePickerReturn* and *saveButtonPress.* The *init* function should create the popup and populate the form with a time selector and the date picker I have already created. The date *datePickerReturn* will handle what happens when the user selects a date on the date picker. The *saveButtonPress* will pass the information back to the form.

I have created the basics of the window, and *init* function. It will take an input called *outputFunction*. I will pass a function to the class so that when the save button is pressed this function is called, passing the information back.

```
from tkinter import *
from DatePicker import *
import datetime
from tkinter import messagebox


class DateTimePopup:

    def __init__(self, outputFunction):
        self.window = Toplevel()
        self.window.wm_title("Pick Date & Time")
        self.window.grab_set()
```

I then created all of the form elements and tested what it looked like.

```
self.outputVariable = outputFunction
self.date = None

hours = list(range(0, 24))
mins = list(range(0, 60))

selectTimeLabel = Label(self.window, text="Select Time:", font='Helvetica 16 bold', fg="orange")
selectTimeLabel.grid(column=0, row=0, columnspan=3,sticky="nsew")

self.startHoursValue = StringVar(self.window)
self.startHoursValue.set(hours[0]) # initial value
self.startHours = OptionMenu(self.window, self.startHoursValue, *hours)
self.startHours.grid(column=0, row=1,sticky="e")

colon = Label(self.window, text=":", font='Helvetica 18 bold')
colon.grid(column=1, row=1)

self.startMinsValue = StringVar(self.window)
self.startMinsValue.set(mins[0]) # initial value
self.startMins = OptionMenu(self.window, self.startMinsValue, *mins)
self.startMins.grid(column=2, row=1,sticky="w")

selectDateLabel = Label(self.window, text="Select Date:", font='Helvetica 16 bold', fg="orange")
selectDateLabel.grid(column=0, row=2, columnspan=3,sticky="nsew")

self.selectedDate = Label(self.window, text="", font='Helvetica 8')
self.selectedDate.grid(column=0, row=3, columnspan=3,sticky="nsew")

self.startDatePicker = datePicker(self.window, command=self.datePickerReturn, borderwidth=2, relief="groove")
self.startDatePicker.grid(column=0, row=4, columnspan=3)

SaveButton = Button(self.window, text="Save", bg="lightblue", command=self.saveButtonPress)
SaveButton.grid(column=0, row=5, padx=10, pady=10, sticky="nsew", columnspan=3)
```



What I now need to do is save the date when the user selects one in the date picker. In the *datePicker* class I have changed the day buttons so they look like this:

```
#Create and pack da button
self.DayButtons.append(Button(self, text=text, command=lambda day=int(text):self.dayButtonPressed(self.Year,self.Month,day)))
```

This creates a temporary variable called ‘day’, which is then passed to a new function I will create (along with the month and year) called *dayButtonPressed.* This function should call the command that the user has passed to a function. Because I do not know if the command passed to the function will work, I have put it in a try except statement to prevent it from throwing up any errors.

```
def dayButtonPressed(self,year,month,day):
    try:
        output = (year,month,day)
        self.command(output)
    except:
        pass
```

I then need the popup window to deal with the press of a day, so I have created a function called *datePickerReturn*, which will take the input from the *datepicker*, save the date the user selected to the popup and then output that date to a label so the user can see the date that is selected.

```
def datePickerReturn(self, dateClicked):
    self.date = dateClicked
    dateObject = datetime.datetime(self.date[0], self.date[1], self.date[2])
    text = dateObject.strftime("%A %d %B %Y")
    self.selectedDate.config(text=text)
```

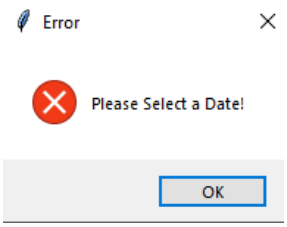
This function worked, and the date is now displayed when the user clicks a date in the *datepicker*.

When the user clicks save, the popup needs to pass all the data back to the *createEvent* form. To do this the *__init__* of the form takes an input called *outputFunction.* I then save this function to a variable called outputVariable within the class - `self.outputVariable = outputFunction`.

When the save button is the pressed I need to validate that everything in the form is safe and then call this function. The first thing that I do is get the values from the form. Because the hour and min are dropdowns, they can only be set values, so I do not need to validate these, just obtain them. I then make sure that a date has been selected. If not, I am going to use a TKinter error box to show an error message. Otherwise it will call the function that I pass to it. This code is shown below:

```
def saveButtonPress(self):
    hour = int(self.startHoursValue.get())
    min = int(self.startMinsValue.get())
    if self.date == None:
        messagebox.showerror("Error", "Please Select a Date!")
    else:
        output = self.date + (hour, min)
        try:
            self.outputVariable(output)
        except:
            pass
        self.window.destroy()
```

When I tested this it worked - if I had not selected a date it showed the error message:



Now that the *DateTimePopup* is working, I now need to integrate this into the 'create event' form. For start time and end time I need a button to open the *dateTime* form and then a text to display the time they have chosen. I have created this below:

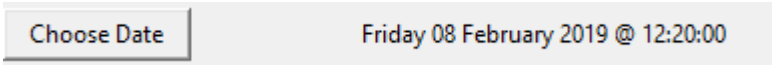
```
#Create Start Time Button / Popup
self.startTime = None
startTimeButton = Button(self, text="Choose Date", width=12, command=lambda: DateTimePopup(self.changeStartTime))
startTimeButton.grid(column=1, row=2, padx=10, pady=10, sticky="w")

#Start Time to de displayed once chosen
self.startTimeText = Label(self, text="")
self.startTimeText.grid(column=2, row=2, sticky="w")

#Create End Time Button / Popup
self.endTime = None
endTimeButton = Button(self, text="Choose Date", width=12, command=lambda: DateTimePopup(self.changeEndTime))
endTimeButton.grid(column=1, row=3, padx=10, pady=10, sticky="w")

#End Time to de displayed once chosen
self.endTimeText = Label(self, text="")
self.endTimeText.grid(column=2, row=3, sticky="w")
```

I then tested this to see what it looks like and it worked as shown.

I then created the functions that the *dateTime* is passed back to once the user selects them. I have two functions – one for start time and one for end time. Each takes the date and the time as an input and converts it to a date-time object, allowing python to manipulate it. It then displays this next to the button so the user knows what time they have selected.

```
def changeStartTime (self, newTime):
    self.startTime = newTime
    dateObject = datetime.datetime(newTime[0], newTime[1], newTime[2], newTime[3], newTime[4])
    text = dateObject.strftime("%A %d %B %Y @ %X")
    self.startTimeText.config(text=text)

def changeEndTime (self, newTime):
    self.endTime = newTime
    dateObject = datetime.datetime(newTime[0], newTime[1], newTime[2], newTime[3], newTime[4])
    text = dateObject.strftime("%A %d %B %Y @ %X")
    self.endTimeText.config(text=text)
```

I tested this and it displayed the date and time as I expected:



Now that I have all of the form elements I need a way to save them. I have first created a button to do this. Pressing this will call a function called *createEvent*.

```
createEventButton = Button(self, text="Create Event", width=12, command=self.CreateEvent)
createEventButton.grid(column=1, row=8, padx=10, pady=10, sticky="w")
```

I tested this and it was not working. I realised that it is because there is a class called *CreateEvent*. This was causing a name error because I had called my function *CreateEvent*. I have renamed this function to *CreateEventButtonPress*. The button code now looks like `command=self.CreateEventButtonPress)`.

The *CreateEventButtonPress* function needs to get all of the form elements, validate it, and then run a database function to save the event. The first thing that I am going to do is get the form elements:

```
def CreateEventButtonPress(self):
    #Get Form Info
    name = self.name.get()
    start = self.startTime
    end = self.endTime
    allDay = self.allDayValue.get()
    description = self.description.get("1.0",END)
    location = self.location.get()
    colour = self.colourValue.get()
```

Then I need to validate the form. I need to make sure that the minimum info is filled out, a name and start date. I also need to make sure that if there is an end time, that it is not after the start date – an event cannot end before it starts! If at any point there is an error, it should show an error message. I have created the validation code below:

```
#Validate Form
if name == "" or start == None:
    messagebox.showinfo("", "At a minimum an event needs a name and a start date.")
    return
StartUnixTime = time.mktime(datetime.datetime(start[0], start[1], start[2], start[3], start[4]).timetuple())

if allDay == 0:
    if EndUnixTime == None:
        messagebox.showinfo("", "No End Time Selected")
        return
    EndUnixTime = time.mktime(datetime.datetime(end[0], end[1], end[2], end[3], end[4]).timetuple())
    if EndUnixTime <= StartUnixTime:
        messagebox.showinfo("", "This event finishes before it starts! Please change either the start or end time.")
        return
```

I have also converted the start and end times into Unix time stamps. This gives me one number that I can input into the database, rather than storing it all individually. To test the validation I have created and filled out a testing table.

Test Num	Input Data	Reason for test	Expected Output	Actual Output	Pass/Fail
1	**Name**: “” **Start Date**: 1/1/2019 at **midnight**	No Name	“At a minimum an event needs a name and a start date.”	“At a minimum an event needs a name and a start date.”	Pass
2	**Name**: “Test Event” **Start Date**: “”	No Start Date		“At a minimum an event needs a name and a start date.”	Pass
3	**Name**: “Test Event” **Start Date**: “5/1/2019 at midnight” **End Date**: “1/1/2019 at midnight”	End Time Before Start Time	“This event finishes before it starts! Please change either the start or end time.”	“This event finishes before it starts! Please change either the start or end time.”	Pass
4	**Name**: “Test Event” **Start Date**: “1/1/2019 at midnight” **End Date**: “1/1/2019 at midnight”	Boundary Test – Event starts and ends at the same time.		“This event finishes before it starts! Please change either the start or end time.”	Pass
5	**Name**: “Test Event” **Start Date**: “1/1/2019 at midnight” **End Date**: “1/1/2019 at 1 AM”	Normal Use	N/A (Nothing programmed Yet)	N/A	Pass

This validation works. The next thing I need to do is create a function in the class *database* to create the event. I have started of by creating the SQL for this statement:

```
INSERT INTO Events ('Name','StartTimestamp', 'EndTimestamp', 'Location', 'AllDay', 'Description', 'Colour', 'CalendarID') VALUES (?,?,?,?,?,?,?,?);
```

In creating this statement, I have realised that when the user creates an account I also need to create a default calendar for them, otherwise the database will not work, as it is relational.

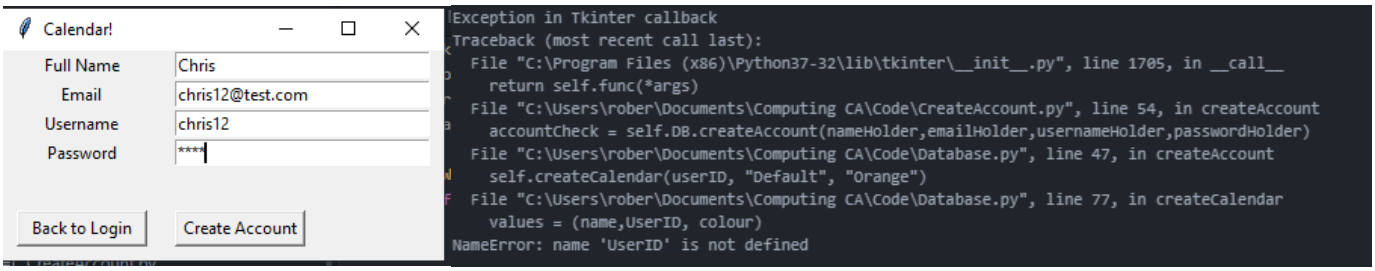
I have created the flowing function to create a calendar:

```
def createCalendar(self, userID, name, colour):
    #Create Values List, and SQL statement
    values = (name,UserID, colour)
    sql = "INSERT INTO Calendars ('Name','UserID', 'Colour') VALUES (?,?,?);"

    #Execute SQL
    self.cur.execute(sql,values)
    self.con.commit()
```

This creates a calendar from the data that is passed to it and saves it to the database. I then created the following code to the *createAccount* function in the *database* class.

```
userID = self.checkLogin(username,password)[1]
self.createCalendar(userID, "Default", "Orange")
```

This gets the users name, and the creates a calendar called default, with a default colour of orange. I tested the function with the flowing data, however I got an error:



It turns out I had added an extra capital letter in the line where the values are turned into a list in the *createCalendar* Function. It now looks like `values = (name,userID, colour)`. I then tested the code with the same input.

This created both a user and a calendar in the database:

Users Table:

18	Chris	chris12@test....	chris12	90a3ed9e32b...

Calendar Table:

1	Default	Orange	18

I have now edited the *getUserData* function to include information about all the calendars the user has associated with them.

```
#Get the calendars from a particular user
sql = "SELECT ID ,Name,Colour FROM Calendars WHERE UserID=?"
self.cur.execute(sql, (userID,))
CalendarRows = self.cur.fetchall()
```

This gets all the calendars that a user is associated with. I have then changed this function to return this data with the user info: `return list(user) + list([CalendarRows])`. I tested this function and it now outputs correctly:

```
['Chris', 'chris12@test.com', 'chris12', [(1, 'Default', 'Orange'), (2, 'Test', 'Pink')]]
```

Now that I have created this, I have gone through and added calendars for all the accounts that I already have. I have now used the SQL I created above to create the *createEvent* function in the database class. As shown below:

```
def createEvent(self, name, startTime, endTime, allDay, description, location, colour, calendarID):
    #Create Values list, and SQL statement
    values = (name, startTime, endTime, allDay, description, location, colour, calendarID)
    sql = "INSERT INTO Events ('Name','StartTimestamp', 'EndTimestamp', 'Location', 'AllDay', 'Description', 'Colour', 'CalendarID') VALUES (?,?,?,?,?,?,?,?);"

    #Execute SQL
    self.cur.execute(sql,values)
    self.con.commit()
```

When the user presses the button to save the event, I then need to call this function. In the code below, I first get the default calendar ID and then I call the function.

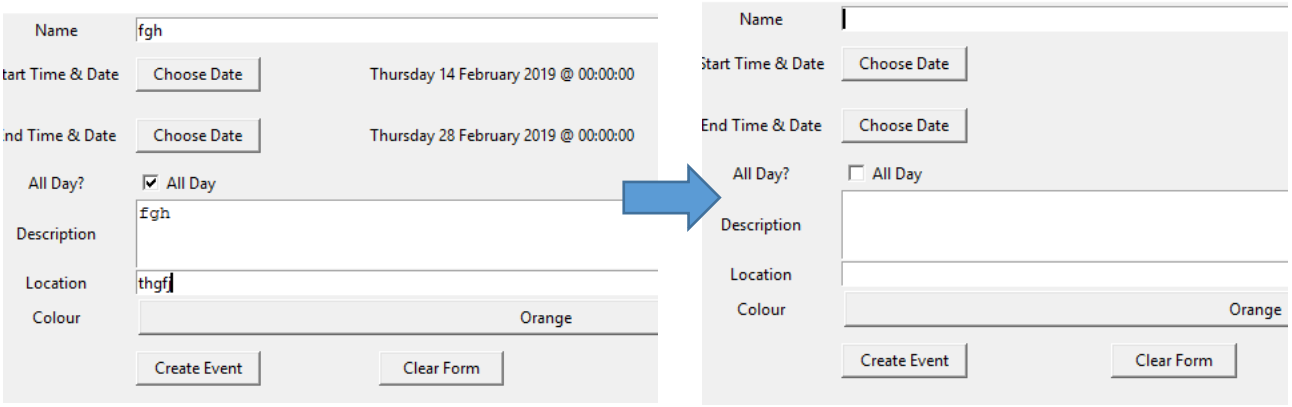
```
#Get Default Calendar ID
calendarID = self.controller.USER_DETAILS[3][0][0]

#Create Event
self.DB.createEvent(name,start,end,allDay, location, colour, calendarID)
```

Once it is created, I then want to clear the form and go back to the *mainpage*. I have created a function called *clearForm*, to clear the form. I also have added a button to clear the form. This is partly to test the function and also, the user might want to clear the form manually, so I am going to leave it showing.

```
clearFormButton = Button(self, text="Clear Form", width=12, command=self.clearForm)
clearFormButton.grid(column=2, row=8, padx=10, pady=10, sticky="w")
```

```
def clearForm(self):
    self.name.delete(0,END)
    self.name.focus_set()
    self.startTime = None
    self.endTime = None
    self.startTimeText.config(text="")
    self.endTimeText.config(text="")
    self.allDay.deselect()
    self.description.delete("1.0",END)
    self.location.delete(0,END)
    self.colourValue.set(self.controller.USER_DETAILS[3][0][2])
```

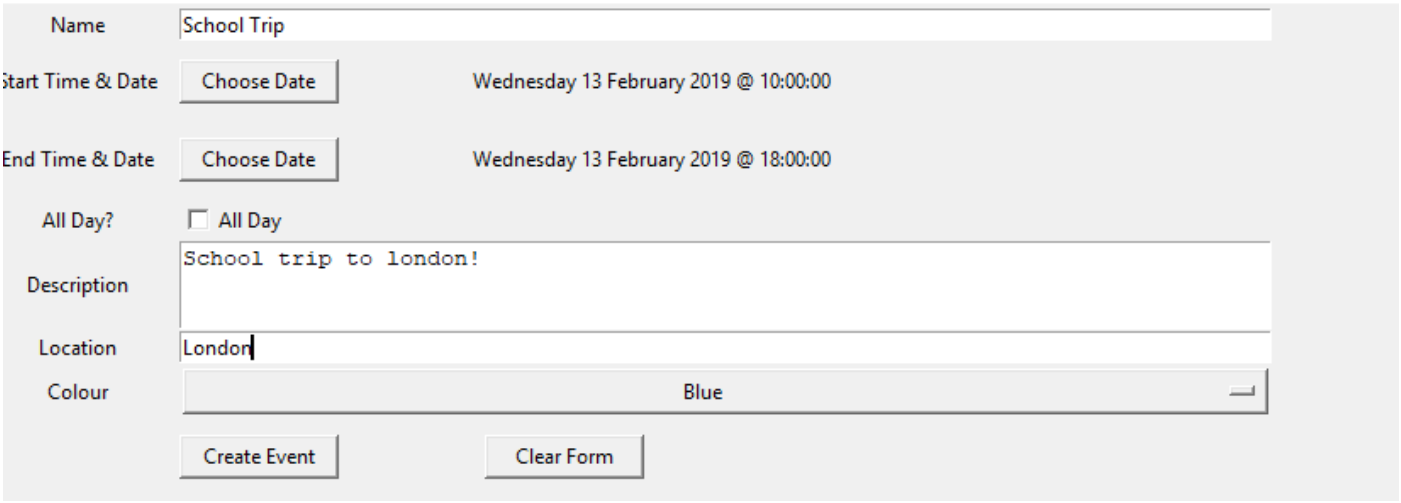
I then tested the clear form function, by pressing the button, and it worked, clearing the form properly.



I now need to make it so that once an event has been created, it clears the form and then changes the frame to the main page. I have done this with the code below:

```
#Clear Form and change frame
self.clearForm()
self.controller.show_frame("mainPage")
```

I am now going to test everything I have done (adding the event SQL, the validation and the changing frame). I have added the flowing data to the form:



When I ran it I got a type error:

```
TypeError: createEvent() missing 1 required positional argument: 'calendarID'
```

. This is because when I called the *createEvent* function from the database class, I realised that I needed to pass it the description as well, so I was one argument short. This line now looks like:

```
#Create Event
self.DB.createEvent(name,start,end,allDay, description, location, colour, calendarID)
```

I then tested the form with the same data. It again did not work, I was getting an SQL error. After a lot of testing I realised that I was passing the wrong start and end time to the function. I was passing the numbers in a list rather than the Unix time that the database is expecting:

```
(2019, 2, 6, 0, 0)
(2019, 2, 9, 11, 0)
```

This means the database was not letting me enter the data as I am not passing the correct data to the function. I have now changed the call again so it looks like:

```
self.DB.createEvent(name,StartUnixTime,EndUnixTime,allDay, description, location, colour, calendarID)
```

I tested this and it went back to the main page as expected. (**This is shown in video 3 in the supporting documents**) When I went back to the form it had cleared / reset it. I then checked the database to make sure everything was there as expected.

ID	Name	StartTimestamp	EndTimestamp	Location	AllDay	CalendarID	Description	Colour
Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	School Trip	1550052000	1550080800	London	0	2	School Trip to...	Blue

All the data was there as expected so this form works. I am now going to use the test data that I created in the planning stages to make sure that it all works:

Create New Event Validation

Test Data	Type	Actual
Name, start time filled in and all day checked	**Valid**	**Valid**
Name, start time, and end time filled in	**Valid**	**Valid**
Name not filled in	**Invalid**	**Invalid**
Start time not filled in	**Invalid**	**Invalid**

Test Data	Type	Actual
Name: “Lunch 1” Start time: “1/1/2019 @ 15:00” End Time: “1/1/2019 @ 17:00”	**Valid**	**Valid**
Name: “Lunch 2” Start time: “1/1/2019 @ 15:00” End Time: “2/1/2019 @ 17:00”	**Valid**	**Valid**
Name: “School 1” Start time: “1/1/2019 @ 15:00” All Day: Checked	**Valid**	**Valid**
Name: “Lunch 3” Start time: “1/1/2019 @ 15:00” End Time: “1/1/2019 @ 12:00”	**Invalid**	**Invalid**
Name: “School 2” Start time: “1/1/2019 @ 15:00” All Day: Unchecked	**Invalid**	**Invalid**

Create New Event Form Complete

Test Data	Type	Actual
Left Click on create account Button	**Valid**	**Valid**
Right Click on event	**Invalid**	**Invalid**
“enter key”	**Invalid**	**Invalid**
Left Click on ‘Back’ Button	**Invalid**	**Invalid**

All these tests worked so I am happy to move on. I spoke to my stakeholder Ryan about this form and he said “Creating an event was really easy. I think it was the right decision to have the event popup rather than to have all of the data on one form. I think that it keeps the page looking cleaner. I wish there were a few more colours to choose from for the events, as it is fairly limited at the moment. Overall it is very simple and quick to create an event.”

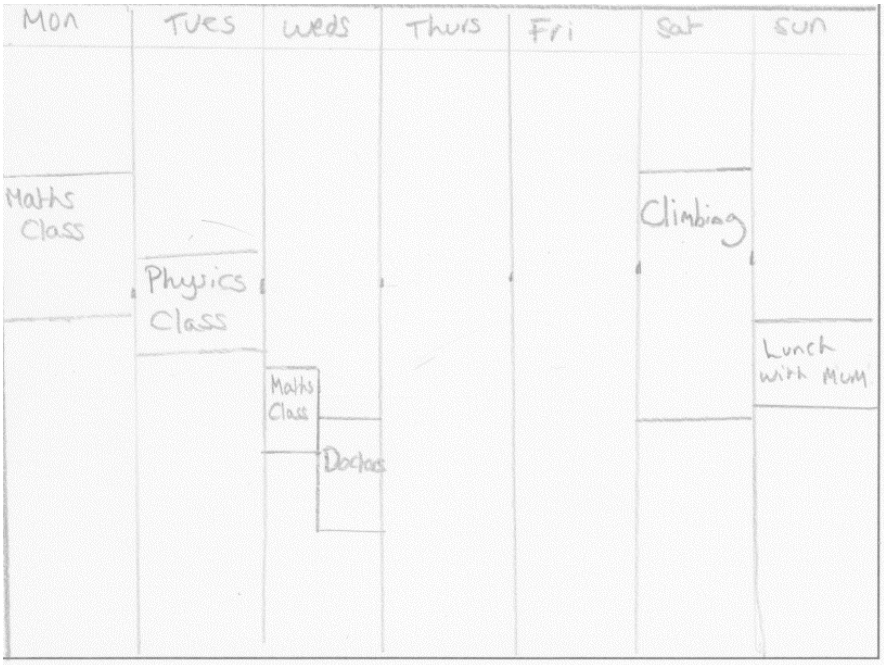
Once I have all of the functionality working, I am going to work on the appearance of the product, and as part of this I will add more colours to the dropdown. I agree with him; it was the right decision to have the popup rather than having the date and time picker on the same page as the create event page. It makes sense to have it separated.

Session 5 – 12/02/2019

The first thing that I am going to do today is design the event viewer. This part of the program is going to follow the following success criteria:

No.	Requirements	Justification
7	A view to see all the events that week	That allows the user to easily see what the is happening in their life that week
8	The week’s events should be split into days and hours in the view on the main page	This is to easily see when events start
9	When viewing events on the main page, the user should see the colour of the event and the name.	This helps the user to easily see different types of events with colour, and can see basic information about that event from the name.

This should be a screen that displays one weeks worth of events. In the design process I designed what it should look like:



I asked my stakeholder Ryan what he thought of this design and he said "It looks good. I think it is important that when two events overlap, like on Wednesday, they change size. This makes easy to see what you are doing on a day as an overview, as you don't have to show or hide events that overlap." I agree with him and I am going to make sure that his happens.

I am now going to create the pseudocode.

```
INCLUDE tkinter, clanedar, datetime

CLASS EventView (tpye = TKINTER Canvas):
    FUNCTION __init__ (self, WIDTH = 150  HEIGHT = 30):

        self.WIDTH = WIDTH
        self.HEIGHT = HEIGHT

        TKINTER INITALISE Canvas

    END FUNCTION

    FUNCTION createGrid (self):
        #Create Vertical Grid Lines
        verticalLength = (HEIGHT * 24 )
        x = self.GRID_START_CORDINATE_X
        y = self.GRID_START_CORDINATE_Y
        for line in range(0, 8):
            self.TKINTER CREATE LINE(x,y,x,y+verticalLength)
            x += self.WIDTH

        #Add Horizontal Grid Lines
        horizontalLength = (WIDTH * 7)
        x = self.GRID_START_CORDINATE_X
        y = self.GRID_START_CORDINATE_Y
```

```
        for line in range(0, 25):
            self.TKINTER CREATE LINE(x, y, x+horizontalLength, y)
            y +=  self.HEIGHT

    END FUNCTION

   FUNCTION createLabels(self)
        #Add Day Labels
        x = self.GRID_START_CORDINATE_X
        y = 10
        days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
"Saturday", "Sunday"]
        for day in days:
            self.TKINTER CREATE WINDOW(x+(WIDTH/2),y, width=self.WIDTH,
window=Label(text=day))
            x += self.WIDTH

        #Add Time Labels
        x = 30
        y = self.GRID_START_CORDINATE_Y + self.HEIGHT

        for time in range(1,24):
            text=str(time) + ":00"
            self.TKINTER CREATE WINDOW(x, y, window=Label(text=text))
            y +=self.HEIGHT

    END FUNCTION


    FUNCTION displayEvent(self, year, month, day):

        startTimestamp = WORK OUT TIMESTAMP FOR DATE (year, month, day)
        endTimestamp = startTimestamp + 7 DAYS

        events = DATABASE.GetEVENTS (startTimestamp,  endTimestamp)

        FOR EVENT in EVENTS:
            CRETATE RECTANGLE ON CORRECT DAY

    END FUNCTION

  FUNCTION clearCanvas(self):
      self.delete("ALL")
```

Using this I am going to create a new file called *EventView.py.* I have started by creating a basic *__init__* function:

```
class EventViewer (Canvas):

    def __init__(self, master, width =150, height=200, ** kw):

        #Define Grid Height
        self.WIDTH = width
        self.HEIGHT = height

        Canvas.__init__(self,master, height=(self.HEIGHT * 25), width=(self.WIDTH * 8),**kw)
```

I have then created the canvas in the *mainPage* with the following code:

```
eventViewerHolder = EventViewer(self, bg="red")
eventViewerHolder.grid(row=0,column=1)
```

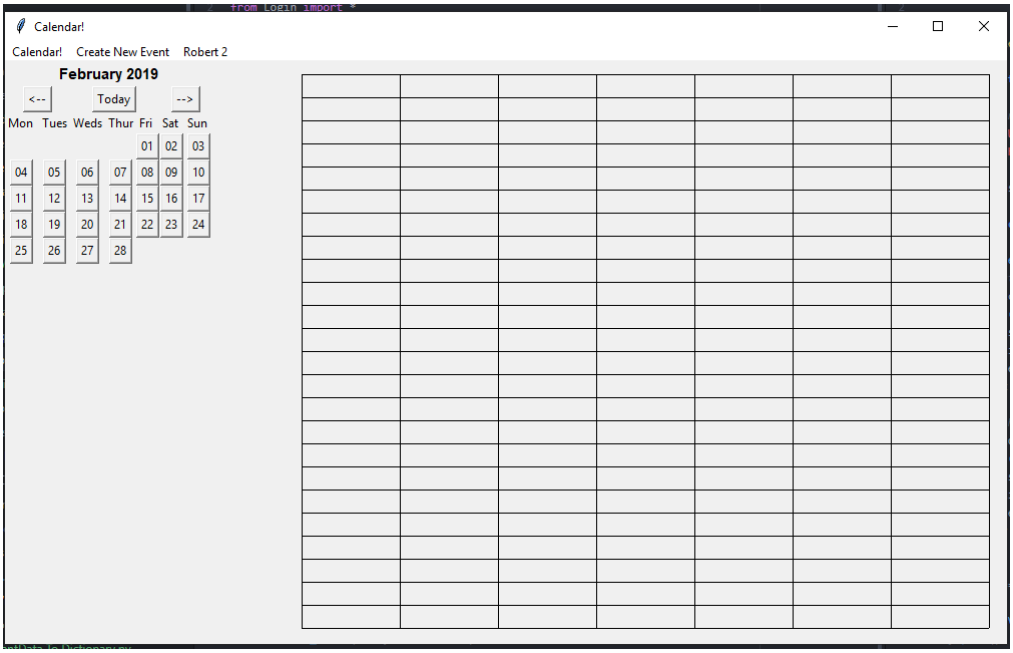
In this I have temporarily created the background red. This is to help me to get the default sizing correct in the window. This makes the page looks like:

Using this I have determined that the correct height and width for the canvas is 23 and 98 respectively. I have changed the *__init__* statement accordingly: `width =98, height=23,`. The next thing I am going to do is create the grid for all of the days.

The *x* and *y* values are the starting values (top left corner) of the grid. The *verticalLength* and *HoritonatalLength* is then the height and width of the grid. For each horizonal and vertical lines it the loops several times to create the grid.

```
def createGrid(self):
    #Create Vertical Grid Lines
    verticalLength = (self.HEIGHT * 24 )
    y = ((self.HEIGHT * 25)/2) - ((self.HEIGHT * 24 )/2)
    x = (self.WIDTH * 8) - (self.WIDTH * 7.1 )
    for line in range(0, 8):
        self.create_line(x,y,x,y+verticalLength)
        x += self.WIDTH

    #Add Horizontal Grid Lines
    horizontalLength = (self.WIDTH * 7)
    y = ((self.HEIGHT * 25)/2) - ((self.HEIGHT * 24 )/2)
    x = (self.WIDTH * 8) - (self.WIDTH * 7.1 )
    for line in range(0, 25):
        self.create_line(x, y, x+horizontalLength, y)
        y +=  self.HEIGHT
```

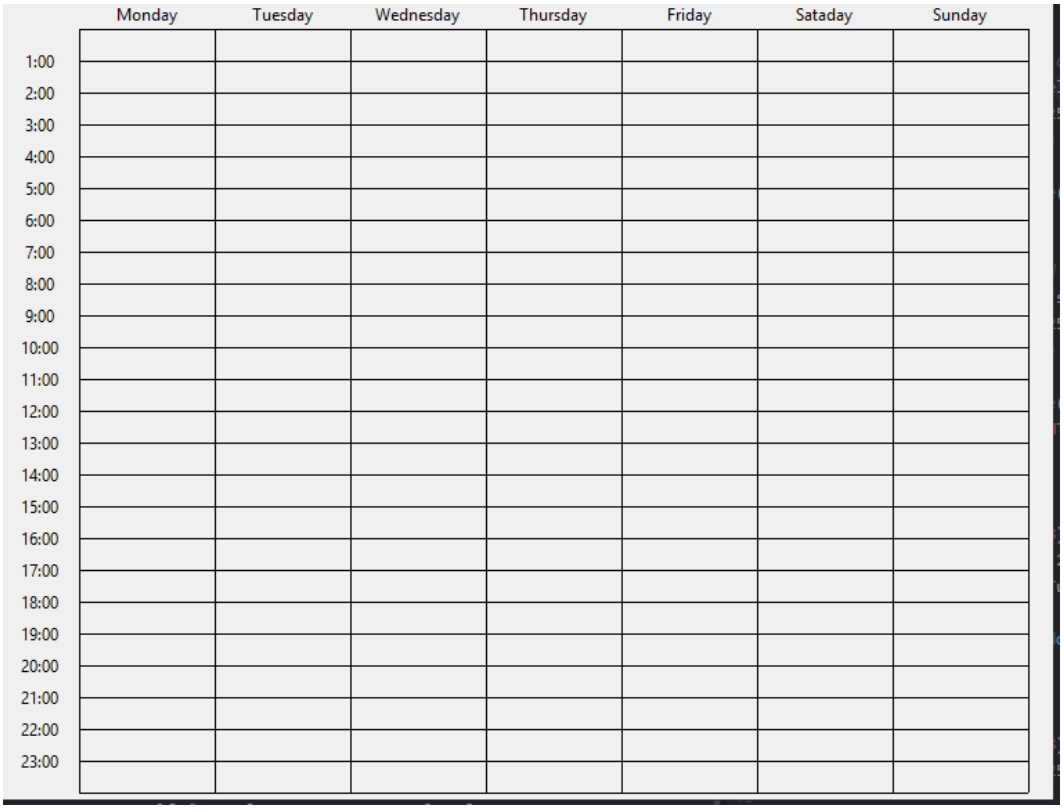
I tested this code and I successfully made a grid of days.



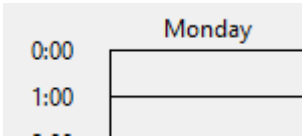
The next thing that I need is to create the labels. In this case, x and y are initially the centre of the first label on each axis. It then loops adding the labels in the correct position.

```
def createLabels(self):
    #Add Day Labels
    x =  (self.WIDTH * 8) - (self.WIDTH * 7.1 )
    y = ((self.HEIGHT * 25) - (self.HEIGHT * 24 ))/2  #OFfset From Top
    days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Sataday", "Sunday"]
    for day in days:
        self.create_window(x+(self.WIDTH/2),y, width=self.WIDTH, window=Label(text=day))
        x += self.WIDTH

    #Add Time Labels
    x = ((self.WIDTH * 8) - (self.WIDTH * 7.1 ))/1.5    # Offset from Side
    y = (self.HEIGHT * 25) - (self.HEIGHT * 24 ) + self.HEIGHT

    for time in range(1,24):
        text=str(time) + ":00"
        self.create_window(x, y, window=Label(text=text))
        y +=self.HEIGHT
```

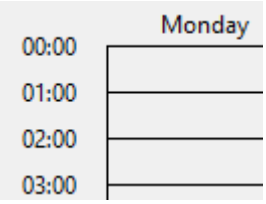
This worked as shown below, however I have not printed midnight (00:00) on the time axis. I am going to fix this as well as making the time be zero padded – 01:00 instead of 1:00.



To add 00:00 I have changed the y starting starting coordinate to `y = (self.HEIGHT * 25) - (self.HEIGHT * 24 )` and then I changed the range of the loop to include 0: `for time in range(0,24):`. I tested this and it worked:



I then wanted to zero pad the times. To zero pad the times I have found a python function called *zfill.* This built in function automatically fills adds in the zero for me. I have added this to the text variable now looks like: `text=str(time).zfill(2) + ":00"`. I tested this function and it worked:



Next I am going to create is the *displayEvents* function, however, before I do this I need to be able to pull events from the database. To do this I have created an SQL statement that I will use to create a function in the *database* class.

```
SELECT * FROM Events WHERE (StartTimestamp BETWEEN ? AND ?) AND
CalendarID=?
```

This will find all of the events that start between two dates, and are in a particular calendar. I used this to create a function called *getEvents:*

```
def getEvents(self, startTimestamp, endTimestamp, calendarID):
    #Get the calendars from a particular user
    sql = "SELECT * FROM Events WHERE (StartTimestamp BETWEEN ? AND ?) AND CalendarID=?"
    self.cur.execute(sql, (startTimestamp,endTimestamp,calendarID))
    Rows = self.cur.fetchall()
    return Rows
```

This function takes three inputs: the start time, the end time and the calendar ID. It then creates and executes the SQL statement returning all the rows that meet the conditions. I then called the function with the following code: `print(db.getEvents(1550041100, 1550397700,1))`. This produces the correct output (below), so this is working:

```
[(4, 'Test 1', 1550041200, 1550091600, '', 0, 1, '1 Day\n', 'Orange'), (5, 'Test 2', 1550102400, 1550188740, '', 0, 1, '1 Day Boundary\n', 'Orange'), (6, 'T
est 3', 1550188800, None, '', 1, 1, 'All Day 1', 'Orange'), (7, 'Test 4', 1550188800, 1550361600, '', 1, 1, 'Multi All Day\n', 'Orange'), (8, 'Test 5', 1550
217600, 1550397600, '', 0, 1, 'Multi Day - Non All Day\n', 'Orange')]
```

I am then going to use this function in the event view. I am going to create a new function within this class called *displayEvent*. This function needs to get the events from the users calendars, split them into all day events and normal events, and then put it in a form that makes it easier to do calculations on. For example, having the timestamp makes it really hard to work out where the event needs to go in the grid. For this reason, I am going to store the day as a number (1-7) and the start and end times as a 24 hour number (E.G. 1145). This will mean that when it comes to drawing the events on the canvas it makes the maths a lot easier to work out positions.

The first thing that the function needed to do was get the events from the database. I am going to take an input of day, year and month into the function, convert this into a timestamp to extract the dates from. I have done this with the following code:

```
def displayEvent(self, year, month, day):
    #Work Out Start TImestamp
    startTime = datetime.datetime(year, month, day, 0,0)
    startTimestamp = time.mktime(startTime.timetuple())

    #Work End Start TImestamp 6 Days Later
    endTime = startTime + datetime.timedelta(days=6,hours=23,minutes=59, seconds=59)
    endTimestamp = time.mktime(endTime.timetuple())
```

This calculates the first timestamp from the input to the function, then it calculates the timestamp 6 days, 23 hours, 59 mins and 59 second later. This is the end of the 7 days that it is going to display. The next thing the function needs to do is get the events. Before I can do that I need to get all of the users calendars. In the *__init__* function of the display event class I have added the following code:

```
self.USERID = userID
self.CALENDARS = self.DB.getUserData(userID)[3]
```

This gets a list of all of the calendars a user has and saves it to a class variable. The next thing is to get all the events. I am going to loop through all of the calendars in *self.CALENDARS* and then add all of the events in that time range to an (initially) empty list called events.

```
#Get Events from all user calendars
events = []
for calendar in self.CALENDARS:
    events += self.DB.getEvents(startTimestamp, endTimestamp, calendar[0])
```

I tested all of this and I go the correct output:

```
[(2, 'Test 5.1', 1550314800, 1550350800, '', 0, 1, '\n', 'Blue'), (6, 'Test 3',
1550188800, None, 'Testing', 1, 1, 'All Day 1', 'Orange'), (7, 'Test 4', 1550188
800, 1550361600, '', 1, 1, 'Multi All Day\n', 'Pink'), (8, 'Test 5', 1550217600,
 1550397600, '', 0, 1, 'Multi Day - Non All Day\n', 'Orange'), (10, 'Test 6', 15
50275200, 1550620800, '', 1, 1, '\n', 'Green')]
```

The next thing that I am going to do is convert this into the more usable list, with day numbers etc. This is not really in the pseudocode that I created – I did not do it very detailed for this part – I just said "CRETATE RECTANGLE ON CORRECT DAY" So I am going to have to think about this before I create it. I want to be able to take an output from the function above like: (2, 'Test 5.1', 1550314800, 1550350800, '', 0, 1, '\n', 'Blue'), and turn it into a dictionary that looks like:

{
"ID":2
"name": "Test 5.1",
"day": 3,
"startTime": 1300,
"endTime": 1700,
"location": "",
"calendarID": 1
"description": "\n",
"colour": "Blue"
}

If an event goes over multiple days I need to do one of these for every day the event goes over.

To achieve all this I need to do three things. I first need to remove all of the all day events and put them in a separate list (in the format above). I then need to work out how many days an event goes across. The final thing is put it in that form and save it to a list.

The first thing that I have done is add the events that are all day to the all day list, in this form:

```
eventsToGrid = []
allDayEvents = []
#For every Event
for event in events:

    #If Event is Allday
    if event[5] == True:

        #Work Out Start Day
        startDay = int((event[2] - startTimestamp) / 86400 ) + 1

        #If Event is only on one day
        if event[3] == None:
            endDay = startDay
        else:
            endDay = int((event[3] - startTimestamp) / 86400) + 1

        #Length of event in days
        eventLength = (endDay - startDay) +1

        #Split event into days
        for x in range(startDay,startDay + eventLength):
            if x <= 7:
                allDayEvents.append({
                    "ID":event[0],
                    "name":event[1],
                    "day": x,
                    "location":event[4],
                    "calendarID":event[6],
                    "description":event[7],
                    "colour":event[8]
                })
```

I then tested this and got the following output for all day events.

```
[{'ID': 6, 'name': 'Test 3', 'day': 1, 'location': 'Testing', 'calendarID': 1, '
description': 'All Day 1', 'colour': 'Orange'}, {'ID': 7, 'name': 'Test 4', 'day
': 1, 'location': '', 'calendarID': 1, 'description': 'Multi All Day\n', 'colour
': 'Orange'}, {'ID': 7, 'name': 'Test 4', 'day': 2, 'location': '', 'calendarID'
: 1, 'description': 'Multi All Day\n', 'colour': 'Orange'}, {'ID': 7, 'name': 'T
est 4', 'day': 3, 'location': '', 'calendarID': 1, 'description': 'Multi All Day
\n', 'colour': 'Orange'}, {'ID': 10, 'name': 'Test 6', 'day': 2, 'location': '',
 'calendarID': 1, 'description': '\n', 'colour': 'Orange'}, {'ID': 10, 'name': '
Test 6', 'day': 3, 'location': '', 'calendarID': 1, 'description': '\n', 'colour
': 'Orange'}, {'ID': 10, 'name': 'Test 6', 'day': 4, 'location': '', 'calendarID
': 1, 'description': '\n', 'colour': 'Orange'}, {'ID': 10, 'name': 'Test 6', 'da
y': 5, 'location': '', 'calendarID': 1, 'description': '\n', 'colour': 'Orange'}
, {'ID': 10, 'name': 'Test 6', 'day': 6, 'location': '', 'calendarID': 1, 'descr
iption': '\n', 'colour': 'Orange'}]
```

I then moved the start and end day calculations out of the if statement as they are going to need to be done regardless of whether the event is all day or not. This helps to make the code more efficient, because without doing this there would have been duplication of code.

The next thing that I am going to do is put normal events in this type of list. This needs to split up events that go over multiple days up. I do this by creating a list called *splitEvent*, which holds all the days an event covers, with the start and end times on those days.

```
else:
    #Get start and end Times
    startTime = int(datetime.datetime.fromtimestamp(event[2]).strftime("%H%M"))
    endTime = int(datetime.datetime.fromtimestamp(event[3]).strftime("%H%M"))

    #Event split into the individual day (if multi day)
    splitEvent = []
    if startDay == endDay:
        #Add a single day event
        splitEvent.append([startDay,startTime, endTime])
    else:
        splitEvent.append([startDay,startTime, 2400])
        if endDay <= 7:
            splitEvent.append([endDay,0000, endTime])

        for y in range(startDay+1,(startDay + eventLength)-1):
            if y <= 7:
                splitEvent.append([y,0000, 2400])

    #For every part of an event add it to the eventsToGrid list
    for part in splitEvent:
        eventsToGrid.append({
            "ID":event[0],
            "name":event[1],
            "day": part[0],
            "startTime": part[1],
            "endTime": part[2],
            "location":event[4],
            "calendarID":event[6],
            "description":event[7],
            "colour":event[8]
        })
```

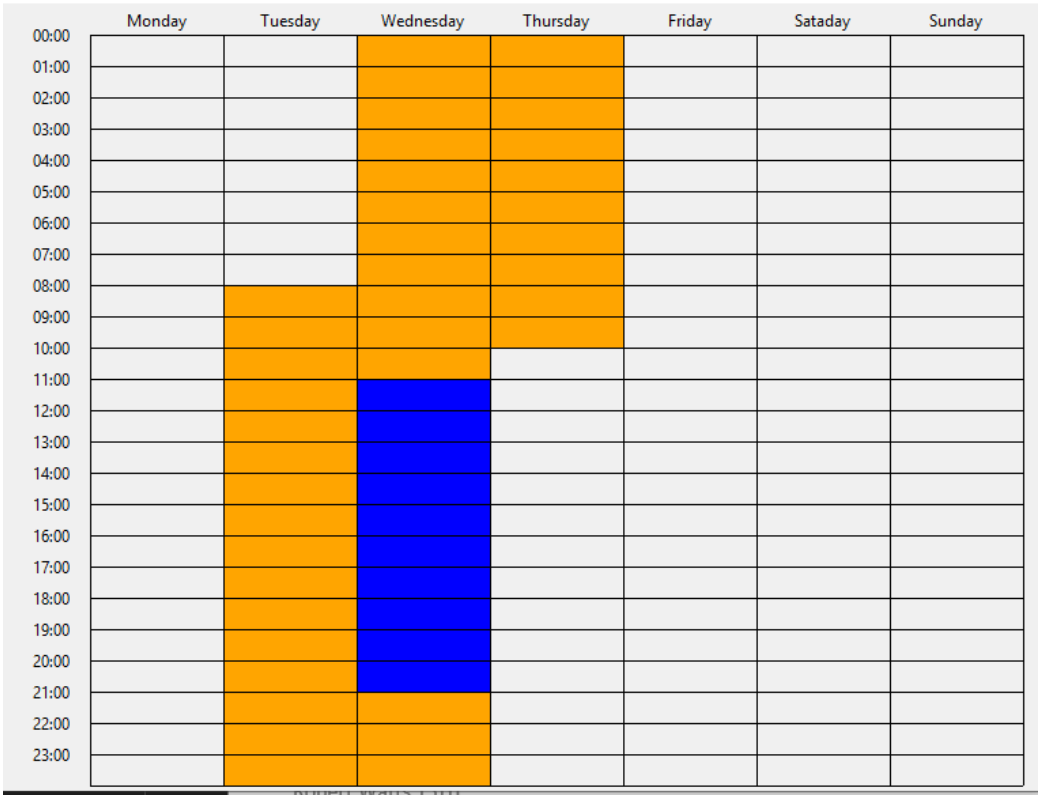
I tested this and it worked as I expected. It has split all of the multi day events up.

```
[{'ID': 8, 'name': 'Test 5', 'day': 1, 'startTime': 800, 'endTime': 2400, 'location': '', 'calendarID':
1, 'description': 'Multi Day - Non All Day\n', 'colour': 'Orange'}, {'ID': 8, 'name': 'Test 5', 'day':
3, 'startTime': 0, 'endTime': 1000, 'location': '', 'calendarID': 1, 'description': 'Multi Day - Non Al
l Day\n', 'colour': 'Orange'}, {'ID': 8, 'name': 'Test 5', 'day': 2, 'startTime': 0, 'endTime': 2400, '
location': '', 'calendarID': 1, 'description': 'Multi Day - Non All Day\n', 'colour': 'Orange'}, {'ID':
9, 'name': 'Test 5.1', 'day': 2, 'startTime': 1100, 'endTime': 2100, 'location': '', 'calendarID': 1, '
description': '\n', 'colour': 'Orange'}]
```


I now need to draw all of these events on the canvas. For every event I need to draw a rectangle. Each rectangle needs two coordinates, top left corner and bottom right corner. To calculate the first coordinate (x1 & y1), I take the corner of the grid, and add the day multiplied by the width of each day before the event. This gives me the x1. Y1 is the corner of the grid, added to the time.

The second set is just the first two with the width and duration added to it, I then draw each rectangle. I created this code and it didn't work initially – I had some BIDMAS errors – so I added some brackets and everything worked (I have also put it in a new function called *drawEvents*):

```
for event in eventsToGrid:
    GRID_START_CORDINATE_Y = (self.HEIGHT * 25) - (self.HEIGHT * 24 )
    GRID_START_CORDINATE_X = (self.WIDTH * 8) - (self.WIDTH * 7.1 )

    x1 = GRID_START_CORDINATE_X + ((event["day"]-1)*self.WIDTH) + ((self.WIDTH))
    y1 = GRID_START_CORDINATE_Y + ((event["startTime"]/100)*self.HEIGHT)
    x2 = x1 + (self.WIDTH)
    y2 = GRID_START_CORDINATE_Y + ((event["endTime"]/100)*self.HEIGHT)


    self.create_rectangle(x1, y1, x2, y2, fill=event["colour"])
```



This piece of code worked, however the events overlap, which is something that my stakeholder, Ryan, said he did not want when I spoke to him at the beginning of this session. To separate them I need to work out how many events there are at any given moment, and then if multiple events overlap I need to make them smaller. I created the following code to do this.

I loop through each day just before the code `for day in range(1,8):` and create the following variables:

```
eventsToday = []
eventDepth = [0] * 1440
```

Event depth is a list that has 1440 zeros in it, one for every minute in a day. This can then be incremented accordingly to see if there is events overlapping.

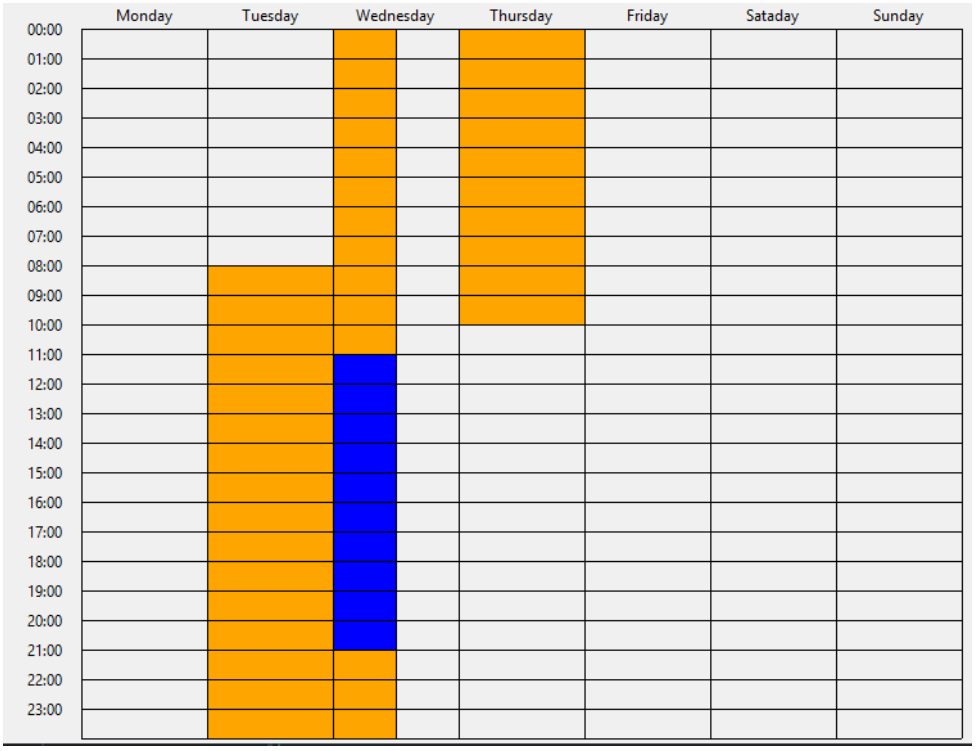
```
#Run through events
for x in range(len(events) - 1, -1, -1):
    #If the event is on the current day
    if events[x]["day"] == day:
        holder = events[x]
        eventsToday.append(holder)

        #for every min in hour
        for hour in range(holder["startTime"],holder["endTime"]-1):
            #Work out hours and mins
            hour = str(hour).zfill(4)
            hoursNum,minsNum = hour[:2], hour[2:]

            #Stops you having 12:76 (mins above 59)
            if int(hoursNum) > 59:
                continue
            #Calculate the number of mins since midnight and increment that min by 1
            mins = (int(hoursNum) * 60) + int(minsNum)
            eventDepth[mins-1] += 1
```

The first thing that I do is loop through all the events and if it is in the current day, I then I loop through every minute in each event and add one to the *eventDepth* list.

In the next section of code I then loop through all of the events that day, and then see what the maximum event depth is. If it is greater than one I change the *widthDivsor*. This variable is used to divide the width, so I know how wide to display the event.

```
for event in eventsToday:
    widthDivider = 1
    for hour in range(event["startTime"],event["endTime"]-1):
        hour = str(hour).zfill(4)
        hoursNum,minsNum = hour[:2], hour[2:]
        if int(minsNum) > 59:
            continue

        mins = (int(hoursNum) * 60) + int(minsNum)
        #See if width bigger
        if eventDepth[mins-1] > widthDivider:
            widthDivider = eventDepth[mins-1]

    #Calculate Cordinates
    x1 = GRID_START_CORDINATE_X + ((event["day"]-1)*self.WIDTH) + ((self.WIDTH))
    y1 = GRID_START_CORDINATE_Y + ((event["startTime"]/100)*self.HEIGHT)
    x2 = x1 + (self.WIDTH/widthDivider)
    y2 = GRID_START_CORDINATE_Y + ((event["endTime"]/100)*self.HEIGHT)


    self.create_rectangle(x1, y1, x2, y2, fill=event["colour"])
```

I then tested all of the code and got the following output:



As you can see this sort of worked, it split the overlapping events up and made them smaller, however they still overlap. I need to shift one of them over, so they do not overlap. I have also noticed that the events are all shifted over by one day. To fix this I have changed the x1 of all of the rectangles:

```
x1 = GRID_START_CORDINATE_X + ((event[0]["day"]-1)*self.WIDTH)
```

To fix the overlaying events I have changed the following code:

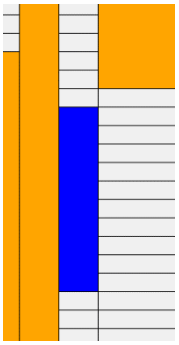
```
if position < eventDepth[mins-1]:
    position = eventDepth[mins-1]
```

I check if the any of the minutes the event goes over has already been incremented by one. If it has, it is increments the position by one, which is then used later to shift the position of the event along as it means that two events are overlapping.

```
#Calculate Cordinates
x1 = GRID_START_CORDINATE_X + ((event[0]["day"]-1)*self.WIDTH) + ((event[1]-1)*(self.WIDTH/widthDivider))
y1 = GRID_START_CORDINATE_Y + ((event[0]["startTime"]/100)*self.HEIGHT)
x2 = x1 + (self.WIDTH/widthDivider)
y2 = GRID_START_CORDINATE_Y + ((event[0]["endTime"]/100)*self.HEIGHT)
```

I then changed the coordinates of the rectangle to reflect these changes.

I tested this and it worked, the events were displayed correctly:



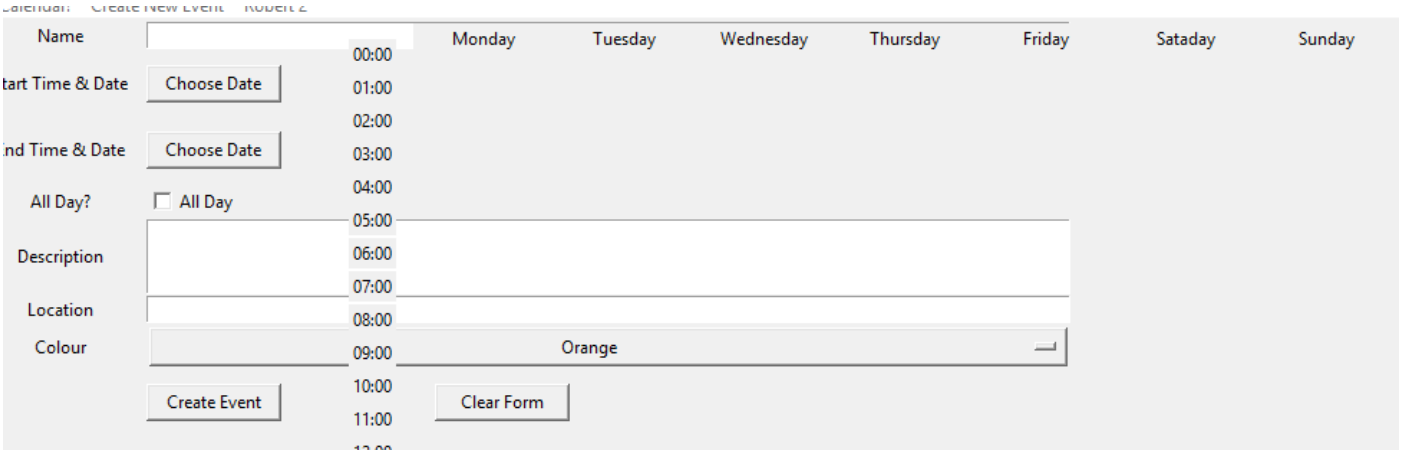
Although this worked, it took 13 seconds to display the events, and there are only 2 events to display! This is really slow, and almost unusable. I believe this is because there is a lot of iteration in my functions. This causes the “Big O” notation do be really high, at least $O(n^3)$. To fix this I need to reduce the amount of iteration that I am doing.

The thing is that this is all single threaded, meaning it only runs one command after the other, and cannot be split across multiple cores. I initial tried to use different threads to speed it up, however this got really complicated really quickly, and I was struggling to trace the algorithms. It also did not fix the underlying problem the original algorithm is slow. In the end I came up with the following solution:

```
for x in eventsToday:
    ID = x[0]
    start = events[ID]["startTime"]
    end = events[ID]["endTime"]
    y = eventDepth[start:end]
    widthDivider = max(y)


    #Calculate Cordinates
    x1 = GRID_START_CORDINATE_X + ((events[ID]["day"] - 1)*self.WIDTH) + ((x[1] -1)*(self.WIDTH/widthDivider))
    y1 = GRID_START_CORDINATE_Y + ((events[ID]["startTime"]/100)*self.HEIGHT)
    x2 = x1 + (self.WIDTH/widthDivider)
    y2 = GRID_START_CORDINATE_Y + ((events[ID]["endTime"]/100)*self.HEIGHT)


    self.create_rectangle(x1, y1, x2, y2, fill=events[ID]["colour"])
```

Rather than me looping through all of the minutes in a day to work out what the largest is, I copy the section of the *eventDepth* between the start and end time to a new variable y. I then work out the *widthDivisor* by using pythons inbuilt function *max()*, which gets the maximum number in the list. The advantage of this, is that the max function uses threading, meaning it can search multiple parts of the list at once. This speeds it up massively, so the load time is hardly noticeable, about 0.5 seconds. This is a lot easier than me trying to deal with the threading. This problem is fixed, and I added comments to it.

The next problem I have noticed is that when you go between the create event from and the main page with the event viewer, the labels stay on:

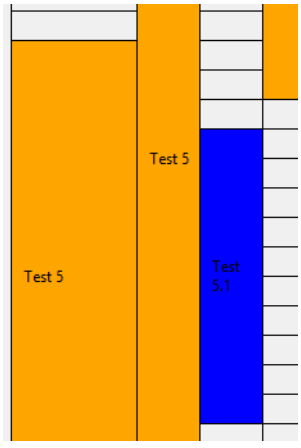


To fix this I have changed the lines `can.create_window(x, y, window=Label(text=text))` and `can.create_window(x+(WIDTH/2),y, width=WIDTH, window=Label(text=day))` to `self.create_text(x+(self.WIDTH/2),y, text=day)`, and `self.create_text(x,y, text=text)`.

This fixed that problem. The next thing that I am going to do is create the labels for the events. I have done this with the line:

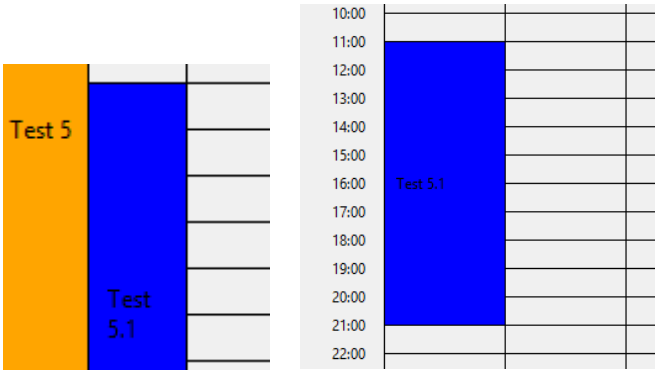
```
self.create_text((x1+10, 0.5*(y2-y1) + y1), text=events[ID]["name"], anchor="w", width=(x2-x1-15))
```

This worked all the events have their name on them.



The next problem that I encountered is that if an event did not start in a week, but ended in it, it was not being displayed. For example, you can see in the first picture below picture that 'Test 5' and 'Test 5.1'

overlap and are displayed next to each other. In the second picture when that day is made the first day of the week, it incorrectly displays the day and forgets about the 'test 5' event.



To fix this problem, I have changed the SQL to:

```
SELECT * FROM Events WHERE ((StartTimestamp BETWEEN ? AND ?) AND
CalendarID=?) OR ((EndTimestamp BETWEEN ? AND ?) AND CalendarID=?)
```

This got all of the events that start or end in the given week. I now need to display them properly. To do that I am going to change the start time of any event the starts before the range, to the start of the range. I have done this with the following code:

```
#Get start and end Times
if event[2] < startTimestamp:
    startTime = int(datetime.datetime.fromtimestamp(startTimestamp).strftime("%H%M"))
else:
    startTime = int(datetime.datetime.fromtimestamp(event[2]).strftime("%H%M"))
```

This worked and the events now sit properly together.

I have spoken to my stake holder Ryan about what he thought of the event viewer so far and he said "It looks really good; I like the fact the events change size depending on how many there are on one day. Two things I have noticed are that the colour blue doesn't really work with the black text, it makes it hard to see the event name because it is black. The other thing is that as I go through the days, the day names at the top of the screen do not change, meaning they are incorrect for the day that I am viewing."

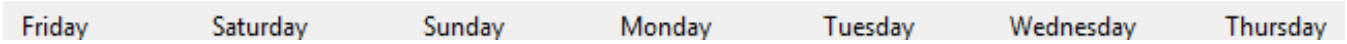
I will remove blue as a colour in the dropdown later, because he is right - it is hard to see. I want to get the functionality of the rest of the program working before this though. The other issue was that the day labels do not change at all. This means that they are incorrect. I have fixed this issue by getting the day number of the start day.

```
startDayWeekdayDecimal = int(startTime.strftime("%w"))
```

I then changed the *createLabels* function to this:

```
days = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Sataday"]
idToDisplay = firstDay
for q in range(0,7):
    print(idToDisplay)
    self.create_text(x+(self.WIDTH/2),y, text=days[idToDisplay])
    x += self.WIDTH
    idToDisplay += 1
    if idToDisplay > 6:
        idToDisplay = 0
```

It takes the decimal weekday and loops through writing out the days.



This means that the days now match up to the day that is being displayed on the calendar.

I checked this with Ryan and he said it "Looks a lot better"

Session 6 – 20/02/2019

The first thing that I am going to do today is to create the all day events, so that the event viewer follows the following success criteria:

No.	Requirements	Justification
14	An all-day event is shown at the top of that day	So the user can see at a glance what is going on all day.

When I created the code that draws the grid and events in the canvas, I forgot about the all day events. This means that I am going to have to modify some of my existing code to draw and create. The things that I need to make the code do is:

- Work out the height of the all day events for the chosen week
- Create a grid below the day labels but above the existing grid for the all day events
- Draw the all day events
- Shift the existing time labels, grid and events down

The first thing that I have done is change the days for all day events. If it starts before the range the user is selected, it is currently still being added to the list allDay events. This is incorrect. I have added the following if statement make sure it is in the correct range.

```
if 1 <= x <= 7 :
```

I have created the following function to draw the all day events.

```
def drawAllDayEvents(self, events, startDay):
    maxEvents = [0,0,0,0,0,0,0]
    for event in events:
        event["eventDepth"] = maxEvents[event["day"] - 1]
        maxEvents[event["day"] - 1] += 1

    maxEventsNum = max(maxEvents)

    for event in events:
        x1 = self.GRID_START_CORDINATE_X + ((event["day"] - 1)*self.WIDTH)
        y1 = self.GRID_START_CORDINATE_Y + (event["eventDepth"] * self.HEIGHT)
        x2 = x1 + self.WIDTH
        y2 = y1 + self.HEIGHT
        tag = "eventClick " + str(event["ID"]) + " event"
        self.create_rectangle(x1, y1, x2, y2, fill=event["colour"], tags="test", dash=(4, 4), tag=tag)
        self.create_text((0.5*(x2-x1) + x1, 0.5*(y2-y1) + y1), text=event["name"], anchor="center", width=(x2-x1-15), tag=tag )

    self.createAllDayGrid(maxEventsNum)
    y = self.GRID_START_CORDINATE_Y
    x = self.GRID_START_CORDINATE_X
    self.GRID_START_CORDINATE_Y += (maxEventsNum * self.HEIGHT) + (self.HEIGHT / 10)
    self.createLabels(startDay, x, y)
    self.createGrid()
```

This calculates the height of them, then loops through all the events and draws them. It then draws all the grids and labels.

I then created a function to draw the grid around the all day events. This works out the height and length required then draws the lines

```
def createAllDayGrid (self, NumOfEvents):
    # verticalLength = (self.HEIGHT * 24 )
    horizontalLength = (self.WIDTH * 7)
    GridHeight = (NumOfEvents )*self.HEIGHT
    #Create Vertical Grid Lines
    y = self.GRID_START_CORDINATE_Y
    x = self.GRID_START_CORDINATE_X

    for line in range(0, 8):
        self.create_line(x,y,x,y+ GridHeight, tag="grid")
        x += self.WIDTH

    y = self.GRID_START_CORDINATE_Y
    x = self.GRID_START_CORDINATE_X
    self.create_line(x, y, x+horizontalLength, y, tag="grid")
    self.create_line(x, y+GridHeight, x+horizontalLength, y+GridHeight, tag="grid")
```

I have added the following line so that I can run both functions.

```
self.drawAllDayEvents(allDayEvents, startDayWeekdayDecimal)
```

This worked as I expected below

The *eventViewer* class, although it works, has become very inefficient. This is because when I wrote the pseudocode initially, I did not foresee several things like the all-day events. This is something that I can come back to if I have time.

I have spoken to Ryan about what he thinks of the event viewer now and he said "Looks good, the all-day events are good, and I like the fact they are at the top of the page. The one thing I think would be useful is that it would be nice to have the date of the day next to the day." I wasn't sure what he meant so I clarified, and he said that rather than have Saturday, Sunday etc. at the top, he would prefer Saturday 21st, Sunday 22nd etc. I will come back to this in another session.

The next thing that I am going to do is the event viewer. When the user clicks on an event it needs to display all of the details about it. It also needs to give the user the option to edit the event. The first thing I need to do is make it so that I know when and which event is clicked.

To do this I have added the following line where the event is drawn:

```
tag = "eventClick " + str(events[ID]["ID"]) + " event"
```

This create a text variable called tag with the event ID. When I create the event I then define the tag with the code `tag=tag`. I have also added the following tags to the grid, and labels respectively: `tag="grid"`, `tag="timeLabels"`, `tag="daysLabel"`. This allows me to easily find these elements on the canvas.

I have decided that I am going to create a new class for viewing the event details called *EventDetails*. I am going to put this in a file called *EventDetails.py.* This needs to get the details about an event, hide everything that is currently on the canvas and then display the details about the event. It also needs to give

the user options to edit and delete the event. I have created an __*init*__ function and called it in at the beginning of the *eventViewer* class:

```
class EventDetails:
    def __init__(self,canvas):
        self.CANVAS = canvas
```

```
self.EventDetails = EventDetails(self
```

I have then used the tags that I created on the events to register an event click to Tkinter. This will then watch out for the user then clicking on the event. This should satisfy the following success criteria points:

No.	Requirements	Justification
11	When an event is clicked on it should display information about it	More details should be hidden initially to stop clutter. When clicking on it should display the info about that event
12	When the user clicks on an event the colour of that event should become the background	This helps to keep the project colourful and exciting. It also helps the user easily identify the event they are viewing

This means that when the user clicks on anything with the tag “eventClick” it will call a function within the event details class.

```
self.tag_bind("eventClick","<Button-1>", self.EventDetails.eventClick)
```

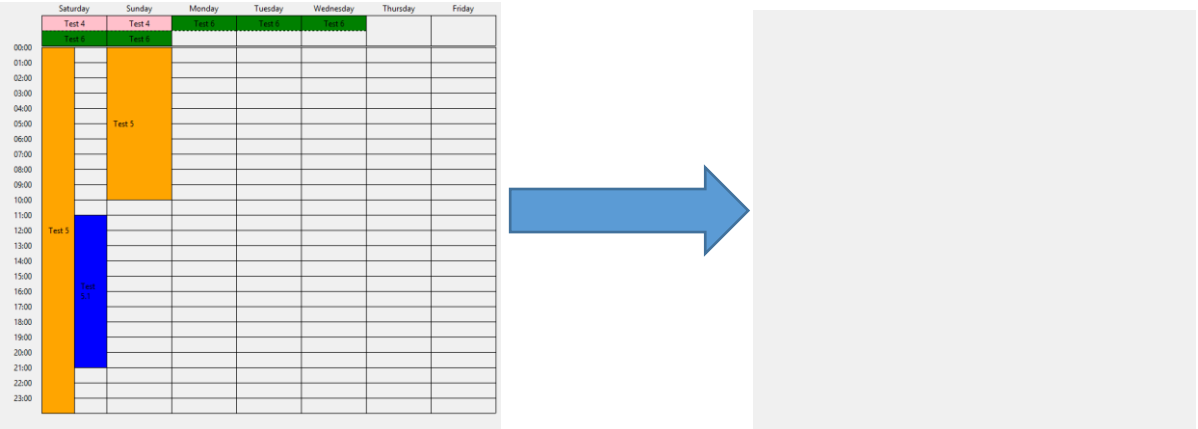
I have created an event details function:

```
def eventClick(self, event, *args):
    #Find Item + Get ID
    CanvasItem = event.widget.find_closest(event.x, event.y)
    eventID = event.widget.gettags(CanvasItem[0])[1]

    #Hide everything on items on the canvas
    thingsOnCanvas = self.CANVAS.find_withtag("event") + self.CANVAS.find_withtag("timeLabels") +
                     self.CANVAS.find_withtag("daysLabel") + self.CANVAS.find_withtag("grid")

    for id in thingsOnCanvas:
        self.CANVAS.itemconfigure(id, state='hidden')
```

It gets the ID of the clicked event from the tag. Then it gets all of the ID’s of everything that is on the canvas. Then it loops through all of those IDs and sets their state to hidden, which means they will not be seen. I tested this by clicking on the orange event (“test 5”) and it outputted the following event ID

```
8
```

and then cleared the canvas:



I tested this with the test that I produced earlier as well:

Event Clicked on to show more details

Test Data	Type	Actual Outcome
Left click on button	**Valid**	**Valid**
Right click on button	**Invalid**	**Invalid**
Click outside event	**Invalid**	**Invalid**

This shows that clicking on the event works. The next thing that this function needs to do is get the details about the event from the database. To do this I need to create a new function in my database class. I have created the following SQL to do this:

```
SELECT * FROM Events WHERE ID=?
```

I used this to get create a function in the database class called getEventDetails. It takes one input of the event ID and then returns the row, as shown below:

```
def getEventDetails(self,eventID):
    #Get the details about a particular event
    sql = "SELECT * FROM Events WHERE ID=?"
    self.cur.execute(sql,(eventID,)))
    for row in self.cur.fetchall():
        return row
```

I tested this function with the following code and got the following outputs:

```
print(db.getEventDetails(1))
print(db.getEventDetails(2))
print(db.getEventDetails(3))
print(db.getEventDetails(4))
print(db.getEventDetails(5))
```

```
(1, 'School Trip', 1550052000, 1550080800, 'London', 0, 2, 'School Trip to London!\n', 'Blue')
(2, 'Test 5.1', 1550314800, 1550350800, '', 0, 1, '\n', 'Blue')
(3, 'School  Trip', 1550052000, 1550080800, 'London', 0, 2, 'School trip to london!\n', 'Blue')
(4, 'Test 1', 1550041200, 1550091600, '', 0, 1, '1 Day\n', 'Green')
(5, 'Test 2', 1550102400, 1550188740, '', 0, 1, '1 Day Boundary\n', 'White')
```

I checked these outputs against the database and these were all correct. The next function that I am going to create is called *drawEventDetails* in the *EventDetails* Class. This function gets the events using the function in database that I just created (*getEventDetails*), works out which items are filled in and adds

them to a list to be displayed. If an item is not filled in in the database, it should not be displayed. I have also set it to change the background colour of the canvas to the colour the user has selected for that event.

```
def drawEventDetails (self, eventID):
    #Get Events
    event = self.DB.getEventDetails(eventID)

    #Change Background Colour
    self.CANVAS.configure(background=event[8])
    x = datetime.fromtimestamp(event[2])

    #Work out which Items the user filled in and added them to the list
    items = [
        ["Name", event[1]],
        ["Start Date", x.strftime("%A %d %B %Y @ %X")]
    ]
    if event[3] != None:
        items.append(["End Date", datetime.fromtimestamp(event[3]).strftime("%A %d %B %Y @ %X")])
    if event[4] != "":
        items.append(["Location", event[4]])
    if event[5] == 1:
        items.append(["All Day?", "Yes"])
    else:
        items.append(["All Day?", "No"])
    items.append(["Description", event[7].strip("\n")])

    print(items)
```

I tested this and it printed all the data that I expected correctly:

```
[['Name', 'Test 5'], ['Start Date', 'Friday 15 February 2019 @ 08:00:00'], ['End
 Date', 'Sunday 17 February 2019 @ 10:00:00'], ['All Day?', 'No'], ['Description
', 'Multi Day - Non All Day\n']]
```

To finish this function off I then needed to draw all the items. For this I will loop through all of the items in the list (*items*) and then display them on the canvas:

```
#Define the inital position for the grid
y = 20
for item in items:
    #Display the infomation about the event
    self.CANVAS.create_text(40,y, text=item[0], anchor="w", tag="eventDetails")
    self.CANVAS.create_text(120,y, text=item[1], anchor="w", tag="eventDetails")
    y += 30
```

I tested this with two different events, event 5 and event 6.

Name	Test 5
Start Date	Friday 15 February 2019 @ 08:00:00
End Date	Sunday 17 February 2019 @ 10:00:00
All Day?	No
Description	Multi Day - Non All Day

Name	Test 6
Start Date	Saturday 16 February 2019 @ 00:00:00
End Date	Wednesday 20 February 2019 @ 00:00:00
All Day?	Yes
Description	

This worked as I expected however the default text is small to read (the screenshots are zoomed in), so I have changed the create text functions to add a font, making the label for each item bold. I created this and tested it and it worked as I expected.

```
#Display the infomation about the event
self.CANVAS.create_text(40,y, text=item[0], anchor="w", tag="eventDetails", font = ('Calibri', 15, 'bold'))
self.CANVAS.create_text(160,y, text=item[1], anchor="w", tag="eventDetails", font = ('Calibri', 15, ''))
```

Name:	Test 5
Start Date:	Friday 15 February 2019 @ 08:00:00
End Date:	Sunday 17 February 2019 @ 10:00:00
Location:	London
All Day?	No
Description:	Multi Day - Non All Day

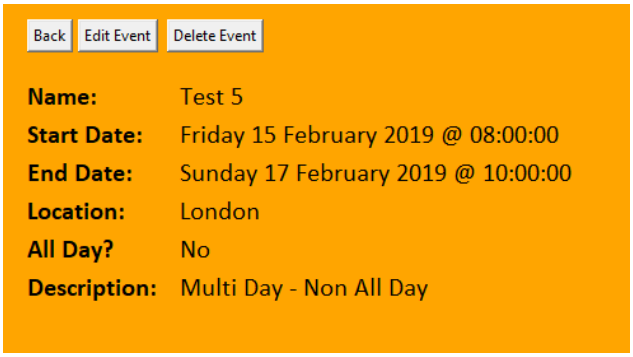
The next thing that I needed to do was create 3 buttons, a back button, edit event button and delete event button. I am going to lay them out above the text that I have just drawn. I have created a new function *drawButtons* to do this:

```
def drawButtons (self):
    #Draw Buttons
    BackButton = Button(self.CANVAS, text = "Back", anchor = W)
    BackButton_window = self.CANVAS.create_window(40, 20, anchor=NW, window=BackButton, tag="eventDetails")

    EditEventButton = Button(self.CANVAS, text = "Edit Event", anchor = W)
    EditEventButton_window = self.CANVAS.create_window(80, 20, anchor=NW, window=EditEventButton, tag="eventDetails")

    DeleteEventButton = Button(self.CANVAS, text = "Delete Event", anchor = W)
    DeleteEventButton_window = self.CANVAS.create_window(150, 20, anchor=NW, window=DeleteEventButton, tag="eventDetails")
```

I then called this function in the *eventClick.*

```
self.drawButtons()
```

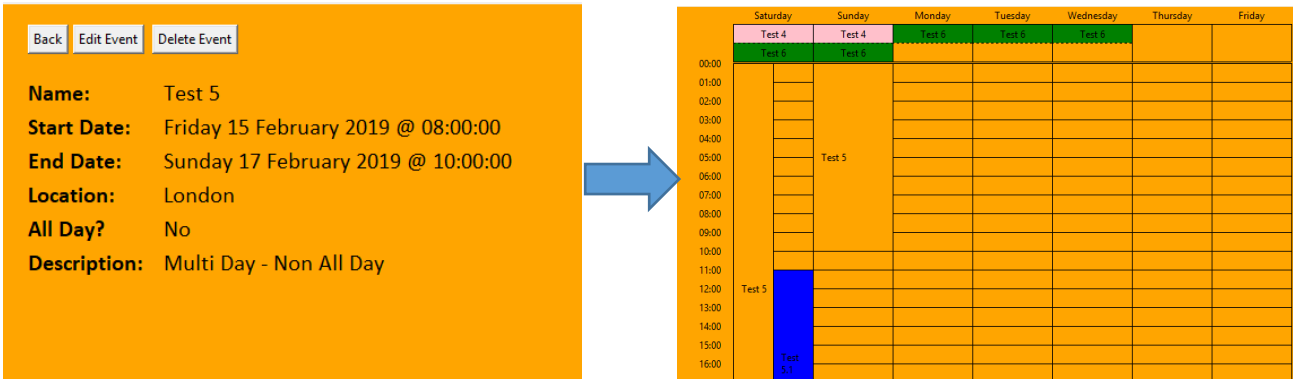
I tested this and got the buttons drawn as I expected.



The next thing that I need to do is get the buttons to work. I am going to create 3 functions: *backButtonPress*, *editButtonPress*, *deleteButtonPress*. Each one will handle a different button press, and needs to complete the respective function of that button. The first one I have created is the *backButtonPress* function:

```
def backButtonPress(self):
    #Delete all of the event detial items
    eventDetailsOnCanvas = self.CANVAS.find_withtag("eventDetails")
    for id in eventDetailsOnCanvas:
        self.CANVAS.delete(id)

    #Display all of the hidden items
    hiddenOnCanvas = self.CANVAS.find_withtag("event") + self.CANVAS.find_withtag("timeLabels") +
                     self.CANVAS.find_withtag("daysLabel") + self.CANVAS.find_withtag("grid")

    for id in hiddenOnCanvas:
        self.CANVAS.itemconfigure(id, state='normal')
```

This gets all of the items that were drawn on the canvas when the user opened the event details. It then deletes all of them, and unhides all of the events for the event viewer. This causes it to go back. I tested this and it worked (as shown below) however the background colour of the event viewer did not change back as I had not written code for this.



To change the background colour back I need to know what the original background colour is. In the event viewer class I have got the background and passed it to the event details class :

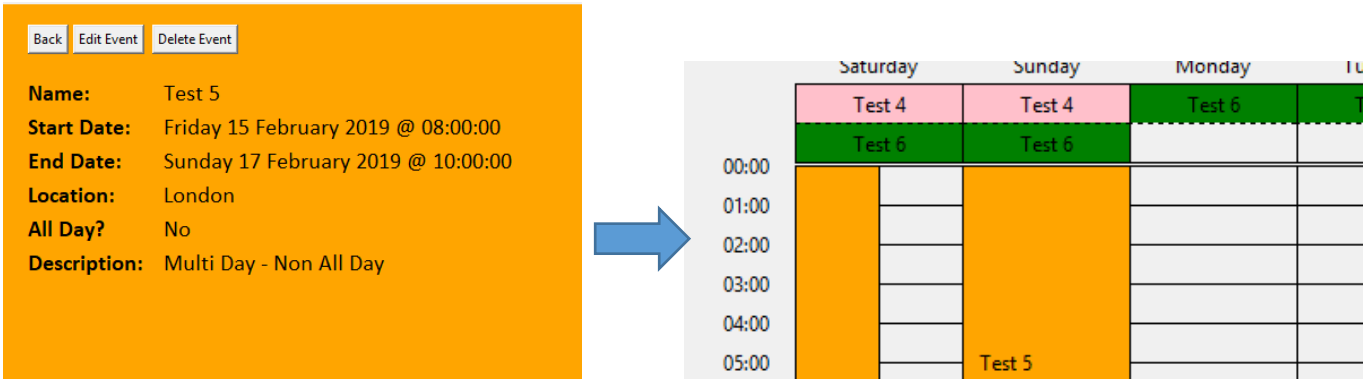
```
self.BACKGROUND = self.master.cget('bg')
self.EventDetails = EventDetails(self, self.DB, self.BACKGROUND)
```

I then changed the *eventDetails* class to accept this input and save it to the class.

```
def __init__(self,canvas, database, defualtBackgroundColour):
self.defaultBackground = defualtBackgroundColour
```

In the back function I have then added the following line to change the background colour of the canvas:

```
self.CANVAS.configure(background=self.defaultBackground)
```



This works now and I can go from event back to the event viewer as many times as I like.

The next function that I am going to look at is the *deleteButtonPress* function, to satisfy the following success criteria:

No.	Requirements	Justification
19	Delete events	The user needs to be able to delete events

Before I can do this I need to create a function in the database class to delete an event. I have created the following SQL to do this:

```
DELETE FROM Events WHERE ID=?
```

I then used this to create a function called deleteEvent, which takes an input of the event ID.

```
def deleteEvent(self,eventID):
    #Delete an event by ID
    sql = "DELETE FROM Events WHERE ID=?"
    #Execute SQL
    self.cur.execute(sql,values)
    self.con.commit()
```

I decided to test it with an event in the database with an ID of 12. I made sure it was in the database and created a basic piece of code to test it.

12	Test	1550052000	1550080800	London	0	1	Test	Pink

```
db= Database()
db.deleteEvent(12)
```

It did not work and I got a nameError: `NameError: name 'values' is not defined`. I realised that I was trying to pass a variable that did not exist to the execute function. I have changed the execute command in the function to `self.cur.execute(sql,(eventID,))` and then I tested it and it worked successfully deleting the event with an ID of 12.

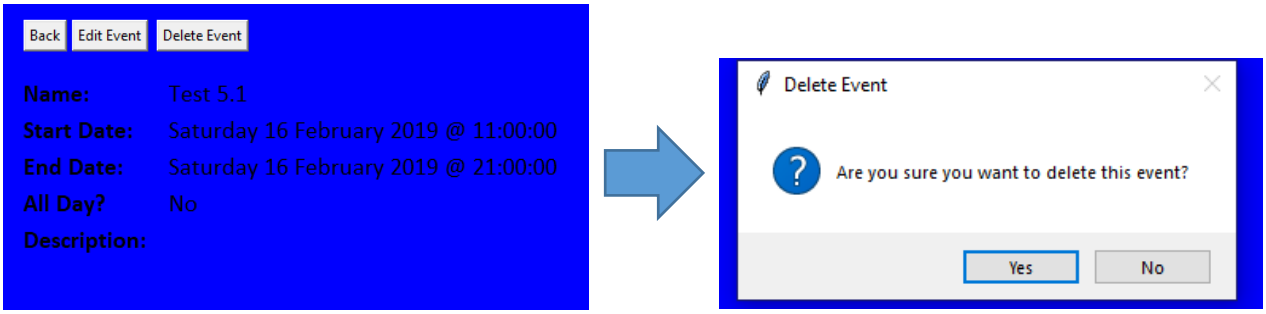
I then created the *deleteButtonPress* function:

```
def deleteButtonPress(self, id):
    comfirm = messagebox.askyesno("Delete Event","Are you sure you want to delete this event?")
    if comfirm == True:
        self.DB.deleteEvent(id)
        self.backButtonPress()
```

This uses the *messagebox* function within Tkinter to confirm whether the user is sure they want to delete the event. If they say yes, it uses the database to delete the event and then 'presses' the back button by calling its function. This is so that once the event is deleted it goes back to the event viewer. I added this function to the delete button with the line: `command=lambda id=eventID: self.deleteButtonPress(id))`.

I then decided to test this with event 5.1 which is blue. I checked it was in the database:

2	Test 5.1	1550314800	1550350800		0	1	Blue



The delete button worked, it removed the event from the database, however when it went back to the event viewer the event was still left in grid, it had not removed it from the grid.

To change this, I need to reload the grid. The first thing that I have changed to do this, is added a variable to the event viewer class so when it loads the date it saves it to a variable called current date.

```
self.currentDate = (year, month, day)
```

Then, rather than virtually pressing the back button, I get this function and then call the display event function. This reloads all of the events from the canvas.

```
self.CANVAS.displayEvent(self.CANVAS.currentDate[0],self.CANVAS.currentDate[1],self.CANVAS.currentDate[2])
```

I tested this and it worked, it removed the deleted event from the grid. Now that I think that it is working I am going to test this button with the tests that I produced in the design phase:

Delete Button

Test Data	Type	Actual Outcome
Left click on button	**Valid**	**Valid**
Right click on button	**Invalid**	**Invalid**
Click outside event	**Invalid**	**Invalid**

These worked showing that the delete button works.

Session 7 – 27/02/2019

The next section of code that I am going to work on is the edit event form. This is to follow the following success criteria points:

No.	Requirements	Justification
20	Edit Events page	A form is needed to change data about existing events.
21	When an event is edited it should go through the same validation as when creating an event	This is to prevent events ending before they start, and so that every event has a title.

My initial plan from my systems diagram was to have a separate class, called *editEvent*, for editing the event. Looking at it again I believe that this will be really inefficient because the form and validation is all the same, it will increase the space complexity of the program unnecessarily to duplicate these. Instead I am going to modify the *CreateEvent* class to accept an input of an event ID and then edit that event rather than create a new one. I have created pseudocode for this function:

```
FUNCTION editEvent (self, eventID):
      self.clearForm()
      eventDetails = GET EVENT DETAILS FROM DATABASE (eventID)

      FOR EACH form element:
            Fill In From EventDetails
      END FOR

      self.IsEditEvent = True
```

```
        self.editID = eventID

END FUNCTION
```

I am going to use the *IsEditEvent* and *editID* functions in the *createEventButtonPress* function. When the user presses the create event button it will see if the *isEditEvent* is true, and if it is it will edit the event rather than create a new one. This can come after all of the validation in the code. I created the edit event function as shown below. I could not loop through all of the form elements as I thought in my pseudocode because all of the different elements require being changed in different ways, for example a text box needs to have the set inserted whereas the all-day button needs to have it selected. Because of this I have hard coded all of the individual form elements.

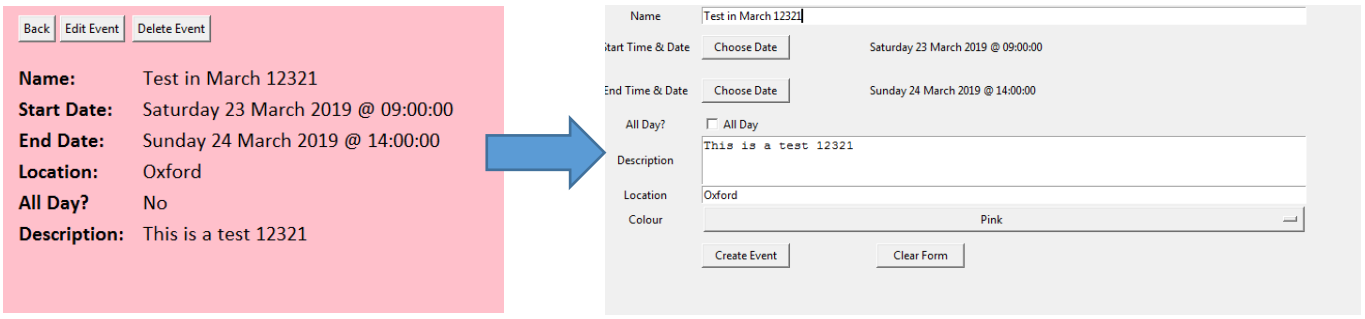
For the start and end times I have used the function changeStartTime that I made earlier in the project. This however takes the input of date and time as numbers, so I convert the timestamp into year, month, day, mins and second and then pass it.

```
def editEvent(self,eventID):
    #Display Frame + Reset Form + Get event Details
    self.controller.show_frame("createEvent")
    self.clearForm()
    eventDetails = self.DB.getEventDetails(eventID)

    #Populate Form
    self.name.insert(END, eventDetails[1])
    startTime = datetime.datetime.fromtimestamp(eventDetails[2])
    self.changeStartTime((int(startTime.strftime("%Y")), int(startTime.strftime("%m")), int(startTime.strftime("%d")), int(startTime
    endTime = datetime.datetime.fromtimestamp(eventDetails[3])
    self.changeEndTime((int(endTime.strftime("%Y")), int(endTime.strftime("%m")), int(endTime.strftime("%d")), int(endTime.strftime(
    if eventDetails[5] == 1:
        self.allDay.select()
    self.description.insert(END,eventDetails[7])
    self.location.insert(END, eventDetails[4])
    self.colourValue.set(eventDetails[8])

    self.IsEditEvent = True
    self.EditID = eventID
```

I then called this button is the *eventDetails* class, on the edit button by adding the parameter command:

```
command=lambda id=eventID: self.controller.frames["createEvent"].editEvent(id))
```

I tested this by opening an event and pressing the edit event button. This opened the create event form as I expected and had all of the information filled in.



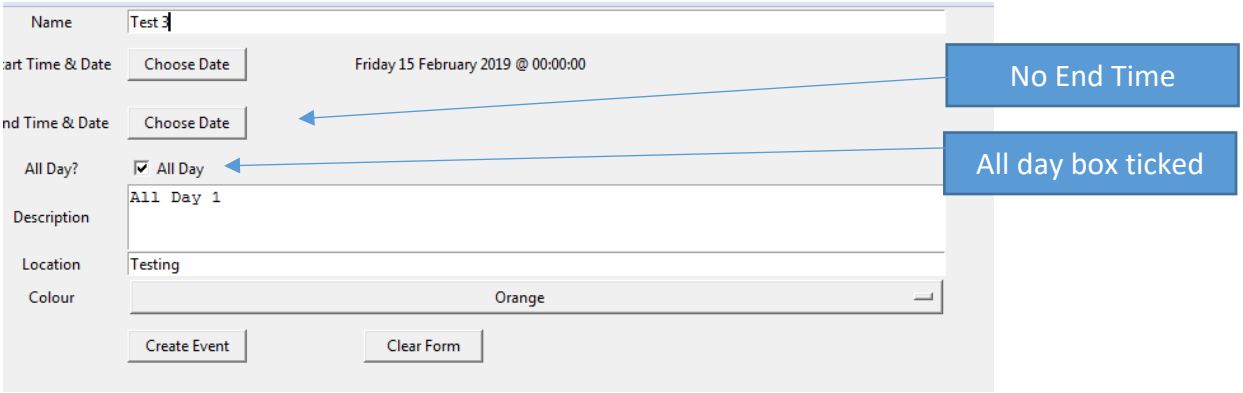
I tested this with an all-day event, and got an error:

```
    endTime = datetime.datetime.fromtimestamp(eventDetails[3])
TypeError: an integer is required (got type NoneType)
```

This is because an all-day event does not need an end time if it only one day long. This means that I am trying to pass a time of 'None' as a number when populating the form. To fix this I am going to check if the event has an end time before passing it. I have added this if statement in to fix it:

```
if eventDetails[3] != None:
    endTime = datetime.datetime.fromtimestamp(eventDetails[3])
    self.changeEndTime((int(endTime.strftime("%Y")), int(endTime.strftime("%m")), int(endTime.strftime("%d")),
```

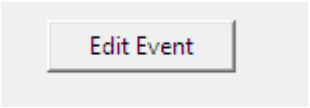
This checks to see if there is an end time set and if there is it will pass it. I tested the function again and it worked as I expected with an all-day event:



The next thing that I have noticed is that it still says 'Create Event' on the button. This is correct, however when editing an event it would make more sense if it said 'Edit Event'. The first thing I need to do is make the variable of the button accessible outside the *__init__* class. I have done this by changing the variable name to `self.createEventButton`. I then need to change the text of the button in the edit event function. I have added the line:

```
self.createEventButton["text"] ="Edit Event"
```

This changed the text of the button successfully:



The next thing I need to do is make this button work. Before I can do that I need to create a in the database class that will allow me to edit an event in the database. To do this I have created the following SQL:

```
UPDATE Events SET Name = ?, StartTimestamp = ?, EndTimestamp = ?,
Location = ?, AllDay=?, Description=?, Colour=? WHERE ID = ?"
```

I have used this to create the following method, that takes all of the inputs required and the executes the SQL:

```
def editEvent(self, eventID, name, startTime, endTime, allDay, description, location, colour):
    sql = "UPDATE Events SET Name = ?, StartTimestamp = ?, EndTimestamp = ?, Location = ?, AllDay=?, Description=?, Colour=? WHERE I
    values = (name,startTime,endTime,location,allDay,description,colour, eventID)
    #Execute SQL
    self.cur.execute(sql,values)
    self.con.commit()
```

I tested this function with the following code:

```
db= Database()
db.editEvent(24,"testing",1554526800, 1554580800, False,
             "This should edit the database", "London", "Pink")
```

Before I ran this I checked to see what it looked like:

ID	Name	StartTimestamp	EndTimestamp	Location	AllDay	CalendarID	Description	Colour
24	New Event 123	1554526800	1554580800		0	1		Orange

After I ran it this line in the database changed so that it looked like:

ID	Name	StartTimestamp	EndTimestamp	Location	AllDay	CalendarID	Description	Colour
24	testing	1554526800	1554580800	London	0	1	This should edit the database	Pink

This worked as I expected and it successfully edited the event. I now need to use this function to edit the event from the form.

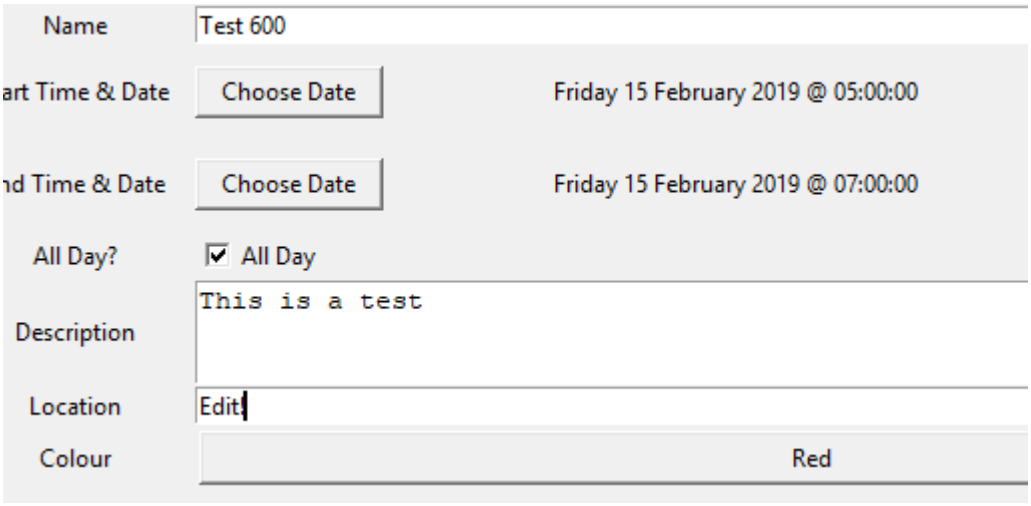
I have modified the *createEventButtonPress* so that if the IsEditEvent variable is false it will create a new event and if not it edits the event. I have placed this under all of the validation that already exists so that the event is still validated when the user edits the event.

```
if self.IsEditEvent == False:
    #Create Event
    self.DB.createEvent(name,StartUnixTime,EndUnixTime,allDay, description, location, colour, calendarID)
else:
    #Edit Event
    self.DB.editEvent(self.EditID, name,StartUnixTime,EndUnixTime,allDay, description, location, colour)
```

I tested this by opening an event to edit. Before I edited it properly I checked what it looked like in the database:

ID	Name	StartTimestamp	EndTimestamp	Location	AllDay	CalendarID	Description	Colour
11	Test 303	1550206800	1550214000		0	1		Orange

I then filled in the form as below:



I then clicked the edit event button to run the edit code. I checked the database and the form had successfully changed that event:

	ID	Name	StartTimestamp	EndTimestamp	Location	AllDay	CalendarID	Description	Colour
1	11	Test 600	1550206800	1550214000	Edit!	1	1	This is a test	Red

The next thing that I need to do is once the event has been edited I need to clear the form. I have modified the *clearForm* function to add the following three lines:

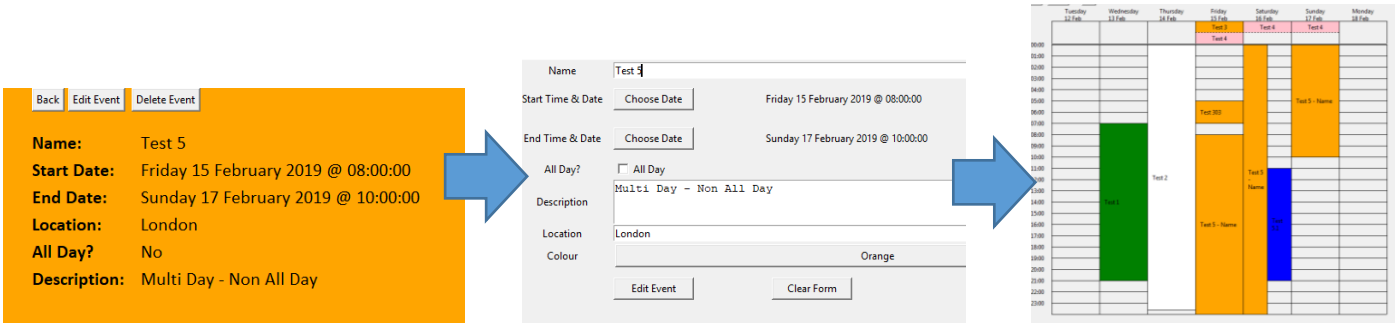
```
self.IsEditEvent = False
self.EditID = None
self.createEventButton["text"] ="Create Event"
```

The first line resets the form to a 'Create' form, so that I am not editing events without meaning to. The second line removes the *EditID* by setting it to a none type. This is ready for the next event to be edited. The final line resets the confirm button text back to 'Create Event'. I will test these functions in a bit however before I can do that I need to change the frame and run the clear form.

The first thing that I need to do is reset the view in the event viewer so that any changes are displayed on the canvas. This line resets the date by displaying the current date:

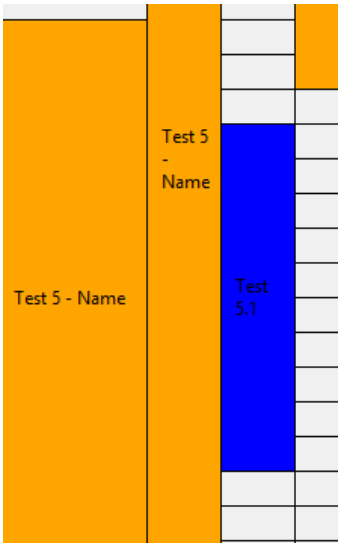
```
self.controller.frames["mainPage"].eventViewerHolder.displayEvent(self.controller.frames["mainPage"].eventViewerHolder.currentDate[0],
                                                                  self.controller.frames["mainPage"].eventViewerHolder.currentDate[1],
                                                                  self.controller.frames["mainPage"].eventViewerHolder.currentDate[2])
```

I tested this by editing an event. I first clicked edit, the changed the name and then clicked edit event. It went back to the main view as I expected with the changes that I made. This means that this function is now all working.



I asked Ryan to test this and he said, "I like that when you click on the event all of the information is diaplayed and the buttons are clear to edit and delete event are clear. I like the fact that when you click edit event the details are already filled in."

```
self.menubar.add_command(label="Calendar!", command=lambda: [self.controller.show_frame("mainPage"),self.controller.frames["createEvent"].clearForm()])
```



Now that this works I am going to run the test that I produced in the design process to make sure that the edit event form is working properly.

Edit Button

Test Data	Type	Actual Outcome
Left click on button	**Valid**	**Valid**
Right click on button	**Invalid**	**Invalid**
Click outside event	**Invalid**	**Invalid**

Edit Event Validation

Test Data	Type	Actual Outcome
Name, start time filled in and all day checked	**Valid**	**Valid**
Name, start time, and end time filled in	**Valid**	**Valid**
Name not filled in	**Invalid**	**Invalid**
Start time not filled in	**Invalid**	**Invalid**

Test Data	Type	Actual Outcome
Name: “Lunch 1” Start time: “1/1/2019 @ 15:00” End Time: “1/1/2019 @ 17:00”	**Valid**	**Valid**
Name: “Lunch 2” Start time: “1/1/2019 @ 15:00” End Time: “2/1/2019 @ 17:00”	**Valid**	**Valid**
Name: “School 1” Start time: “1/1/2019 @ 15:00” All Day: Checked	**Valid**	**Valid**
Name: “Lunch 3” Start time: “1/1/2019 @ 15:00” End Time: “1/1/2019 @ 12:00”	**Invalid**	**Invalid**
Name: “School 2” Start time: “1/1/2019 @ 15:00” All Day: Unchecked	**Invalid**	**Invalid**

Edit Event Form Complete

Test Data	Type	
Left Click on create account Button	**Valid**	**Valid**
Right Click on event	**Invalid**	**Invalid**
“enter key”	**Invalid**	**Invalid**
Left Click on ‘Back’ Button	**Invalid**	**Invalid**

All of these tests were successful so, the next thing that I am going to look at is the navigation buttons at the top of the window. This is to satisfy these success criteria:

No.	Requirements	Justification
15	Navigation Buttons at top of page	To make it easy to navigate between weeks
16	A “Today” Button to skip to today’s date	Quick way to find the today, so you can see what you should be doing now

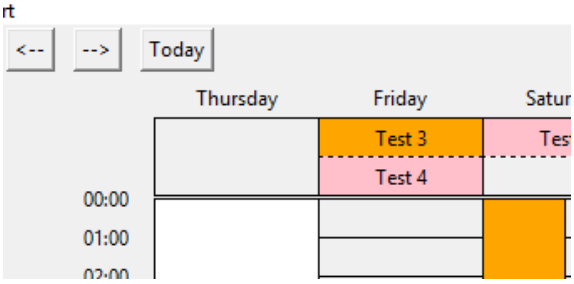
The first thing I need to do is shift the whole grid down so that there is room for the buttons. I have added 30 to the variable GRID_START_CORDINATE_Y to shift the whole grid down by 30 pixels.

```
self.GRID_START_CORDINATE_Y = (self.HEIGHT * 25) - (self.HEIGHT * 24 ) + 30
```

I then created a new function called *createNavButtons*, which creates the 3 navigation buttons that I need. It then adds them to the canvas at particular locations.

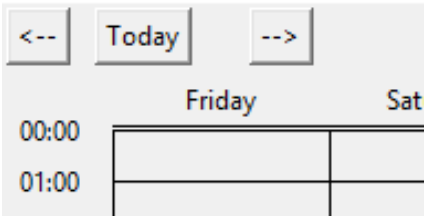
```
def createNavButtons(self):
    leftButton =Button(self, text="<--")
    self.create_window(0, 0, anchor=NW, window=leftButton, tag="grid")

    rightButton = Button(self, text="-->")
    self.create_window(40, 0, anchor=NW, window=rightButton, tag="grid")

    todayButton = Button(self, text="Today")
    self.create_window(80, 0, anchor=NW, window=todayButton, tag="grid")
```

I called this function displayEvent function, so that when it draws everything else it also draws the navigation buttons. `self.createNavButtons()`. I then tested this function and it drew the buttons in the top left corner of the canvas as I expected.



They are in the wrong order so I changed the code to reorder them and move them all across so they are more in line with the rest of the grid and not hanging out to the left:

```
leftButton =Button(self, text="<--",  command=self.leftButtonClick)
self.create_window(40, 0, anchor=NW, window=leftButton, tag="grid")

todayButton = Button(self, text="Today", command=self.todayButtonClick)
self.create_window(80, 0, anchor=NW, window=todayButton, tag="grid")

rightButton = Button(self, text="-->", command=self.rightButtonClick)
self.create_window(150, 0, anchor=NW, window=rightButton, tag="grid")
```

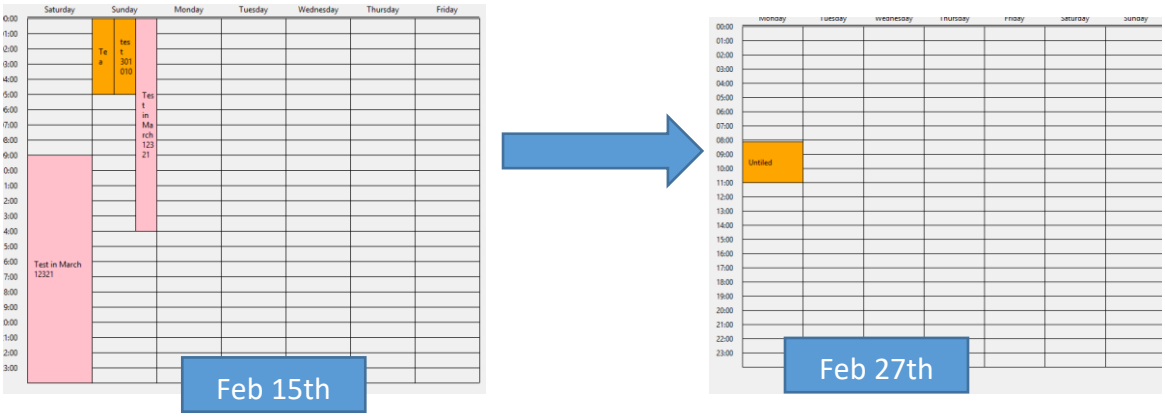


I now need to get the buttons to work with the rest of the code. The first one I am going to look at is the today button. I have created a new function *todayButtonClick.* This gets the time now, and then runs the display event function using this time.

```
def todayButtonClick(self):
    now = datetime.datetime.now()
    self.displayEvent(now.year,now.month,now.day)
```

I have added this attribute to the button with the command `command=self.todayButtonClick)`.

I tested this by pressing the day button and it displayed the events that were on today.



I then needed to get the other two buttons to work. To do this I need to get the date that is currently being displayed and the for the left button minus seven days and for the right button add seven days. I have created two functions to do this *leftButtonClick* and *rightButtonClick.*

```
def leftButtonClick(self):
    currentDateObject = datetime.datetime(self.currentDate[0], self.currentDate[1], self.currentDate[2], 0,0)
    newDateObject = currentDateObject - datetime.timedelta(days=7)
    self.displayEvent(newDateObject.year,newDateObject.month,newDateObject.day)
```

```
def rightButtonClick(self):
    currentDateObject = datetime.datetime(self.currentDate[0], self.currentDate[1], self.currentDate[2], 0,0)
    newDateObject = currentDateObject + datetime.timedelta(days=7)
    self.displayEvent(newDateObject.year,newDateObject.month,newDateObject.day)
```

I am going to struggle to test this as I do not know what exact date I am viewing. I am going to take Ryan's advice from earlier and add the date. This will be the first thing that I do next time.

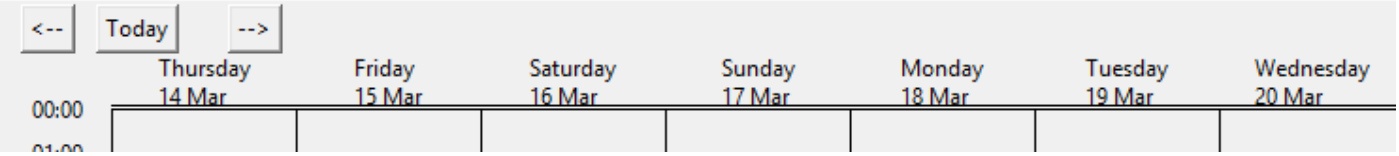
Session 8 – 1/03/2019

The first thing that I am going to do in this session it to change the day labels so that they include the date being viewed. This is something that Ryan, my stakeholder, suggested previously, and as I am close to finishing this section of the program I am going to complete this now.

I have changed the *createLabels* function so that it looks like this.

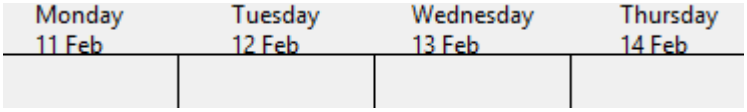
```
DateObject = datetime.datetime(self.currentDate[0], self.currentDate[1], self.currentDate[2], 0,0)
for q in range(0,7):
    text = DateObject.strftime("%A\n%d %b")
    self.create_text(x+(self.WIDTH/2),y, text=text, tag="daysLabel")
    x += self.WIDTH
    DateObject += datetime.timedelta(days=1)
```

This creates a date object for the first day the user is viewing. It then loops through all of the days creating that day's labels, and then adding 24 hours to the date object. I tested this and it worked, displaying the date successfully:



Now that this is working, I can now test my day left and right buttons. I pressed the left button and it changed the week to the 11th Feb, which was correct for the week I was on when I pressed it.

I then pressed the right button and this went 7 days forward as I expected to the 18th Feb



This worked as I expected taking me back to the date that I started on. I tested these buttons with the tests that I produced during the design process:

Changing date using the left button

Test Data	Type	Actual Outcome
Left click on button	**Valid**	**Valid**
Right click on button	**Invalid**	**Invalid**
Click outside button	**Invalid**	**Invalid**

Changing date using the right button

Test Data	Type	Actual Outcome
Left click on button	**Valid**	**Valid**
Right click on button	**Invalid**	**Invalid**
Click outside button	**Invalid**	**Invalid**

Changing date using the today button

Test Data	Type	Actual Outcome
Left click on button	**Valid**	**Valid**
Right click on button	**Invalid**	**Invalid**
Click outside button	**Invalid**	**Invalid**

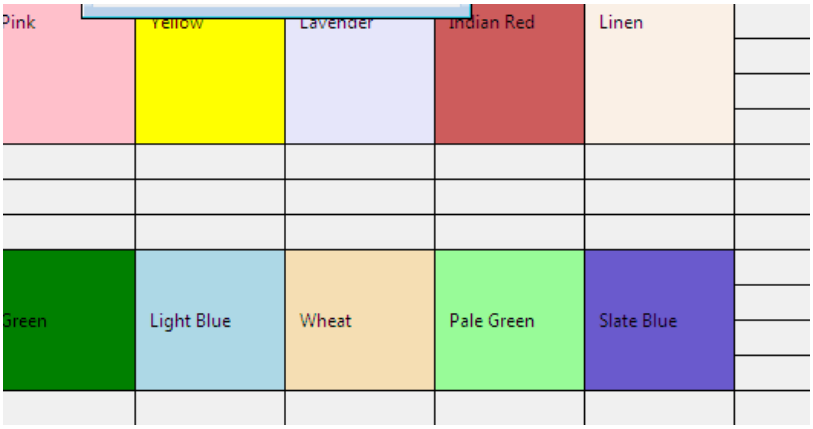
The next thing that I have worked on is the colours for the events. This is something that my stake holder, Ryan suggested that I do earlier when I was creating the event viewer. I did not want to do it at that point because I wanted to get on with the rest of the project, however I have come back to it now that I know

that I have enough time left. I have changed the colours array so that it looks like:

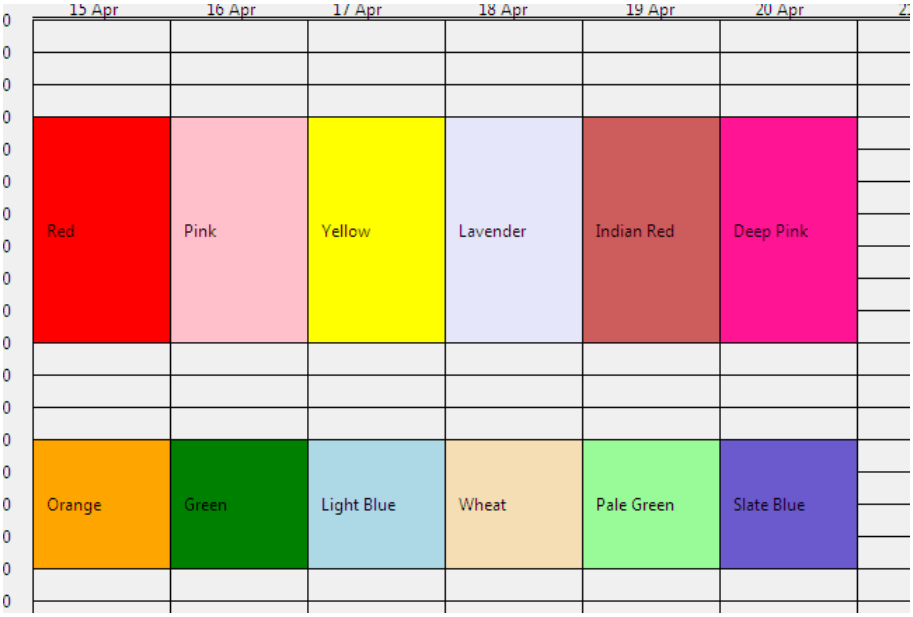
```
#Create Dropdown
colours = ["Red","Orange","Pink","Green", "Yellow", "Light Blue", "Lavender",
           "Plum 4", "Indian Red", "Pale Green", "Linen", "Slate Blue"]
```

I the created a list of events with different colours so that I could see what they looked like:

Ryan said "I think that pink and plum are very similar, so I would change one of those two. Other than those two I like all of the extra colours." Because of this I have changed plum to a colour called wheat: `"Wheat"`. I tested this and it looked like this:



Looking at this I think wheat and linen are also very similar colours. Also, if I did not have a colour list in front of me I would not know what shade linen would be, so I think a user will struggle as well. Because of this I am going to change linen to a colour called deep pink. I did this by changing the array, so it had: `"Deep Pink"`. I then tested this and it looks like this, which I think is a good selection of colours for the user to have.



The next thing that I am going to do is make it so that the user can edit their data from within the program. This makes the program follow the success criteria points:

No.	Requirements	Justification
24	A user can change the information stored about them	The Data Protection Act says data should be kept up to date and relevant.
25	When a user changes the information about them if goes through the same validation as when they create a new account	This is so that two different users cannot have the same name and email

In my initial designs I had a class called *editAccount*. I now think that it seems like a waste of space to have a completely new class for editing an account. For this reason, I am just going to modify the *createAccount* class and modify it so that it I can also edit an event from within the same form.

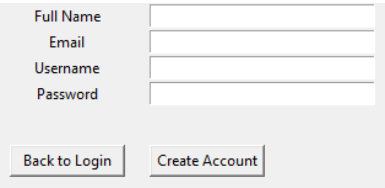
To do this, I am going to add a function called *editUser* which will get the data currently stored about a user and then insert this into the form. I am then going to have to change the *createAccount* function so that if it is editing a user it edits it rather than creates a new one. The first thing I have done is added the following variable to the __init__ function of the class. This means that I can test to see whether I am editing a user or creating a new one by testing whether this function is None.

```
self.userID = None
```

I then changed the following line in the mainPage class.

```
self.menubar.add_command(label=self.USER_DETAILS[0],
                         command=lambda: [self.controller.show_frame("createAccount"),
                                          self.controller.frames["createAccount"].editUser()])
```

This means that when I add the users name to the menu bar, I have added two commands to it, to show the create account page and to run the edit user function. I tested this and it showed an empty create event form as I expected.



I also tested this using the test table that I created in the design process:

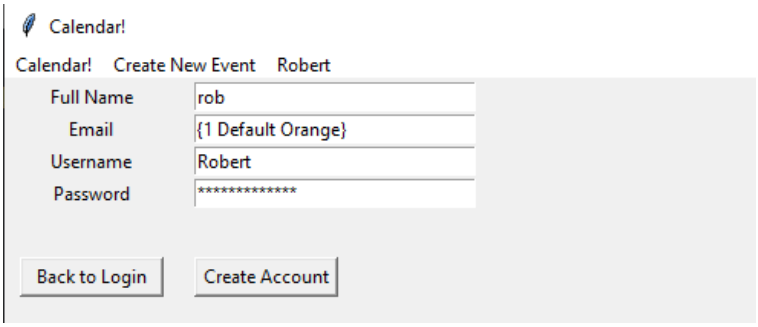
Account details button

Test Data	Type	Actual Outcome
Left click on button	**Valid**	**Valid**
Right click on button	**Invalid**	**Invalid**
Click outside button	**Invalid**	**Invalid**

These worked. The next thing that am going to do is write the *editUser* function. As I worked out earlier this needs to get the data about a user, and then populate the form.

```
def editUser(self):
    #Get User Details
    self.userID = self.controller.USERID
    details = self.DB.getUserData(self.controller.USERID)

    #Put user details info form
    self.username.insert(END, details[0])
    self.password.insert(END, details[1])
    self.name.insert(END, details[2])
    self.email.insert(END, details[3])
```

I have used my database function *getUserData* to get the data about the user I then go through all of the form elements and add the data to that text box. I tested this and got the following output:



This worked to some extent however all of the data is in the wrong place, for example the email is clearly not an email! I decided to print the users details because I probably have them in the wrong order from the output of the database function. I did this with the following line and then got the following output:

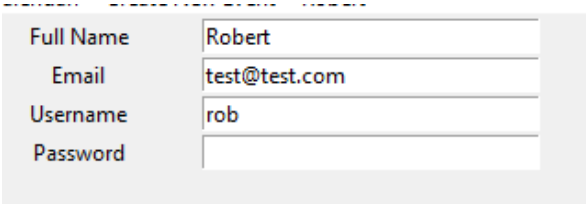
```
print(details)
```

```
['Robert', 'test@test.com', 'rob', [(1, 'Default', 'Orange')]]
```

What I have realised is that the password is not in this list. It does make sense why I did not put the password in the list; as the password is hashed, so it is of no benefit to the user to have the password displayed, in fact it would be dangerous as if the user left themselves logged in anyone would then be able to see their password. For this reason, I do not want to display any password in the form, so I have removed this line. I have also changed the others to the correct data from the array. The *editUser* function now looks like:

```
def editUser(self):
    #Get User Details
    self.userID = self.controller.USERID
    details = self.DB.getUserData(self.controller.USERID)

    #Put user details info form
    self.username.insert(END, details[2])
    self.name.insert(END, details[0])
    self.email.insert(END, details[1])
```

I tested this and it gave the output:



This worked as expected as all of the data is in the correct place in the form, with no password. The next thing that I need to do is create an 'edit user' function within the database class. This will allow me to edit the user in the database. I have created the following SQL to do this:

```
UPDATE Users SET Name = ?, Email = ?, Username = ?, Password = ?  WHERE
ID = ?
```

I started creating the edit user function, however I realised that was all of the validation for the email and username is in the create account function which means that I cannot reuse it when I am editing the user, because I still need to be able to validate everything when the users details are being edited. For this reason, I have taken the validation for the email and username from the create event function and put it into a new function called *checkIfUserExist.* This function takes an input of email and a username and checks to see if it already exists in the database. If it is in the database it returns an error message otherwise it returns false as the user does not exist.

```
def checkIfUserExist(self,email,username):
    #See if a user already exists
    sql = "SELECT Email,Username FROM users WHERE Email=? OR Username=?"
    self.cur.execute(sql, (email,username))
    rows = self.cur.fetchall()
    for user in rows:
        if user[0] == email:
            return "That email aready exists!"
        else:
            return "That username already Exists! Try Another one."
    return False
```

I then changed the create account function to use this function to validate if the user exists:

```
#See if a user already exists
check = checkIfUserExist
if  check != False:
    return check
```

I checked to make sure the *checkIfUserExist* function works, by trying to create a new user in the program. When I clicked create event I got the following error. This didn't work and produced the following error:

```
    check = checkIfUserExist
UnboundLocalError: local variable 'checkIfUserExist' referenced before assignment
```

I realised that I needed to add the function location when calling the function so I have changed a line in the *createAccount* function so that it looks like:

```
check = self.checkIfUserExist(email, username)
```

I then checked to make sure the validation still worked, and it did, as I expected. Now that I know that this works, I have created the *editUser* function. This takes 4 inputs name, email, username and user id. It can take a password; however, this is not required as a password may not be changed. This function validates everything and then edits the database.

```
def editUser(self, name, email, username, id, password=None):
    email=email.lower()

    #Make sure email is valid
    if self.validateEmail(email) == False:
        return "Email Not Valid"

    #Makes sure all othe feilds are valid
    if name == "" or username == "":
        return "Fill In Missing Values"

    #See if a user already exists
    check = self.checkIfUserExist(email, username)
    if  check != False:
        return check

    #Create SQL Statement and Values to pass
    if password == None:
        sql = "UPDATE Users SET Name = ?,Email = ?,Username = ?  WHERE ID = ?"
        values = (name,email, username,id)
    else:
        sql = "UPDATE Users SET Name = ?,Email = ?,Username = ?,Password = ?  WHERE ID = ?"
        values = (name,email, username,self.hashPassword(password),id)


    #Execute SQL
    self.cur.execute(sql,values)
    self.con.commit()
```

I initially tested the basic functionality of this function with the following code.

```
db= Database()
print(db.editUser("George","George@test.com","george",16))
```

Before I ran it the user looked like this

ID	Name	Email	Username	Password
16	Robert 2	test1@test.com	rob	ccc9c73a37651c6b35de64c3a37858ccae045d285...

I ran it and got the output of *None* which is what I expected and then checked the database and the 3 fields I was expecting to change had changed, which means this basic functionality works.

```
None
```

ID	Name	Email	Username	Password
16	George	george@test.com	george	ccc9c73a37651c6b35de64c3a37858ccae045d285...

I then decided to test this with a new password as well. I tested this with the following code:

```
db= Database()
print(db.editUser("Bill","Bill@test.com","bill",16,password="qwerty"))
```

Before I tested the function the user looks like this:

ID	Name	Email	Username	Password
16	George	george@test.com	george	ccc9c73a37651c6b35de64c3a37858ccae045d285...

I then ran it and got an output of *None* and all the fields had changed, including the password hash. This means that everything that I was expecting to change worked, so this part of the function is successful.

```
None
```

ID	Name	Email	Username	Password
16	Bill	bill@test.com	bill	5154aaa49392fb275ce7e12a7d3e00901cf9cf3ab...

The final thing that I want to test with the *editUser* function is missing values to make sure it handles these correctly. I created the following code, with each one I have removed one value to make sure that each part of the validation is working. In theory nothing in the database should change because they are all missing values.

```
db= Database()
print(db.editUser("","Bill@test.com","bill",16))
print(db.editUser("Bill","","bill",16))
print(db.editUser("Bill","Bill@test.com","",16))
```

I checked what this user looks like in the database before the test:

ID	Name	Email	Username	Password
16	Bill	bill@test.com	bill	5154aaa49392fb275ce7e12a7d3e00901cf9cf3ab...

I then ran it and got the following three outputs which is what I expected from this test. I then checked the database to make sure that nothing had changed and it had not. This shows that the validation is working on the *editUser* function.

```
Fill In Missing Values
Email Not Valid
Fill In Missing Values
```

ID	Name	Email	Username	Password
16	Bill	bill@test.com	bill	5154aaa49392fb275ce7e12a7d3e00901cf9cf3ab...

I have realised that if the user is successfully edited in the database with no problems, I do not want it to return *None,* I want it to return *True* so that I know that it is successful. For this reason, I have added the following line at the bottom of the *editUser* function to make it return True.

```
return True
```

Now that I have all of the database functions required to get the edit account working, I now need to get this working with the form in the GUI. What needs to happen is when the user clicks the create account button it needs to check if the user is being edited and it needs to do the function the user wants (either edit or create account). If the user is not being edited it still needs to create a new account. I am going to create a new if statement for this.

I believe that I have done this by changing the *createAccountButton* function in the *createAccount* class:

```
#Check if user Exists and run
if self.userID = None:
    accountCheck = self.DB.createAccount(nameHolder,emailHolder,usernameHolder,passwordHolder)
else:
    if passwordHolder == "":
        passwordHolder = None
    accountCheck = self.DB.editUser(nameHolder,emailHolder,usernameHolder,self.userID,password = passwordHolder)
```

This checks to see if there is a user Id that is being edited and if there is, it edits the account, and if there isn't it creates a new one. If it edits an account, it also checks to see if a new password has been inputted. If it hasn't it sets the *passwordHolder* variable to a None type.

Once the account is edited, I also need to confirm it was successful, it will then go back to the main viewer. I have changed part of the *createAccount* function to look like this to do this

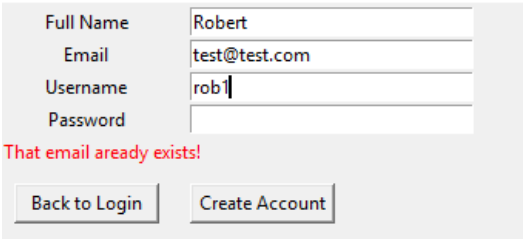
```
if accountCheck == True:
    if self.userID == None:
        #Log user in
        loginCheck = self.DB.checkLogin(usernameHolder,passwordHolder)
        self.controller.USERID = loginCheck[1]
        #Add Frame and Open
        self.controller.createFrame(mainPage)

    self.controller.show_frame("mainPage")

else:
    #Display Error Message
    self.errorLabel.config(text=accountCheck)
```

If an account is successfully edited or created then it goes back to the main page, if not it displays the error message.

I then decided to check this by changing the username of this user.

6	Robert	test@test.com	rob	d63dc919e20...

I opened the account and sent to the edit account screen and changed just the username to 'rob1'. I then got an error message:



This is technically correct, the email does exist within the database, however it exists in the user that is being edited, so I know that it exists. This is not what I want it to do, the validation is prevent a user from editing their own account. To fix this I need to make an exception in the validation, check all accounts that exist to see if an email or username already exists apart from the that users account.

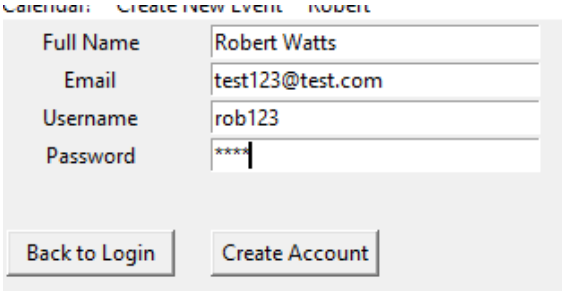
To do this I have modified the *checkIfUserExist* function, that is validating if the user already exists. I have added a parameter ID for the function, this defaults to None, so is not required when calling the function. If the ID is None it checks all of the users in the database when creating a new account. When editing an account, I can set the user ID I want it to ignore. This looks for all the accounts where the email or username are not equal to the one the user has entered, and it is not part of the current user. This makes sure that the current user is not checked against the database when validating them.

```
def checkIfUserExist(self,email,username, id=None):
    #See if a user already exists
    if id == None:
        sql = "SELECT Email,Username FROM users WHERE Email=? OR Username=?"
        self.cur.execute(sql, (email,username))
    else:
        sql = "SELECT Email,Username FROM users WHERE (Email=? OR Username=?) AND (ID <>  ?)"
        self.cur.execute(sql, (email,username,id))
```

I then ran the same test, just changing the username to ‘rob1’. This worked as I expected changing the username in the database.

6	Robert	test@test.com	rob1	d63dc919e20...

I then tested this function further by changing all of the values about this user as shown in the form below:

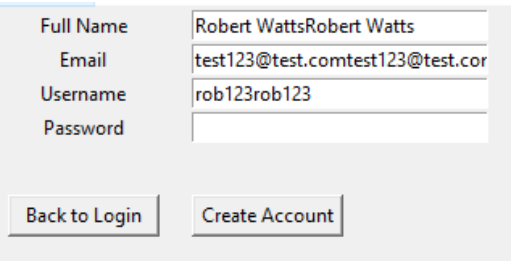


I then clicked the create account button and it went back to the main view. I checked the database and all of the items changed to what was in the database including the password hash.

6	Robert Watts	test123@test....	rob123	ccc9c73a3765...

This shows that this function was working as I expected. I then asked Ryan, my stake holder, what he thought about the edit account page. He said “It would be nice for instructions on the page explaining how to change the password. This would help tell users what to do. The ‘Create Account’ button could also change to an ‘Edit Account’ button as well. Something else that I have noticed is that when you go off the form, back to the calendar view and then back to the form the data in the text boxes is duplicated.”

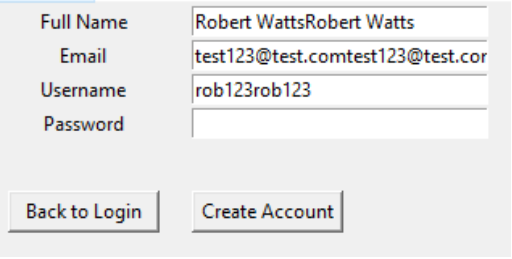
The first thing that I looked at is the duplication of data. I did as he said, and went on and off the form and he was correct, the data is duplicated within the text boxes as shown below:



This is an issue as it will cause errors when the user try's to edit their account. I think that I need to clear the form every time the create account button is pressed. I have added the following code before it shows the frame main page

```
#Clear Form and change frame
self.name.delete(0,END)
self.username.delete(0,END)
self.password.delete(0,END)
self.email.delete(0,END)
self.controller.show_frame("mainPage")
```

This clears all of the textboxes and sets the cursor back to position 0 in each one. I tested this and it did not work as shown below:



I think that the reason that this is happening is because it only clears it when the user clicks the button but if they just click away from the page using the menu bar at the top, it does not clear the form. Because of this I have created a new procedure called *resetForm.* This procedure needs to reset the form back to its normal state. The first thing I have added is the delete commands to clear all of the text boxes.

```
def resetForm(self):
    #Clears the form
    self.name.delete(0, 'end')
    self.username.delete(0, 'end')
    self.password.delete(0, 'end')
    self.email.delete(0, 'end')
```

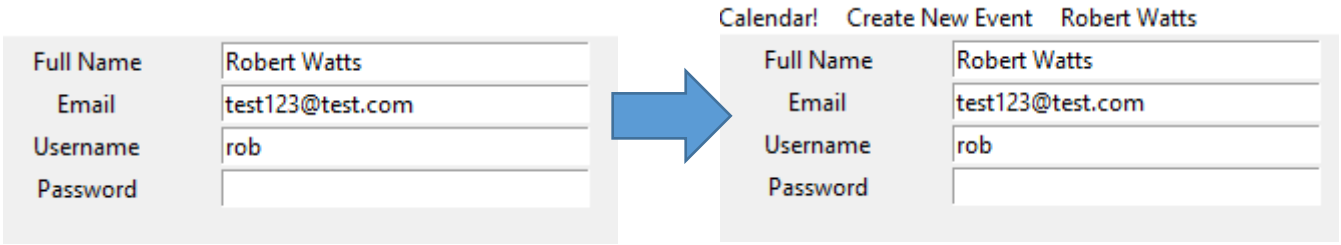
I then changed the button press function so that it looks like this – calling the function rather than deleting them with the commands.

```
#Clear Form and change frame
self.resetForm()
self.controller.show_frame("mainPage")
```

I then also called the *resetForm* function in the *editUser* function, so that before it populates the form, it clears all of the values that are in it:

```
#Get User Details + resets form
self.userID = self.controller.USERID
details = self.DB.getUserData(self.controller.USERID)
self.resetForm()
```

I tested this and now when I go from the edit account to the main view and back, it works, and there are no duplicated words in the form.



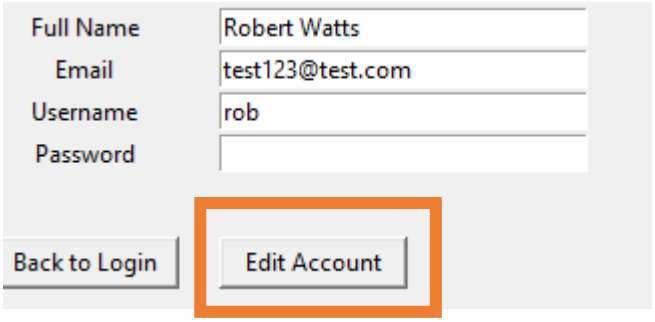
Ryan also suggested that I change the button so that it said “edit account” rather than having a “create account” button. Rather than delete and recreate a new button, I am just going to change the text on the button. I have added the following line to the edit user function so that the button text is changed:

```
self.createAccountButton["text"] ="Edit Account"
```

I then added the following line to the reset form function so that the button text is changed back to its default value:

```
self.createAccountButton["text"] ="Create Account"
```

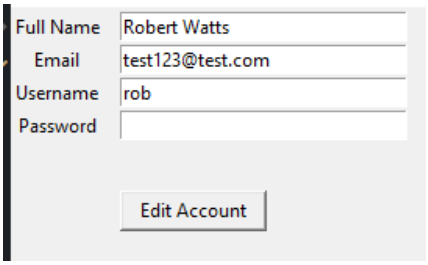
I tested this and it worked – the button text has been changed.



The “back to login” button is still there, which is unnecessary, as at this point the user is actually logged in. It is there because I am reusing the code from earlier, but it is still inelegant. I am going to remove this button when editing an account. I have done this with the following line of code added to the *editUser* function:

```
self.LoginButton.grid_forget()
```

I tested this and the button successfully disappeared:



I have also realised the need to reset the error label so that there is no text on it, as if you get an error and you click off of the form and back onto it the error message is still there even though the rest of the form is reset. To do this I have added the following line to the *resetForm* function:

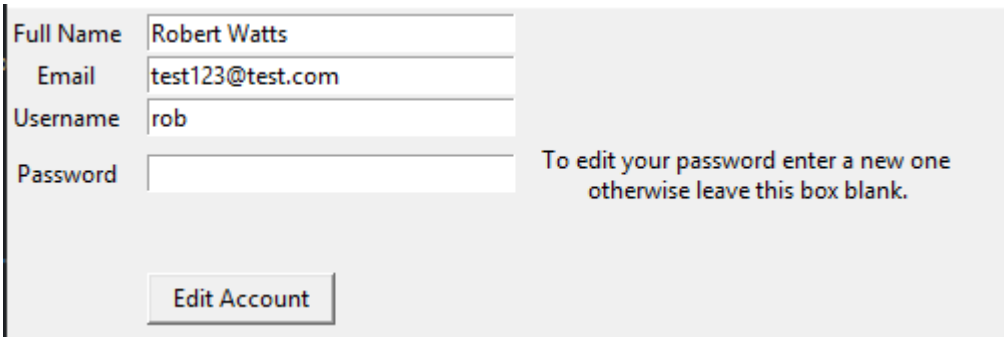
```
self.errorLabel.config(text="")
```

This clears the error label every time the form is reset. This works, meaning that error messages are deleted with the form.

The final thing that Ryan suggested was that I have some instructions on how to edit your password. I have added the following text label in to the *editUser* function.

```
#Add Instructions Text
instructionsText = Label(self, text="To edit your password enter a new one \notherwise leave this box blank.")
instructionsText.grid(column=2, row=3)
```

I tested this and it works – The instructions are successfully there.



The final thing that I have done in this session is to use the tests data that I produced at the beginning design process to test the edit account form:

Edit Account Form Validation

Test Data	Type	Actual Outcome
All form elements filled in	**Valid**	**Valid**
One form element not filled in	**Invalid**	**Invalid**

Test Data	Type	Actual Outcome
Username or email not already attributed to another user	**Valid**	**Valid**
Username attributed to another user	**Invalid**	**Invalid**

Email attributed to another user	**Invalid**	**Invalid**

Edit Account Form Complete

Test Data	Type	Actual Outcome
Left Click on create account Button	**Valid**	**Valid**
Right Click on event	**Invalid**	**Invalid**
"Enter" Key	**Invalid**	**Invalid**
Left Click on 'Back' Button	**Invalid**	**Invalid**

These tests worked as I expected meaning there are no issues with the edit account form. I am happy to move on.

Session 9 – 8/03/2019

The first thing I did today is have a look at the success criteria to see what I had not yet achieved. I believe I have done the majority of it, however one thing that I have not done, is "If the event name is too long to display on the main page, then it should cut the name so that it fits":

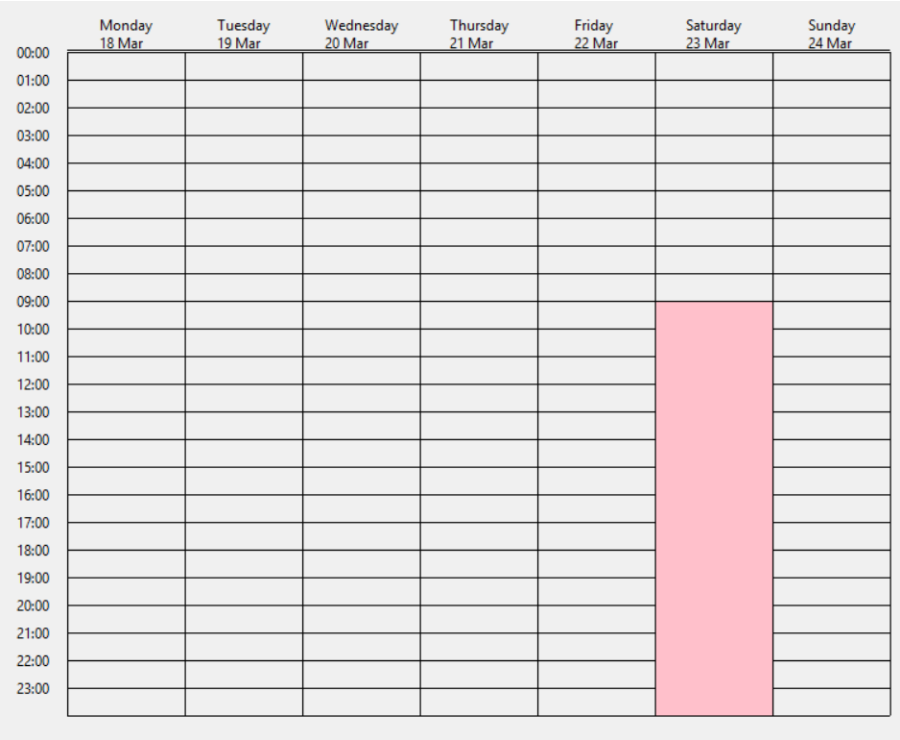
No.	Requirements	Justification
10	If the event name is too long to display on the main page, then it should cut the name so that it fits	This helps to keep the name presentable because you aren't trying to fit really long text in a small area.

To do this I need to modify the *EventViewer* class. I have changed the *drawEvents* functions so that instead of just creating text it looks at the length and depending on the length, it cuts it:

```
#Calculate event name width + shorten it
eventName = events[ID]["name"]
eventNameLength = len(events[ID]["name"]) * (0.2*widthDivider)

#Find the first space and cut it there, and add a ...
for x in range(eventNameLength, -1, -1):
    if eventName[x] == " ":
        eventName = eventName[0:x] + "..."
        break
#Draw Text
self.create_text((x1 + ((x2-x1)*0.1), 0.5*(y2-y1) + y1), text=eventName, anchor="w", width=(x2-x1)*0.8, tags=tag)
```

This gets the length of the *eventName*, and then makes it shorter depending on how wide the event is within the grid. It then loops each character in the event name backwards, from that point, checking to see if that character is a space, as I do not want to cut the event name mid word. If it is a space, then it gets rid of everything after it and the adds a '…' so that the user knows that it has been cut.

I tested this function but going to a date that I knew had different widths of events. I got the following output, which is wrong as I know there should be more than one event, and there also should have the event name on it.



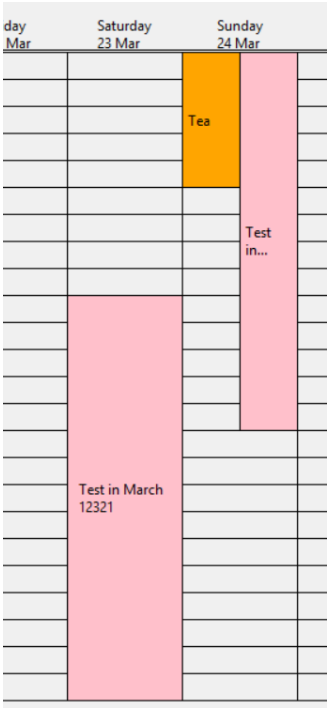
I had a look in the console and I had the following error:

```
'float' object cannot be interpreted as an integer
```

What I have realised is that currently when I do the maths to calculate the event length I am multiplying it by a decimal so that I can make it smaller. This has the effect that the *eventNameLength* is now float not integer. This becomes an issue when I start looping through the event name, because at the point it cannot be a float. I have made this number integer with the line below:

```
eventNameLength = int(len(events[ID]["name"]) * (0.2*widthDivider))
```

I tested this part of the function again and this time it worked. The events displayed correctly and on the one event where the length of the text was too long it got cut and the '…' was added.



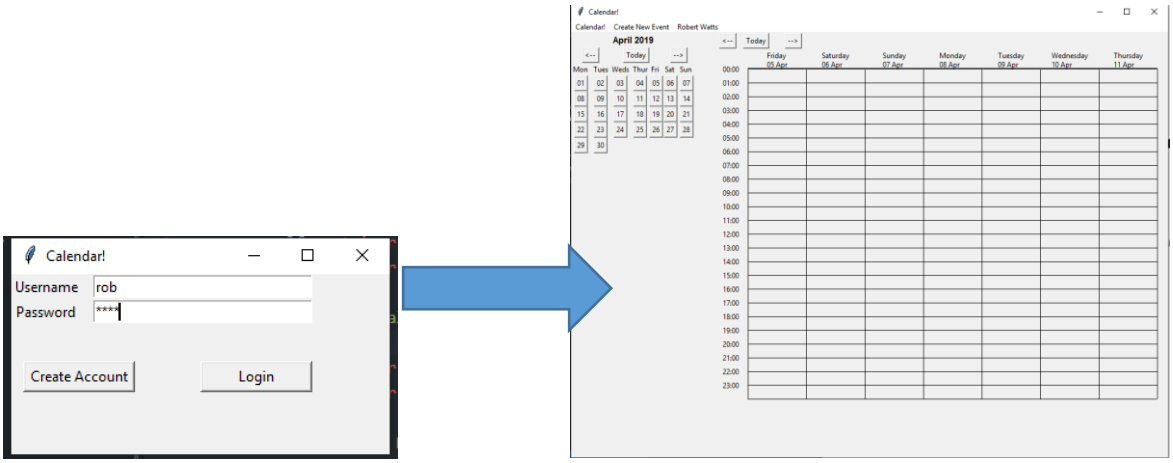
At this point I asked my stake holder Ryan to test my program properly as a whole, not just the individual parts, to see his thoughts and comments. You can see him using it in in **Video 4** in the supporting documents. The first thing that he did was create a new account, this worked for him and there were no problems. He logged in and created some events, a short one during a day and an all-day one to see this functionality. He then modified his username on the edit account page. He then closed the program, which logged him out.

Once he had done this, I asked him what he thought about how the program looked, felt, and how it was to use. He said "The program looks really good. Overall, I think that it fits the requirements that we looked at the beginning of your project. I think it is really easy to use and navigate. I really like the small calendar; it really helps with navigation around different dates. Navigation around the program is really easy because of the menu at the top. One thing that I did notice was that when I first logged in there was no calendar displayed until I selected a date. This is a little odd I think because it is just a blank page until that point. The other thing that I noticed was that the when I created an event it was not automatically put onto the calendar, I had to refresh the view from the mini calendar. This is a little annoying as it slows down creating events. Overall it is looking like a really solid project though!"

I agree with Ryan on both improvements that he suggested. This first one I am going to fix is when you login I want it to display the current date. I have added the following two lines of code the *__init__* function of the *mainView* class:

```
#Display the events on Current Date
now = datetime.datetime.now()
self.calendarClick((now.year, now.month, now.day))
```

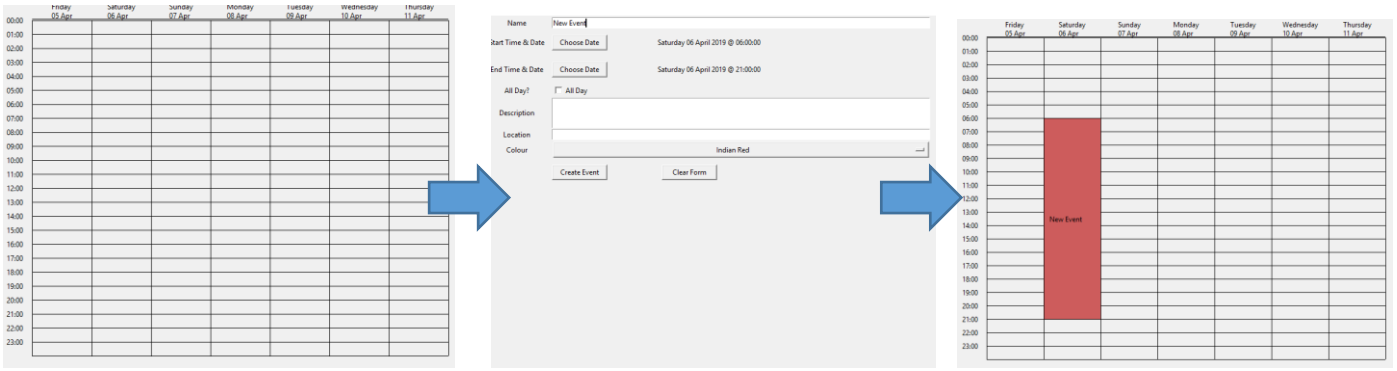
This gets the current date saving it to a variable called *now*, and then simulates a calendar click passing it to my *calendarClick* function. I tested this function and as you can see in the screen shots below the calendar displayed the current date in the on the canvas:



The other thing my stakeholder Ryan said was that he did not like that that he had to refresh the calendar when a new event is created. I agree with him on this it is a bit of a pain. When I was looking at the problem, I remembered that I made the calendar refresh when I edited an event. I have taken that line out of an if statement in the *CreateEventButtonPress* function.

```
#Refresh Calendar View
self.controller.frames["mainPage"].eventViewerHolder.displayEvent(self.controller.frames["mainPage"].eventViewerHolder.currentDate[0],
                                                                  self.controller.frames["mainPage"].eventViewerHolder.currentDate[1],
                                                                  self.controller.frames["mainPage"].eventViewerHolder.currentDate[2])
```

This line now runs whenever there is a new event created as well as when an event is edited. I tested this by starting off with an empty calendar. I then created a new event, and it displayed it this time.



This worked and I did not need to refresh the page for it to display the event.

Evaluation

Testing for Evaluation

Account

This is testing all of the sections that deal with the users personal information.

Table I: Account Testing

No.	Requirements	Input	Expected Outcome	Actual Outcome	Video Reference
1	Login Screen	**N/A**	There is a form that allows the user to login	As Expected	**Video 5 – Login.mp4**
2	When the user clicks login it checks that they have access from a database	**Valid**: A left mouse click on a login button checks with username and password filled in	**Valid**: Check the username and password against the database, and give an error message if not correct	As Expected	**Video 5 – Login.mp4**
		Invalid: A left mouse click where either the username or password are not filled it	**Invalid**: An error message telling the user to fill in the username and password	As Expected	**Video 5 – Login.mp4**
3	Create Account	Clicks Create Account button	There is a form that allows the user to create an account	As Expected	**Video 6 - Create Account.mp4**
4	Checks user does not already have an account	**Valid**: The username, email, password, and name filled out, and a left click on the create account button	**Valid**: The inputs are validated and checked against the database	As Expected	**Video 6 - Create Account.mp4**
		Invalid: At least one of the following not filled out: username, email, password, and name. Then a left click on the create account button	**Invalid**: An error message is shown to get the user to fill in the “Create Account” form.	As Expected	**Video 6 - Create Account.mp4**
24	A user can change the information stored about them	**Valid**: A user can change their email, name, username and password.	**Valid**: There is a form that allows the user to change all the information about them	As Expected	**Video 7 - Edit Account.mp4**

Navigation

This section will test how easy it is to navigate through the program, and different dates.

Table II: Navigation Testing

No.	Requirements	Input	Expected Outcome	Actual Outcome	Video Reference
6	A mini calendar for quick navigation	**N/A**	There is a mini calendar in the top left corner so that the user can quickly move between dates.	As Expected	**Video 8 – Navigation.mp4**
13	A menu bar at the top detailing different options	**N/A**	A menu at the top with the different options for a user to choose from.	As Expected	**Video 8 – Navigation.mp4**
15	Navigation Buttons at top of page	**N/A**	A left and right button to move the event viewer should be at the top of the window.	As Expected	**Video 8 – Navigation.mp4**
16	A "Today" Button to skip to today's date	**Valid**: Left click on "Today" Button	**Valid**: The event viewer is moved to that days date	As Expected	**Video 8 – Navigation.mp4**
			Invalid: The event viewer does not go to today's date.	As Expected	**Video 8 – Navigation.mp4**

Events

This section will test all of the parts that look at events. It will test the creating, editing and deleting of events. It will also test viewing of events.

Table III: Events Testing

No.	Requirements	Input	Expected Outcome	Actual Outcome	Video Reference
7	A view to see all the events that week	**N/A**	The user should be able to see all their events within a week on the screen in chronological order.	As Expected	**Video 9 - Event Veiwer.mp4**
8	The week's events should be split into days and hours in the view on the main page	**N/A**	The user should be able to see the events on a grid with each day split into hours when logged in and navigating around the program.	As Expected	**Video 9 - Event Veiwer.mp4**
9	When viewing events on the main page, the user should see the	**N/A**	The events should be coloured when viewing them in the main view	As Expected	**Video 9 - Event Veiwer.mp4**

	colour of the event and the name.		and the text should be black.		
10	If the event name is too long to display on the main page, then it should trim the name so that it fits	**Valid**: Have an event called "This is a really long name for an event" and then navigate to it in the event viewer	**Valid**: The name of the event is trimmed depending on the width of the event for example it might be "This is a…."	As Expected	**Video 9 - Event Veiwer.mp4**
		Invalid: Have a event with a short name, called "Lunch" and navigate to it in the event viewer	**Invalid**: The event name is not cut and is displayed in full.	As Expected	**Video 9 - Event Veiwer.mp4**
11	When an event is clicked on it should display information about it	**Valid**: Left click on an event	**Valid**: The information about the event clicked on is displayed.	As Expected	**Video 9 - Event Veiwer.mp4**
		Invalid: Not clicking on an event or using a right click	**Invalid**: Nothing should happen	As Expected	**Video 9 - Event Veiwer.mp4**
14	An all-day event is shown at the top of that day	**Valid**: A event that is marked as all day in the database	**Valid**: The event is displayed at the top of the day, before midnight.	As Expected	**Video 9 - Event Veiwer.mp4**
		Invalid: A event that is not marked as all day in the database	**Invalid**: The event is not displayed at the top of the day but within the rest of the grid.	As Expected	**Video 9 - Event Veiwer.mp4**
17	Create event page	**N/A**	There is a form that a user can use to create a new event	As Expected	**Video 10 - Create, Edit & Delete Events.mp4**
19	Delete events	**N/A**	A user can delete an event when viewing more details about the event.	As Expected	**Video 10 - Create, Edit & Delete Events.mp4**
20	Edit Events page	**N/A**	There is a page that allows the user to edit the information about the event.	As Expected	**Video 10 - Create, Edit & Delete Events.mp4**
27	User can change colour of event	**Valid**: The user can change the colour of the event	**Valid**: When creating or editing an event there is a dropdown allowing the user to edit the event colour	As Expected	**Video 10 - Create, Edit & Delete Events.mp4**

			Invalid: The user cannot change the colour of an event	As Expected	**N/A**
28	There are at least 5 colours that a user can choose from for an event	**N/A**	**Valid**: There are at least 5 colours in the dropdown	As Expected	**Video 10 - Create, Edit & Delete Events.mp4**
			Invalid: There are less than 5 colours	As Expected	**N/A**
29	The colour of an event defaults to the colour of the calendar	**N/A**	When creating an event the default colour in the dropdown is the same colour as the calendar colour.	As Expected	**Video 10 - Create, Edit & Delete Events.mp4**
30	An event can be set to all day	**N/A**	There is a checkbox when editing or deleting an event to set it to all day	As Expected	**Video 10 - Create, Edit & Delete Events.mp4**

While testing this part of the program I found an error in the event viewer when editing two events. This can be seen in "***Video 13 - Error in event Veiwer.mp4***" in the supporting documents. In this video once I have edited one event, they overlap which is correct, however when I change the colour of one, the colour of both changes. This is not what I would have expected to happen. This is the first time I have come across this issue, it has not come up before. Due to time constraints on the project, I do not have time to fix this, however this is something that I would improve if I had more time. In normal operation the event viewer, and the edit event functions work fine without issue. I am doing it on a very slow school computer rather than at home. Because of this I think that this is causing some issues, but it is something that I would have to have a look at in the future.

Database

In this section I will test the database against my success criteria.

Table IV: Database Testing

No.	Requirements	Input	Expected Outcome	Actual Outcome	Video Reference
5	If an account already exists then an error message will be shown	**Valid**: An username or email that is in the database	**Valid**: An error message something like "Account already exists in the database"	As Expected	**Video 6 - Create Account.mp4**
		Invalid: A username or email that is not in the database	**Invalid**: No error message is shown to the user.	As Expected	**N/A**

18	An event must have a name and start date at a minimum, otherwise an error message is shown.	**Valid**: The event has a start time, end time and a name. The end time is after the start time.	**Valid**: A event is created in the database.	As Expected	**Video 11 - Create Event Validation.mp4**
		Invalid: The event is missing either a start time, end time or a name. The start time may also be after the end time.	**Invalid**: Display an error message because it is either missing data or it is starting before it ends, and an event cannot do this.	As Expected	**Video 11 - Create Event Validation.mp4**
		Boundary: The event has a start time, end time and a name. The end time is equal to the start time.	**Boundary:** Display an error message because an event cannot have a length of 0	As Expected	**Video 11 - Create Event Validation.mp4**
21	When an event is edited it should go through the same validation as when creating an event	**Valid**: The event has a start time, end time and a name. The end time is after the start time.	**Valid**: A event is created in the database.	As Expected	**Video 12 - Edit Event Validation.mp4**
		Invalid: The event is missing either a start time, end time or a name. The start time may also be after the end time.	**Invalid**: Display an error message because it is either missing data or it is starting before it ends and an event cannot do this.	As Expected	**Video 12 - Edit Event Validation.mp4**
		Boundary: The event has a start time, end time and a name. The end time is equal to the start time.	**Boundary:** Display an error message because an event cannot have a length of 0.	As Expected	**Video 12 - Edit Event Validation.mp4**
22	A database to store user data, events etc	**N/A**	There is a database file to store all the information about a user and their events	As Expected	**N/A**
23	The personal data about a user is email, name, username and password	**Valid**: The information stored about a user is: email, name,	**Valid**: This is the information about a user	As Expected	**Video 6 - Create Account.mp4**
			Invalid: There should be no more	As Expected	**N/A**

		username and password.	information collected about the user when they create the account nor at any other point.		
25	When a user changes the information about them it goes through the same validation as when they create a new account	**Valid**: A valid email and a username that is not attributed to another user.	**Valid**: The account information is edited in the database	As Expected	**Video 7 - Edit Account.mp4**
		Invalid: A email address that is not valid or a username that is attributed to another user.	**Invalid**: An error is displayed to the user.	As Expected	**Video 7 - Edit Account.mp4**
26	Store title, description, and location of an event. As well as start and end times	**Valid**: The information stored about an event is: title, description, location, start and end times.	**Valid**: This is the information about an event	As Expected	**Video 10 - Create, Edit & Delete Events.mp4**

Window

This section is to test that the window operates correctly.

Table V: Window Testing

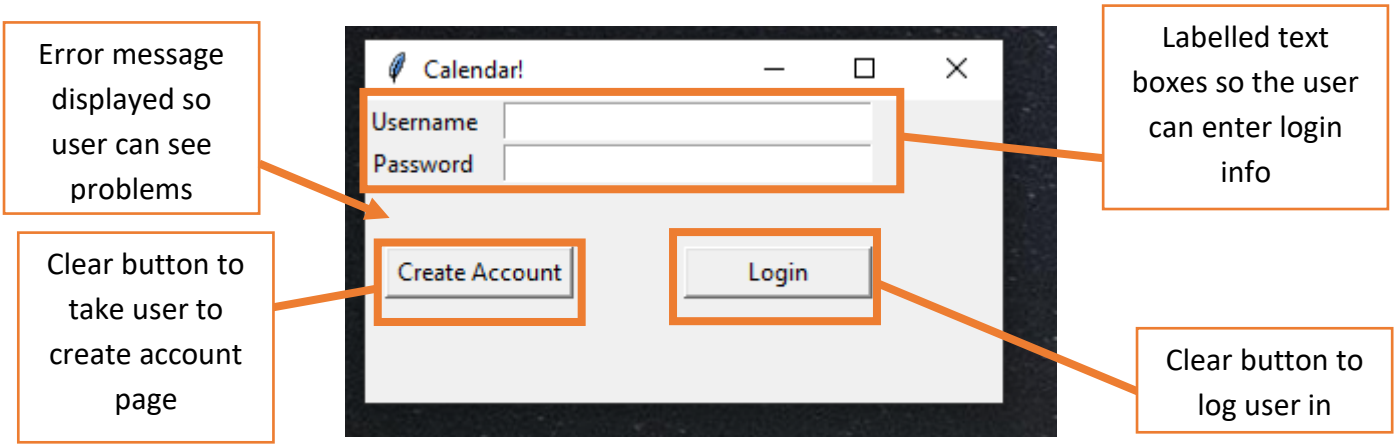
No.	Requirements	Input	Expected Outcome	Actual Outcome	Video Reference
12	When the user clicks on an event the colour of that event should become the background	**Valid**: Left click on an orange event	**Valid**: The background of the window should become orange	As Expected	**Video 10 - Create, Edit & Delete Events.mp4**
		Invalid: Not clicking on an event or using a right click	**Invalid**: Nothing should happen	As Expected	**Video 10 - Create, Edit & Delete Events.mp4**
31	The title of the window should say “Calendar !”	**N/A**	The top of the window should say “Calendar!”	As Expected	**Video 5 – Login.mp4**

Usability Features

In this section I will annotate images of the calendar showing all the usable features. I am also going to follow each one was a comment from my stakeholder Ryan and myself.

Account

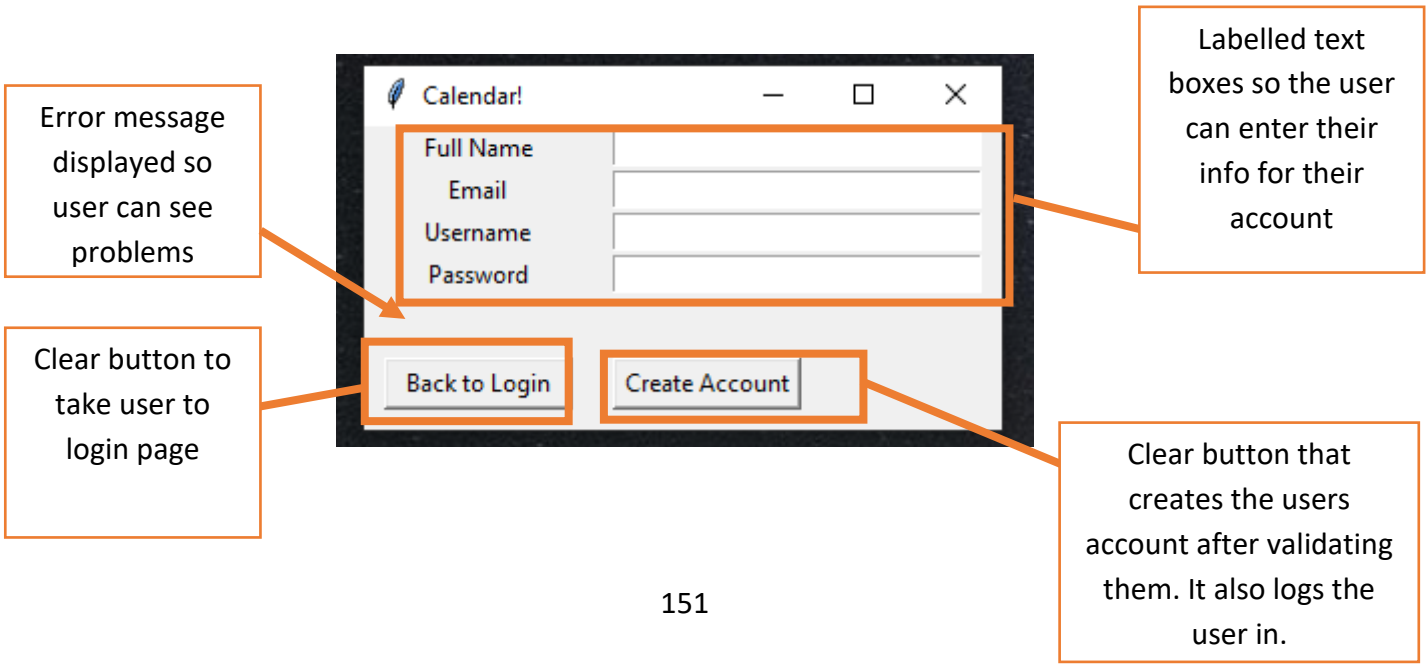
Login



Me: This form works really well, it is easy to use and it is clear where all of the information should go. It is like other login forms which allows the user to already know how to use it. We did speak about making this form fit the page, as currently there is a bit of a gap. We also spoke about styling it a little better as at the moment it does look a little rough and ready, however due to time constraints on the project I have not had time to implement either of these. It does work though.

Ryan: I agree with everything you said. It is clear and easy to use. I remember us speaking about this when you created the login form and I do think that you could do jazz it up a bit more with a logo and a bit of styling.

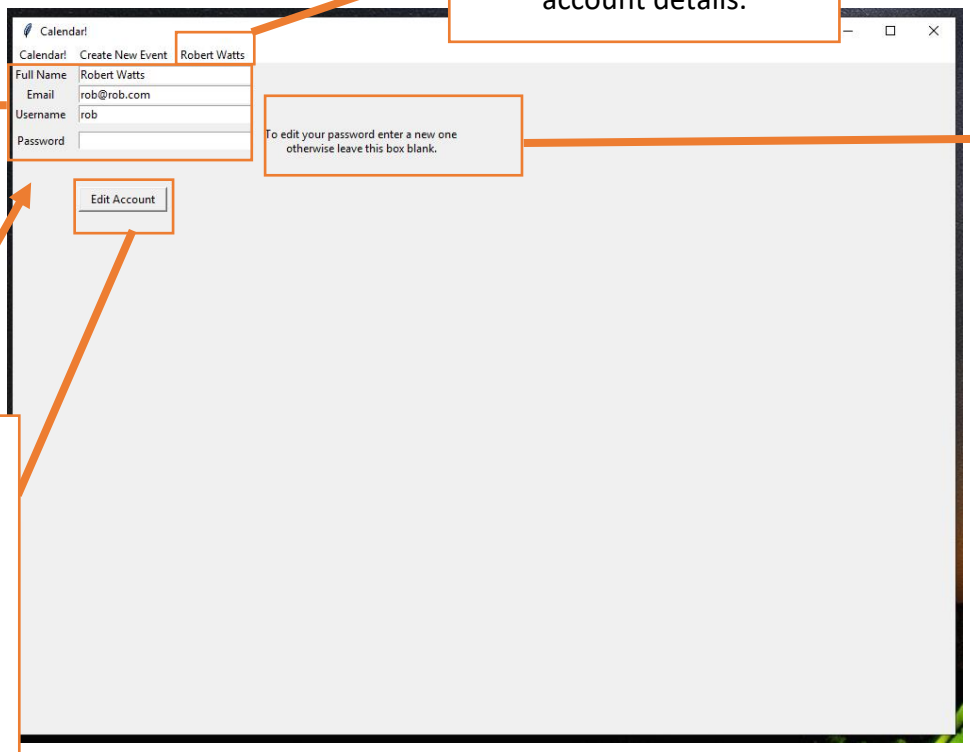
Create Account



Me: The form is a clean and fits the page well. The controls on the page are clear and well defined. The fact that the user cannot have multiple accounts helps to keep the program compliant with the Data Protection Act. The form is easy to use so a user shouldn't have any problems. I could add instructions in future version as well as email verification, but as a basic form this works.

Ryan: Like I said with the login form, it would be nice to have a bit of colour – Jazz it up! You could also add a logo. Another thing that a lot of programs have is terms and conditions which may be something that could be implemented before production to protect yourself. The form itself is functional and easy to use, so no problems there.

Edit Account



Labelled text boxes so the user can enter their info for their account

Users name is displayed. When clicked on it opens a form to edit their account details.

Clear instructions on how to edit the password

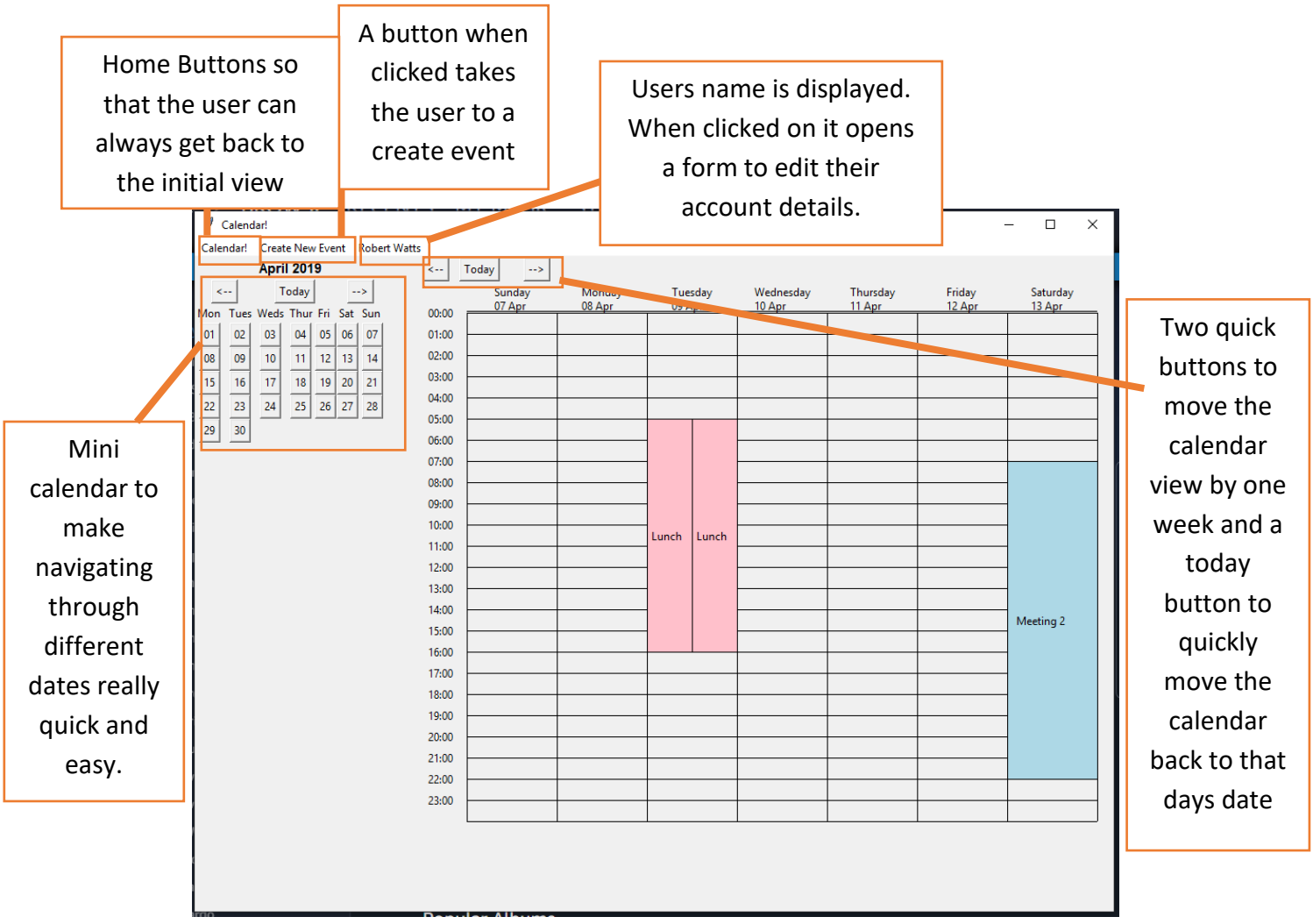
Error message displayed so user can see problems

Clear button to validate the data and change it within the database. This then takes the user back to the main page.

Me: This form is clear and easy to use. I agree with Ryan's suggestion while I was developing this form that it should have the instructions. I think it does look a bit odd because it is a small form right in the top corner of the window. This form could be made bigger, and make it look a bit better.

Ryan: I am glad you took the feedback I suggested with adding the instructions for the password. I agree with you when you say the form needs to be bigger, it does. Ultimately the form does do what it is suppose to. Another thing you could add in the future is add a title to the page, so the user knows what page they are on.

Navigation

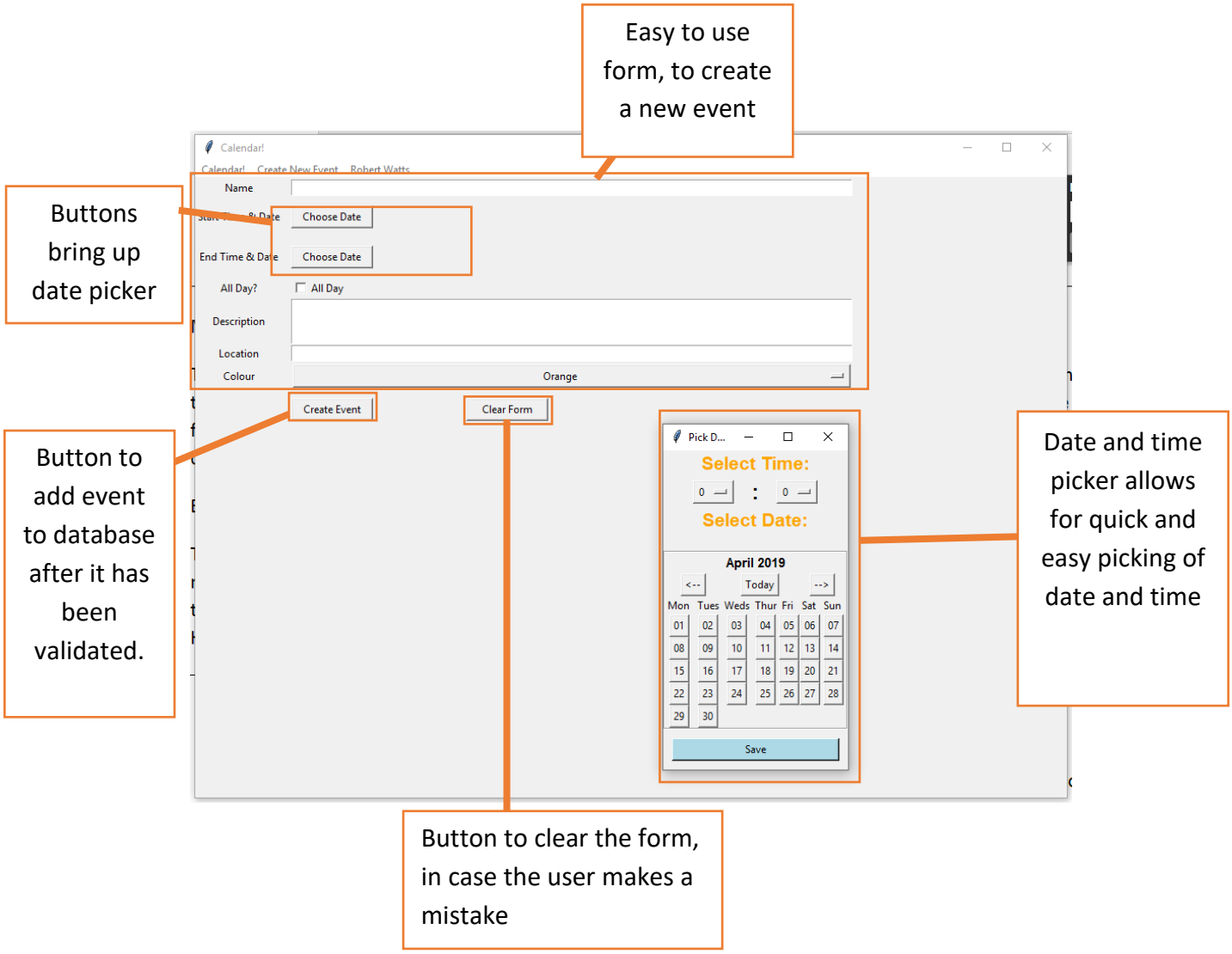


Me: I think that navigation around the program is fairly easy. I am really proud of the mini calendar; I think this came out really well! I think that the nav bar at the top is a little confusing. The Calendar button does not immediately scream 'Home' so can be a little confusing, I could change this. It is also not immediately obvious that clicking your name in the navbar will take you to edit your account details. These are things that I would look at. The 3 buttons that go Left, Right and Today work really well, and I think they are clear for the user.

Ryan: I agree most of what you have said. The mini calendar is cool! I think that the three buttons can work as they are. I think that people are used to clicking a logo to go home, from websites so you could change the home button to something like this. Ultimately it does work. A logout button might be nice but as you can log out by closing the program it is not the end of the world.

Events

Create & Edit Event Page



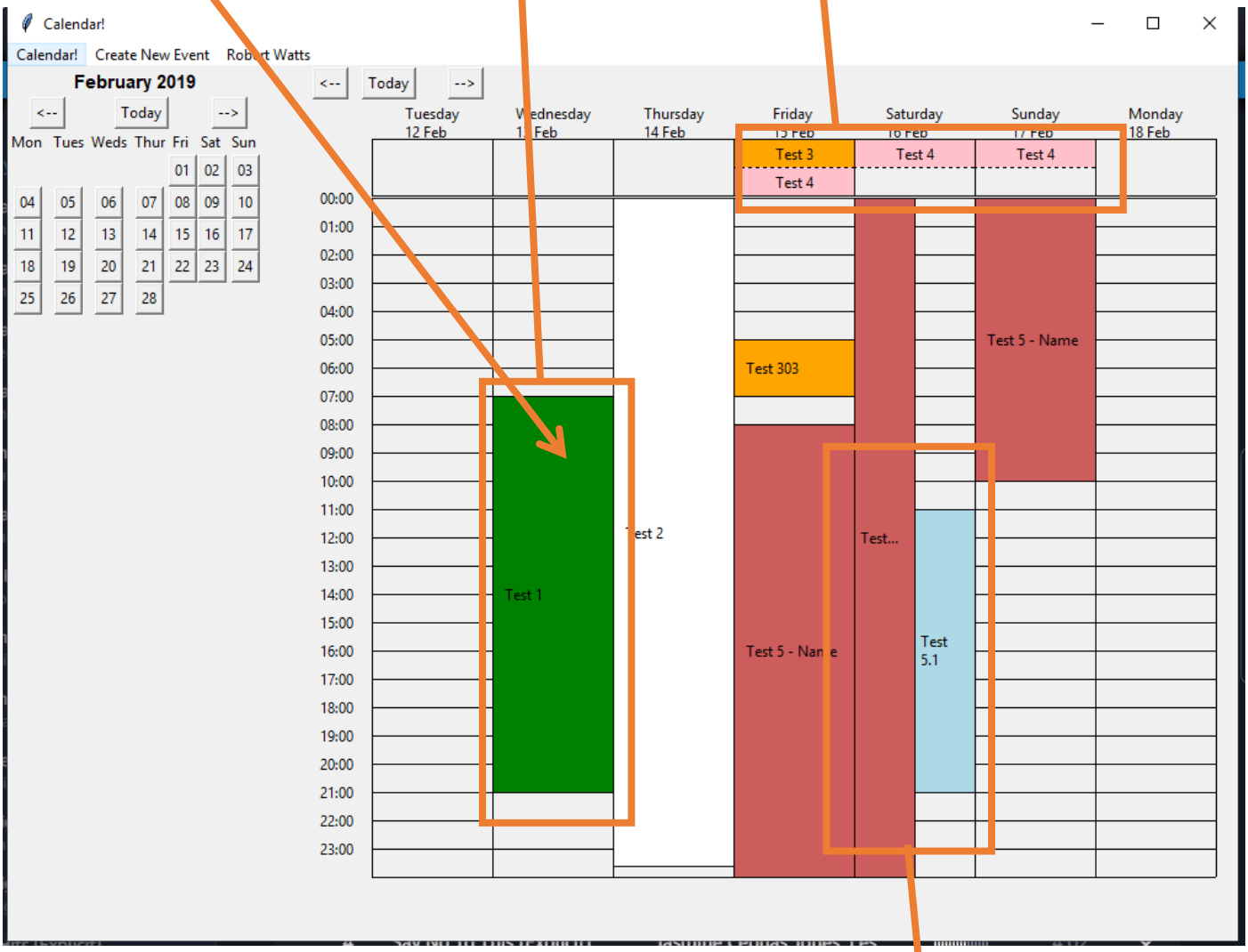
Me: This form came out well. Ryan and I discussed how it should look and the date time picker popup was the right idea I think. It keeps the initial form clean, but adds a lot of functionality by opening up. I think it might be nice to have a title at the top of the page so the user knows that they are creating a new event.

Ryan: I agree with what you have said there. It was the right idea to have the popup. I think it would be nice to have a title at the top of the form but I don't think it is the end of the world overall.

Click on an event to get more information about it

Events are in the grid with their text and colour

Events that are marked as all day are put at the top of the page.

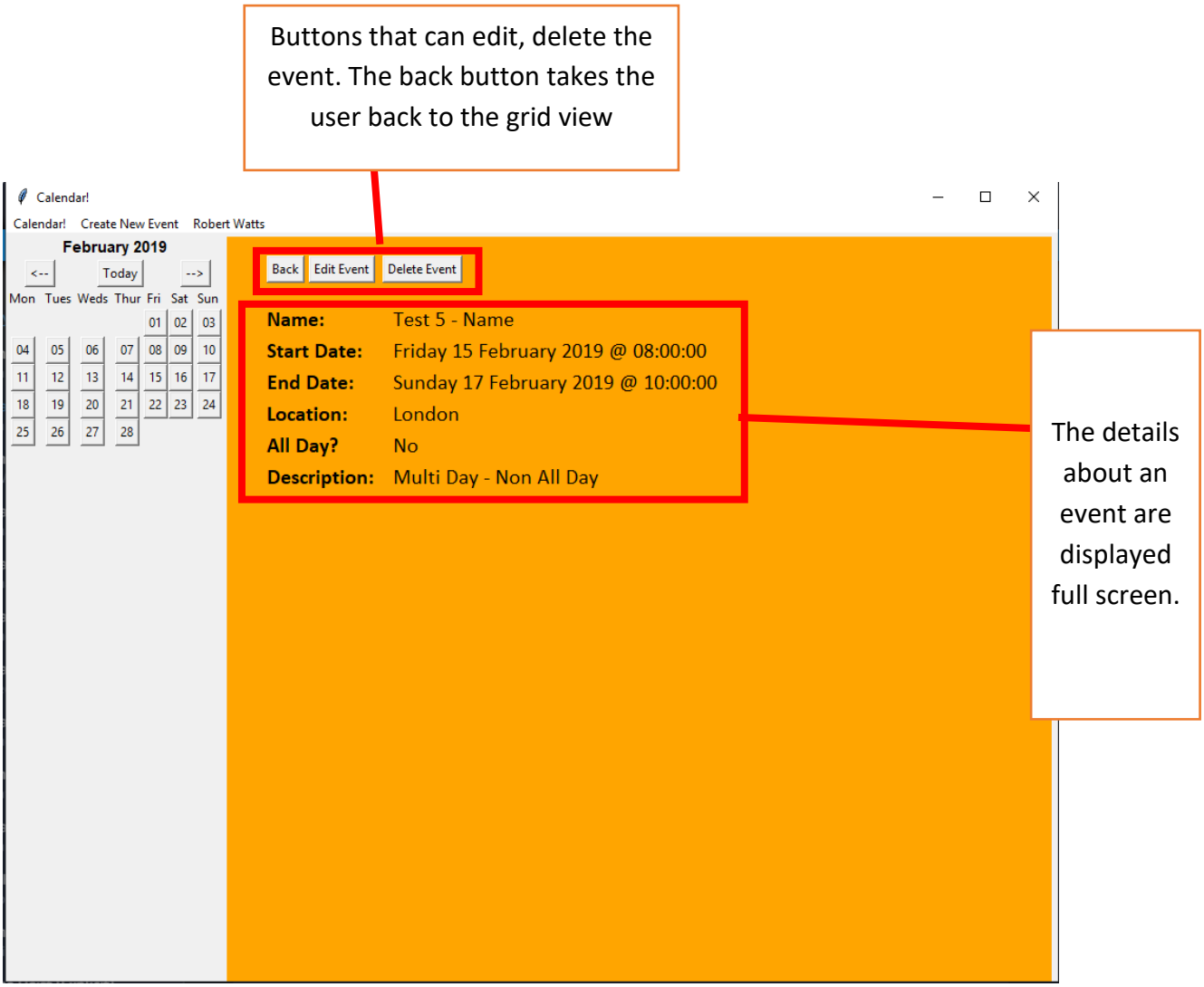


Events that are on at the same time scale so that they all fit.

Me: I really like how this all came together. Seeing all the events in a view like this really helps the user see what events are going on at any time. I think that the fact that the events scale works really well overall, it means that the events do not overlap which I think is important for the user to see what is going on. The text does get shortened if the event name is too long to fit in the box, and I think this also helps. Having all day events at the top works really well. The different colours are good as well.

Ryan: I agree with everything you have said this is an amazing screen. I think it is very clear on what is going on in my week.

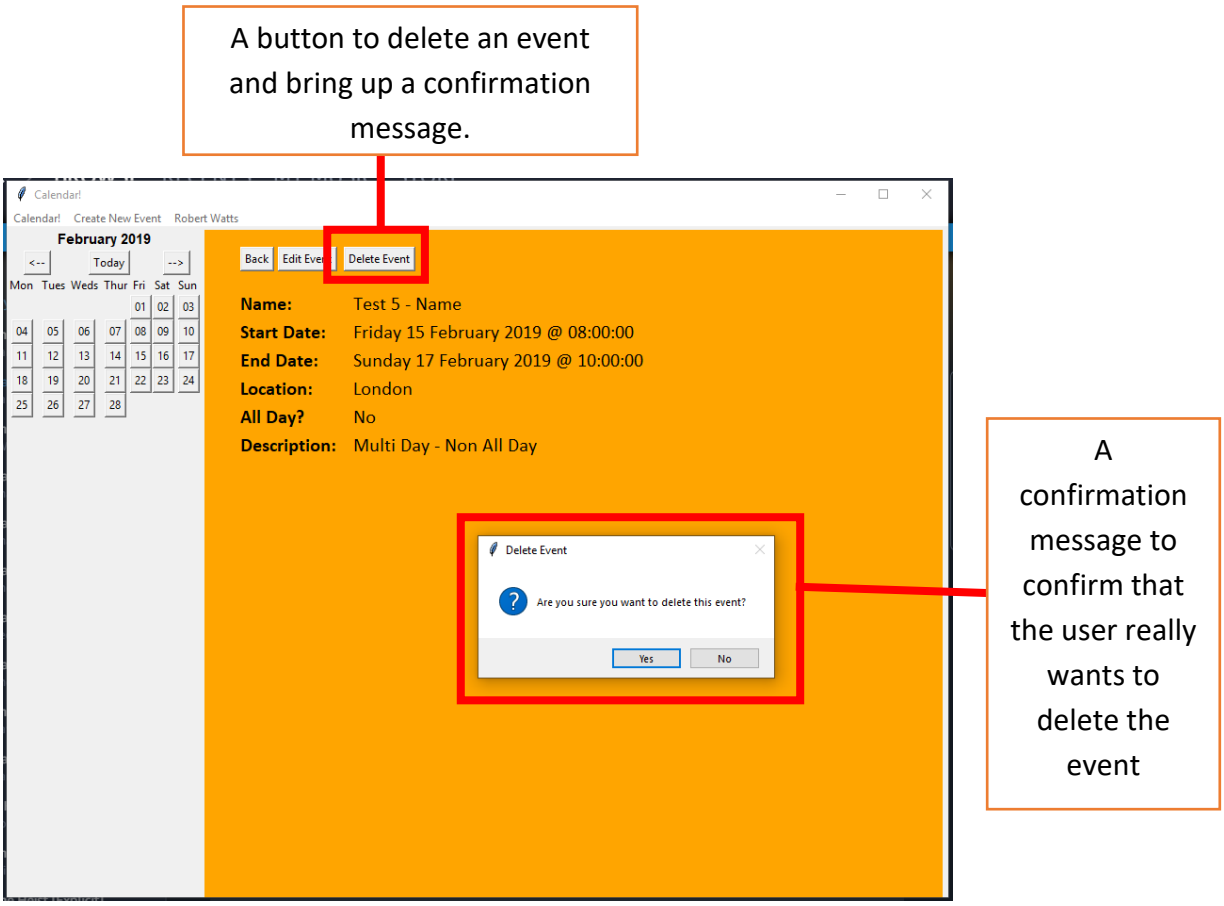
Event Details



Me: This full screen event details viewer works well. The details about an event are very clear; I like how big they are. I like the fact that if one of the details is not in the database it is not displayed on this page; it helps to keep the page clear. The buttons are good, clear and I think that they are in the right place on the page. One thing that I think is that this page could be more of a popup, meaning that you could view all of the events in the grid while viewing the details of an event.

Ryan: I agree that this works well. I personally disagree with you about this page being a popup, I think this fits the program really well. The buttons are really clear, and I agree that they are in the right place on the page.

Delete Event



Me: This works well. I think it is really self-explanatory how to delete an event, and I think that it is good the program confirms that the user definitely wants to delete the event.

Ryan: I agree with all of that. I like the fact that it is not too easy to delete an event – I cannot do it by accident.

How Well Does The Solution Match The Success Criteria

In the following section I will look at the success criteria that I created after the research phase. I will look at how well the solution matches each point in the success criteria, and any changes were made during the development phase.

Account

No.	Requirements	Justification
1	Login Screen	To protect the users events from unauthorised access
2	When the user click login it checks that they have access from a database	This stops unauthorized access to the calendar. By using a database it make easy to change data about the user.
3	Create Account	A new user will need to create an account
4	Checks user does not already have an account	This is to stop duplication of data. This also helps to keep data relevant about a user

24	A user can change the information stored about them	The Data Protection Act says data should be kept up to date and relevant.

Requirement number 1 was completed. The Login screen is the first screen that a user comes to when they open the program. Requirement number 3 was also completed, and the user can get to the create account page from a button on the Login page. I did not specifically say this in the success criteria, but both forms were in a GUI as was the rest of the program.

When I drew the GUI diagrams, I was collecting the date of birth in the Create Account form. This would have been part of requirements 3 and 24. In the end I decided that it was not needed for my program to store the date of birth of a user, so I did not have this on the final form. This form is fairly extendable so if in the future I wanted to add this back in, I could, by drawing it on the form. I would also need to add some columns to the database to do this.

Requirement 2 was completed, there is a relational database that has all of the data held by the project. The testing for this success criteria can be found on page 145 This is same database that is used to complete requirement 4. The program does successfully prevent two users having the same username or email. The testing for this can be found on page 145.

Requirement 24 was completed when I created the edit account page. In my initial classes diagram the class I created in the design process I was called *editAccount*. Overall, I felt that it was inefficient to have this in a separate class, as a lot of the code is, if not the same, very similar. For this reason, I decided to put all of these functions in the *createAccount* class. This helped to decrease the space complexity of the program, by not having duplication of code. If I had thought about this in the planning stage I would have renamed the *createAccount* class to just *Account*, as at the moment the name does not really describe all of what it does. This is something that I would look at changing in the future.

Navigation

No.	Requirements	Justification
6	A mini calendar for quick navigation	To make it really easy for a user to find the dates that they want.
13	A menu bar at the top detailing different options	Navigation should be easy, keeping everything at the top is where the user will expect it to be.
15	Navigation Buttons at top of page	To make it easy to navigate between weeks
16	A “Today” Button to skip to today’s date	Quick to find the today, so you can see what you should be doing now

Requirement number 6 was completed. This works really well, and because it is its own class, it allowed me to reuse the mini calendar in the create event form, which helps to make the program as efficient as possible. The fact this is separate to the rest of the event viewer really speeds up navigation.

I did have a menu bar at the top of the page to satisfy Requirement 13. In the initial GUI diagrams that I drew for what I wanted the program to look like, I had a button to take the user to edit their account details on the right of the bar, with all of the other buttons on the left. Like this:

In the end the users name ended up being on the left with the rest of the buttons. This is something that I was going to come back to, however I ran out of time to complete this properly. Ultimately though this requirement was fulfilled.

Requirement 15 and 16 are very similar. They are both buttons that ended up at the top of the page. I realised while I was developing the solution that these buttons were omitted in the GUI diagrams that I created. I ended up putting them at the top of the page, and I think that position works really well.

Events

No.	Requirements	Justification
7	A view to see all the events that week	That allows the user to easily see what the is happening in their life that week
8	The week's events should be split into days and hours in the view on the main page	This is to easily see when events start
9	When viewing events on the main page, the user should see the colour of the event and the name.	This helps the user to easily see different types of events with colour, and can see basic information about that event from the name.
10	If the event name is too long to display on the main page, then it should cut the name so that it fits	This helps to keep the name presentable because you aren't trying to fit really long text in a small area.
11	When an event is clicked on it should display information about it	More details should be hidden initially to stop clutter. When clicking on it should display the info about that event
14	An all-day event is shown at the top of that day	So the user can see at a glance what is going on all day.
17	Create event page	A form is needed to input data about new events.
19	Delete events	The user needs to be able to delete events
20	Edit Events page	A form is needed to change data about existing events.
27	User can change colour of event	This helps the user to easily see different events with colours
28	There are at least 5 colours that a user can choose from for an event	This helps the user to split their calendar into different colour to easily see what is going on.
29	The colour of an event defaults to the colour of the calendar	This means that a calendar can have a colour which all events default to
30	An event can be set to all day	Some events do not have times but you still need reminding

Requirements 7, 8 and 9 are all part of the event viewer. This follows the GUI diagrams that I initially created at the start of the project. The events scale really well when there are multiple events, and the

events have colour and text. One thing that I did change from my initial design was having the date of the day displayed. This really helps as with just the week day displayed, which is what is on the original design, it is not very clear which week you are on.

Requirement 10 was also satisfied in the Event Viewer. This cuts the text successfully. I had forgotten to plan this out, so it was added at the end. That being said, the solution works and it cuts text to the correct length.

Requirement 11 worked, when you clicked on an event you got all of the information about an event. When I first talked about this requirement in the research and designing phases I had thought about having more of a popup rather than this taking over the entire screen. I think the solution I came up with – taking over the screen – works. It could be something that I look at in the future but the solution that I have currently works and satisfies the requirement. To complete this I created a class that I had not planned on in the design phase called *EventDetails*. I started off with doing it within the event viewer, but it got to the point where I thought it was confusing to have it in the same file, because I was struggling to find the functions that I was trying to edit. I did not need to separate it out into the separate class, but I think it helped my development, as it made it easier to code.

Requirement 14 was completed, All Day Events are at the top of the page. This wasn't in the GUI diagrams that I did at the beginning of the project, but I am happy with the position of them and they work well.

Requirements 17, 19 and 20 are all completed. You can create, edit and delete events. When creating and editing the event I had several discussions with my stakeholder Ryan about how to allow the user to select dates and times. In the end I added a new class to do this called *DateTimePopup.* This solution worked better than the one that I had originally designed or implemented. Being able to reuse the mini date picker helped because it makes the program consistent. Deleting the event worked and had a good confirmation message with it. In my initial classes that I created in the design process I had a separate class called *editEvent.* In the end I felt that having it in a separate class would be inefficient so I decide to include this function in the *createEvent* form . This allowed me to reuse a lot of code, which decreases the space complexity of the program. If I had decided to do this while planning the project, I would have called the create event class just *Event*, because the class name does not really describe what the function actually does.

Requirements 27 and 28 are very similar; they were both completed. The colours that the user can chose for their event to be are: Red, Orange, Pink, Green, Yellow, Light Blue, Lavender, Wheat, Indian Red, Pale Green, Deep Pink, and Slate Blue. This is obviously greater than 5. I had a conversation with my stakeholder Ryan about what colours to use, and in the end, we chose colours that work with black text in the foreground. These work well.

Requirement 29 is related to these requirements, the default colour of any event that the user creates is the colour of the selected calendar. I did this because one of the success criteria that I did not complete (numbers 36,37, & 38) was that a user could have multiple calendars. By having default colours for each calendar, I could have event default to the colour of their parent calendar. This would have worked had I had more time to implement those functions.

Requirement 30 is completed. The user can set a check box when creating or editing an event. When the event is all day, it is shown at the top of the page.

Database

No.	Requirements	Justification
5	If an account already exists then an error message will be shown	This is so the user knows that there is a problem
18	An event must have a name and start date at a minimum, otherwise an error message is shown.	This is to prevent events ending before they start, and so that every event has a title.
21	When an event is edited it should go through the same validation as when creating an event	This is to prevent events ending before they start, and so that every event has a title.
22	A database to store user data, events etc	This allows the program to be expandable. The database could be linked to other programs like a phone app in the future.
23	The personal data about a user is email, name, username and password	I don't need more data, this keeps me compliant with the Data Protection Act.
25	When a user changes the information about them if goes through the same validation as when they create a new account	This is so that two different users cannot have the same name and email
26	Store title, description, and location of an event. As well as start and end times	This is the data that is needed in the form.

Requirement 5 is completed; this is shown in **Video 6 - Create Account.mp4** in the supporting documents. The Data Protection Act says that data stored about a user should be relevant and up to date. If a user can have multiple accounts then these conditions may not be met, so this is important.

Requirement 18 and 21 has also been met. The testing for this is shown in **Video 11 - Create Event Validation.mp4** and **Video 12 - Edit Event Validation.mp4** in the supporting documents. Both of these work, events must follow the general laws of time (they must not end before they start) before they are put into the database. This is also checked if the user is editing an event as well. Both of these came out as I expected.

Requirement 22 is very definitely met. This was the first thing I did. I did have to change some of my original database designs as when I had done the initial database design, I had forgotten to add several columns, so when I created the SQL for this database, I added some columns. I did have a few problems with syntax when creating this SQL, because SQLite – the database engine I am using – uses a slightly different SQL variation to what I was used to. I got these problems fixed and ended up with a relational database that works. The relational aspect was not in the requirements, but this was important to include, as it has saved me from having lots of data duplication in the database.

Requirement 25 has been completed; it goes through the same validation. This validates the email and makes sure that the username does not already exist in the database.

Requirement 23 and 26 are very similar, both detailing the data that the database should store about different things. Both of these were completed. Requirement 23 was kept the same, however in requirement 26 I had to add the colour of the event, as I needed to store this information but had

forgotten to put it in the success criteria. This hasn't been an issue and allows me to have events that are different coloured.

Window

No.	Requirements	Justification
12	When the user clicks on an event the colour of that event should become the background	This helps to keep the project colourful and exciting. It also helps the user easily identify the event they are viewing
31	The title of the window should say "Calendar !"	So the user knows what the window is and the window purpose is clear to the user

Requirement 12 was completed, the background did change colour to that of an event. It is not the whole window, but just the canvas that the event viewer is placed into. I think it would have looked a little odd to have everything change, so it is just that canvas that changes.

Requirement 31 was also completed, the calendar does have that text.

Maintenance

Current and Future Maintenance

Currently there are no maintenance features in the calendar. If the project was going to be distributed professionally features that allowed updates would be required to allow the developers to add more features and fix bugs.

One way to do this is to add a feature that checks a web server at periodic intervals to see if an update is available for the product. There could also be a button in a settings page to force the program to do this as a manual over ride. I would also need to add a feature that would allow a user to report a bug that they have found which a future developer could then act upon, releasing a new patch. As part of this it may be useful to add a log file that is sent with the bug report so that any problems could be investigated by a programmer, allowing them to recreate these problems.

More features would also need to be added, such as being able to have multiple calendars and an online / app version of the calendar. This would be to keep my target market happy, as the sort of business people that are going to be using this may be on the move a lot, so would want to have this. This may mean the database has to be hosted on a central server, and my program interfaces with that rather than accessing a database on the user's computer. I believe that my program is a good start to what could be a bigger project.

Future Ideas

Back at the start I created a secondary success criteria to complete if I had time. Unfortunately, I did not so I will now go through each one explain how they could be implemented.

No.	Requirements	Justification
32	Loading Page	So the user knows the page is loading

To complete this requirement, I would create a new page that has a loading graphic on it as an image. I would add this to the window as a frame, meaning that I could switch to it whenever something is loading.

I could add this while the user is logging in or creating an account, so the user know that when they have clicked submit on the form, it has actually gone.

No.	Requirements	Justification
33	Scalable Forms	All of the forms adjust their size to the size of the window that they are in
34	Scalable Main Page	The main page will be scalable
35	Scalable Event Viewer	The event viewer will change width and height. The height can be changed by adding a scroll bar, and the width can be changed by reducing the number of days shown

This could be done by registering an event within Tkinter so that whenever the window changes in size it calls a function. This could then loop through all of the frames and change the size of the widgets on that page. Of course, there is a point above which there is no point making it bigger. There is also a minimum size that the form could be before I would need to start adding things like a scroll bar in as it would have become too small for the form to still look usable.

No.	Requirements	Justification
36	Create New Calendars	Creates a form for the user to create a new calendar.
37	Edit Calendars	Allows the user to change the name and default colour of a calendar

This could be done by creating a new form that can create and edit calendars. Each calendar would require a name and a dropdown for its colour. I would then also need to add a dropdown of all of the users calendars, when creating or editing an event. To access the new forms I would need to modify my menu so that you could do this.

No.	Requirements	Justification
38	Delete Calendars	Allows the user to remove a calendar and all events associated with it

I would need more of a settings page, with editing account details, and listing all calendars so they can be modified or deleted. There could be a button to bring up a popup confirm this. Because the calendars and events are relational in the database, I would need to delete all of the events associated with the calendar before I could delete the calendar itself.

No.	Requirements	Justification
39	Pictures to do with events	Add a small picture when viewing event details from looking at event names

To complete this I would first need to either find or create lots of little cartoon pictures of different events, for example small pictures of football, cricket, rugby etc. When I the draw the events I could then check the event name for a list of key words, and if there is a key word, put the associated picture below the text.

No.	Requirements	Justification
40	ICAL Output	Output all of the user's events in an ICAL file for them to upload elsewhere
41	ICAL Import	Allow the user to import an ICAL file from another calendar application.

To do both of these I would need some sort of settings page where both of these features could be accessed from. I would then need to create an ICAL parser so that I could import all of the events and the put them into the database. I would then need to also create some sort of algorithm that can take the database details and convert them into an ICAL format.

Project Appendices

File Structure

This is the final file structure of the program:

```
.
├── DB Creator
│   └── DB Creator.py
├── CreateAccount.py
├── CreateEvent.py
├── Database.py
├── DatePicker.py
├── DateTimePopup.py
├── EventDetails.py
├── EventViewer.py
├── Login.py
├── mainPage.py
├── Window.py
├── Calendar.db
└── SQL.sql
```

Code

CreateEvent.py

```
'''
Author: Robert Watts
Project: Computer Science Controlled Assessment 2019
File Name: CreateEvent.py
Description: Creates a form to create and edit
             event details.
'''

from tkinter import *
from DateTimePopup import *
import datetime
from tkinter import messagebox
import time

class createEvent(Frame):
    def  __init__ (self, parent, controller, database):
        #Initalise Frame
        Frame.__init__(self,parent)
        self.DB = database
        self.controller = controller

        #Create and Pack Labels
        labelsText = ["Name","Start Time & Date","End Time & Date","All Day?","Description","Location","Colour"]
        labels = []
        row = 1
        for text in labelsText:
            labels.append(Label(self, text=text))
            labels[-1].grid(column=0, row=row)
            row += 1

        #Create Text Boxes
        self.name = Entry(self)
        self.name.grid(column=1, row=1, padx=10,sticky="ew", columnspan=5)
        self.location = Entry(self)
        self.location.grid(column=1, row=6, padx=10,sticky="ew", columnspan=5)

        #Create Check Box
        self.allDayValue = IntVar()
        self.allDay = Checkbutton(self, text="All Day",variable=self.allDayValue)
        self.allDay.grid(column=1, row=4, padx=10,sticky="w", columnspan=5)

        #Create Text Area
        self.description = Text(self,height=3)
        self.description.grid(column=1, row=5, padx=10, sticky="ew", columnspan=5)

        #Create Dropdown
        colours = ["Red","Orange","Pink","Green", "Yellow", "Light Blue", "Lavender",
                   "Wheat", "Indian Red", "Pale Green", "Deep Pink", "Slate Blue"]
        self.colourValue = StringVar(self)
        self.colourValue.set(self.controller.USER_DETAILS[3][0][2]) # initial value
        self.colour = OptionMenu(self, self.colourValue, *colours)
        self.colour.grid(column=1, row=7, padx=10, sticky="ew", columnspan=5)

        #Create Start Time Button / Popup
        self.startTime = None
        startTimeButton = Button(self, text="Choose Date", width=12, command=lambda: DateTimePopup(self.changeStartTime))
        startTimeButton.grid(column=1, row=2, padx=10, pady=10, sticky="w")

        #Start Time to de displayed once chosen
        self.startTimeText = Label(self, text="")
        self.startTimeText.grid(column=2, row=2, sticky="w")

        #Create End Time Button / Popup
        self.endTime = None
        endTimeButton = Button(self, text="Choose Date", width=12, command=lambda: DateTimePopup(self.changeEndTime))
        endTimeButton.grid(column=1, row=3, padx=10, pady=10, sticky="w")

        #End Time to de displayed once chosen
        self.endTimeText = Label(self, text="")
        self.endTimeText.grid(column=2, row=3, sticky="w")
```

```
        self.createEventButton = Button(self, text="Create Event", width=12, command=self.CreateEventButtonPress)
        self.createEventButton.grid(column=1, row=8, padx=10, pady=10, sticky="w")

        clearFormButton = Button(self, text="Clear Form", width=12, command=self.clearForm)
        clearFormButton.grid(column=2, row=8, padx=10, pady=10, sticky="w")

        self.IsEditEvent = False
        self.EditID = None

    def changeStartTime (self, newTime):
        self.startTime = newTime
        dateObject = datetime.datetime(newTime[0], newTime[1], newTime[2], newTime[3], newTime[4])
        text = dateObject.strftime("%A %d %B %Y @ %X")
        self.startTimeText.config(text=text)

    def changeEndTime (self, newTime):
        self.endTime = newTime
        dateObject = datetime.datetime(newTime[0], newTime[1], newTime[2], newTime[3], newTime[4])
        text = dateObject.strftime("%A %d %B %Y @ %X")
        self.endTimeText.config(text=text)

    def editEvent(self,eventID):
        #Display Frame + Reset Form + Get event Details
        self.controller.show_frame("createEvent")
        self.clearForm()
        eventDetails = self.DB.getEventDetails(eventID)

        #Populate Form
        self.name.insert(END, eventDetails[1])
        startTime = datetime.datetime.fromtimestamp(eventDetails[2])
        self.changeStartTime((int(startTime.strftime("%Y")), int(startTime.strftime("%m")), int(startTime.strftime("%d")), int(startTime
        if eventDetails[3] != None:
            endTime = datetime.datetime.fromtimestamp(eventDetails[3])
            self.changeEndTime((int(endTime.strftime("%Y")), int(endTime.strftime("%m")), int(endTime.strftime("%d")), int(endTime.strft
        if eventDetails[5] == 1:
            self.allDay.select()
        self.description.insert(END,eventDetails[7])
        self.location.insert(END, eventDetails[4])
        self.colourValue.set(eventDetails[8])

        self.createEventButton["text"] ="Edit Event"

        self.IsEditEvent = True
        self.EditID = eventID


    def CreateEventButtonPress(self):
        #Get Form Info
        name = self.name.get()
        start = self.startTime
        end = self.endTime
        allDay = self.allDayValue.get()
        description = self.description.get("1.0",END)
        location = self.location.get()
        colour = self.colourValue.get()

        #Validate Form
        if name == "" or start == None:
            messagebox.showinfo("", "At a minimum an event needs a name and a start date.")
            return
        StartUnixTime = time.mktime(datetime.datetime(start[0], start[1], start[2], start[3], start[4]).timetuple())
        try:
            EndUnixTime = time.mktime(datetime.datetime(end[0], end[1], end[2], end[3], end[4]).timetuple())
        except:
            EndUnixTime = None
        if allDay == 0:
            if end == None:
                messagebox.showinfo("", "No End Time Selected")
                return
```

```
141
142             if EndUnixTime <= StartUnixTime:
143                 messagebox.showinfo("", "This event finishes before it starts! Please change either the start or end time.")
144                 return
145
146         #Get Default Calendar ID
147         calendarID = self.controller.USER_DETAILS[3][0][0]
148
149         if self.IsEditEvent == False:
150             #Create Event
151             self.DB.createEvent(name,StartUnixTime,EndUnixTime,allDay, description, location, colour, calendarID)
152         else:
153             #Edit Event
154             self.DB.editEvent(self.EditID, name,StartUnixTime,EndUnixTime,allDay, description, location, colour)
155
156         #Refresh Calendar View
157         self.controller.frames["mainPage"].eventViewerHolder.displayEvent(self.controller.frames["mainPage"].eventViewerHolder.currentDa
158                                                                           self.controller.frames["mainPage"].eventViewerHolder.currentDa
159                                                                           self.controller.frames["mainPage"].eventViewerHolder.currentDa
160
161         #Clear Form and change frame
162         self.clearForm()
163         self.controller.show_frame("mainPage")
164
165     def clearForm(self):
166         #Reset the form
167         self.name.delete(0,END)
168         self.name.focus_set()
169         self.startTime = None
170         self.endTime = None
171         self.startTimeText.config(text="")
172         self.endTimeText.config(text="")
173         self.allDay.deselect()
174         self.description.delete("1.0",END)
175         self.location.delete(0,END)
176         self.colourValue.set(self.controller.USER_DETAILS[3][0][2])
177         self.IsEditEvent = False
178         self.EditID = None
179         self.createEventButton["text"] ="Create Event"
180
```

CreateAccount.py

```
'''
Author: Robert Watts
Project: Computer Science Controlled Assessment 2019
File Name: CreateAccount.py
Description: Creates a form to create and edit
             account details.
'''

from tkinter import *
from mainPage import *

class createAccount(Frame):
    def  __init__ (self, parent, controller, database):
        #Initalise program
        Frame.__init__(self,parent)
        self.DB = database
        self.controller = controller

        self.userID = None

        #Create Labels
        nameText = Label(self, text="Full Name")
        emailText = Label(self, text="Email")
        usernameText = Label(self, text="Username")
        passwordText = Label(self, text="Password")

        #Put Labels on Grid
        nameText.grid(column=0, row=0)
        emailText.grid(column=0, row=1)
        usernameText.grid(column=0, row=2)
        passwordText.grid(column=0, row=3)

        #Create entry
        self.name = Entry(self,width=30)
        self.name.grid(column=1, row=0, padx=10)

        self.email = Entry(self,width=30)
        self.email.grid(column=1, row=1, padx=10)

        self.username = Entry(self,width=30)
        self.username.grid(column=1, row=2, padx=10)

        #Create password entry
        self.password = Entry(self,width=30)
        self.password.grid(column=1, row=3)
        self.password.config(show="*")

        #Create error message label
        self.errorLabel = Label(self, text="")
        self.errorLabel.grid(column=0, row=4, sticky=W, columnspan=2)
        self.errorLabel.config(fg='red')

        #Create Login Button + Create account button
        self.createAccountButton = Button(self, text="Create Account", width=12, command=self.createAccount)
        self.createAccountButton.grid(column=1, row=5, columnspan=2, sticky=W, padx=10, pady=10)
        self.LoginButton = Button(self, text="Back to Login", width=12, command=lambda: controller.show_frame("login"))
        self.LoginButton.grid(column=0, row=5, padx=10, pady=10, sticky=E)

    def createAccount(self):
        #Get Data from form
        usernameHolder = self.username.get()
        passwordHolder = self.password.get()
        nameHolder = self.name.get()
        emailHolder = self.email.get()

        #Check if user Exists and run
        if self.userID == None:
            accountCheck = self.DB.createAccount(nameHolder,emailHolder,usernameHolder,passwordHolder)
        else:
            if passwordHolder == "":
```

```
                passwordHolder = None
            accountCheck = self.DB.editUser(nameHolder,emailHolder,usernameHolder,self.userID,password = passwordHolder)

        if accountCheck == True:
            if self.userID == None:
                #log user in
                loginCheck = self.DB.checkLogin(usernameHolder,passwordHolder)
                self.controller.USERID = loginCheck[1]
                #Add Frame and Open
                self.controller.createFrame(mainPage)


            #Clear Form and change frame
            self.resetForm()
            self.controller.show_frame("mainPage")

        else:
            #Display Error Message
            self.errorLabel.config(text=accountCheck)

    def resetForm(self):
        #Clears the form
        self.name.delete(0, 'end')
        self.username.delete(0, 'end')
        self.password.delete(0, 'end')
        self.email.delete(0, 'end')
        self.createAccountButton["text"] ="Create Account"
        self.errorLabel.config(text="")

    def editUser(self):
        #Get User Details + resets form
        self.userID = self.controller.USERID
        details = self.DB.getUserData(self.controller.USERID)
        self.resetForm()

        #Put user details info form
        self.username.insert(END, details[2])
        self.name.insert(END, details[0])
        self.email.insert(END, details[1])
        self.createAccountButton["text"] ="Edit Account"
        self.LoginButton.grid_forget()

        #Add Instructions Text
        instructionsText = Label(self, text="To edit your password enter a new one \notherwise leave this box blank.")
        instructionsText.grid(column=2, row=3)

```

Database.py

```
'''
Author: Robert Watts
Project: Computer Science Controlled Assessment 2019
File Name: Database.py
Description: Interfaces with the database file, holding all the
             SQL that is needed throught the program.
'''

import sqlite3 as lite
import os
import re
import hashlib

class Database:
    def __init__(self):
        #Find location of database
        dir_path = os.path.dirname(os.path.realpath(__file__))
        databaseLocation =  dir_path + "/Calendar.db"

        #Create Database connection
        self.con = lite.connect(databaseLocation)
        self.cur = self.con.cursor()


    def createAccount(self, name, email, username, password):
        email=email.lower()

        #Make sure email is valid
        if self.validateEmail(email) == False:
            return "Email Not Valid"

        #Makes sure all othe feilds are valid
        if name == "" or username == "" or password == "":
            return "Fill In Missing Values"

        #See if a user already exists
        check = self.checkIfUserExist(email, username)
        if  check != False:
            return check


        #Create Values list, and SQL statement
        values = (name, email, username, self.hashPassword(password))
        sql = "INSERT INTO Users ('Name','Email', 'Username', 'Password') VALUES (?,?,?,?);"

        #Execute SQL
        self.cur.execute(sql,values)
        self.con.commit()

        userID = self.checkLogin(username,password)[1]
        self.createCalendar(userID, "Default", "Orange")

        return True

    def checkIfUserExist(self,email,username, id=None):
        #See if a user already exists
        if id == None:
            sql = "SELECT Email,Username FROM users WHERE Email=? OR Username=?"
            self.cur.execute(sql, (email,username))
        else:
            sql = "SELECT Email,Username FROM users WHERE (Email=? OR Username=?) AND (ID <>  ?)"
            self.cur.execute(sql, (email,username,id))

        rows = self.cur.fetchall()
        for user in rows:
            if user[0] == email:
                return "That email aready exists!"
            else:
                return "That username already Exists! Try Another one."
        return False
```

```
71
72      def checkLogin(self, username, password):
73          #See if a username and password is valid
74          sql = "SELECT ID,Password FROM users WHERE Username=?"
75          self.cur.execute(sql, (username,))
76          rows = self.cur.fetchall()
77          for user in rows:
78              if self.hashPassword(password) == user[1]:
79                  return True, user[0]
80          return False, None
81
82
83      def validateEmail(self, email):
84          #Makes sure an email is valid
85          regx = "(^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$)"
86          p = re.compile(regx)
87
88          if p.match(email):
89              return True
90          else:
91              return False
92
93      def hashPassword(self,password):
94          #Hash the string.
95          return hashlib.sha224(password.encode()).hexdigest()
96
97      def createCalendar(self, userID, name, colour):
98          #Create Values list, and SQL statement
99          values = (name,userID, colour)
100         sql = "INSERT INTO Calendars ('Name','UserID', 'Colour') VALUES (?,?,?);"
101
102         #Execute SQL
103         self.cur.execute(sql,values)
104         self.con.commit()
105
106
107     def getUserData(self,userID):
108         #Get the calendars from a particular user
109         sql = "SELECT ID ,Name,Colour FROM Calendars WHERE UserID=?"
110         self.cur.execute(sql, (userID,))
111         CalendarRows = self.cur.fetchall()
112
113         #Get the details from a particular user
114         sql = "SELECT Name,Email,Username FROM users WHERE ID=?"
115         self.cur.execute(sql, (userID,))
116         rows = self.cur.fetchall()
117         for user in rows:
118                 return list(user) + list([CalendarRows])
119
120
121     def createEvent(self, name, startTime, endTime, allDay, description, location, colour, calendarID):
122         #Create Values list, and SQL statement
123         values = (name, startTime, endTime,location,allDay, description, colour, calendarID)
124         sql = "INSERT INTO Events ('Name','StartTimestamp', 'EndTimestamp', 'Location', 'AllDay', 'Description', 'Colour', 'CalendarID'
125
126         #Execute SQL
127         self.cur.execute(sql,values)
128         self.con.commit()
129
130
131     def getEvents(self, startTimestamp, endTimestamp, calendarID):
132         #Get the calendars from a particular user
133         sql = "SELECT * FROM Events WHERE ((StartTimestamp BETWEEN ? AND ?) AND CalendarID=?) OR ((EndTimestamp BETWEEN ? AND ?) AND Ca
134         self.cur.execute(sql, (startTimestamp,endTimestamp,calendarID, startTimestamp,endTimestamp,calendarID))
135         Rows = self.cur.fetchall()
136         return Rows
137
138     def getEventDetails(self,eventID):
139         #Get the details about a particular event
140         sql = "SELECT * FROM Events WHERE ID=?"
```

```
        self.cur.execute(sql,(eventID,))
        for row in self.cur.fetchall():
            return row

    def deleteEvent(self,eventID):
        #Delete an event by ID
        sql = "DELETE FROM Events WHERE ID=?"
        #Execute SQL
        self.cur.execute(sql,(eventID,))
        self.con.commit()

    def editEvent(self, eventID, name, startTime, endTime, allDay, description, location, colour):
        #Edits an event
        sql = "UPDATE Events SET Name = ?, StartTimestamp = ?, EndTimestamp = ?, Location = ?, AllDay=?, Description=?, Colour=? WHERE I
        values = (name,startTime,endTime,location,allDay,description,colour, eventID)
        #Execute SQL
        self.cur.execute(sql,values)
        self.con.commit()

    def editUser(self, name, email, username, id, password=None):
        #EDITS USER
        email=email.lower()

        #Make sure email is valid
        if self.validateEmail(email) == False:
            return "Email Not Valid"

        #Makes sure all othe feilds are valid
        if name == "" or username == "":
            return "Fill In Missing Values"

        #See if a user already exists
        check = self.checkIfUserExist(email, username, id=id)
        if  check != False:
            return check

        #Create SQL Statement and Values to pass
        if password == None:
            sql = "UPDATE Users SET Name = ?,Email = ?,Username = ?  WHERE ID = ?"
            values = (name,email, username,id)
        else:
            sql = "UPDATE Users SET Name = ?,Email = ?,Username = ?,Password = ?  WHERE ID = ?"
            values = (name,email, username,self.hashPassword(password),id)


        #Execute SQL
        self.cur.execute(sql,values)
        self.con.commit()

        return True

```

DatePicker.py

```
'''
Author: Robert Watts
Project: Computer Science Controlled Assessment 2019
File Name: DatePicker.py
Description: Creates a date picker that allows the user to select
             a day from any month and year. Outputs their selection
             to the command variable.
'''

from tkinter import *
from calendar import monthrange, month_name
import datetime


class datePicker(Frame):
    def __init__(self, master=None, command=None, ** kw):
        self.command = command
        Frame.__init__(self,master,**kw)

        self.populateForm()
        self.navButtonClick("t")

    def populateForm(self):
        #Create Title
        self.title = Label(self, font='comic-sans-ms 11 bold')
        self.title.grid(in_=self, column=0, row=0, columnspan=7)

        #Create Nav DayButtons
        self.navButtons = []

        self.navButtons.append(Button(self, text="<--", command=lambda: self.navButtonClick("l")))
        self.navButtons[-1].grid(in_=self, column=0, row=1, columnspan=2)

        self.navButtons.append(Button(self, text="-->", command=lambda: self.navButtonClick("r")))
        self.navButtons[-1].grid(in_=self, column=5, row=1, columnspan=2)

        self.navButtons.append(Button(self, text="Today", command=lambda: self.navButtonClick("t")))
        self.navButtons[-1].grid(in_=self, column=2, row=1, columnspan=3)

        #Create Labels
        days = ["Mon ", "Tues", "Weds", "Thur", "Fri ", "Sat ", "Sun "]
        self.DayLabels = []
        ColumnCount = 0
        for day in days:
            self.DayLabels.append(Label(self,text=day))
            self.DayLabels[-1].grid(in_=self, column=ColumnCount, row=2)
            ColumnCount += 1

    def navButtonClick(self, command):
        if command == "l":
            self.Year, self.Month = self.PreviousYearMonth(self.Year, self.Month)
        elif command == "r":
            self.Year, self.Month = self.NextYearMonth(self.Year, self.Month)
        elif command == "t":
            now = datetime.datetime.now()
            self.Year = now.year
            self.Month = now.month

        self.DisplayMonth(self.Year,self.Month)

    def DisplayMonth(self, year, month):
        #Delete Month
        try:
            for x in self.DayButtons:
                x.destroy()
        except:
            pass
        #Change Title Text
        text = month_name[month]  + " " + str(year)
        self.title['text'] = text
```

```
71
72          #Create new buttons
73          self.DayButtons = []
74
75          #Get details about the month in question:
76          monthDetails = monthrange(year, month)
77          gridPosition = [monthDetails[0], 3]
78
79          for x in range(0, monthDetails[1]):
80              #Work out position in grid
81              if gridPosition[0] % 7 == 0:
82                  gridPosition[0] = 0
83                  gridPosition[1] += 1
84              gridPosition[0] += 1
85              text = x+1
86              if text <= 9:
87                  text = "0" + str(text)
88
89              #Create and pack da button
90              self.DayButtons.append(Button(self, text=text, command=lambda day=int(text):self.dayButtonPressed(self.Year,self.Month,day)
91              self.DayButtons[-1].grid(in_=self, column=gridPosition[0]-1, row=gridPosition[1])
92
93      def dayButtonPressed(self,year,month,day):
94          try:
95              output = (year,month,day)
96              self.command(output)
97          except Exception as e:
98              print("Date Picker Error: ",e)
99
100
101
102     def NextYearMonth(self, year, month):
103         if month == 12:
104             return year + 1, 1
105         else:
106             return year, month+1
107
108     def PreviousYearMonth(self, year, month):
109         if month == 1:
110             return year - 1, 12
111         else:
112             return year, month-1
113
```

DateTimePopup.py

```
'''
Author: Robert Watts
Project: Computer Science Controlled Assessment 2019
File Name: DateTimePopup.py
Description: Creates a popup that allows the user to select
             a day from any month and year, as well as a time.
             Outputs their selection to the outputFunction
             variable.
'''


from tkinter import *
from DatePicker import *
import datetime
from tkinter import messagebox


class DateTimePopup:

    def __init__(self, outputFunction):
        #Create Popup Window
        self.window = Toplevel()
        self.window.wm_title("Pick Date & Time")
        self.window.grab_set()

        #Define the variables
        self.outputVariable = outputFunction
        self.date = None

        #Create a list of all hours and mins avaliable
        hours = list(range(0, 24))
        mins = list(range(0, 60))

        #Create Time Label
        selectTimeLabel = Label(self.window, text="Select Time:", font='Helvetica 16 bold', fg="orange")
        selectTimeLabel.grid(column=0, row=0, columnspan=3,sticky="nsew")

        #Create Hours dropdown
        self.startHoursValue = StringVar(self.window)
        self.startHoursValue.set(hours[0]) # initial value
        self.startHours = OptionMenu(self.window, self.startHoursValue, *hours)
        self.startHours.grid(column=0, row=1,sticky="e")

        #Create a colon label
        colon = Label(self.window, text=":", font='Helvetica 18 bold')
        colon.grid(column=1, row=1)

        #Create a mins dropdown
        self.startMinsValue = StringVar(self.window)
        self.startMinsValue.set(mins[0]) # initial value
        self.startMins = OptionMenu(self.window, self.startMinsValue, *mins)
        self.startMins.grid(column=2, row=1,sticky="w")

        #Create a select date Label
        selectDateLabel = Label(self.window, text="Select Date:", font='Helvetica 16 bold', fg="orange")
        selectDateLabel.grid(column=0, row=2, columnspan=3,sticky="nsew")

        #Create a date picker
        self.selectedDate = Label(self.window, text="", font='Helvetica 8')
        self.selectedDate.grid(column=0, row=3, columnspan=3,sticky="nsew")
        self.startDatePicker = datePicker(self.window, command=self.datePickerReturn, borderwidth=2, relief="groove")
        self.startDatePicker.grid(column=0, row=4, columnspan=3)

        #Create save button.
        SaveButton = Button(self.window, text="Save", bg="lightblue", command=self.saveButtonPress)
        SaveButton.grid(column=0, row=5, padx=10, pady=10, sticky="nsew", columnspan=3)

    def datePickerReturn(self, dateClicked):
        #Save date from date picker output
        self.date = dateClicked
```

```
        dateObject = datetime.datetime(self.date[0], self.date[1], self.date[2])
        text = dateObject.strftime("%A %d %B %Y")
        self.selectedDate.config(text=text)

    def saveButtonPress(self):.
        #Handels save button press
        #Get values from form + Validate
        hour = int(self.startHoursValue.get())
        min = int(self.startMinsValue.get())
        if self.date == None:
            messagebox.showerror("Error", "Please Select a Date!")
        else:
            #Output data to output variable.
            output = self.date + (hour, min)
            try:
                self.outputVariable(output)
            except:
                pass
            self.window.destroy()

```

EventDetails.py

```
'''
Author: Robert Watts
Project: Computer Science Controlled Assessment 2019
File Name: EventDetails.py
Description: Display the details for an event within
             a canvas. Provides the users options to
             edit and delete an event.
'''

from tkinter import *
from datetime import datetime
from tkinter import messagebox

class EventDetails:
    def __init__(self,canvas, database, defualtBackgroundColour, controller):
        #Inialise Class
        self.CANVAS = canvas
        self.DB = database
        self.defaultBackground = defualtBackgroundColour
        self.controller = controller

    def eventClick(self, event, *args):
        #Find Item + Get ID
        CanvasItem = event.widget.find_closest(event.x, event.y)
        eventID = event.widget.gettags(CanvasItem[0])[1]

        #Hide everything on items on the canvas
        thingsOnCanvas = self.CANVAS.find_withtag("event") + self.CANVAS.find_withtag("timeLabels") + self.CANVAS.find_withtag("daysLabel
        for id in thingsOnCanvas:
            self.CANVAS.itemconfigure(id, state='hidden')

        self.drawEventDetails(eventID)
        self.drawButtons(eventID)

    def drawEventDetails (self, eventID):
        #Get Events
        event = self.DB.getEventDetails(eventID)

        #Change Background Colour
        self.CANVAS.configure(background=event[8])
        x = datetime.fromtimestamp(event[2])

        #Work out which Items the user filled in and added them to the list
        items = [
            ["Name:", event[1]],
            ["Start Date:", x.strftime("%A %d %B %Y @ %X")]
        ]
        if event[3] != None:
            items.append(["End Date:", datetime.fromtimestamp(event[3]).strftime("%A %d %B %Y @ %X")])
        if event[4] != "":
            items.append(["Location:", event[4]])
        if event[5] == 1:
            items.append(["All Day?", "Yes"])
        else:
            items.append(["All Day?", "No"])
        items.append(["Description:", event[7].strip("\n")])

        #Define the inital position for the grid
        y = 80
        for item in items:
            #Display the infomation about the event
            self.CANVAS.create_text(40,y, text=item[0], anchor="w", tag="eventDetails", font = ('Calibri', 15, 'bold'))
            self.CANVAS.create_text(160,y, text=item[1], anchor="w", tag="eventDetails", font = ('Calibri', 15, ''))
            y += 30

    def drawButtons (self, eventID):
        #Draw Buttons
        BackButton = Button(self.CANVAS, text = "Back", anchor = W, command=self.backButtonPress)
        BackButton_window = self.CANVAS.create_window(40, 20, anchor=NW, window=BackButton, tag="eventDetails")

```

```
        EditEventButton = Button(self.CANVAS, text = "Edit Event", anchor = W, command=lambda id=eventID: self.controller.frames["create
        EditEventButton_window = self.CANVAS.create_window(80, 20, anchor=NW, window=EditEventButton, tag="eventDetails")

        DeleteEventButton = Button(self.CANVAS, text = "Delete Event", anchor = W, command=lambda id=eventID: self.deleteButtonPress(id)
        DeleteEventButton_window = self.CANVAS.create_window(150, 20, anchor=NW, window=DeleteEventButton, tag="eventDetails")

    def backButtonPress(self):
        #Delete all of the event detial items
        eventDetailsOnCanvas = self.CANVAS.find_withtag("eventDetails")
        for id in eventDetailsOnCanvas:
            self.CANVAS.delete(id)

        #Display all of the hidden items
        hiddenOnCanvas = self.CANVAS.find_withtag("event") + self.CANVAS.find_withtag("timeLabels") + self.CANVAS.find_withtag("daysLabe
        for id in hiddenOnCanvas:
            self.CANVAS.itemconfigure(id, state='normal')

        self.CANVAS.configure(background=self.defaultBackground)

    def deleteButtonPress(self, id):
        #Delete Button press
        comfirm = messagebox.askyesno("Delete Event","Are you sure you want to delete this event?")
        if comfirm == True:
            self.DB.deleteEvent(id)
            self.CANVAS.displayEvent(self.CANVAS.currentDate[0],self.CANVAS.currentDate[1],self.CANVAS.currentDate[2])

```

EventViewer.py

```
'''
Author: Robert Watts
Project: Computer Science Controlled Assessment 2019
File Name: EventViewer.py
Description: Displays the events on a canvas in a grid.
             Handles all user clicking on an event and
             user changing date.
'''
from tkinter import *
import time
import datetime
from EventDetails import EventDetails

class EventViewer (Canvas):

    def __init__(self, master, controller, database, userID, width =98, height=23, ** kw):
        #Initalise Class
        self.DB = database
        self.USERID = userID
        self.CALENDARS = self.DB.getUserData(self.USERID)[3]
        self.controller = controller

        #Define height and width of each cell
        self.WIDTH = width
        self.HEIGHT = height


        Canvas.__init__(self,master, height=(self.HEIGHT * 31), width=(self.WIDTH * 8), **kw)

        self.BACKGROUND = self.master.cget('bg')
        self.EventDetails = EventDetails(self, self.DB, self.BACKGROUND, self.controller)


    def createGrid(self):
        verticalLength = (self.HEIGHT * 24 )
        horizontalLength = (self.WIDTH * 7)

        #Create Vertical Grid Lines
        y = self.GRID_START_CORDINATE_Y
        x = self.GRID_START_CORDINATE_X
        for line in range(0, 8):
            self.create_line(x,y,x,y+verticalLength, tag="grid")
            x += self.WIDTH

        #Add Horizontal Grid Lines
        y = self.GRID_START_CORDINATE_Y
        x = self.GRID_START_CORDINATE_X
        for line in range(0, 25):
            self.create_line(x, y, x+horizontalLength, y, tag="grid")
            y +=  self.HEIGHT

    def createLabels(self, initalX, initalY):
        #Add Day Labels
        x =  initalX
        y = (initalY)/1.3 #OFfset From Top

        DateObject = datetime.datetime(self.currentDate[0], self.currentDate[1], self.currentDate[2], 0,0)
        for q in range(0,7):
            text = DateObject.strftime("%A\n%d %b")
            self.create_text(x+(self.WIDTH/2),y, text=text, tag="daysLabel")
            x += self.WIDTH
            DateObject += datetime.timedelta(days=1)

        #Add Time Labels
        x = (self.GRID_START_CORDINATE_X)/1.5    # Offset from Side
        y = self.GRID_START_CORDINATE_Y

        for time in range(0,24):
            text=str(time).zfill(2) + ":00"
            self.create_text(x,y, text=text, tag="timeLabels")
```

```
            y +=self.HEIGHT

    def displayEvent(self, year, month, day):
        self.currentDate = (year, month, day)

        self.configure(background=self.BACKGROUND)

        self.GRID_START_CORDINATE_Y = (self.HEIGHT * 25) - (self.HEIGHT * 24 ) + 35
        self.GRID_START_CORDINATE_X = (self.WIDTH * 8) - (self.WIDTH * 7.1 )
        #Work Out Start TImestamp
        startTime = datetime.datetime(year, month, day, 0,0)
        startTimestamp = time.mktime(startTime.timetuple())


        #Work End Start TImestamp 6 Days Later
        endTime = startTime + datetime.timedelta(days=6,hours=23,minutes=59, seconds=59)
        endTimestamp = time.mktime(endTime.timetuple())



        #Get Events from all user calendars
        events = []
        for calendar in self.CALENDARS:
            events += self.DB.getEvents(startTimestamp, endTimestamp, calendar[0])


        eventsToGrid = []
        allDayEvents = []
        #For every Event
        for event in events:

            #Work Out Start Day
            startDay = int((event[2] - startTimestamp) / 86400 ) + 1
            #If Event is only on one day
            if event[3] == None:
                endDay = startDay
            else:
                endDay = int((event[3] - startTimestamp) / 86400) + 1

            #Length of event in days
            eventLength = (endDay - startDay) +1

            #If Event is Allday
            if event[5] == True:


                #Split event into days
                for x in range(startDay,startDay + eventLength):
                    if 1 <= x <= 7 :
                        allDayEvents.append({
                            "ID":event[0],
                            "name":event[1],
                            "day": x,
                            "location":event[4],
                            "calendarID":event[6],
                            "description":event[7],
                            "colour":event[8]
                        })

            else:
                #Get start and end Times
                if event[2] < startTimestamp:
                    startTime = int(datetime.datetime.fromtimestamp(startTimestamp).strftime("%H%M"))
                else:
                    startTime = int(datetime.datetime.fromtimestamp(event[2]).strftime("%H%M"))

                endTime = int(datetime.datetime.fromtimestamp(event[3]).strftime("%H%M"))

                #Event split into the individual day (if multi day)
                splitEvent = []
```

```
                if startDay == endDay:
                    #Add a single day event
                    splitEvent.append([startDay,startTime, endTime])
                else:
                    splitEvent.append([startDay,startTime, 2400])
                    if endDay <= 7:
                        splitEvent.append([endDay,0000, endTime])

                    for y in range(startDay+1,(startDay + eventLength)-1):
                        if y <= 7:
                            splitEvent.append([y,0000, 2400])

                #For every part of an event add it to the eventsToGrid list
                for part in splitEvent:
                    eventsToGrid.append({
                        "ID":event[0],
                        "name":event[1],
                        "day": part[0],
                        "startTime": part[1],
                        "endTime": part[2],
                        "location":event[4],
                        "calendarID":event[6],
                        "description":event[7],
                        "colour":event[8],
                        "startTimestamp": event[2],
                        "endTimestamp": event[3]
                    })



        self.clearCanvas()
        self.drawAllDayEvents(allDayEvents)
        #self.createGrid()

        self.drawEvents(eventsToGrid)
        self.createNavButtons()

    def drawEvents(self,events):
        #For every day
        for day in range(1,8):
            eventsToday = []
            eventDepth = [0] * 1440

            #Run through events
            for x in range(len(events) - 1, -1, -1):
                #If the event is on the current day
                if events[x]["day"] == day:
                    holder = events[x]
                    position = 0

                    #for every min in hour
                    for hour in range(holder["startTime"],holder["endTime"]-1):
                        #Work out hours and mins
                        hour = str(hour).zfill(4)
                        hoursNum,minsNum = hour[:2], hour[2:]

                        #Stops you having 12:76 (mins above 59)
                        if int(minsNum) > 59:
                            continue
                        #Calculate the number of mins since midnight and increment that min by 1
                        mins = (int(hoursNum) * 60) + int(minsNum)
                        eventDepth[mins-1] += 1

                        if position < eventDepth[mins-1]:
                            position = eventDepth[mins-1]

                        eventsToday.append([x, position])


```

```
            for x in eventsToday:
                ID = x[0]
                start = events[ID]["startTime"]
                end = events[ID]["endTime"]
                y = eventDepth[start:end]
                widthDivider = max(y)

                if widthDivider == 0:
                    widthDivider=1

                #Calculate Cordinates
                x1 = self.GRID_START_CORDINATE_X + ((events[ID]["day"] - 1)*self.WIDTH) + ((x[1] -1)*(self.WIDTH/widthDivider))
                y1 = self.GRID_START_CORDINATE_Y + ((events[ID]["startTime"]/100)*self.HEIGHT)
                x2 = x1 + (self.WIDTH/widthDivider)
                y2 = self.GRID_START_CORDINATE_Y + ((events[ID]["endTime"]/100)*self.HEIGHT)


                tag = "eventClick " + str(events[ID]["ID"]) + " event"
                self.create_rectangle(x1, y1, x2, y2, fill=events[ID]["colour"], tags=tag)

                #Calculate event name width + shorten it
                eventName = events[ID]["name"]
                eventNameLength = int(len(events[ID]["name"]) * (0.2*widthDivider))

                #Find the first space and cut it there, and add a ...
                for x in range(eventNameLength, -1, -1):
                    if eventName[x] == " ":
                        eventName = eventName[0:x] + "..."
                        break
                #Draw Text
                self.create_text((x1 + ((x2-x1)*0.1), 0.5*(y2-y1) + y1), text=eventName, anchor="w", width=(x2-x1)*0.8, tags=tag)

        self.tag_bind("eventClick","<Button-1>", self.EventDetails.eventClick)

    def drawAllDayEvents(self, events):
        maxEvents = [0,0,0,0,0,0,0]
        for event in events:
            event["eventDepth"] = maxEvents[event["day"] - 1]
            maxEvents[event["day"] - 1] += 1

        maxEventsNum = max(maxEvents)

        for event in events:
            x1 = self.GRID_START_CORDINATE_X + ((event["day"] - 1)*self.WIDTH)
            y1 = self.GRID_START_CORDINATE_Y + (event["eventDepth"] * self.HEIGHT)
            x2 = x1 + self.WIDTH
            y2 = y1 + self.HEIGHT
            tag = "eventClick " + str(event["ID"]) + " event"
            self.create_rectangle(x1, y1, x2, y2, fill=event["colour"], tags="test", dash=(4, 4), tag=tag)
            self.create_text((0.5*(x2-x1) + x1, 0.5*(y2-y1) + y1), text=event["name"], anchor="center", width=(x2-x1-15), tag=tag )

        self.createAllDayGrid(maxEventsNum)
        y = self.GRID_START_CORDINATE_Y
        x = self.GRID_START_CORDINATE_X
        self.GRID_START_CORDINATE_Y += (maxEventsNum * self.HEIGHT) + (self.HEIGHT / 10)
        self.createLabels(x, y)
        self.createGrid()

    def createAllDayGrid (self, NumOfEvents):
        # verticalLength = (self.HEIGHT * 24 )
        horizontalLength = (self.WIDTH * 7)
        GridHeight = (NumOfEvents )*self.HEIGHT
        #Create Vertical Grid Lines
        y = self.GRID_START_CORDINATE_Y
        x = self.GRID_START_CORDINATE_X

        for line in range(0, 8):
            self.create_line(x,y,x,y+ GridHeight, tag="grid")
            x += self.WIDTH

```

```
281         y = self.GRID_START_CORDINATE_Y
282         x = self.GRID_START_CORDINATE_X
283         self.create_line(x, y, x+horizontalLength, y, tag="grid")
284         self.create_line(x, y+GridHeight, x+horizontalLength, y+GridHeight, tag="grid")
285 
286 
287     def createNavButtons(self):
288         leftButton =Button(self, text="<--",  command=self.leftButtonClick)
289         self.create_window(40, 0, anchor=NW, window=leftButton, tag="grid")
290 
291         todayButton = Button(self, text="Today", command=self.todayButtonClick)
292         self.create_window(80, 0, anchor=NW, window=todayButton, tag="grid")
293 
294         rightButton = Button(self, text="-->", command=self.rightButtonClick)
295         self.create_window(150, 0, anchor=NW, window=rightButton, tag="grid")
296 
297     def todayButtonClick(self):
298         now = datetime.datetime.now()
299         self.displayEvent(now.year,now.month,now.day)
300 
301     def leftButtonClick(self):
302         currentDateObject = datetime.datetime(self.currentDate[0], self.currentDate[1], self.currentDate[2], 0,0)
303         newDateObject = currentDateObject - datetime.timedelta(days=7)
304         self.displayEvent(newDateObject.year,newDateObject.month,newDateObject.day)
305 
306     def rightButtonClick(self):
307         currentDateObject = datetime.datetime(self.currentDate[0], self.currentDate[1], self.currentDate[2], 0,0)
308         newDateObject = currentDateObject + datetime.timedelta(days=7)
309         self.displayEvent(newDateObject.year,newDateObject.month,newDateObject.day)
310 
311     def clearCanvas(self):
312         self.delete("all")
313 
```

Login.py

```
'''
Author: Robert Watts
Project: Computer Science Controlled Assessment 2019
File Name: Login.py
Description: Handles the user logging into the program.
'''

from tkinter import *
from mainPage import *

class login(Frame):
    def  __init__ (self, parent, controller, database):
        Frame.__init__(self,parent)
        self.DB = database
        self.controller = controller
        #Create  Labels
        usernameText = Label(self, text="Username")
        passwordText = Label(self, text="Password")
        usernameText.grid(column=0, row=1)
        passwordText.grid(column=0, row=2)

        #Create Username entry
        self.username = Entry(self,width=30)
        self.username.grid(column=1, row=1, padx=10)

        #Create password entry
        self.password = Entry(self,width=30)
        self.password.grid(column=1, row=2)
        self.password.config(show="*")

        #Create error message label
        self.errorLabel = Label(self, text="")
        self.errorLabel.grid(column=0, row=3, sticky=W, columnspan=2)
        self.errorLabel.config(fg='red')

        #Create Login Button + Create account button
        createAccountButton = Button(self, text="Create Account", width=12, command=lambda: controller.show_frame("createAccount"))
        createAccountButton.grid(column=0, row=4, columnspan=2, sticky=W, padx=10, pady=10)
        LoginButton = Button(self, text="Login", width=12, command=self.Login)
        LoginButton.grid(column=1, row=4, padx=10, pady=10, sticky=E)

    def Login(self):
        #Get Data from form
        usernameHolder = self.username.get()
        passwordHolder = self.password.get()

        #Check if user Exists and run
        loginCheck = self.DB.checkLogin(usernameHolder,passwordHolder)
        if loginCheck[0]:
            #Save the users ID
            self.controller.USERID = loginCheck[1]
            #Add Frame and Open
            self.controller.createFrame(mainPage)
            self.controller.show_frame("mainPage")

        else:
            #Display Error Message
            self.errorLabel.config(text="Username or Password Incorect")

```

mainPage.py

```
'''
Author: Robert Watts
Project: Computer Science Controlled Assessment 2019
File Name: Login.py
Description: Handles the main window that the user
             comes to. Creates menus and the event
             viwer.
'''

from tkinter import *
from CreateEvent import *
from EventViewer import *
from DatePicker import datePicker


class mainPage(Frame):
    def  __init__ (self, parent, controller, database):
        Frame.__init__(self,parent)
        self.DB = database
        self.controller = controller
        self.parent = parent

        #Resize Window
        self.controller.geometry("1000x730")

        #Get User Details
        self.controller.USER_DETAILS = self.DB.getUserData(self.controller.USERID)
        self.USER_DETAILS = self.controller.USER_DETAILS

        self.createMenu()

        self.createSideBar()
        self.addFrames()

        self.eventViewerHolder = EventViewer(self,self.controller, self.DB, self.controller.USERID)
        self.eventViewerHolder.grid(row=0,column=1)

        #Display the events on Current Date
        now = datetime.datetime.now()
        self.calendarClick((now.year, now.month, now.day))

    def createMenu(self):
        #Create Menu
        self.menubar = Menu(self)

        #Create Menu Buttons
        self.menubar.add_command(label="Calendar!", command=lambda: [self.controller.show_frame("mainPage"),self.controller.frames["creat
        self.menubar.add_command(label="Create New Event", command=lambda: self.controller.show_frame("createEvent"))
        self.menubar.add_command(label=self.USER_DETAILS[0],command=lambda: [self.controller.show_frame("createAccount"),self.controller

        #Pack Menu
        self.controller.config(menu=self.menubar)

    def createSideBar(self):
        datePickerSide = datePicker(self, command=self.calendarClick)
        datePickerSide.grid(row=0,column=0, sticky="n")

    def addFrames(self):
        self.controller.createFrame(createEvent)

    def calendarClick(self, input):
        self.eventViewerHolder.displayEvent(input[0], input[1], input[2])

```

Window.py

```
'''
Author: Robert Watts
Project: Computer Science Controlled Assessment 2019
File Name: Login.py
Description: Hanndles all of the frames within the program.
             Allows switching between frames and storing
             them for use later.
'''

from tkinter import *
from Login import *
from CreateAccount import createAccount
from Database import *

class Window(Tk):
    def __init__(self, *args, **kwargs):
        #Define Tkinter Window
        Tk.__init__(self, *args, **kwargs)

        #Define Database Connection
        self.DB = Database()

        #Create Container in window
        self.container = Frame(self)
        self.container.pack(side="top", fill="both", expand = True)
        self.container.grid_rowconfigure(0, weight=1)
        self.container.grid_columnconfigure(0, weight=1)

        #Change window text
        self.title("Calendar!")

        #User Id when logged in
        self.USERID = None

        self.frames = {}

        #Create Long and create account frames
        for F in [login, createAccount]:
            self.createFrame(F)

        #Display login page
        self.show_frame("login")


    def createFrame(self,newFrame):
        #create a new frame
        frame = newFrame(self.container, self, self.DB)
        self.frames[newFrame.__name__] = frame
        frame.grid(row=0, column=0, sticky="nsew")


    def show_frame(self, FrametoShow):
        #Display a frame
        frame = self.frames[FrametoShow]
        frame.tkraise()

if __name__ == "__main__":
    app = Window()
    app.mainloop()

```

SQL.sql

```sql
/*
Author: Robert Watts
Project: Computer Science Controlled Assessment 2019
File Name: SQL.sql
Description: The SQL for the database
*/

/*Create Users Table - Holds User Data*/
CREATE TABLE Users (
  ID int,
  Name varchar(255) NOT NULL,
  Email varchar(255) NOT NULL,
  Username varchar(255) NOT NULL,
  Password varchar(255) NOT NULL,

  PRIMARY KEY (ID)
);

/*Create Calendars Table - Holds data about all the calendars that exist */
CREATE TABLE Calendars (
  ID int,
  Name varchar(255) NOT NULL,
  Colour int,
  UserID int NOT NULL,

  PRIMARY KEY (ID),
  FOREIGN KEY (UserID) REFERENCES Users(ID)
);

/*Create Calendars Table - Holds data about all the events that exist */
CREATE TABLE Events(
  ID int,
  Name varchar(255),
  StartTimestamp int NOT NULL,
  EndTimestamp int NOT NULL,
  Location varchar(255),
  AllDay bollean,
  CalendarID int NOT NULL,

  PRIMARY KEY (ID),
  FOREIGN KEY (CalendarID) REFERENCES Calendars(ID)
);

```

DB Creator.py

```
'''
Author: Robert Watts
Project: Computer Science Controlled Assessment 2019
File Name: Login.py
Description: Handles the main window that the user
             comes to. Creates menus and the event
             viwer.
'''

from tkinter import *
from CreateEvent import *
from EventViewer import *
from DatePicker import datePicker


class mainPage(Frame):
    def  __init__ (self, parent, controller, database):
        Frame.__init__(self,parent)
        self.DB = database
        self.controller = controller
        self.parent = parent

        #Resize Window
        self.controller.geometry("1000x730")

        #Get User Details
        self.controller.USER_DETAILS = self.DB.getUserData(self.controller.USERID)
        self.USER_DETAILS = self.controller.USER_DETAILS

        self.createMenu()

        self.createSideBar()
        self.addFrames()

        self.eventViewerHolder = EventViewer(self,self.controller, self.DB, self.controller.USERID)
        self.eventViewerHolder.grid(row=0,column=1)

        #Display the events on Current Date
        now = datetime.datetime.now()
        self.calendarClick((now.year, now.month, now.day))

    def createMenu(self):
        #Create Menu
        self.menubar = Menu(self)

        #Create Menu Buttons
        self.menubar.add_command(label="Calendar!", command=lambda: [self.controller.show_frame("mainPage"),self.controller.frames["creat
        self.menubar.add_command(label="Create New Event", command=lambda: self.controller.show_frame("createEvent"))
        self.menubar.add_command(label=self.USER_DETAILS[0],command=lambda: [self.controller.show_frame("createAccount"),self.controller

        #Pack Menu
        self.controller.config(menu=self.menubar)

    def createSideBar(self):
        datePickerSide = datePicker(self, command=self.calendarClick)
        datePickerSide.grid(row=0,column=0, sticky="n")

    def addFrames(self):
        self.controller.createFrame(createEvent)

    def calendarClick(self, input):
        self.eventViewerHolder.displayEvent(input[0], input[1], input[2])

```