NEA Programming Project - Jonathan Nwebube

Centre name:
Watford Grammar School for Boys

Centre number:
17655

Contents

Contents **2**
Analysis **5**
1.1 Definition of the problem 6
1.1.2 How is it solvable by computational methods? 7
1.2 Stakeholders 9
1.3 Overview of the problem 9
1.3.1 Interview plan 9
1.3.2 Interview transcript 11
1.3.3 Interview review 12
Research - Existing solutions 13
Limitations of this solution: 13
1.4 Key Features 17
1.5 Limitations 18
1.6 Software and Hardware Requirements 19
1.6.3 Success Criteria 20
1.6.4 Usability success criteria 22
Design **24**
Introduction 25
2.1 Problem breakdown 26
2.2 Top down modular design 27
2.3 Explaining modules 27
2.4 & 2.5 Algorithms 30
2.6 Usability features 36
2.7 Key variables and Objects 39
2.8 Order of development plan 41
2.9 Testing 42
2.9.1 Iterative development testing 42
2.9.2 Post development testing 44
Development **46**
Overview 47
Module 1: Web server 48
Prototype 1 48
Validation 50
Relation to top down modular diagram 51
Testing 52
Prototype 2 54
Relation to top down modular diagram 55
Testing 56

Conclusion of module 1 testing 56
Module 2: Web pages - backend 58
Prototype 1 58
Relation to top down modular diagram 61
Testing 62
Prototype 2 65
Validation 67
Validation 70
Relation to top down modular design 72
Testing 73
Prototype 3 77
Testing 78
Relation to top down modular design 80
Comments for maintenance: 80
Module 3: HTML & Authentication Templates 82
Prototype 1 82
Validation 89
Relation to top down modular design 90
Testing 90
Prototype 2 93
Relation to top down modular design 95
Testing 96
Prototype 3 97
Relation to top down modular design 99
Testing 100
Prototype 4 101
Relation to top down modular design 107
Testing 108
Prototype 5 113
Relation to top down modular design 114
Testing 115
Comments for maintenance 116
Module 4: Database 118
Prototype 1 118
Validation 121
Relation to top down modular design 123
Testing 124
Prototype 2 126
Relation to top down modular design 127
Testing 128
Prototype 3 133

Relation to top down modular design 134
Testing 136
Relation to top down modular design 137
Comments for maintenance 138
Testing and Evaluation 139
4.1 Post development testing for function and robustness 140
4.1.1 Authentication: 140
Account Creation 140
Account login 142
Account logout 143
4.1.2 Views: 144
4.2 Post development testing for usability 150
4.3 Success against objectives of functional components against success criteria 151
4.4 Addressing unmet/partially met tests of the system in the future 152
4.5 Evaluation of maintenance of the solution 152
4.6 Limitations of the current program 152
Bibliography 154
Appendix 155

Analysis

1.1 Definition of the problem

Many new businessmen in the truck driving business struggle to manage their drivers' hours due to the complex and sometimes confusing EU and AETR driving laws. This confusion is primarily due to a lack of accessible programs with specific inbuilt rules that adhere to the AETR and its changing rules. Businessmen need to manage and access readable data easily and quickly, to make a profit and expand a truck driving business.

My dad has started a truck-driving business, delivering containers from site to site. He finds it extremely difficult to manage each driver's data because a high amount must be recorded and accessible for calculations that will be used over 17 weeks.
Another problem is a misallocation of the hours a driver can legally do that week with no indicator of this by a program. Reminders aren't being sent to him to record hours or helpful tips to help allocate hours better.

This project aims to provide a scheduling service specific to the driving laws. It looks to include in-built features that may help to schedule better, such as emails to different devices about possibilities. It also aims to use machine learning to auto-fill driving hours that the leading company allocates to my dad's firm, which adheres to the driving laws with the option to edit and change them around correctly.

The current system involves over 20 excel sheets in different folders with more additional files. The biggest problem is navigating multiple files to find the driving schedules for the current week. Another con is that the schedules are updated weekly, referring to 3 additional documents.

- This includes the information about the truck to be driven, the sites to be visited and the truck's contents.

My solution looks to limit the different files needed to manage a business. The business will be manageable in one program with a page for each task. The pros of this are that there will be a reduced time in making and allocating schedules and an organised and efficient system for creating schedules.

Pros and cons of current method used:

Pros	Cons
Can easily share files to other managers in the company	Many folders and files which may make it hard to navigate
Folders can be used for less important things but related things such as their payment details or a history of payslips	Prone to a lot of human error because there is no validation
Any changes can be traced back with a	There is less control and security of the files

revision history	isn’t guaranteed
	It requires a constant internet connection to access the solution

1.1.2 How is it solvable by computational methods?

Visualisation

- I will have a GUI so the user can easily select different program features. It will also allow the user to easily enter data for correct dates with clear and logical prompts that feel natural. It will be constructed with images and text.
- I will use this to get a clear understanding of the problem and its solution allowing me to have an idea of multiple solutions and what problems I could face before starting the implementation. This would help me pick a better solution design.

Problem decomposition

- This program is easily decomposed as its multiple features include inputting hours worked, displaying hours worked, and a function to show how many hours can be worked for each upcoming day and week.
- The code will also be in the form of structured programming.
- I can work on multiple modules at the same time, which would help increase productivity as I can take breaks from a module when faced with a problem without a discovered solution as I work on something else until I figure a solution to the other module
- This will help me save a lot of time such as the possibility to reuse code.

Problem recognition

- There is currently a lack of systems to manage driving times according to the always changing AETR and EU driving laws.
- To automate the system as much as possible with minimal user input to prevent as many mistakes
    - I will use problem recognition because it will help me understand the problem better, allowing me to make a more focused program. This will reduce the stress of time to complete the program as there would now be more possible solutions I can try instead of exploring a limited scope.

Algorithms

- Using algorithms can reduce the time spent coding because features like autofill can be reused.
- I will use algorithms because they are easy to understand and will also help reduce programming time as I don’t have to repeat code as I can reuse them. It will also help with the readability of my code, allowing me to run tests easier and locate errors quicker.

Abstraction

- The use of abstraction in this program is essential as it disregards unnecessary components not needed to run the system. For example removing the locations of the truck's destination as the focus of this system is to track distance.
- This ensures the program is more focused on managing and displaying schedules because the design's complexity is reduced.
- It will allow me to finish the program implementation within my time period due to reduced unnecessary features.

Thinking procedurally

- After decomposing the main problem at hand into independently solvable modules, how do I go about developing and testing them, building my way up to linking them all together and solving the main problem?
- This would allow me to understand smaller sections of the solution easier so that it is easier to design specific features

Performance modelling

- I will use performance modelling to test my code against. This process of evaluating the system performs as expected before being released. For example, I could simulate many drivers and the completed times at once to stress test the system.
- As I have a budget with no money, this would allow me to test my solution cheaper than other ways.
- It allows me to test my solution before implementing it so that I narrow my final solution down
- It will help me find unexpected problems so that a fix for them can be made quicker allowing me to meet the deadlines for this program

Data retention

- Due to the nature of this program handling numbers that represent hours, with every hour being used for further calculations, data retention is necessary to produce this program as data from a period over 17 weeks will need to be stored due to the rule that drivers cannot exceed an average of 48 hours a week over a 17 week period. It is essential for any scheduling program to store data over prolonged periods without the fear of deletion because it could result in incorrect calculations, which can cause laws to be broken.
- I will use this as it will give me an improved data quality, especially as there are going to be a lot of calculations for this program
- It will also make implementation and design easier for me as it makes it simpler by grouping and organising data types.

1.2 Stakeholders

The stakeholders of my program would have to be driving services such as Knight-Swift Transportation Holdings Inc for the truck driving the market. I have chosen my dad to be my primary stakeholder as he represents a majority of new businesses getting into this market. He will tell me what he would expect from the software and what features should be programmed. I think this would be useful for new businessmen as they can see all the relevant information without an information overload and clear functions.

The most important thing will be readability, usability and the program's layout. This means I have to research and understand the laws to help guide them to manage their drivers' time correctly to adhere to the laws. This would be done by removing less important things via efficient abstraction and using intuitive designs

1.3 Overview of the problem

1.3.1 Interview plan

Interview to be conducted on two stakeholders
I want to have specific questions for the manager of this business and the employees. My program has to have meaningful features because this program is specific to one type of business only therefore features can easily become useless thus making the program inefficient. The program has to be able to help manage a business to any scale hence its required efficiency. My questions would help me determine how successful the program is.

The interviews will take place on 12 April 2022, in a coffee shop. The setting is in a comfortable less formal place as it is vital for the stakeholders to feel comfortable which allows for more information to be passed by them. This makes the interview seem more like a running conversation which is important as subconsciously people speak more freely in a running conversation and speak their mind. Therefore the room layout of this interview isn't totally controllable, however if possible it would take place in the quieter area of the shop and on cushioned chairs.

The interview will be recorded on a phone so it can be played back later on, this prevents unnatural pauses writing down notes about what is said. The conversation flows better this way. However a notepad will be present to write down minimal things, This is so that the stakeholder feels they are being listened to and their points are going to be remembered.

Plan for interview 1 - Manager

Question	Justification
Question 1 What is the difficulty of schedule management?	This question will be asked so that we can highlight the bigger problems of schedule management. These highlighted issues will be ranked in a hierarchy system, with the program focusing more on making these problems as easy and straightforward for the users

Question 2 How are schedules currently managed?	This question will be asked to gain an insight of the current solution. Highlighted features of this solution proved to be very helpful will be included in the implementation of the new solution to add a sense of formality.

Follow up questions

Question	**Justification**
Question A What features would you like to see?	This is a follow up question to the second question in the event the stakeholder doesn't go into detail about already used features in their current solution that would be useful in the new solution
Question B Are there any other features that sometimes come in handy?	Follow up question to question 2 and question a if the interviewee doesn't go into much detail about every feature that could be of possible use. This interviewee could possibly be asked 3 times the features they would like to see, without knowing it. It allows me to get more information than they had thought to share.

Plan for interview 2 - Driver

Question	**Justification**
Question 3 Are there problems with the current solution?	A general open ended question for drivers to get them to think about all possible problems no matter how minor they are.
Question 4 What features would be useful to you	This question allows the driver to request features that they may have nit been given the opportunity to have added. Managers and drivers see different things and as a result the manager may not realise that viewing the schedule their way is easy to understand for a driver, as they may focus on different information of the schedule

Follow up questions

Question	**Justification**
Question C How do you view your schedules right now?	This is a follow up question to question 3 in case the driver doesn't mention the viewing of the schedules as a problem. It looks to see if they had forgotten to speak about it or to tell us the good things about how they view their schedules so it can be incorporated into the new solution

1.3.2 Interview transcript

Interview 1 - New Manager to this line of work

Is it difficult to manage workers schedules?

"Of course. I found it to be the biggest difficulty in managing a business like this. As an experienced manager in a varied line of work such as restaurants to office companies, managing time was simple because all I had to do was allocate shifts to people, however in this driving field there are now rules to the shifts I can allocate which I have to take into consideration. On top of this keeping track of so many driving laws for so many drivers becomes tiring and near impossible. This is mainly due to the amount of hours worked even 24 weeks ago has to also be taken into consideration for when I allocate an employee another shift"

How do you currently manage the allocation of shifts?

"Currently, I use an excel sheet to track the shifts, and have been colour coding different factors to help with the readability of the files, such as highlighting the last two weeks of an employee's shift in yellow and also highlighting the last 26 weeks of an employee's shifts in purple. These help me calculate how many hours I can allocate per person in the next week to conform to the 17:26 rules and also the maximum of 90 hours in two consecutive weeks. Currently I am doing a lot of repeated maths, working out averages and subtractions per driver"

What features would you like to see?

"I would like to see how many hours I can allocate a driver in a week without having to do any of the maths myself as I am prone to a lot of mistakes due to handing a lot of data. I would also like to see recommendations of who to allocate driving times to"

What non-crucial features would you like to see?

"Seeing the drivers and then a profile of them opening up with details about them such as their previous trips and total damages they have caused or their contact information would also be very convenient overall but not a direct need to my scheduling problem, I suppose a program that allowed me to limit my job to one app would be very convenient too, for example, an inbuilt messaging or notifying feature would help so I can alert drivers of their new weekly schedules in the same app or even allow them to decline or request changes in their schedules"

Interview 2 - Driver

What problems do you face with receiving your schedules?

"A lot of the time the schedule has to be reformed either due to errors with how many hours I am legally allowed to work or the swapping of someone else's shift due to other reasons. But the main problem is the format of the schedules we get which then results in a lot of confusion and

phone calls to try and understand what's in front of me. Right now, I receive my schedule through WhatsApp, and it either comes via a link to an excel spreadsheet or a photo of one, sometimes it is annoying to access"

What features would you like to see?

"Probably a dedicated app that shows me day by day and hour by hour where I should be and the information of the containers I'm meant to pick up and deliver. Even information about which truck I should be driving"

What non crucial features would you like to see?

"The distance of the travel and estimated times as well as what site entrances to go to instead of asking site workers where to go each time depending on the container"

1.3.3 Interview review

The first interviewee is new to this sector and has been in it for less than 7 months. They have made it clear that they've struggled to increase the scale of the business by hiring more drivers and as a result delivering more containers due to the limited resources.
The employee I interviewed has been in this line of work for 14 years but has recently been employed to a new small company and has had the same problem throughout which was the out-dated systems companies use to manage its employees.

It is clear the users require a platform that displays schedules via a GUI. This is how both stakeholders want to either view or input schedules in my program. The main stakeholder was the employee and they wanted more features than the employees due to the fact they have to manage an unlimited amount of employees. He wanted to see more automatic calculated features, whereas the employee wanted to see a better schedule design with more information. This means that when I develop the program I have to allow the manager a lot of optional prompts to enter details for the employees resulting in me considering the layout of the GUI I use as there could be more information on the screen than I originally thought. With this I need to consider how readable information is to ensure the program looks simple to understand for the different possible users. I will make sure the program is well tested to keep the focus of the program which is time management

Overall, both stakeholders want to see

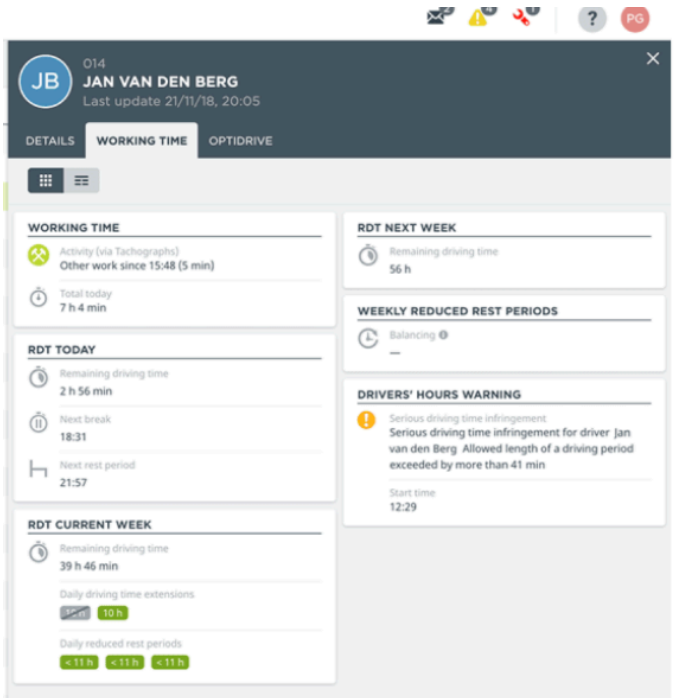
- An intuitive GUI for their schedule
- More information either about the schedules or the drivers'

Research - Existing solutions

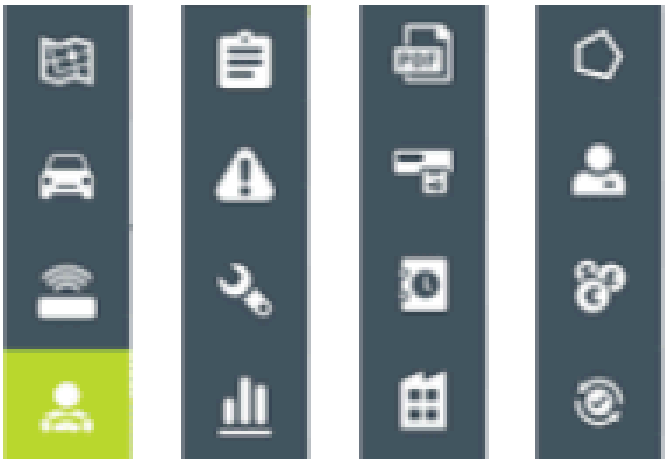
A few solutions can be found on the internet however can be hard to locate. They tend to be hard to navigate due to confusing and wordy websites.

Example 1 - WEBFLEET Tachograph Manager

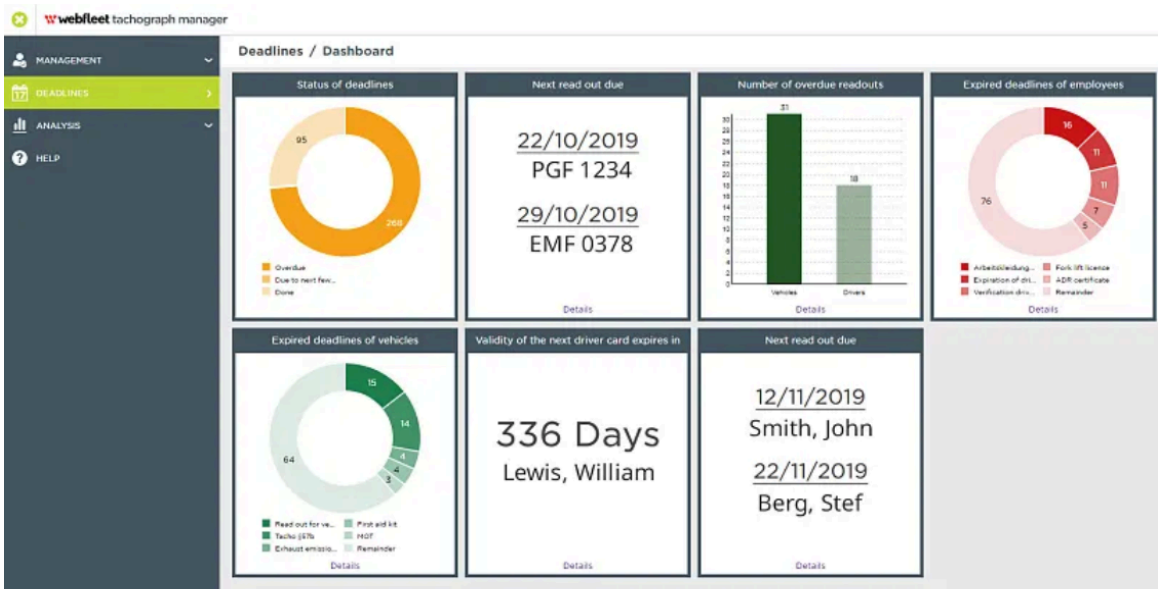
Overview of this solution:
The best solution I found was WEBFLEET Tachograph Manager because it has a lot of useful features such as its GUI. The solution is more advanced and much more intuitive than the other solutions on the internet.
A noticeable feature is has is the clicking on a drivers profile and a smaller screen displays all relevant information of that selected driver, such as the possible hours in a shift they can work in a given time period specific to the AETR laws



Limitations of this solution:
Although this solution could help a manager manage its hours amongst the drivers appropriately, there isn't a method to autofill and recommend driving times to the end user.
The main aim of my program is to make it user-friendly to prevent any mistakes occurring and help relieve workload on the manager therefore a feature that displays recommendations would be incorporated.
This solution has a long menu with icons that are confusing and making it difficult to understand the features of the program or how to actually add the information to access the main function of the program; displaying schedules per driver



What I would like to include from this solution:
From this solution I will incorporate the feature of displaying more specific information to a selected driver when interacted with, such as showing me the possible hours in each time frame they can drive.



Example 2 - AETRControl

Overview of this solution:
This solution is accessible via mobile app, allowing for easy accessibility of driving hours at any given time. It also provides a range of tachograph devices which can work directly with the app when connected. This method to solving driving management allows drivers to view their times however would prevent the truck driving managers of companies to easily keep track of all of their drivers times

Limitations of this solution:
The main flaw in this design is that each tachograph has to connect to a phone and must be physically present to update. This would become a problem in the long run when companies grow and hire hundreds of drivers.
The design of the app is basic and feels out of date as it uses old generic icons and colours. There is also a lot of information available however a poor menu design makes navigating the app hard and tedious.

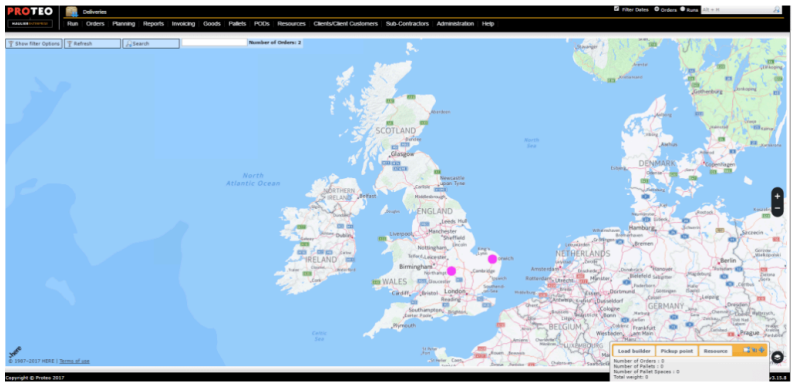
What I would like to include from this solution:

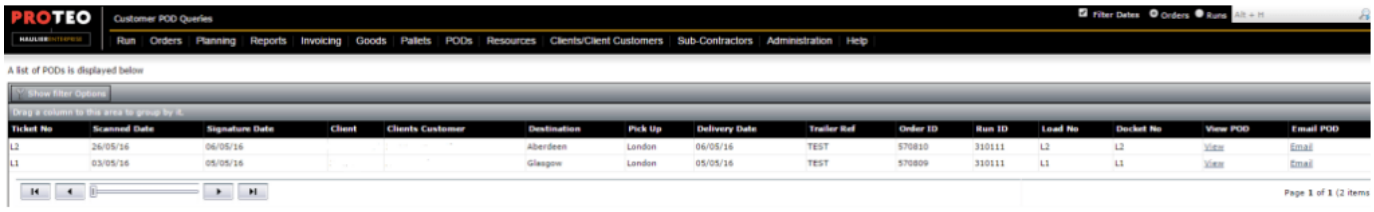
From this solution I will incorporate the feature that allows drivers to see their shifts

Example 3 - Proteo

Overview of this solution

This solution is a web based system designed for managing hauling business with less than 25 trucks. It has features such as drag and drop resourcing and maps to help display schedules. In comparison with the other two similar solutions has the map feature which would cut down the amount of text on screen and allow easier readability.





Limitations of the solution

The solution is a web based system, meaning internet connection would be needed to access the program at any time. The system needs to be made accessible when not connected to the internet so that a driver or manager can have an idea of their schedules. This issue could be fixed with a downloadable function.

What I would like to include from this solution

I would like to include a map to show the distances between the sites driver's move between which will help with the time management

1.4 Key Features

After a discussion with the stakeholder, I have key features that he has requested I implement into the system and some I believe will help with the functionality of the program

- **Login**
  Login for each driver and the manager. The manager and drivers have different access. For example the manager can view the schedule of each driver and see their driving history, however the drivers can only see their own driving schedules and history and not any others.
  This is needed to allow the manager's account to have access to change times around, however the driver would not. This adds levels of security and ensures hierarchy roles in the system are in place.

- **Messaging**
  In-built messaging is not essential, however it would allow the drivers to communicate to the manager about their schedule after viewing it in the same program. This is needed to allow an easier way of communication for possible scheduling changes.

- **Notifications**
  Notify the drivers when the schedules come out, however this is not essential. This feature is to guarantee flow of business with less disruptions due to human error such as forgetting to check the schedule.

  A possible extension feature to this is the ability for drivers to accept their schedules and notify the manager they have seen their schedules and are happy with it. On the managers end they could see a green box or a red box representative of if the driver has seen it or not

- **Drivers hours**
  A list of all the drivers will be displayed alongside how many hours they can drive that week. This allows the user to easily see the availability of drivers, which then helps them manage future driving for that week.
  Upon selection of a driver via mouse click, more information will be displayed to show:
  - How many hours they have driven in the last week and the current week combined
  - How many hours they have driven in current week alone
  - How many hours they can drive during the current week
  - How many hours they can drive the following week
  - How many more times they can extend a driving day to 10 hours

  The extra information isn’t on the screen right away to keep the main screen simple and easy to read

- **Scheduling aid**
  A calendar would be added to show what days drivers will be driving in future. This would also visually help break down management. This feature can be built upon and see some automation as the program fills in driving slots that adhere to the law taking into account how many hours they have already driven
  Due to automation being involved, the driving hours that need to be shared out will be forwarded in bulk to a program that can then fill out the times based on the information it receives, such as distances, and time

- **Ability to add and remove drivers**
  The stakeholder requested this be included as they have plans to upscale the company in the future, therefore they want to get this program future proofed

1.5 Limitations

Limitations to this program is that auto completion may not be of use to many users as the main basis of this program is to keep track of drivers and their work to comply with the AETR and EU laws. This would be a limitation because it would require more memory space for the program even though the feature wouldn't be used. This would also increase the time to complete the program and could result in the incompleteness of other parts of the program.

Another limitation is that the security may be compromised as the program won't be local. Information is accessible from any device signed in and thus, information will be sent from device to device. The possible messaging feature would not be encrypted so if hackers get a hold of their messages they are able to read it. Only forms of security is the sign in feature with accounts having more or less access than others. Passwords would be secure due to a hash function and encryption where passwords can't be reversed search even by owners of the source code. This is a problem because employee information needs to be protected to adhere to the GDPR and data protection act. This would become more of a problem as this solution might not be widely used due to this factor.

Another limitation is with the website because the customer's device features, for example the website would be dysfunctional on a phone if it was made for computers. This would mean this program has to have adapted designs for multiple device types and also take into consideration the maximal specs smaller devices can have to make sure the program can run on them.
This is a limitation because more computational methods would be needed, increasing the design and implementation times. There are likely to be more irregularities between the devices.

1.6 Software and Hardware Requirements

This program will require the user to have tachographs that can be interpreted by the program, however due to the function of the program to be a user-friendly solution, options to enter data in manually will be more viable

Software specifications	Justification
Google Chrome	Extensions to the app would be displayed in a web browser using html, so for this to work google chrome is required The web browser will be used to access some parts of the program so would be needed to have full accessibility of the program
WIndows 7+ / Linux / Mac OS	This is needed because python needs an operating system to run on
Notepad++	This is needed to store the data of the drivers somewhere and also to help with CSS files
CSS	This is needed to design the user interface as it will be a website

Hardware specifications	Justification
2GHz or greater CPU	This is needed to calculate instructions such as to open a file if opened
4Gb of RAM	The RAM is needed to run the program as external files need to be quickly fetched from main memory
Intel Pentium 4 processor or later that's SSE3 capable	This would help run the program smoother due to a lot of number crunching and high data usage

Peripheral specifications	Justification
Keyboard	It is needed to input information for the program
Mouse/Mouse pad	It is needed to click on drivers and navigate the program
Monitor	It is needed to view the program

1.6.3 Success Criteria

Criteria	Justification	Measure of success and expected outcome	Minimum score to pass
1. The user should be able to sign in	If the user is unable to sign in after creating an account then all information is lost for each driver. If the user is able to sign in with an incorrect password, they will feel unsafe using this program due to other people being able to access the account without knowing the correct password	A user signs in using their correct password and are given access to their account with their schedules This will be measured by entering incorrect usernames with correct passwords and correct usernames with incorrect passwords to see if access to the program is accessible with invalid entries. - Expected outcome is a pass Then it will be measured by entering the correct username and password. If a user is able to gain access to the program and account specified information tailored to them and them only then a pass will be given. - Expected outcome is a pass If the user is unable to sign in or are given account specified information for another user then a fail will be given - Expected outcome is a fail A fail = 0 A pass = 1 Will evidence this by screenshotting the page after signing in	1
2. Must have a menu with an intuitive design	To keep the program beginner-proof, a functional unambiguous menu is needed The program is meant to be as user-friendly as possible - It should be able to be used by someone who has no knowledge of this hauling business	A random friend who knows nothing about this program is able to navigate through it and work out its function after using it for half an hour The friend will be told to Make an account and log out then sign in Navigate through the menu to somewhere i tell them to Display the hours of a random diver I tell them to Enter information to a given driver	2 and above

		Add drivers and remove drivers ---------- The friend will then rate the program on a scale of 1 - 3 with 3 being easy to use and 1 being confusing to use Will evidence this via feedback from the friend either screenshotting an email from them with their feedback or another method	
3. Must be able to display correct hours	For the program to be useful, the user needs to be able to see how many hours the driver has driven in a time period and see how many hours they can drive in a time period	Putting in some normal and boundary data in the program for it to calculate the hours in each time period left and comparing it with manual calculations The manual calculations should equal the programs calculations Fail = 0 Pass = 1 Will evidence this by screenshotting the output	1
4. Must be able to enter information	For the program work, the user needs to be able to enter the hours driven so the program can do its calculations and return something back	Enter values into the program Fail = 0 Pass = 1 Will evidence this by printing what is entered, into a terminal and screenshotting it	1
5. Add drivers / remove drivers	The user needs to be able to add and remove drivers to record their hours specific to them otherwise the main focus of the program wouldn't work	Create and delete user accounts and checking the files of all accounts to see the changes in them Fail = 0 Pass = 1 Will evidence this by screenshotting the SQL tables of user information such as a dummy username to maintain privacy of real users.	1

1.6.4 Usability success criteria

1.6.4.1 Design usability success criteria

Criteria	Justification	Measure of success and expected outcome	Minimum score to pass
6. The web page will fit any screen size	This program is meant to be a portable solution resulting in it being accessed by people on their mobile phones when away from a desktop. Desktop monitors and laptop screens also differ in size and must be accommodate for	The website should be run on a phone and the web pages should fit on the screen and have buttons accessible. This awards a point The website should be able to display on a random laptop. It should then be used in full screen with the schedules still viewable This awards a point The website on a laptop should be accessible when its not in full screen and is a window. The buttons need to be clickable and return a new web page This awards a point This will be evidenced by screenshotting the website on various devices in multiple screen modes	2
7. The program should have a minimalist feel	In interview 2 the user finds reading the current solution confusing. This is because in interview 1 the user says that an excel spreadsheet is used to manage the schedules. A Lot of small boxes with different information makes it distracting and hard to read so a minimalist solution eliminates any distractions using abstraction	The website should be intuitive and easily navigable. Anyone should be able to understand what the program is for after using it for 5 minutes with no prior information The program will be given to someone with no knowledge of what it is for. And they have 5 minutes to guess what it's for. They start off at 5 marks, every minute without the correct guess a mark is taken off. After 5 minutes a 0 is rewarded	1

1.6.4.2 Input usability success criteria

Criteria	Justification	Measure of success and expected outcome	Minimum score to pass
8.	This is needed information to sign	A html form showing these options is accessible	1

The user can enter their first name, second name, email and password	up and create account to be added to the table in the database to query them later	when logged out Pass mark is 1	
9. Sending a message across networks from different computer systems	In interview 1 the user had said a useful feature would be an in built messaging service to keep all work related things in one app.	A message sent to one computer is able to be displayed on another computer Pass mark is 1	1
10. Inputting jobs for schedules, such as driver name, time, date and place	This is so the drivers can view their schedules later on. Interview 2, the user wanted an easier less confusing method to view their program	A table adds a record of the newly added details. This passes if the table can query the recently added record correctly Pass mark is 1	1

1.6.4.3 Output usability success criteria

Criteria	Justification	Measure of success and expected outcome	Minimum score to pass as a success
11. Displaying messages	To have a built in messaging service, sent messages muse be displayed on corresponding computers to have a conversation	A message sent to one computer is able to be displayed on another computer Pass mark is 1	1
12. Displaying driver names	Manager needs to see all its drivers in the database, to select who to add a job for to update their schedule	List of all drivers name from a query of the table in the database is displayed Pass mark is 1	1
13. Displaying schedule	Once a manager has added a schedule, its drivers need to see this update.	All jobs in the table can be displayed correctly This awards 1 mark The jobs should only be for the driver signed in and not anyone else This awards 1 mark	2

Design

Introduction

My project will be written in Python using VScode IDE. Python is an ideal language for developing my scheduling program because python code can be written faster than other programming languages due to its simple syntax. Python has quick automation and simple web development which is valuable to this program.

I use this integrated development environment because it allows me to use multiple programming languages for my program with all the relevant files displayed and accessible in the IDE. This makes it easier to work on multiple scripts at the same time.

The models which I will be using is:

- Flask: This will be used to make the website front of the program
- Tkinter: This is to make the GUI
- SQL: For handling the data and communicating with a database

2.1 Problem breakdown

The program can be broken down into 5 main modules

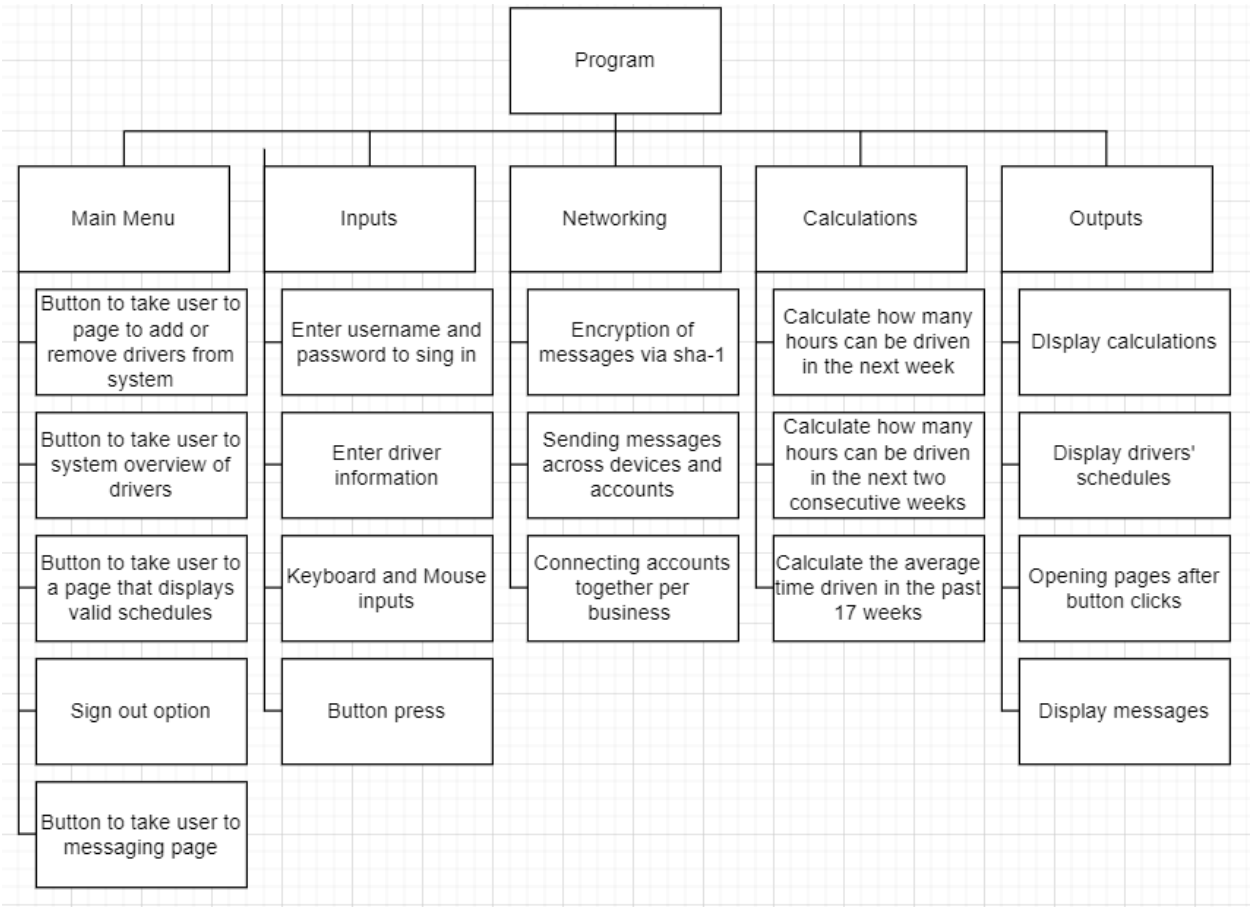
- User interface: To do with the interface between the users and the schedule manager
- Calculations: This is to calculate possible schedules to help manage workload
- Networking: This is to do with the transmission of data between managers and drivers
- Input: Handle data inputs
- Output: Return data from inputs

Stage 1	Haulage website that helps with scheduling and managing a company					
Stage 2	User interface			Calculations		Networking
Stage 3	Main menu algorithm **Justification** The menu can be programmed in one algorithm	Schedule		Mean hours algorithm **Justification** Working out the hours driven in the last set weeks for multiple reasons is needed to make sure the stakeholder follows AETR laws. This can be programmed in one algorithm	Comparison algorithm **Justification** This tells the driver the results of the calculations thus helping them make decisions on who to schedule a job for This can be programmed in one algorithm	Messages algorithm **Justification** The messages need to be sent across different networks This can be programmed in one algorithm
Stage 4		Edit schedule algorithm **Justification** This is an important part of the program as it allows the manager to add, update or delete jobs in a schedule and can be programmed in one algorithm This can be programmed in	View schedule algorithm **Justiciation** This is an important part of the program as it displays the drivers' schedules by day This can be programmed in one algorithm			

		one algorithm				

2.2 Top down modular design

Program Structure: Top Down Diagram



2.3 Explaining modules

<u>Main Menu</u>
The main menu will have multiple buttons to take the users to different parts of the program. Each part of the program would be coded in subroutines, as a result when a button is pressed on the main menu, it will call on the subroutine needed.
The menu would be the first page the user sees after signing in. The user will see icons which double up to act as buttons.
There will also be a sign out option because users may prefer to log out after each session

This is important because the user needs to be able to navigate from page to page quickly and correctly. WIthout a main menu the user will be stuck on one page with a limited program which may not fulfil user requirements

Inputs

Depending on the page the user opens, there are many opportunities to enter information into the program. The main inputs will be the input of driver schedules. The user may enter a number representing the hours driven within a range of 1-9, however there will also be an option to increase this range up to 10 twice a week. To allow this, there will be a button on the side to increase range.

The inputs will need to be checked to make sure they are valid. This can be done by having a limited range of possible values to be entered to reduce input mistakes

This is important to make sure the user can create or modify schedules so that the other users can see their work plans. Inputs are needed to make this program useful otherwise users cannot use the program

Networking

A connection sequence needs to be designed to allow the users (managers and drivers) to communicate with each other such as messaging and updating schedules and having it automatically displayed on another's account.

This is important because without this, the drivers will not receive their schedules. Networking is needed to transmit data from account to account allowing users to share information efficiently and effortlessly to increase productivity

Calculations

Calculations will be part of the back end in this program. The use of calculations range from validation (such as validating user details to sign into an account) as well as calculating the averages of data quicker than a human could.

Calculations for driver hours will be done via calling on data within the last 17 weeks maximum and working out three different numbers. This means the system doesn’t need to store information from over 17 weeks ago.

This is an important module because it allows the computer to do hundreds of calculations for multiple accounts quicker than a human could and with less errors than a human.

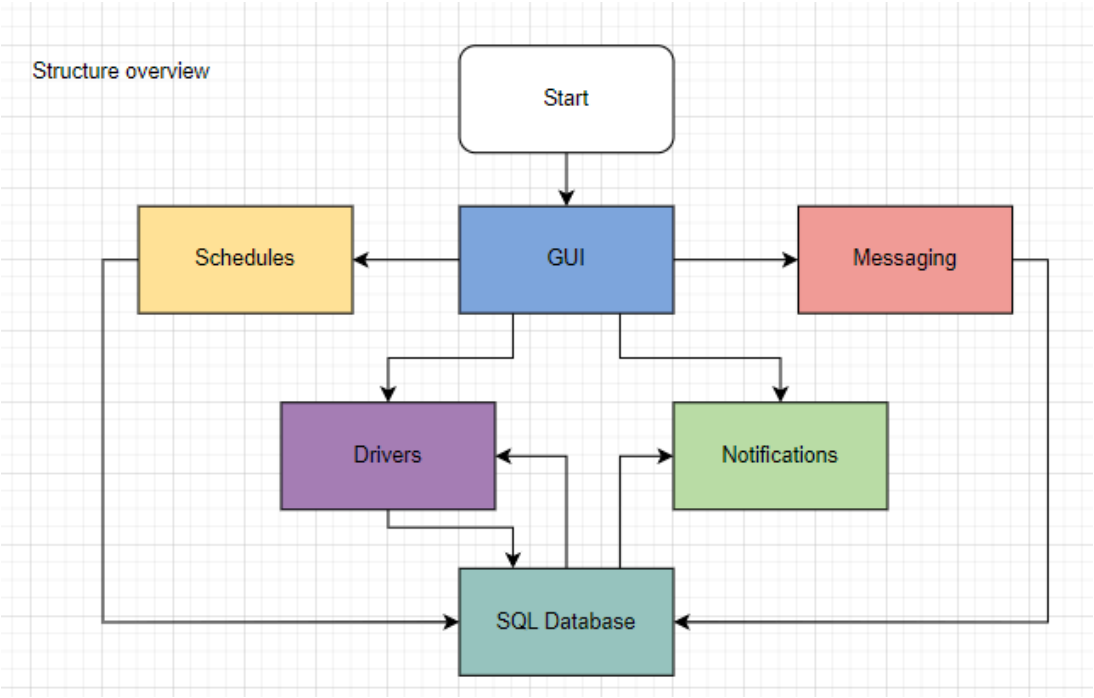
Outputs

This would need to display the information from the calculation module so that the user can see its information about the drivers. As the program will be written using OOP, subroutines called may return values, these values will be displayed for the user to access.

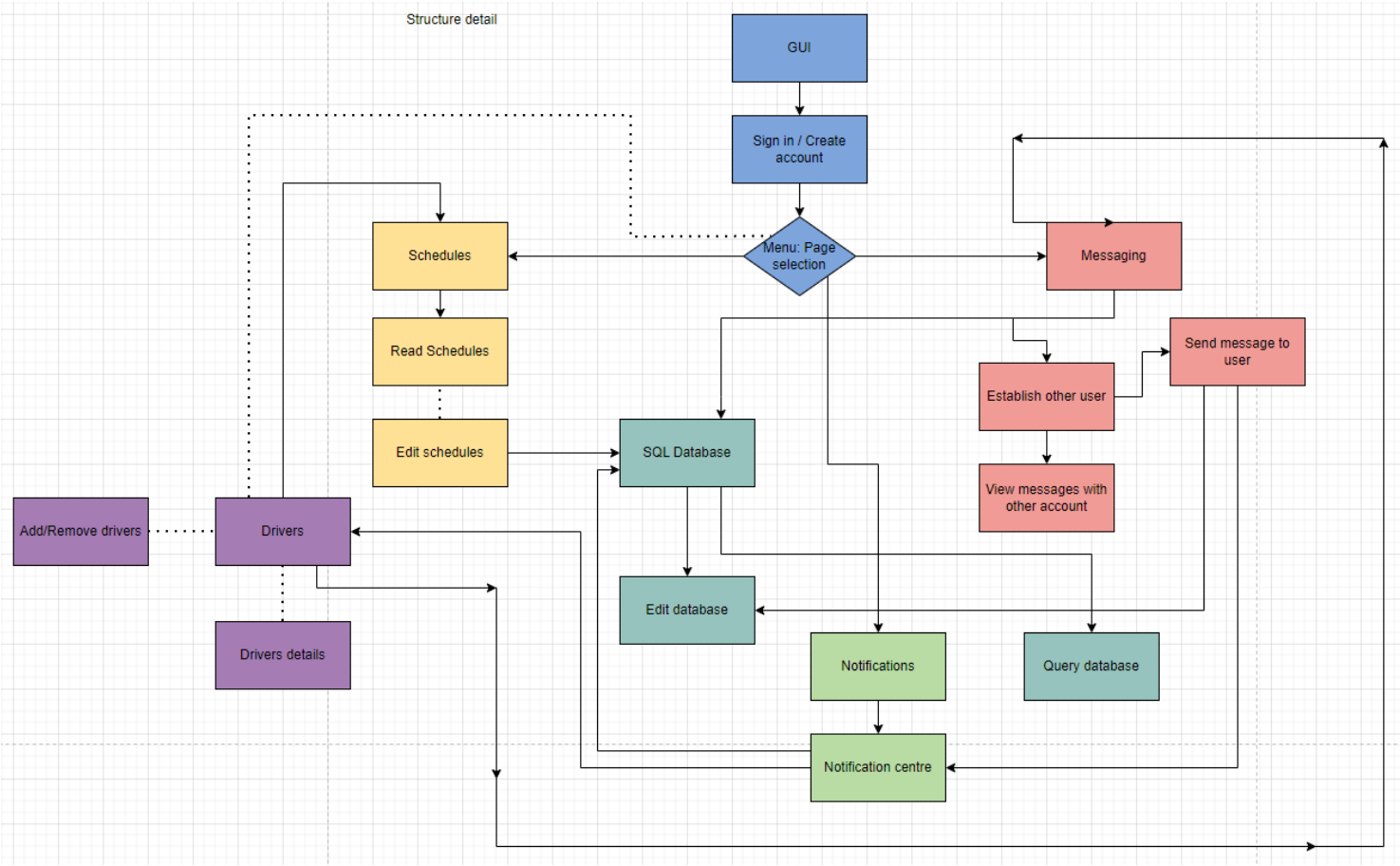
This module is important in this program so that the user can access data processed by the computer in a human-friendly form.

Structure

Structure overview:



Structure detail:



Structure explanation:

These diagrams show how the program will work. The GUI will be the main way users interact with the solution. The user, once signed in, will have options to go to various pages (i.e Schedules, Drivers, Messaging, Notifications). The SQL database will be running in the background to link all the features together, for example messages will need to be sent into the correct user's messages and notification centre - when messages are sent it will first be saved in the SQL database messaging table. When the account it was sent to opens the notification or messaging function a query of the messaging table will display the message.
The dotted lines show features only managers have access to.

2.4 & 2.5 Algorithms

Each section of this program will be completed as individual tasks, and once all are completed combined together to form the end solution.
Each section will be tested during and after its creation, before starting a new section.
The program will be created in this order below - it is like this because I will create the basic and fundamentals of the program first, then move on to sections with lower priorities:

1. Creating a database
2. Creating an account
3. Signing in to account
4. Creating the web pages template: using HTML and external CSS files to make the different pages unique
    - Account creation page
    - Login page
    - Main page
    - Schedule page
    - Messaging page
5. Main menu
6. Scheduler manager to edit and display the schedules
7. Messaging module between members of the company

Creating a database

The database will be created first because it will make testing different subroutines easier, such as viewing a schedule or logging into an account. The database is arguably the most important element of the program because the majority of the functions will rely on the database to complete its function.

I will be using SQLAlchemy for my database. This is because it is a relational database and not a file-based database approach. Relational databases are more flexible than file-based databases as they can contain multiple tables of data that relate to each other through special key fields.

There are going to be multiple tables in this database because there are multiple attributes in this program. This means that I will have to establish foreign keys early to connect these tables together.

- This will allow me to repeat less in my tables

Creates tables if not already made

```
db = SQLAlchemy()
DB_NAME = "database.db"

FUNCTION create_database(app) THEN
    IF NOT path.exists 'website/' + DB_NAME THEN
        db.create_all(app=app)
        OUTPUT ('Created Database!')
```

This database algorithm helps complete the solution because it stores all the data the program needs to output the schedules. For example, a user can access their schedule by querying the Schedules table.

It is a necessary algorithm because queries will need to be made for each user as they have different driving schedules

The database tables

```
from flask_sqlalchemy import SQLAlchemy

CLASS Accounts
    id int,
    username string,
    password string,
    first_name string,
    second_name string,
    manager int,
    message relationship 'Messages'


CLASS Messages
    id int,
    data string 1000,
    date DateTime(timezone=True),
    accounts_id int, ForeignKey 'accounts.id'

CLASS Schedules
    id int,
    day string,
    time int,
    truck string,
    destination string,
    accounts_id int, ForeignKey 'accounts.id'
```

Creating / Signing into an account

There will be two options to create or sign into an account:

- Create/Sign into accounts for a manager
- Create/Sign into accounts for drivers

There are two options in order to keep the program simple and split up between the different access between a manager and its drivers accounts.

```
FUNCTION login
    Username <- request.form.get 'Username'
    password <- request.form.get 'password'
        IF Username in Accounts and password in Accounts THEN
            OUTPUT 'Logged in successfully!'
        ELSE THEN
            OUTPUT 'Incorrect username or password, try again.'

FUNCTION sign_up
    first_name <- request.form.get 'firstName'
    second_name <- request.form.get 'secondName'
    password1 <- request.form.get 'password1'
    password2 <- request.form.get 'password2'

    firstThree <- first_name[0:3]
    secondThree <- second_name[0:3]
    Username <- firstThree + secondThree

    IF Username in Accounts THEN
        OUTPUT 'Username already exists.'
    ELSE IF LEN first_name < 2 THEN
        OUTPUT 'First name must have at least 3 characters.'
    ELSE IF LEN second_name < 2 THEN
        OUTPUT 'Second name must have at least 3 characters.'
    ELSE IF password1 != password2 THEN
        OUTPUT 'Passwords don\'t match.'
    ELSE IF LEN password1 < 7 THEN
        OUTPUT 'Password must be at least 7 characters.'
    ELSE THEN
        new_user <- User Username=Username, first_name=first_name, second_name=second_name, password=generate_password_hash(
            password1, method='sha256')
        db.ADD new_user
        db.COMMIT
        OUTPUT 'Account created!'
```

This login function checks what the user has entered on the sign in page and verifies the information matches and is on the database. This helps complete the solution as users may need to sign in to their accounts on multiple devices to access the same information. Being able to sign in allows them to gain access to their driving schedule and other features.

This sign_up function is what creating an account may look like depending on what is required to make an account. It receives user input and uploads it to the database once verified. This helps create the solution of the problem because an account needs to be created for every user to view their driving schedules

Creating web pages

Using HTML and CSS the web pages will be made following a base template so that the program has a followed through themed design. The other html files will be for each page in the program and will be an extension of the base file.

- Time will be saved from repeatedly typing the same lines of html to maintain the theme of the program by extending one file.

Most programs are now website based. So to make sure this program is competitive the front end needs to be accessible via web instead of downloading.

- The pages will be simple and easily navigated

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <link
      rel="stylesheet"
      href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css"
      integrity="sha384-Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh"
      crossorigin="anonymous"
    />
    <link
      rel="stylesheet"
      href="https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css"
      crossorigin="anonymous"
    />

    <title>{% block title %}Home{% endblock %}</title>
  </head>
```

This is a HTML algorithm to link bootstrap to the web pages.
This helps complete the final solution because HTML can manipulate the web design.

This is the HTML algorithm to include a navbar on the web pages. Helps complete the solution by providing a menu accessible on any web page for the user to navigate the program

```
<nav class="navbar navbar-expand-lg navbar-dark bg-dark">
  <button
    class="navbar-toggler"
    type="button"
    data-toggle="collapse"
    data-target="#navbar"
  >
    <span class="navbar-toggler-icon"></span>
  </button>
  <div class="collapse navbar-collapse" id="navbar">
    <div class="navbar-nav">
      {% if user.is_authenticated %}
      <a class="nav-item nav-link" id="home" href="/">Home</a>
      <a class="nav-item nav-link" id="logout" href="/logout">Logout</a>
      {% else %}
      <a class="nav-item nav-link" id="login" href="/login">Login</a>
      <a class="nav-item nav-link" id="signUp" href="/sign-up">Sign Up</a>
      {% endif %}
    </div>
  </div>
</nav>
```

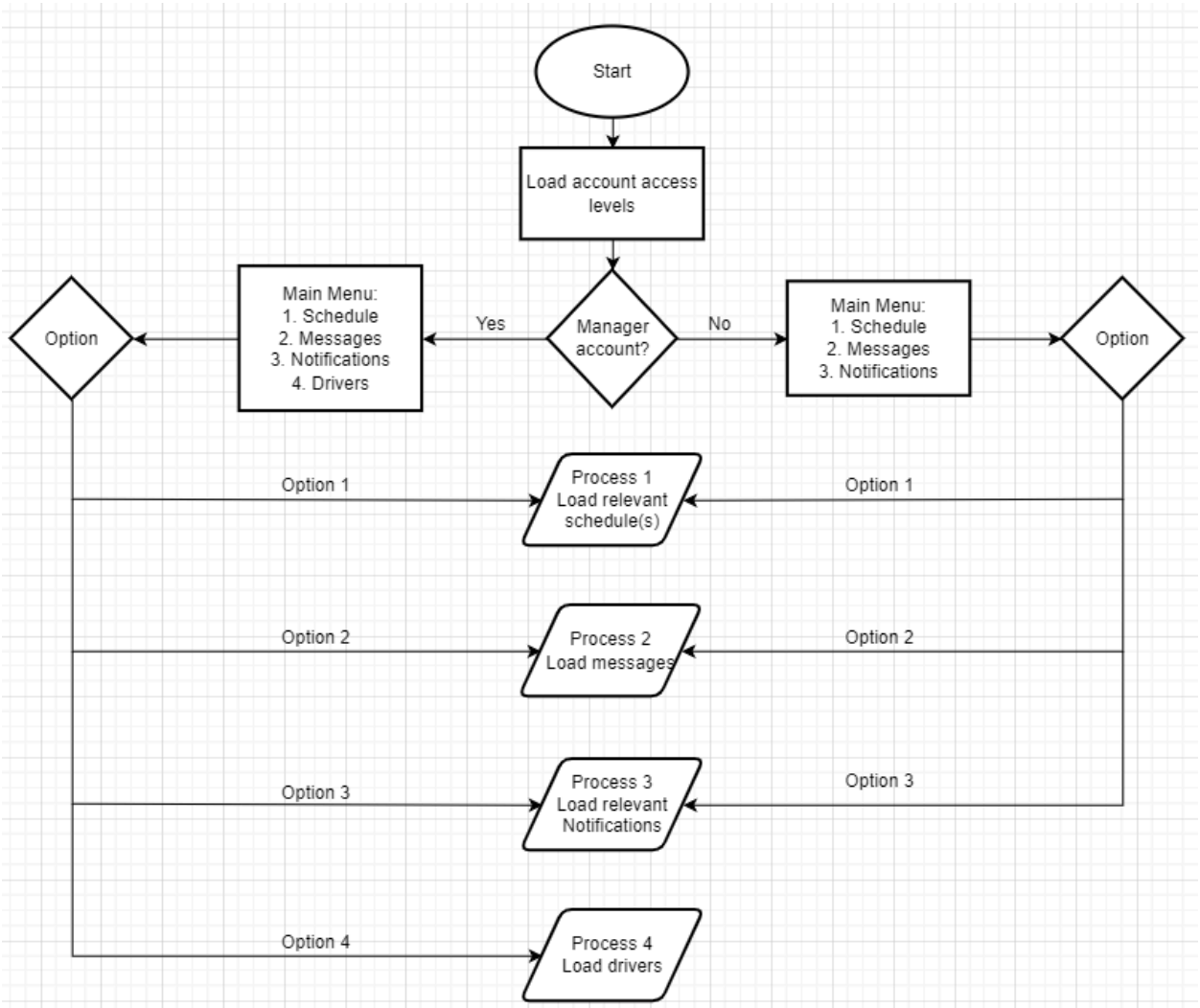
```
{% extends "base.html" %} {% block title %}Home{% endblock %} {% block content%}
```

This line of HTML would be used at the top of the other html templates to extend features of the base template to each web page. This reduces the lines of code and programming time per web page template due to reusability.

Main Menu

The main menu will follow a navbar design on any border of the screen so that it is accessible on every page.

- This means a user can be on one page and be 1 click away from all other pages instead of going to a hub themed main menu which can be time consuming and tedious to navigate while doing multiple tasks on the program



This is the main menus algorithm to show how the main menu will work in redirecting to the other pages. This is necessary and helps to complete the solution by providing the user the means to access the program's features. The main menu options will be buttons and also made accessible on the navbar to reduce the time needed to traverse the program.

Scheduling

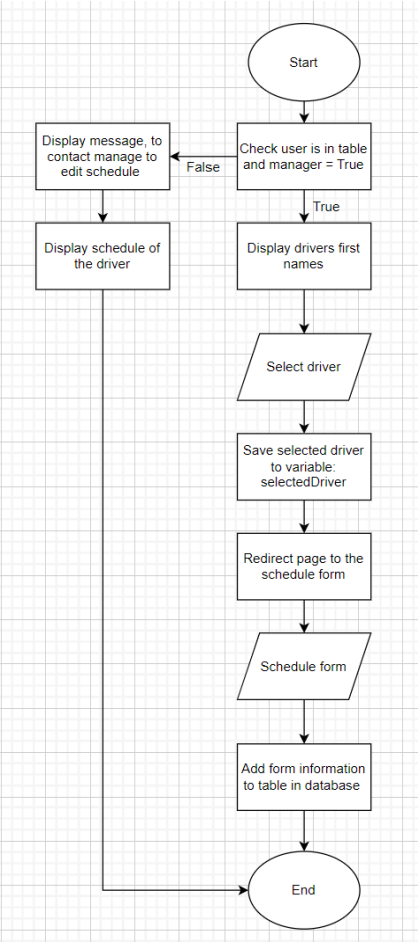
The manager account will have access to all schedules of drivers in its company, however the drivers will only have access to their personal schedule.
This code will have to align with the multiple tables in the database in order to make queries and as a result, display relevant tables or edit relevant tables as a manager.

Validation of user access - pseudocode

```
IF user.manager = True THEN
    FOR Schedule = 0 to length(Schedules)
        OUTPUT Schedule
    next Schedule
ELSE IF user.driver = True THEN
    MySchedule = Query(Schedule) where Schedule.firstname = driver.firstname
    FOR Job = 0 to length(MySchedule)
        OUTPUT Job
    nect Job
ENDIF
```

If the user is a manager then all the schedules are displayed, however if the user is a driver then the Schedule table in the database will be queried for all the records where the drivers firstname is the same as the name in the first name column in the table. Then this goes through a definite loop, to output each job.

Editing schedules

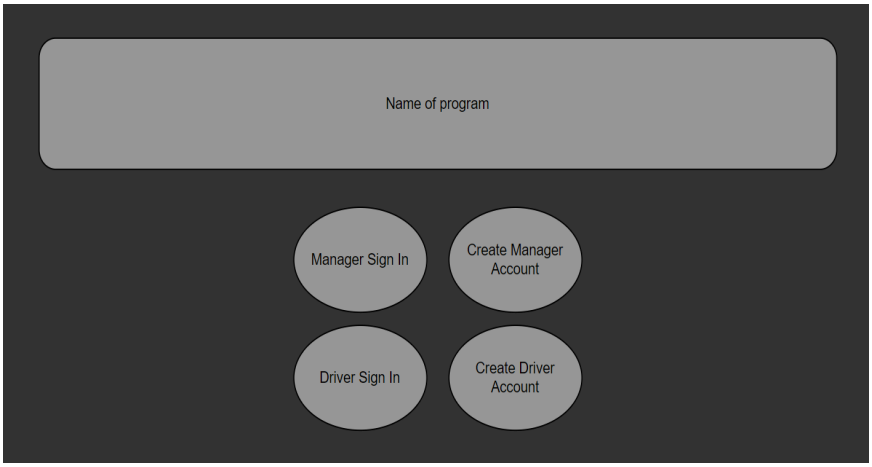


This is the algorithm for the adding schedules. This algorithm has various validation to ensure data integrity. It helps complete the solution as this information will be used later to navigate schedules and make the program easy to use.

This algorithm allows the manager to select who to assign a job to. It helps complete the solution as the information from this will be used later to navigate the schedules and make the program easy to use.

2.6 Usability features

Startup page:

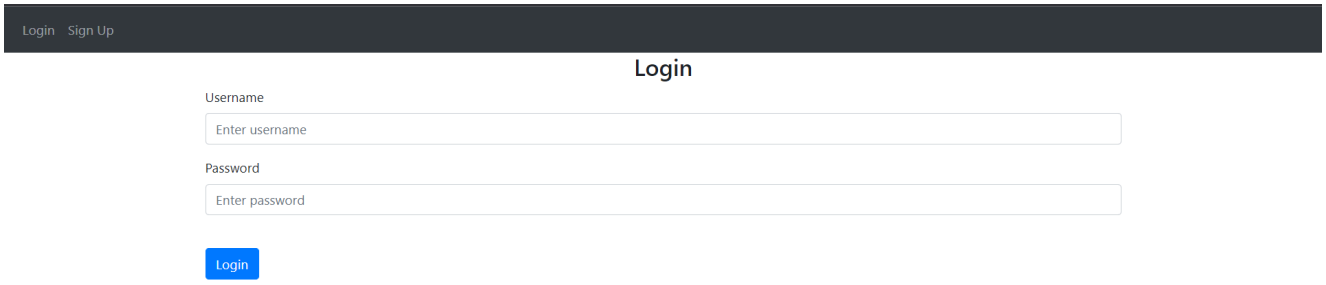


The sign-in options menu will have 4 options due to the two different possible roles in this program, as the manager would have to access all its driver's schedules but the drivers only have to access theirs.

The buttons are positioned in the middle and the log-in page is kept simple because there isn't any other necessary information to have

User Feedback

The main stakeholder is unsure about the shape, font and colour scheme used in this mock up design, however agrees with the general layout.

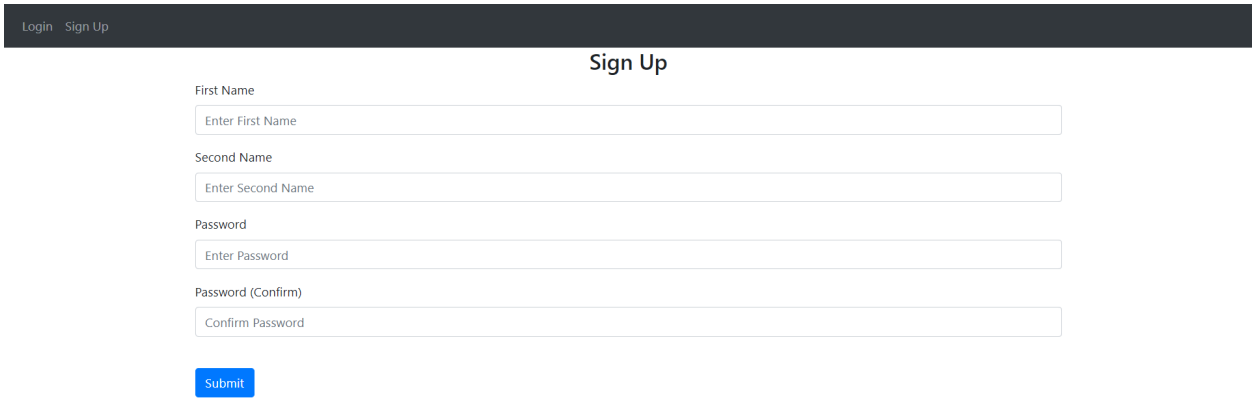


The login page will be kept simple to meet the requirements - an intuitive and easy to understand solution.

This will include a navbar that has the option to go to the Sign Up page. The navbar is used to maintain the same style and format throughout the solution as the main menu once inside an account will be accessible from all pages via navbar
This is an important feature of the program so that users can gain access to account information and driving schedules.

User Feedback

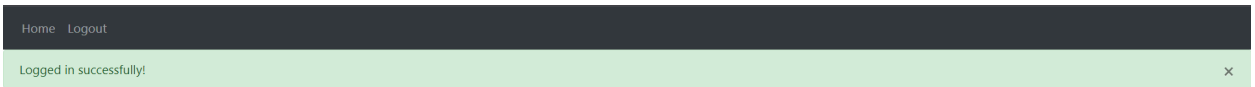
The main stakeholder likes the minimalist design and the navbar incorporation and the layout of the boxes



The Sign Up page will be kept simple and will look similar to these with a navbar that has the option to go to the login page. This is an important feature of the program so that users can add themselves to the database and receive their schedules

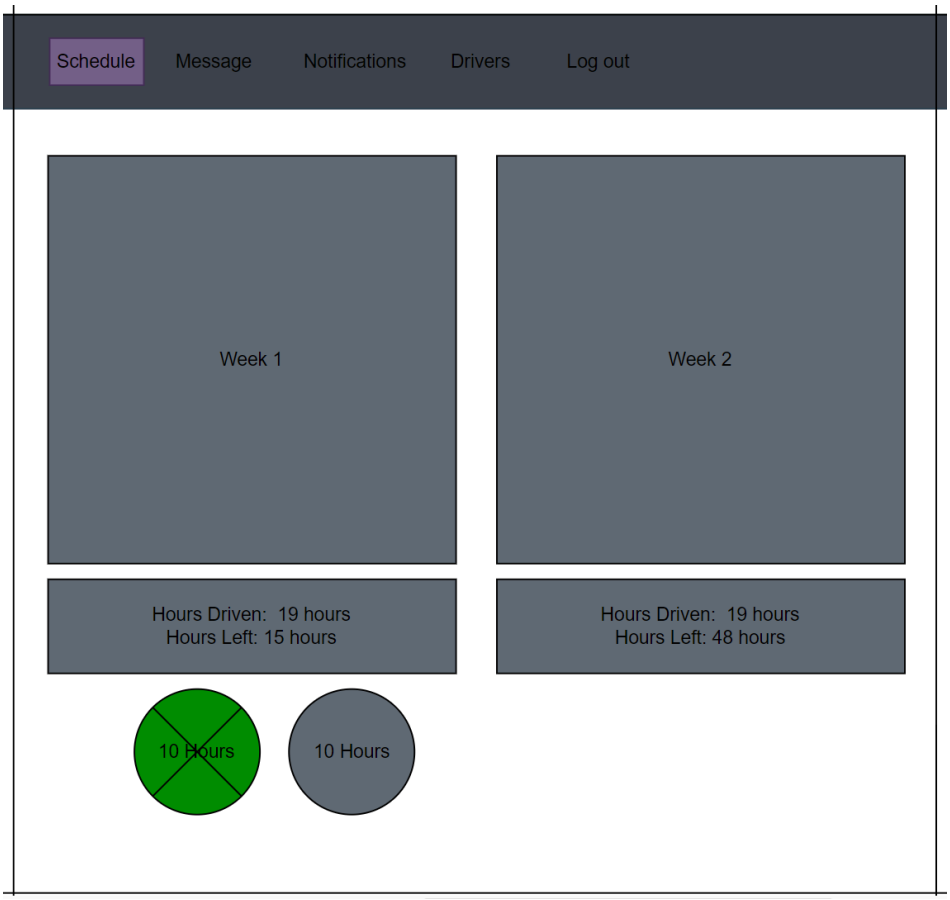
User Feedback

The main stakeholder likes the minimalist design and the navbar incorporation and the layout of the boxes - has requested additional boxes to help create the account: such as option to be manager account or driver account



Logout button at the top that when clicked takes you to the login page again. This is a security feature to maintain the safety of users information when they're done with a session

A message in green pops up after you log in



This is the layout of the main part of the program. The schedule page will only show week 1 and week 2 and will calculate and display the hours driven and left for them two weeks
The circles below are checked if and when the driver uses those two options - it turns green and is crossed out when used

The boxes and colours are not final design, the boxes are place holders for the specific information to drivers.

User Feedback

The main stakeholder is happy with the layout but would change colour scheme
Says the main focus of the page is the week 1 schedule and so should be the only thing displayed at first.
The other bits of information would be an option to view and extend the page.

- This is because the amount driven isn't necessary to drivers as much as the schedule and is just additional information

2.7 Key variables and Objects

Key variables	Data type	Sample Value Example	Validation	Description & Justification
WeekHours	Integer	15	If WeekHours > 56 Then an error message flashes, as driver can only work less than 56 hours in a week	This variable stores the number of hours driven in a week per driver
AVG48	Integer	48	N/A	This is a constant variable with the value of 48 to be used for comparison with the drivers current average hours driven
averageHours48	Float	45	If AVG48 > 48 Then an error message flashes, as drivers musn't drive on average more than 48 hours a week	This variable is needed to store the calculations of the drivers hours worked in a 17 week period to be compared with another variable
TwoWeekHours	Integer	39	If TwoWeekHours > 90 then error message flashes, as drivers musn't work more than 90 hours in 2 consecutive weeks	This variable stores the number of hours driven in the last two consecutive weeks
EXT_HOURS	Integer	1	If EXT_HOURS > 2 then error message as drivers can only extend their hours driven up to 2 times in a week	This variable will be a constant of the number of times a driver can extend their driving limit in a week
MAX90	Integer	90	N/A	This is a constant equal to 90 because the maximum hours allowed to be driven in

				two weeks is 90. Will be used to keep track of possible hours that can be driven in a time period
MAX56	Integer	56	N/A	This is a constant equal to 56 because the maximum hours allowed to be driven in a week is 56. Will be used to keep track of possible hours that can be driven in a week time period
MAX9	Integer	9	N/A	This is a constant equal to 9 because the maximum hours allowed to be driven in a day is 9 hours unless extended to 10 thus will work with variable 'EXTEND_HOURS'
drivenHours	Integer	37	N/A	This will store the hours driven per driver
Name	String	'John Doe'	If Name = "" Driver does not exist	This will store the names of the drivers
TractorID	List	KS14 HRX	If TractorID = "" Truck does not exist	This will store the number plates of the trucks
TripID	Integer	QWE123PL90	If TripID != to Trips(i) Then trips does not exist	This will store the number of the trip ID

Object name	Function	Description and justification
Truck	This is responsible for creating and adding trucks to the system	Allows the business to assign trucks to its drivers and their jobs, preventing clashes
Schedule	This is responsible for displaying the schedules for drivers on a table	Allows a user to see when they are working and where they are to go
Menu	This will hold all the buttons for the menu to navigate through the program	Allows a user to navigate the website and access features such as their schedule
CreateAccount	This is responsible for allowing a user to create an account and add it to the relevant tables in the database	Allows a user to create an account and being able to see their schedule
Login	This will be used to login a user to access the program. It queries the table where the account creations were previously stored, and then compares entered value to the table	Allows a user to log into created account

AddSchedule	This will allow a user to add information on a html form which is then to be saved in a table and queried later to be displayed	Allows a user to make changes to schedule and update it so that the schedule is kept accurate

2.8 Order of development plan

There are 5 modules to be developed

- Creating a web server
- Creating web pages - back end
- Creating the web pages template
- Creating a database
- Messaging module between members of the company

Order	Algorithm/function	Justification of position in the implementation
1	Web server	This if first, because to carry out the development of the other modules a working web server is necessary for iterative testing
2	Web pages	Web pages and the templates could be before or after one another, however due to the nature of the program to calculate and display schedules, this is done second so that all the information is ready to be manipulated in the next stage, making it easier to code
3	Templates	This stage is third as it works on displaying and manipulating the data from the previous stage
4	Database	As the program may change how it operates, the database tables may require column additions or removals or the creation of more unexpected tables. This would be made clear after completion of the second module
5	Messaging	This is last because in 1.3.2 Interview transcript, the manager had said that this feature isn't a completely necessary feature for the completion of the program, and as a result, if challenged for time to complete this program, this module will be unlikely to have been finished

2.9 Testing

Alpha testing

- This focuses on testing the code in the early stages of development of the program. Tests will be carried out per data entry and data calculations or returns while programming a module. More tests will then be done after the completion of a module
- The tests will be looking for syntax errors more than logical errors but may detect some logical errors along the way

White box testing

- This focuses on testing every possible combination, it tests the source code and the internal structure. As the test will be based from the developers point of view I will carry out tests during and after the completion of every module using erroneous boundral and normal data entries.
- The test results will also be record in a table of pre set tests made prior

Beta testing

- This focuses on testing the solution from the end users point of view. As a result beta testing will take place near the end of completion of the program.
- This will involve testing every area of the program

Black box testing

- This focuses on testing on functionality without checking what the code is supposed to do. It will find gaps in the functionality and usability of the program. This form of testing can be done by my stakeholder. Due to the requirements of a black box test involving the tester to understand the requirements specifications of the program a friend can test modules such as creating an account as they will have an understanding of the requirements of logging in and creating an account due to everyday use of account access

Acceptance testing

- This testing will focus on making sure the requirements of the client are met. This can be done during and after the development of the program. I will test that the requirements are met by checking there is a module per requirement. Tests related to the modules will be carried out to ensure they produce the results needed.

2.9.1 Iterative development testing

Test No.	Purpose of test	Justification	Test Data	Expected Result
1	This test is for the web	The program should only be	Running file from	Web server created when

	server function to ensure the program runs correctly	able to run from one python file and not any other to ensure the program starts up correctly	main.py and any other file	run from main.py file. Web server refuse to connect when run from any other file
2	Test for the web pages function A different screen is displayed for the authentication URL for the login page	The program needs to display different information or prompt information to be entered by the user. As the front end of the project is a website design, each page needs a unique identifier to display its contents The login page is required for the program	/login	Two fields for the user to enter their email and password
3	Test for the web pages function A different screen is displayed for the authentication URL for the create account page	The program needs to display different information or prompt information to be entered by the user. As the front end of the project is a website design, each page needs a unique identifier to display its contents The create account page is required for the program	/createaccount	5 fields for the user to enter their first name, second name, email, password, and password verification
4	Test for the web pages function A different screen is displayed for the authentication URL for the log out page This URL is different to the other authentication tests as this requires the program to then redirect the user to the login page	The program needs to display different information or prompt information to be entered by the user. As the front end of the project is a website design, each page needs a unique identifier to display its contents The log out page is required for the program and	/logout	It redirects the user to the login page after
5	Test that the home page with AETR rules, is displayed when the URL is the home page URL	The home page is often the most important page of a website and so this page will have limited information to not overwhelm the cluster of the program on startup. The URL here is an absolute path as it is common practice	/	Page with AETR rules displayed

6	Loading schedules page and validation of schedules	The program must have validation to ensure data integrity throughout in the database	Submitting a job that doesn't have any fields filled out	An error message should be displayed saying what is missing
7	Test to filter jobs by date, driver, location, number plate	The users may need to have a closer look at the schedules and view the jobs for different scenarios and needs	Enter a drivers name to view their schedule	The schedule of one driver should be displayed

2.9.2 Post development testing

Test No.	Purpose of test	Justification	Test Data	Expected Result
1	To test if navbar menu is functional and buttons redirect pages	To access their schedules and see up to date schedules, the buttons on the navbar need to be working. Without this the website becomes unusable	Clicking navbar buttons: 1. Home button 2. Edit schedule button 3. Schedule button 4. Timesheet button 5. Logout button	User is redirected to corresponding pages 1. Home page 2. Add schedule page 3. Page full of schedules 4. Page to enter hours worked 5. Login page
2	For a user to create an account	To access data from the program the user would need a registered account. Schedules are saved together with the relevant account in a table	Enter first name: - John Enter second name: - Doe Enter a email: - JohnDoe@gmail.com Enter a password - 123456789	Account to be created and added to a table, user is then redirected to the home page
3	Logout	The user may want to log out of their account on shared devices. This is an extra security measure. The test is important to make sure no details of previous account can be accessed when not signed in with credentials	Click logout button	Redirected to the login page and unable to access or view any schedules or edit anything else.

4	Login	The user needs to be able to log in to their account to view their schedule saved to that account	Entering email: - JohnDoe@gmail.com Entering password: - 123456789	The user is redirected to the home page and has access to schedule
5	For a user to update their schedule by deleting and adding jobs	Schedule would need to be filled out by someone in the business most likely the manager for their driver. They must be able to inform their drivers their schedules to work	Enter date: - 2/2/2022 Enter time: - 22:22 Enter tractorID: - NW13 AWN Enter location: - Arenson	The job to be displayed on the screen for the user to see the changes made. An x on the jobs so a user can delete them
6	Purpose of this test is for a user to view their schedule	This is important as drivers need to see when they work to maintain organisation as a business	Click schedule	List of all jobs in their schedule

Development

Overview

There are 5 modules to be developed

- Creating a web server
- Creating web pages - back end
- Creating the web pages template
- Creating a database
- Messaging module between members of the company

Through this section of the project I will be coding and testing, validating the parts of the program. It will be separately done in modules with different functions.

The website element framework base layer was coded with the help of the youtube tutorial:
https://www.youtube.com/watch?v=dam0GPOAvVI&t=1292s

The stakeholder has asked for the program to be completed as a functional website which has led me to use the python framework Flask.

- This is a web application framework in python

Iterative development

Developing this program I used Visual Studio Code to write and test the solution.
This proved useful due to its combination of web and language specific technologies

The code was developed on a modular basis and iterative method, with each module generally residing in its own separate file with self-descriptive file names. The benefit of coding a multi-file program keeps files short enough to edit easily, and sharing code with other programs

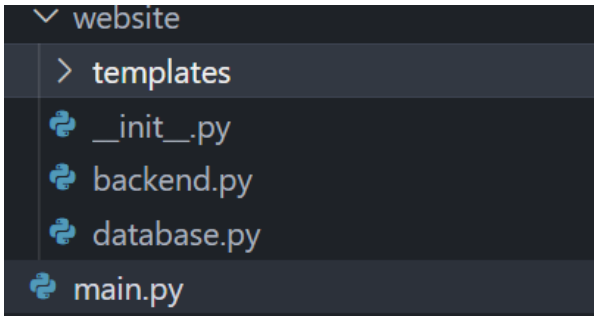
Module 1: Web server
Prototype 1

The python libraries downloaded and used are

- flask
    - Web application framework written in python
- flask-login
    - Provides user session management for Flask
- flask-sqlalchemy
    - Database library for flask
    - It is an extension for Flask adding support for SQLAlchemy applications

To start off I created a file called main.py and a folder called Website which contains a folder called template and the files:

- __init__.py
- backend.py: Contains the authentication modules, such as account related functions and contains the routes of the website, such as a homepage
- database.py: contains the SQL tables used to store account information, schedules etc...



Splitting the program into multiple files will ensure easy readability and makes testing easier as errors are easier to spot
The template folder is used to store the different HTML scripts for each web page

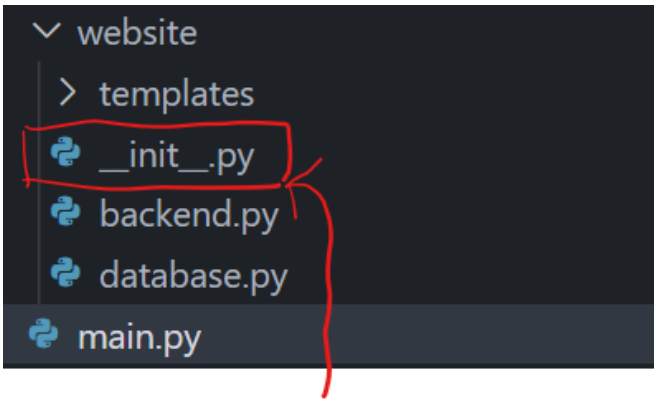
__init__.py

```python
# the name of this file makes the folder 'website' a python package

# setting up the flask application
from flask import Flask

# Creating an instance of the Flask class
def create_app():
    app = Flask(__name__) # initialising the app, __name__ used so that Flask knows where to look for templates etc...
    app.config['SECRET_KEY'] = 'sfibdjkfhb hdfjbs' # This is to sign session cookies for protection

    return app
```

Firstly, an instance of Flask class had to be created - this was done in the file '__init__.py'

```
app.config['SECRET_KEY'] = 'sfibdjkfhb hdfjbs'
```

A secret key is used to sign session cookies for protection against cookie data tampering as shown by the red
The key in this case is in blue and can be anything random or a specific word. The developer needs to ensure an attacker cannot access the value of the key



This file name makes the folder it's in a python package which means it contains several modules that all need to be accessed when run.

- It allows you to import modules from another folder into the project

Justification

The files are organised like this as it follows a common practice by other flask users. This means the code is likely to be understandable to other programmers for maintenance

Reference

Github and Youtube by TechWithTim:
Github: https://github.com/techwithtim/Flask-Web-App-Tutorial/blob/main/website/__init__.py
Youtube: https://www.youtube.com/watch?v=dam0GPOAvVI&t=7354s

```python
1  from website import create_app #importing website package from __init.py__ to create an application and run it
2  # Website folder is a python package so when imported it will run all files in folder
3
4  app = create_app() # Function in the __init__.py file
5
6  # This runs a flask application
7  if __name__ == '__main__': # Only when this file is ran the following line is executed
8      app.run(debug=True) # Starts a web server
9                          # debug=True change in code will auto rerun web server
```

This is code taken from the main.py file and its function is to run the flask program - this is the only file that the program can be ran from due to line 7 as the file is ran as a script and not an imported module

- It is boilerplate code which prevents someone from accidentally invoking the program in another file
- This syntax is used for programs with multiple python files

```python
if __name__ == '__main__':
    app.run(debug=True)
```

__name__ is used to denote the name of the file currently being ran

```python
if __name__ == '__main__':
    app.run(debug=True)
```

__main__ is the value given to a file currently being executed

```python
if __name__ == '__main__':
    app.run(debug=True)
```

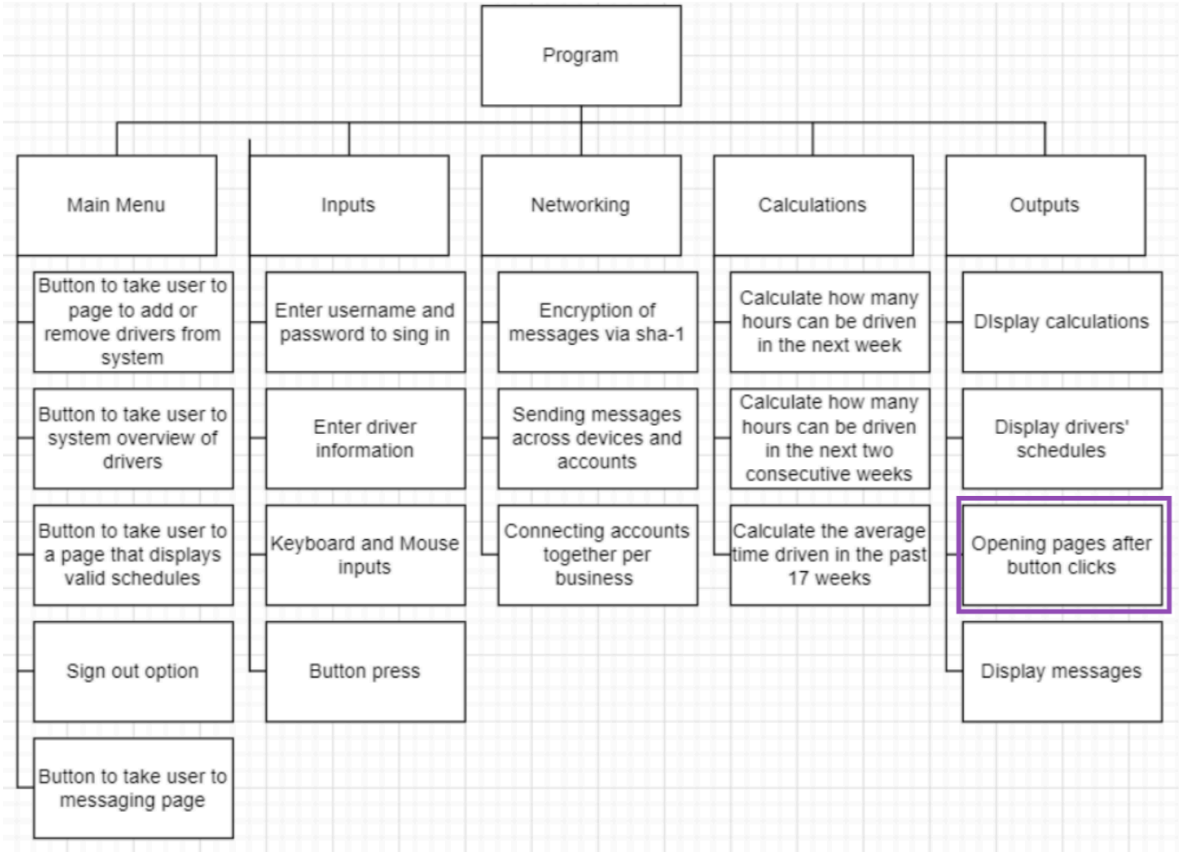
debug is set to True to allow me to locate errors and their locations by logging a traceback of the error. It also allows the web server to update every time a change is made and saved to the code without having to restart the webserver

Validation

The program must be run from the file main.py, or else the program will not run#

```python
if __name__ == '__main__':
```

Relation to top down modular diagram

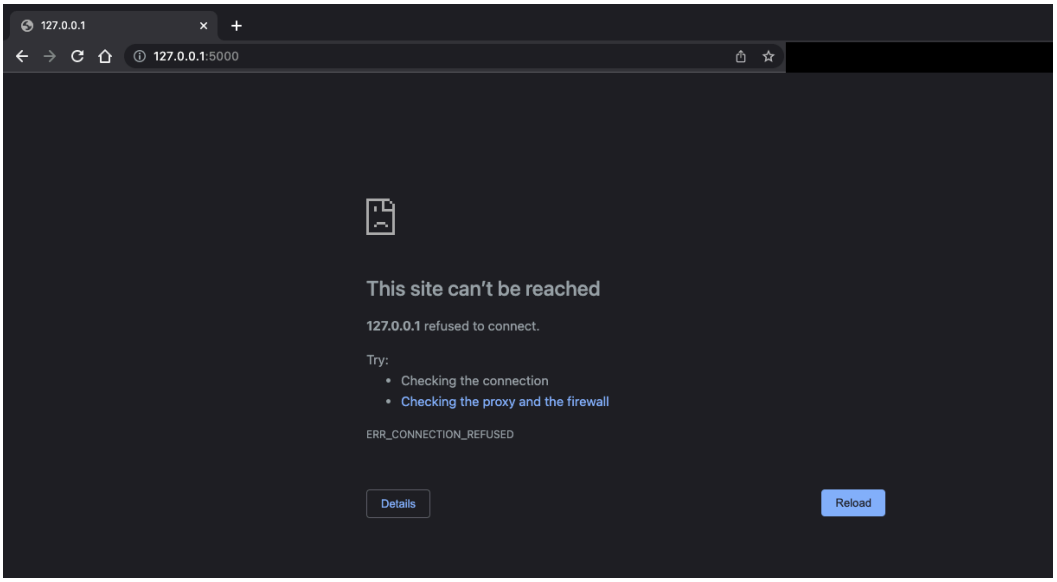


This section's focus comes underneath the output box in the top down modular diagram stated in the Design section: This is because a website runs and displays a web page after the program has been run.

Testing

Creating a web server correctly is important to the success of the project, the test for this section is to see if a web server is created and run. If the test fails then the code would need to be rectified immediately.

Test No.	Explanation and justification	Expected outcome / criteria	Actual outcome	Pass or fail
1	Running the program from the file __init__.py This is to test whether or not a web server can be created when the file, 'main.py' is imported as a module and not run as a script. It is important that the program cannot run like this as it may cause distorted results as the point of entry into the program is unknown and wrong and possibly execute operations on the wrong file and output wrong results.	File not to be executed	File wasn't executed	pass
2	Running the program from the file, main.py This is to test whether or not a web server can be created when running the script in the desired manner - from the 'main.py' script This is important because a web server is the frame of the whole program and interacts with all the other modules.	A web server to be created in the form of a page on a web browser to be ran as a local host and the message: 'Not Found, the requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.' - This is the outcome because no routes have been created in this module to keep simplicity and limit areas of errors	The web page refused to connect	fail
	Evidence: Web server result:			



Terminal result:

```
 * Serving Flask app 'website'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
0.01s - Debugger warning: It seems that frozen modules are being used, which may
0.00s - make the debugger miss breakpoints. Please pass -Xfrozen_modules=off
0.00s - to python to disable frozen modules.
0.00s - Note: Debugging will proceed. Set PYDEVD_DISABLE_FILE_VALIDATION=1 to disable this validation.
No module named main
```

Prototype 2

Remedial action:

```
if __name__ == '__main__':
    app.run(debug=True)
```

The error is due to a boolean value on line 8 where the debug is set to True, the error occurred when on a MacBook laptop

As a result the test was also ran on a windows laptop where it had passed
From this test it is clear that the different operating systems may affect the output of this program

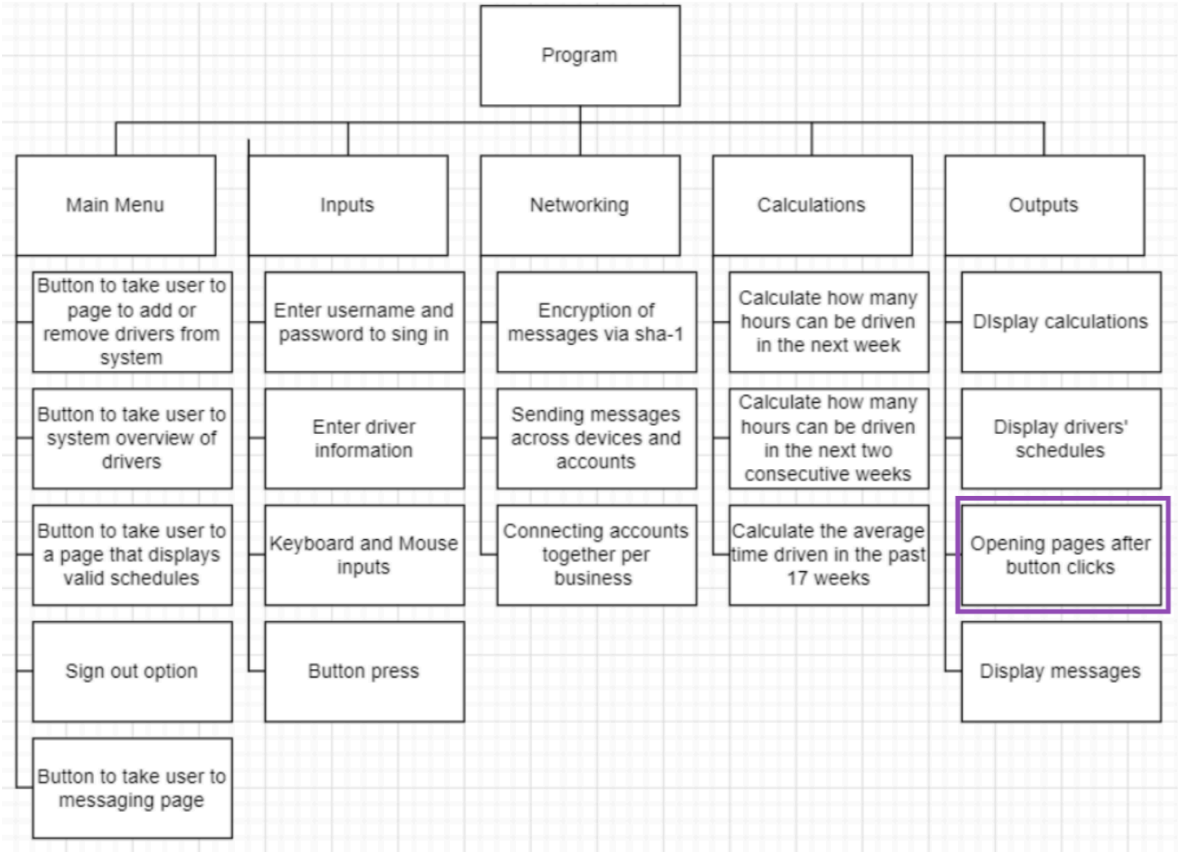
The action that took place to produce a webserver on a MacBook laptop was changing the value of debug to False

```
app.run(debug=False)
```

However this will make development of the program much more difficult as there will be no log of tracebacks.
To conclude the program would continue to be developed but on windows, with the assumption the majority of people accessing this program in the future will be on a windows device

This remedial action was necessary to allow a web server to be made

Relation to top down modular diagram



This section's focus comes underneath the output box in the top down modular diagram stated in the Design section: This is because a website runs and displays a web page after the program has been run.

Testing

Test no.	Explanation and justification	Expected outcome / criteria	Actual outcome	Pass or fail
1	Running the program from the file, main.py This is to test whether or not a web server can be created when running the script in the desired manner - from the 'main.py' script This is important because a web server is the frame of the whole program and interacts with all the other modules.	A web server to be created in the form of a page on a web browser to be ran as a local host and the message: 'Not Found, the requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.' - This is the outcome because no routes have been created in this module to keep simplicity and limit areas of errors	The web page refused to connect	pass

Evidence:

Web server result:

Not Found

The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.

Terminal result:

```
 * Serving Flask app 'website'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

Conclusion of module 1 testing

This module meets requirements and is functional on multiple devices, however the test is variable to change in the future, as the test initially failed due to the specifics of the device used, meaning there was no programming error.

Annotations for maintenance:

```
# the name of this file makes the folder 'Website' a python package
```

This is commented at the top of the __init__.py file to ensure the file name doesn't change as it has special functions due to its name

```
app = create_app() # Function in the __init__.py file
```

States where this function comes from as it is not written in the file it is invoked in - this prevents confusion for what the function is

```
# This runs a flask application
```

Someone new to flask may not know how the program is ran or how to run it properly, this comment eliminates that issue as it explains what part of the code is responsible for running the program

```
#This file is where the program is to be ran from
```

In case a developer is unsure how to run the program from the backend, this comment tells them how to run the program.
All files will have comments at the top of them to explain the use and functionality of it

```
# the name of this file makes the folder 'website' a python package
```

This comment is at the top of the __init__.py file to explain its purpose in the folder for the program

```
app.config['SECRET_KEY'] = 'sfibdjkfhb hdfjbs' # This is to sign session cookies for protection
```

The secret key can be random - this is commented so that future developers know where to find the key and what it means

Module 2: Web pages - backend
Prototype 1

The main file used in this module is backend.py file
This is where the loading of different routes takes place, and in another module will add a style via bootstrap with custom web pages.

To start of with creating the base of all web pages also known as routes, in a file called backend to this module

- backend.py
  - The file handles the routes
  - This is where the functions for authenticity routes will be coded: such as Creating accounts, logging into accounts, logging out of accounts, deleting accounts…
  - It will also be responsible for the the displaying of other web pages

Justification

The program makes use of an extra file, making the maintenance of code for another programmer. It also helps track tests and errors easier.

backend.py

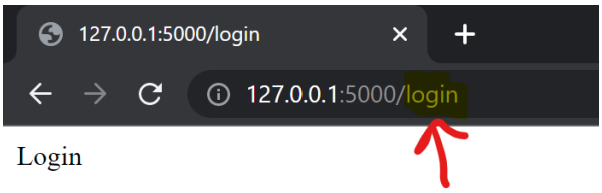
```
from flask import Blueprint
```

To begin with the blueprint library needs to be imported, used to create blueprints - a blueprint defines a collection of views, templates, static files and other elements that can be applied to an application.

Justification

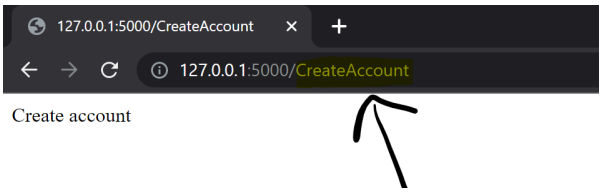
This is used because it allows me to structure and organise the code into subdirectories

```
@backend.route('/login') # creating a decorator, when the URL is /login the function below is invoked
def login():
    return "<p>Login</p>"
```



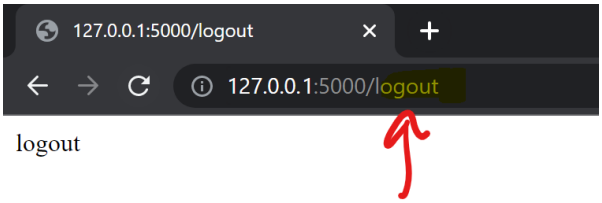
When the path for the URL is /login then the login function is invoked in the backend.py file thus returning a p tag reading login
The code is subject to change but for now acts as passing values that can be used for testing

```python
@backend.route('/CreateAccount') # creating a decorator, when the URL is /CreateAccount the function below is invoked
def CreateAccount():
    return "<p>Create Account</p>"
```



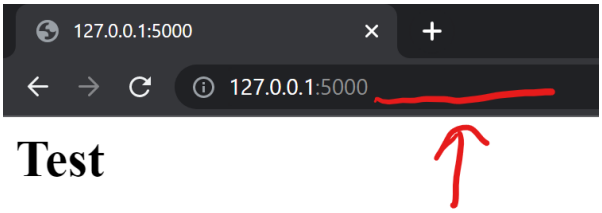
When the path of the URL is /CreateAccount then the CreateAccount function is invoked in the backend.py file thus returning a p tag reading Create Account.

```python
@backend.route('/logout') # creating a decorator, when the URL is /logout the function below is invoked
def logout():
    return "<p>logout</p>"
```



When the path of the URL is /logout then the logout function is invoked in the backend.py file thus returning a p tag reading logout.

```python
@backend.route('/') #defining the hompage route - this is a decorator
def home(): # This function runs when the URL is /
    return "<h1>home</h1>" # along the way testing to ensure the right routes are loaded
```



When there is an absolute URL-path the page loaded would be the homepage, currently represented by a h1 tag reading Test
When there is an absolute-URL path the home function is invoked in the backend.py file

```
return render_template("create_account.html")
```

```
return render_template("login.html")
```

render_template is a flask library used to generate an output from the stated template file - based on the applications templates folder where the html files are stored

These lines of code direct the program to the correct html page to display the relevant content. The html files have bodies of code as the pages have different uses

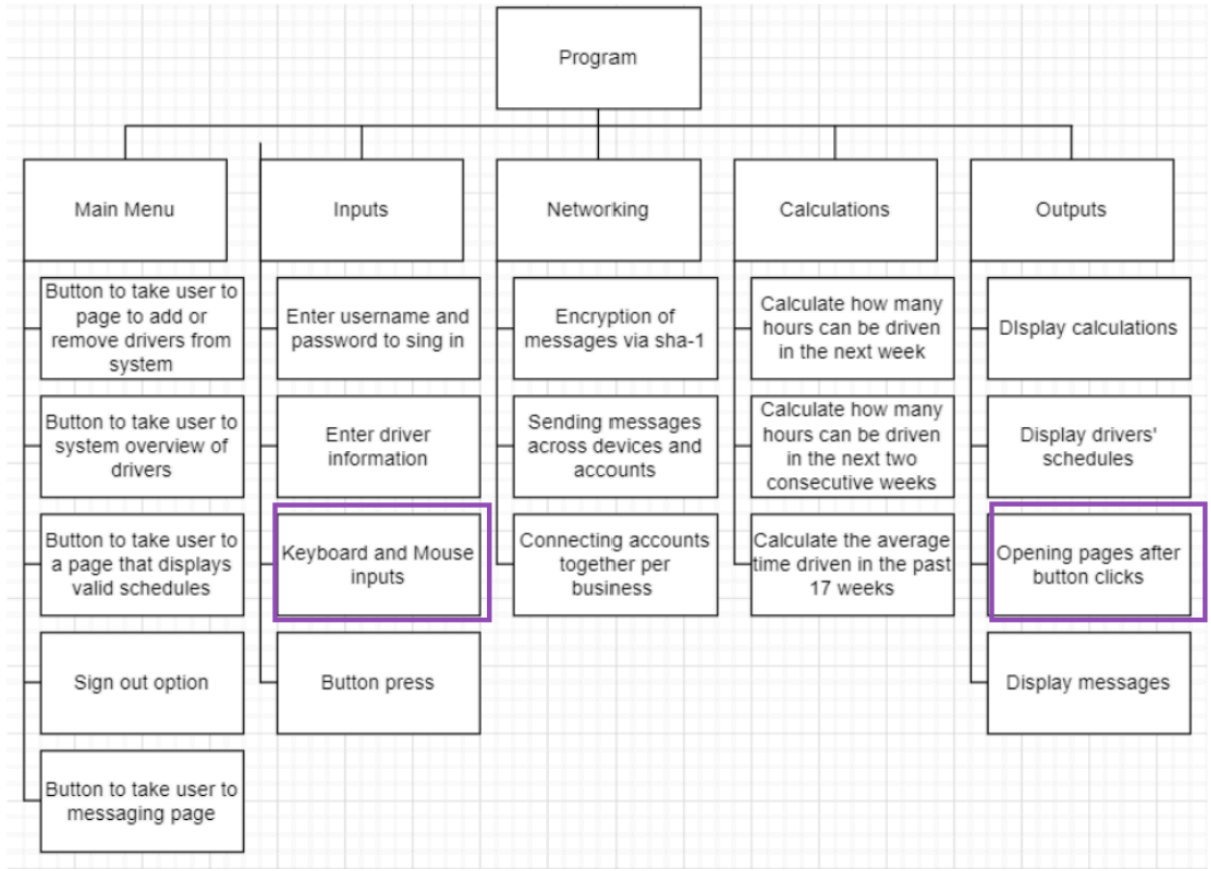
These lines of code will replace the temporary lines used for testing to complete the functionality of the program, such as:

- return “ Login ”
- return”<h1>Test</h1>”

Justification

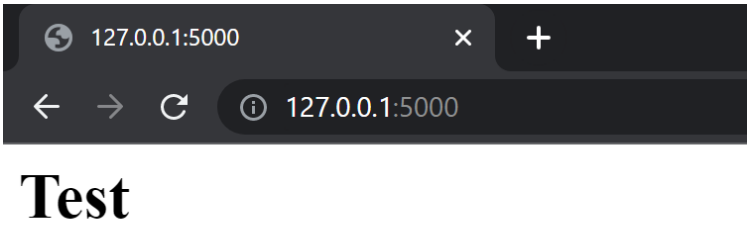
This is needed to render a template which may have static data and placeholders for dynamic data. It renders a template to display on the user's web browser to make the program have a front end for the user to interact with.

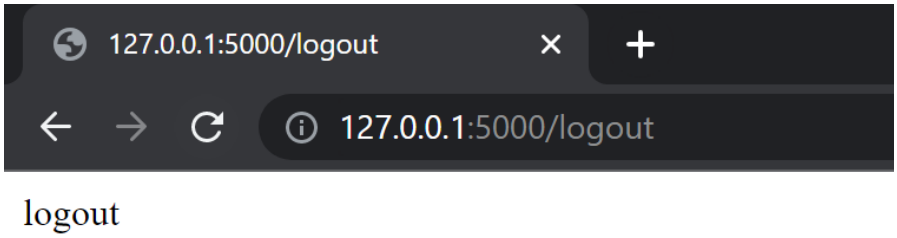
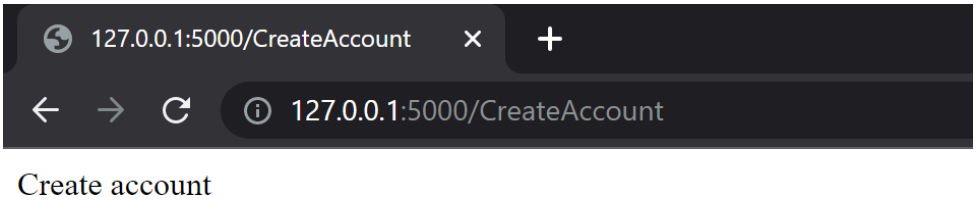
Relation to top down modular diagram

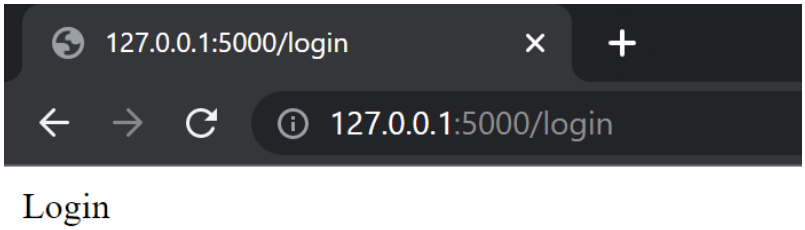


This relates to the top down modular diagram because the URLs return different pages. This is necessary for the programs buttons on the menu to work.

Testing

Test No.	Explanation and justification	Expected outcome / criteria	Actual outcome	Pass or fail
1	To carry out this test, I will run the program. This URL is an absolute URL path so I will not input anything in the URL for the test. When the URL is '127.0.0.1:5000/', the loaded page will return the h1 tag Test. This is set to be the homepage or startup page when the website is run. This is the default URL and is needed to access the rest of the website. This is important for a user to gain access to their schedules and complete their work. The page will not stay as this as the final product, however it must be tested before moving onto the next modules which would expand on this page.	The h1 tag 'Test' will be displayed on the screen	The h1 tag 'Test' was displayed on the screen	Pass
	Evidence:  			
2	To carry out this test I will run the program and enter in the search bar '/logout' after the current URLs domain name and port number. When the URL is '127.0.0.1:5000/logout', the loaded page will return the p tag, 'logout'. The importance of the test is to see whether the logout function is invoked or not when the URL is loaded, because when the function is	The p tag, 'logout' will be displayed on the screen	The p tag 'logout' was displayed on the screen	pass

	fully implemented, we will be unable to see if the function is invoked or not.			
	Evidence: 			
3	To carry out this test I will run the program and manually enter '/CreateAccount'. When the URL is '127.0.0.1:5000/CreateAccount', the create account page should be loaded so that users can join their company's system and gain access to schedules. This is important to allow them to sign into their account later by adding themselves to the database.	The p tag 'Create Account' should be displayed on the screen	The p tag 'Create Account' was displayed on the screen	pass
	Evidence: 			
4	To carry this test out I will run the program and enter the URL path, '/login' This will display a page for users to enter account information to gain access to their schedules and other program features.	The p tag 'Login' will be displayed on the screen	The p tag 'Login' was displayed on the screen	pass

	This is important because the stakeholder has requested security measures and privacy to allow only intended recipients to see their schedules and other information such as direct messages between two people. It is important this test passes so that more code can be written around the invoked login function.			
	Evidence: 			

Prototype 2

This prototype focuses on the views unrelated to authentication. This builds on the previous prototype, by sending multiple sets of information to the html server to display more than the previously used H tags.

backend.py
At the top of the file, I imported date and datetime from datetime. These libraries are used to get the current day.

```
from datetime import date, datetime


weekdays = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]
day = date.today()
dt = datetime.now()
weekdayNumber = dt.weekday()
weekday = weekdays[weekdayNumber]
```

day and weekday are the variables that will be used during throughout the program
These variables have been defined at the start of this file outside of a function so can be accessed by multiple functions instead of repeating code

The weekdays are all in a list because the datetime module does not output the weekday name but a corresponding number, starting at 0 for Monday and ending with 6 for Sunday.

The variable day uses the date function imported to capture the current day outputting the day as YYYY-MM-DD

weekdayNumber is assigned a number between 0 and 6 depending on the day it is.
weekday is assigned the weekdayNumber's value weekday from the weekdays list

- Example: if it is a Friday, weekdayNumber = 4, so weekday = the 4th item in the list weekday indexing at 0 which is Friday

Justification

This provides the user with extra information on screen, in the hopes it may make scheduling easier for them as they have access to a direct cross reference, thus reducing the time manipulating the schedule as the program initially intended to do.

backend.py, home function
This function is the first loaded page when a user signs in. It is the default web page, made clear as it has an absolute URL

```
@backend.route('/') # defining the homepage route - this is a decorator
@login_required # decorator - cannot get to the home page unless logged in
def home(): #this function runs when the URL is /
```

The first line of code in the snippet above is a decorator which is a function that takes in another function as a parameter and then returns a function, returning the function 'home'.

The html file linked with this function displays the AETR rules for the frontend, however this function also serves as another purpose as it is the first page run when the website is loaded.

Its backend process is to increment the week number every monday by 1, as it will then be used to update the week column in the Days and Week tables.

```
96      #Have to get last week_no from Week database and increment that number by 1 every time it is a new monday
97      weekQuery = Week.query.order_by(Week.id.desc()).first()
98      lastMonday = weekQuery.monday_date
99
100     if str(day) != str(lastMonday) and weekday=="Monday":
101         week = weekQuery.week_no
102         week += 1
103         new_week = Week(week_no=week, monday_date=date.today())
104         db.session.add(new_week)
105         db.session.commit()
106
107     return render_template("home.html", user = current_user) #will be able to reference the current user and check if it is authentic
108
```

Line 97:

- To begin with the table Week is queried to find the last record in the table as this will be the most recent week. It orders the table in descending order of id numbers and selects one record which is now the first in the variable.

Reference:

```
weekQuery = Week.query.order_by(Week.id.desc()).first()
```

This line of code was modified from the code in link:
https://stackoverflow.com/questions/49607031/last-record-in-column-sqlalchemy

Line 98:

- Then the variable lastMonday stores the data from the column monday_date of the record previously queried.
- This is used to compare the current day to this day.

Line 100:

- Next the program needs to add the date to the calendar and increment the week number by 1 if it meets certain requirements. Validation is used.

Line 101:

- When the validation is done the program, a variable called week is assigned the week number retrieved from the query of Week table in the database prior.

Line 102 - 105:

- This number is then incremented by one and added to the Week table.

- A variable called new_week creates the python object. Week is the table the record is being added to.

```
new_week = Week(week_no=week, monday_date=date.today())
```

- The circled text are the column names in the Week table, these are assigned values
- monday_date is assigned the current date using the imported library date from datetime

Justification

This function is required for the user to have something on the first screen instead of it being blank. A home page is used on most programs on the web. As mentioned in the design section the program is meant to be intuitive, and to keep this true a home page is required as most users will be affiliated with a home page and its purpose.

Validation

backend.py, home function

```
if str(day) != str(lastMonday) and weekday=="Monday":
```

This line of code is the validation to make sure a date is not added twice in the table and that the only dates to be added to the table are mondays.

To begin with ‘day’ is the variable defined at the top of this file. This is changed to a string. lastMonday is the date of the most recent monday queried from the table. This is also changed to a string so that the values can be compared properly.

day and lastMonday are compared to make sure they are not the same, as this would mean that the date and week has already been added to the table. This prevents duplicates in the table.

The last check is to make sure the day of the week is Monday. ‘weekday’ is a predefined variable that has the current day of the week stored. This bit of validation ensures that the only time the week table can be updated to increment the week number is on a monday. This is because each Monday will be the start of the new week.

Justification

This is needed to ensure the week number is correctly incremented so as to not cause logical errors and make the understanding of schedules difficult resulting in a confusing solution. A confusing solution would be a failed solution as its purpose is to eliminate confusion in the haulage business.

backend.py, edit_schedules function

This function allows the drivers in the table User to be queried and sent to the server for the HTML web pages to manipulate it and display information.

```
110 @backend.route('/EditSchedules', methods=['GET','POST']) # defining the route for the manager to edit the drivers schedules
111 @login_required # decorator so that you cannot access schedules without being logged in
112 def edit_schedules():#when the URL path is /EditSchedules then this function is ran
113
114     driver = User.query.all()
115     if request.method == 'POST':
116         driverSelect = request.form.get('drivers')
117         return redirect(url_for('backend.schedules'))
118
119     weekdays = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]
120     return render_template("edit_schedules.html", user=current_user, drivers=driver, weekdays=weekdays)
```

Line 110:
- A decorator for when the url is ‘EditSchedules’ the edit_schedules function is called and run.
- Methods for this function have been defined; GET and POST
    - This allows users to send HTML form data to the server

Line 111:
- This page is only accessible when the user is logged in

Line 112:
- Defining the function edit_schedules

Line 114:
- Querying all information from the User table and saving to the variable ‘driver’

Line 115:
- When the form is submitted this line is run and its block of code

Line 116:
- The drivers field from the web page form has its submitted value assigned to the variable driverSelect

Line 117:
- After this block of code is ran, the user is redirected to another page

Line 119:
- Is a list of all weekdays to be used by html file to show a drop down menu of weekdays

Line 120:
- This displays the html file content and passes values that are accessed by that html file

Justification

This function is required so that the manager can select who to assign a job to. Without this the schedule would have jobs assigned to no drivers, resulting in a useless schedule. It helps complete the solution as this information will be used later to navigate schedules and make the program easy to use.

backend.py, schedules function
This function is called after the user selects the user to add the job for on the previous web page. This page is only accessed through the edit_schedules function after a redirect from button press.

This is because the driver selected from the previous function will need to be recorded with the new job added

```
136 @backend.route('/AddSchedule', methods=['GET', 'POST'])
137 @login_required
138 def schedules():
139     if request.method == 'POST':
140         date = request.form.get('date')
141         time = request.form.get('time') # .get is used to get a specific attribute from the form
142         tractorID = request.form.get('tractorID')
143         start_location = request.form.get('start_location').capitalize() # capitalize function used to make first letter a capital
144         end_location = request.form.get('end_location').capitalize() # capitalize function used to make first letter a capital
145
```

The web page displays another form for the user to fill out. They are prompted to fill out 5 fields for the jobs.
When the user submits the form, line 139 is run as this is a POST request.

The fields from the form are assigned to a variable as it is needed to add to the Job table

start_location and end_location use the capitalise function to change the first letter to a capital letter and the following letters into lowercase letters as these entries are city names and will make the program more professional.

After the validation, the fields from the form are added to the Job table. This is when the else statement is ran

```
156         else:
157             new_job = Job(date=date, time=time, tractorID=tractorID,
158                           start_location=start_location, end_location=end_location,
159                           user = driverEdit, user_id = driverEditID)
160             db.session.add(new_job) # add user to database
161             db.session.commit() # update the database
162             flash('Schedule updated', category = 'success')
163             return redirect(url_for('backend.view_schedule'))
```

A variable; ‘new_job’ is used to add a record into the table Job
Once the job is added to the table a green message will pop up to notify the user the job has been added correctly to the schedule.

On line 163:

- The user is redirected to another page so that they can see all the schedules and see the schedule they had just added

Justification

This allows the user to query jobs in future and view the schedule. This is needed to complete the functionality of the program as the purpose of it is to manipulate and manage schedules. The manager must be able to add jobs for each driver giving them details on what vehicle to take and where to take it to and from.

Validation

backend.py, schedules function

Before the program can add the job to the table and update it, the program must check all the correct information has been added through validation.

```
### Validation
if date == "":
    flash('Must enter a date', category='error')
elif time == "":
    flash('Must enter a time', category='error')
elif tractorID == "":
    flash('Must enter a tractorID', category='error')
elif start_location == "":
    flash('Must enter the start location', category='error')
elif end_location ==="":
    flash('Must enter the end location', category='error')
```

All of these checks are similar in that they check to make sure the user has added something into each field. If they have not done so then a red error message will pop up on the top of the screen telling them what to add. This is to ensure there is no missing information from each job. It checks that the user hasn't left the fields blank.

Justification

The validation prevents compromisation of data integrity. Data integrity must be maintained as it ensures consistency and traceability, allowing the database to search efficiently

backend.py, view_schedule function

This function is used to display the schedules on a web page.
To do this the records in the Job table must be queried and assigned to a variable which will be passed into the html file to then display its contents.

```
@backend.route('/ViewSchedule', methods=['GET', 'POST'])
@login_required
def view_schedule():

    job = Job.query.all()
    driver = User.query.all()

    if request.method == 'POST':
        global specificDriver
        specificDriver = request.form.get('drivers')
        return redirect(url_for('backend.specific_schedule'))

    return render_template("view_schedule.html", user = current_user, jobs = job, drivers=driver, day=day, weekday=weekday)
```

Next the drivers need to be queried. This is to allow the manager to choose one driver to view only their schedule instead of all the schedules per driver. This is to make the schedules easier to read.

Reference

https://flask-sqlalchemy.palletsprojects.com/en/2.x/queries/

```
global specificDriver
```

If the manager selects a driver to view only their schedule, a POST request is sent. Line 173: is run
In this block of code, specfifcDriver is made global as it will need to be accessed in another function called specific_schedule which is responsible for displaying the schedule of the driver selected.
The driver selected is assigned to the variable 'specificDriver'. The user is then redirected to the page that will display the schedule of the driver selected

Justification

The jobs are ordered by date because in the table they are not in order of date but in the order they were added. This would result in jobs being displayed in what seems to be a random order. By sorting them out in order of date, order is made and scanning over the schedule for a particular job is made easier.

The manager has the option to select a single driver's schedule to view, and this is an important feature as a manager may have hundreds of different drivers and as a result thousands of jobs, making it impossible to find the schedule of a specific driver.

backend.py, specific_schedule function
The purpose of this function is to send the relevant records from the database to the HTML file to then display it.

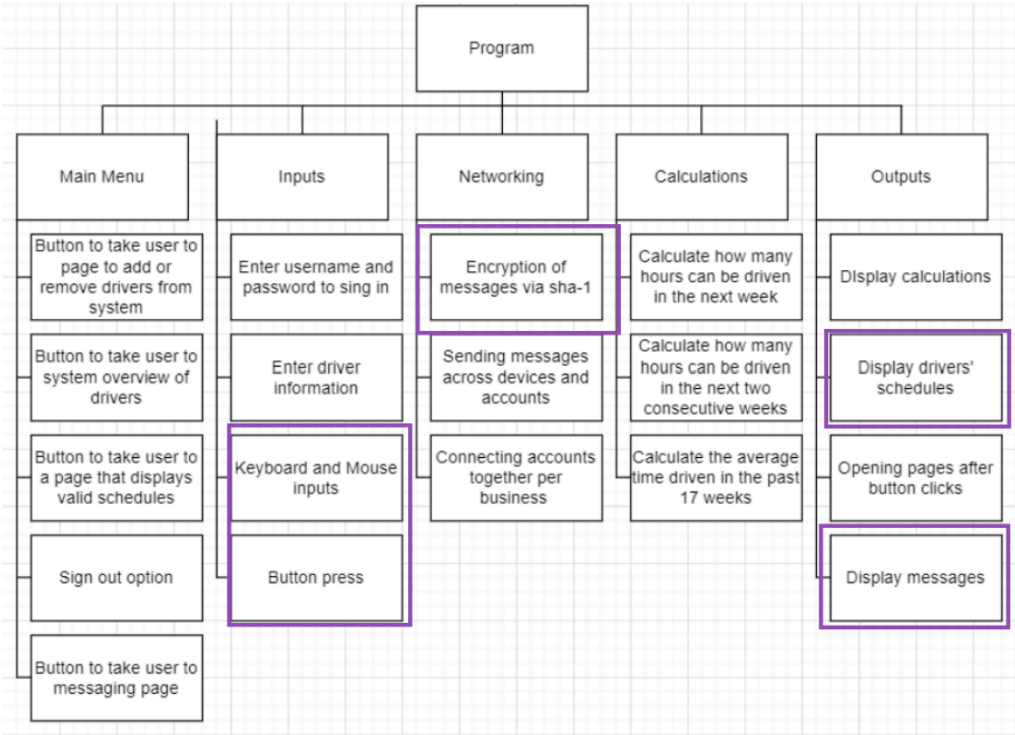
```
@backend.route('/DriverSchedule', methods=['GET', 'POST'])
@login_required
def specific_schedule():

    job = Job.query.order_by(Job.date).filter_by(user=specificDriver).all()

    return render_template("specific_schedule.html", user=current_user, jobs = job, specificDriver = specificDriver, day=day, weekday=weekday)
```

This function queries the Job table, selecting every record where the first_name column has the same value as the variable specificDriver, which has gotten its value from the previous function in its POST request.

Justification

This function is needed to allow a manager to filter through all the jobs and easily find a job in someone's schedule. Filters are widely used as they are helpful and reduce time taken to navigate through extended blocks of information

Relation to top down modular design



This relates to the diagram because when a button is pressed, it redirects a user to a different URL, once a certain URL is sent to the server, it invokes a relevant function.
The passwords are also hashed and encrypted using sha-1
Drivers schedules are queried in this module to allow them to be displayed in the next module.
The validation functions also display messages on screen.

Testing

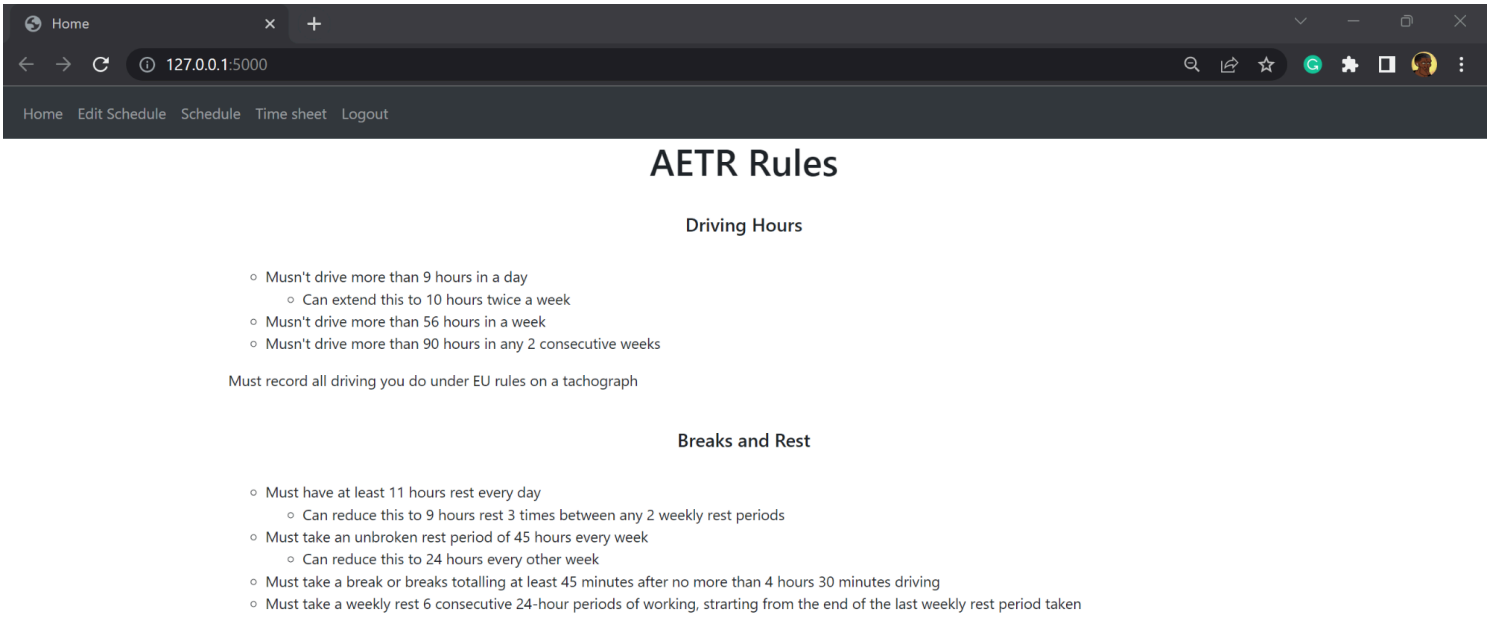
Test No.	Explanation and justification	Expected outcome / criteria	Actual outcome	Pass or fail
1	This test checks whether the home page is loaded when the user signs into an account	Page that states the AETR laws	A page is loaded showing the AETR laws	pass
	Evidence: 			
2	Test to make sure the week number is incremented correctly and stored correctly	Week number will increment by one because the day of testing is monday	The week number incremented by 1 and was added to the table	pass
	Evidence:			

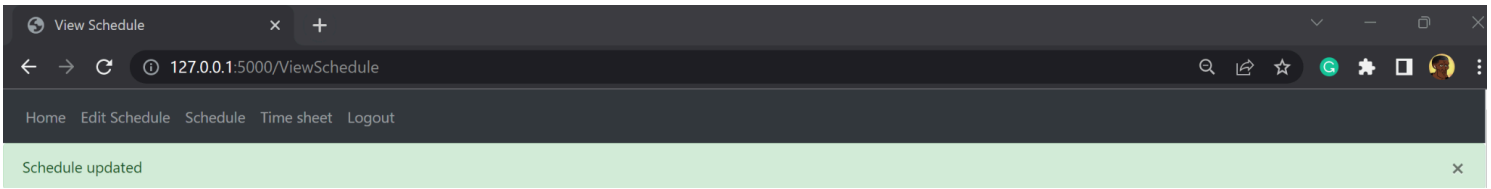
Table: week

	id	week_no	monday_date
	Filter	Filter	Filter
1	1	1	2023-02-20
2	2	2	2023-02-27
3	3	3	2023-03-03

3	Test to see if a new record for the table jobs is added	A new job is added to the table	A new job is added to the table	pass

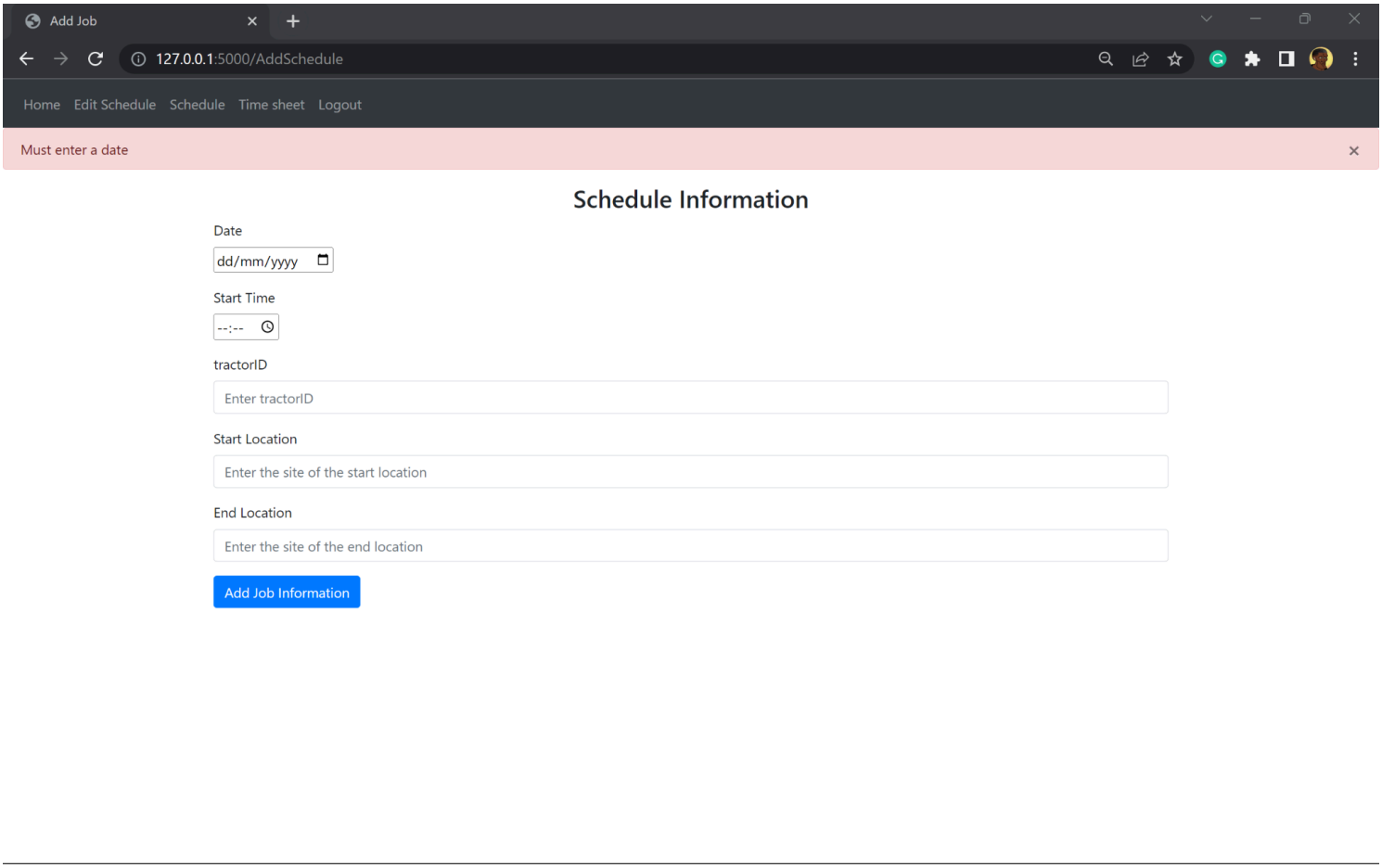
Evidence

9	9	2023-02-26	13:00	WE23 KBC	Northampton	Arenson	Austin	7

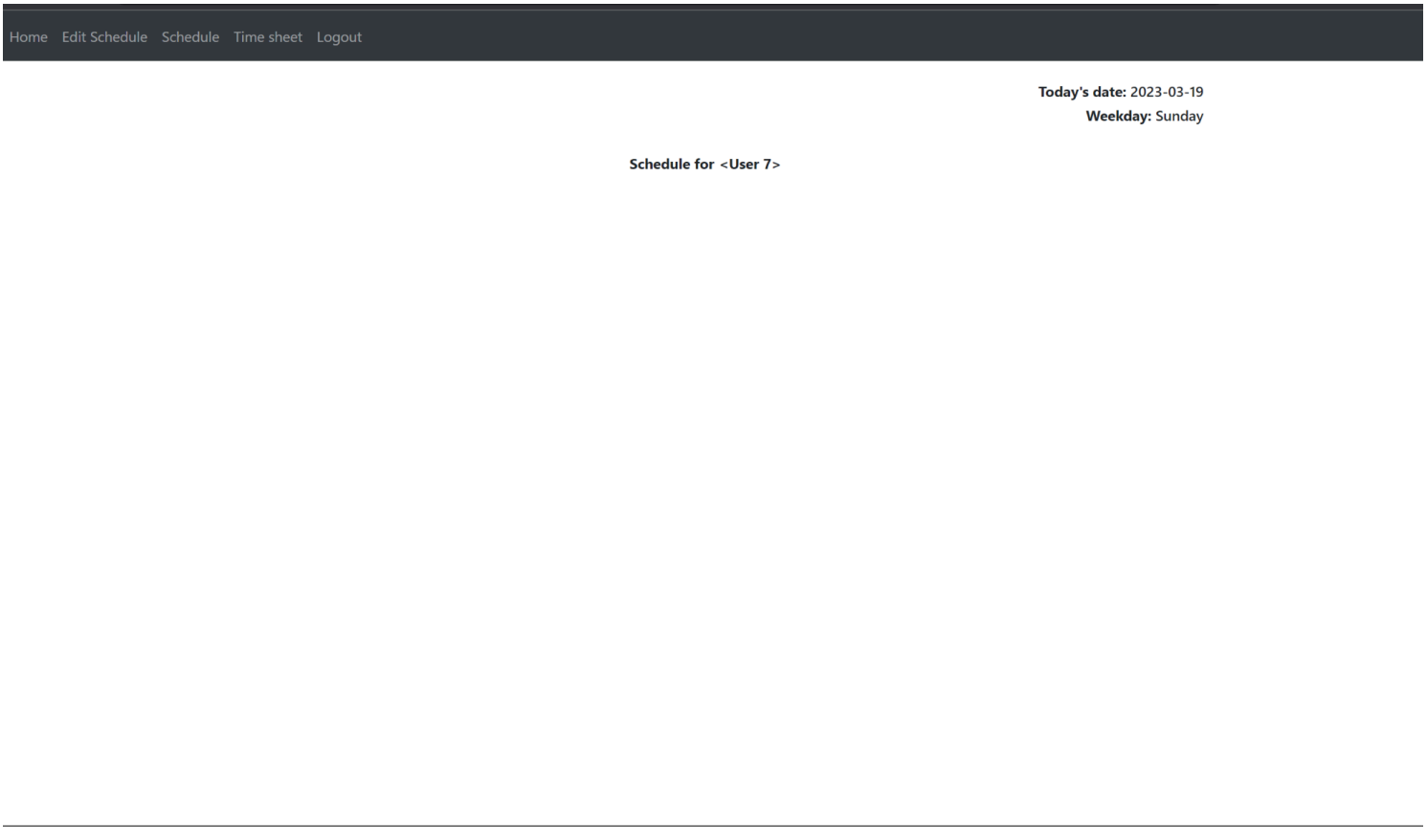


4	To make sure the validation of the the job being added works This will be carried out by not entering any information for the form to add a schedule	An error message should be displayed	An error message was displayed	pass

Evidence:



5	Test if the user can filter the schedules by name	The schedule for a single driver selected will be displayed	The record identifier was displayed instead of the driver's name. As a result no jobs were shown as this identifier could not query the table	fail
	Evidence:			



Conclusion of module 2, prototype 2 testing

The user is unable to filter viewings of the schedule by driver. This has occurred because of different formatting from receiving the form from the server to processing it, and querying the job table from this form. As a result no schedule is displayed.

Stakeholder feedback

The URL should have the driver's name inside it. But on the whole the layout of the website looks great.
The schedule itself is in a random order, making it hard to look for schedules by day

Prototype 3

It also fixes the errors of getting the driver name from the form in the view_schedule and specific schedule function. The schedules are also in a sorted order instead of an unordered list.

backend.py, view_schedule function

Remedial action:
 	
Here the records in the Job table are sorted by the date column from the earliest date to latest date
`job = Job.query.order_by(Job.date).all()`
 	
In prototype 2 the incorrect format was gotten from the form causing confusion and hard to understand outputs.
This block of code fixes that issue.
The issue:
- If the user selects Austin it returns <User 7> and not Austin
 	
`<User 7>`
`7>`
`7`
 	
`if request.method == 'POST':`
`global specificDriver`
`specificDriver = request.form.get('drivers')`
`sliceSpecificDriver = specificDriver [6:] # slices the variable value`
`sliceSpecificDriver = sliceSpecificDriver[:1] #slices the variable again`
`specificDriver = User.query.filter_by(id=sliceSpecificDriver).first()`
`specificDriver = specificDriver.first_name`
`return redirect(url_for('backend.specific_schedule', specificDriver=specificDriver))`
 	
To begin with, specificDriver is globalised so that it can be accessed in the specific_schedule function
`global specificDriver`
 	
The information entered in the form on the html page is assigned to the variable; 'specificDriver'.
The value is sliced so that the it has the the number and the following characters
This new value is sliced again so it only has the number and removes the following characters
 	
`sliceSpecificDriver = specificDriver [6:] # slices the variable value`
`sliceSpecificDriver = sliceSpecificDriver[:1] #slices the variable again`
 	
Now that only the number is assigned to the variable, the User table is queried where the id column has a record the same as the variable sliceSpecificDriver

```
specificDriver = User.query.filter_by(id=sliceSpecificDriver).first()
specificDriver = specificDriver.first_name
```

Justification

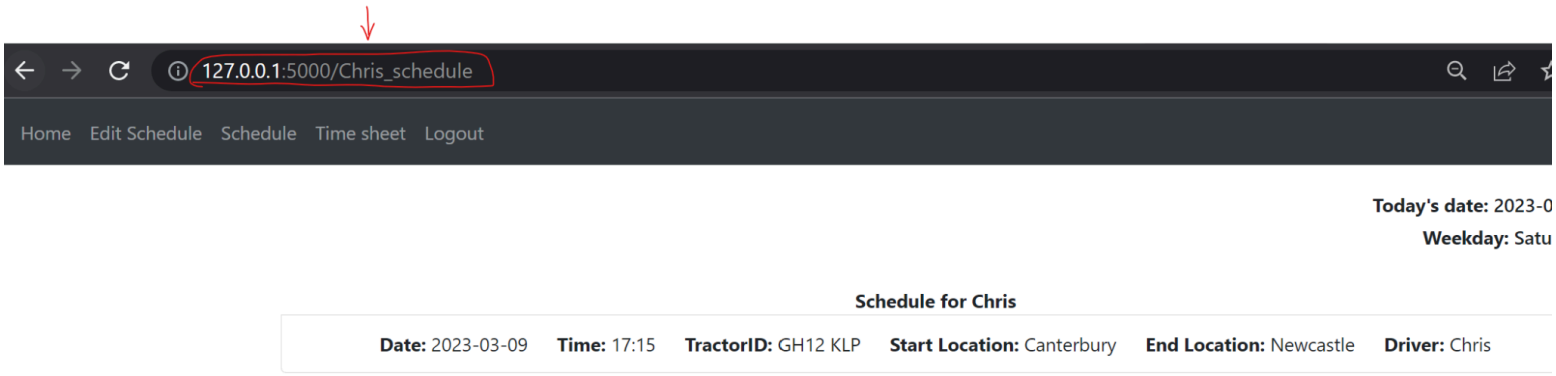
These extra lines of code are needed because it allows the program to know who was picked in the form. The information from the form is received in a different format than needed. For example picking Devin on the form and getting it from the form would be as <User 3>, and not Devin.
This block of code takes the number 3 from the value received by slicing it twice, it then queries the table for the record with that id number, this is because the id number will always be the same number.

backend.py, specific_schedule function

A dynamic URL is used here, where the URL for the same html file can be different. This is done by encasing a variable in < and >. Here the variable is a specificDriver, which is passed in the redirect from view_schedule function above.

```
@backend.route('/<specificDriver>/schedule', methods=['GET', 'POST'])
@login_required
def specific_schedule(specificDriver):

    job = Job.query.order_by(Job.date).filter_by(user=specificDriver).all()
```

The URL is made of the variable and also schedule. This is so the URL is able to explain what the page is showing



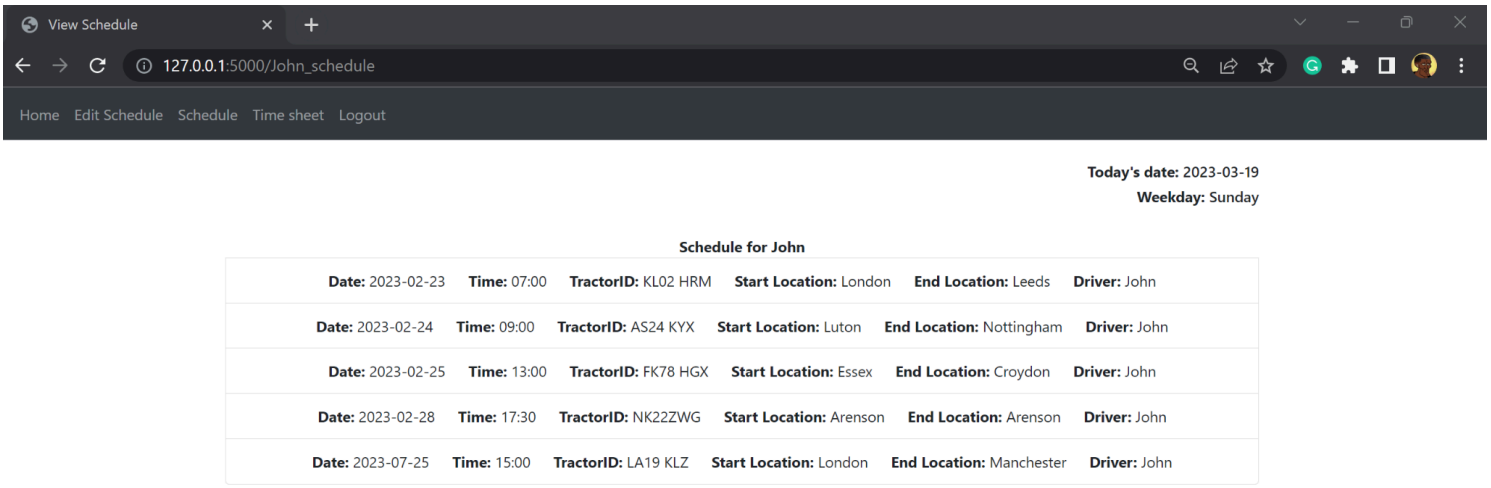
Justification

This remedial action took place because it allows the program to be more self explanatory to all users.

Testing

Test No.	Explanation and justification	Expected outcome / criteria	Actual outcome	Pass or fail
1	Test if the user can filter schedule by driver and redirect user to a page which has the URL displaying the driver selected	The schedule of one driver is displayed on a redirected page, with the URL containing their name	The schedule of one driver is displayed on a redirected page, with the URL containing their name	Pass

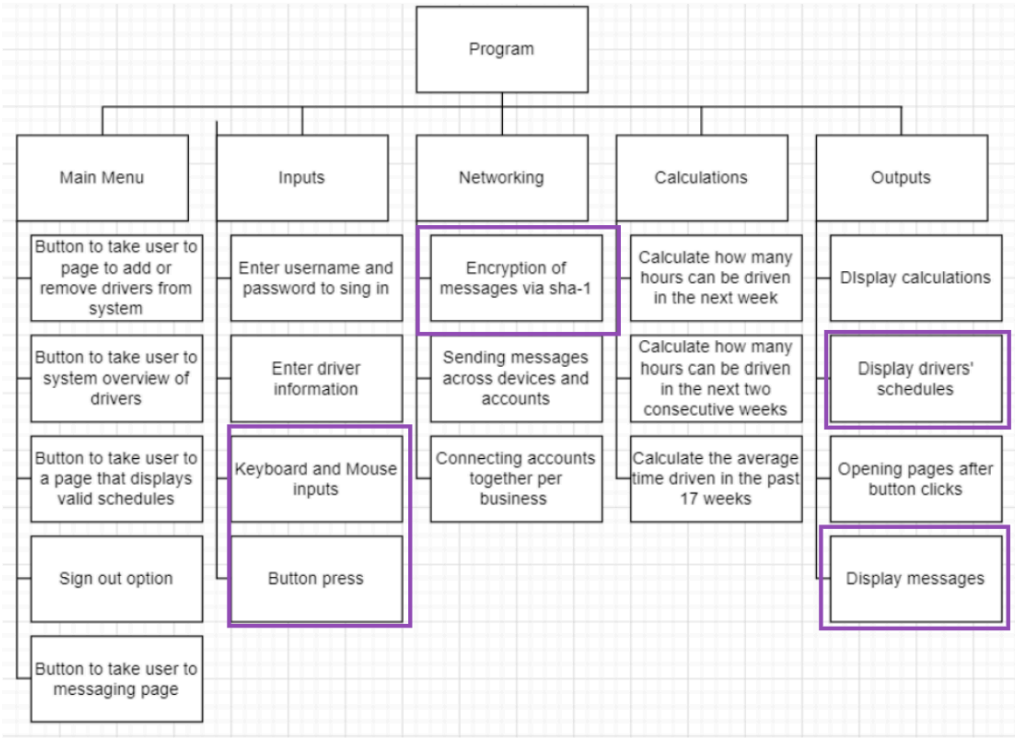
Evidence:



Conclusion of module 2:

All the tests meet the passing criteria making the program fit to move onto the next module. The testing allowed me to see how different functions were invoked giving an insight to where code should and shouldn't be written and formatted. Such as creating html files per page instead of reusing one - this would mean a base html file will need to be made to have child files extend from this file to make coding easier.

Relation to top down modular design



This relates to the diagram because when a button is pressed, it redirects a user to a different URL, once a certain URL is sent to the server, it invokes a relevant function.
The passwords are also hashed and encrypted using sha-1
Drivers schedules are queried in this module to allow them to be displayed in the next module.
The validation functions also display messages on screen.

Comments for maintenance:

```
# Shows how to access all the URLs from the corresponding blueprints
app.register_blueprint(backend, url_prefix='/') #sets the prefix of a URL to /
```

Comment to explain what new syntax means - makes it so a new coder is able to register more blueprints following the syntax

It also explains what the line does, in this example it sets the url_prefix to make an absolute-URL path so that whenever a new route is made repetition is not needed for the each URL as they all start the same

```
# This file is a blueprint of the program, containing multiple routes inside it
```

Comment in the backend.py file to explain the purpose of it

```
@backend.route('/') #defining the hompage route - this is a decorator
def home(): # This function runs when the URL is /
    return "<h1>home</h1>" # along the way testing to ensure the right routes are loaded
```

A decorator is a function that takes in another function as a parameter and then returns a function
This comment is to help a coder understand the purpose of the @ in the case they don't understand this syntax
Another comment explaining when and how the function 'home' is called on and ran as it isn't called on by writing a line of code saying 'home()'

```
backend = Blueprint('backend',__name__) #Defining the name of the blueprint
```

This comment shows how a Blueprint is created

Module 3: HTML & Authentication Templates
Prototype 1

The majority of this modules code will be written in the templates folder

- The templates folder will have multiple html files which represent different pages of this website

When coding the HTML, commenting was an issue as it was read as code when the program was run. To resolve this issue copies of the HTML files were made with annotations. These files aren't called on but are involved in the program to help with the maintenance of the code in the future.

The base.html file is used to act as the general theme of the website. Other child html files will inherit the base template and use this as its framework and override parts.

- Any block that is defined in the base template can be overwritten and changed by a child template

Bootstrap: A CSS framework that has built in classes to help style this website

- CDN (Content delivery network) - load without downloading the file some custom CSS and JavaScript

base.html

The base.html is created to set the theme of the website and other pages. Other html files will use this file to override blocks of this code to output unique pages.

<head> element

This is the head section of the html file. This is where metadata is defined.

```
4    <meta charset="utf-8" />
5    <meta name="viewport" content="width=device-width, initial-scale=1" />
```

The character set used for this document is defined on line 4. UTF-8 is UCS Transformation Format 8 and means each character is represented by 1 to 4 bytes.

Line 5 sets the page size to take up the entire width of the screen the program is being accessed on.
initial-scale is equal to 1 as this zoom level is most common among other websites, which would help meet the requirement of intuitiveness as it has a natural feel.

In the head section of this file, there are also two link elements for the bootstrap and stylesheets.

- It allows me to use different classes to help format different parts of the program.

```
<title>{% block title %}Home{% endblock %}</title>
```

The final bit of code in this head element was the title. This is defining a block for a title. Blocks are parts of the base file that are able to be overwritten by its child files. Code that isn't in a block will be the same for the child html files that extend this one, and code written inside the blocks may vary from child file.

Justification

The title needs to change per different web page, so that the user knows what page they are on to better navigate the website. So having this in a block allows for it to be changed by html files that extend this one.
The design of the website takes into consideration multiple devices with a wide range of screen sizes and accessibility; this would result in maximising the number of users who can access the program. It increases the flexibility of this program.
The charset used is because it can represent ASCII text and allow international characters. It is one of the most popular encoding systems, thus other programmers will have a better understanding of the program as they are likely to also use the same charset.

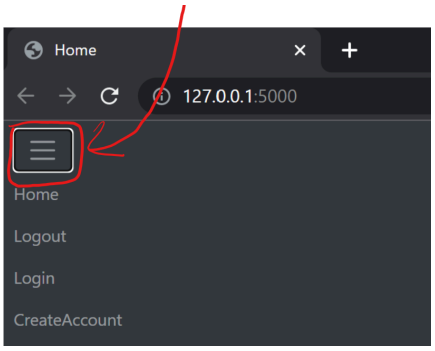
<body> element

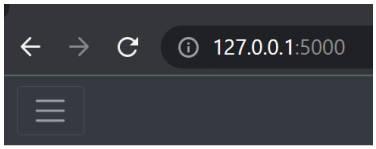
This is the main body of the html file which contains all the content of the HTML file and is to be displayed on the screen for a user to see.

```
21  <body>
22    <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
23      <button
24        class="navbar-toggler"
25        type="button"
26        data-toggle="collapse"
27        data-target="#navbar"
28      >
29        <span class="navbar-toggler-icon"></span>
30      </button>
31      <div class="collapse navbar-collapse" id="navbar">
32        <div class="navbar-nav">
33          <a class="nav-item nav-link" id="home" href="/">Home</a>
34          <a class="nav-item nav-link" id="logout" href="/logout">Logout</a>
35
36          <a class="nav-item nav-link" id="login" href="/login">Login</a>
37          <a class="nav-item nav-link" id="CreateAccount" href="/CreateAccount">Create Account</a>
38
39        </div>
40      </div>
41    </nav>
```

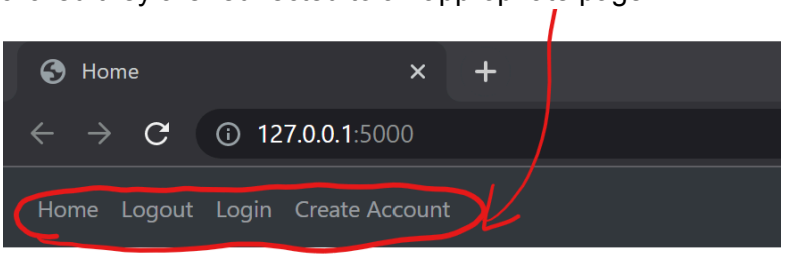
To begin with a navigation bar is created, aka navbar. The purpose of the navbar is to have the menu accessible on all pages eliminating the need for a main menu.

Line 22 uses a class from the bootstrap which can be found on the bootstrap website, this is what gives the design of the navbar.
Multiple buttons are created on the navbar. The first button is a button only accessible when the screen is not adjusted to full screen or if not all buttons on the navbar can be accessed. This button will collapse or expand the menu in a list format, as seen below:





The next buttons are the buttons that redirect the user to another page. There are 4 buttons and when the buttons are clicked they are redirected to an appropriate page.



`class="nav-item nav-link"` the class used here, tells the computer that when the button is clicked it will redirect

the user to something else

```
href="/logout"
```

This is responsible for redirecting the user to the correct page. The value equal to the href is the path in the URL. /logout is the path here, and when redirected to this page the correct function created in module 2 is run. In this example, the decorator is found in the backend.py file:

```
@backend.route('/logout')
```

```
<a class="nav-item nav-link" id="logout" href="/logout">Logout</a>
```

This is what the user sees for the button

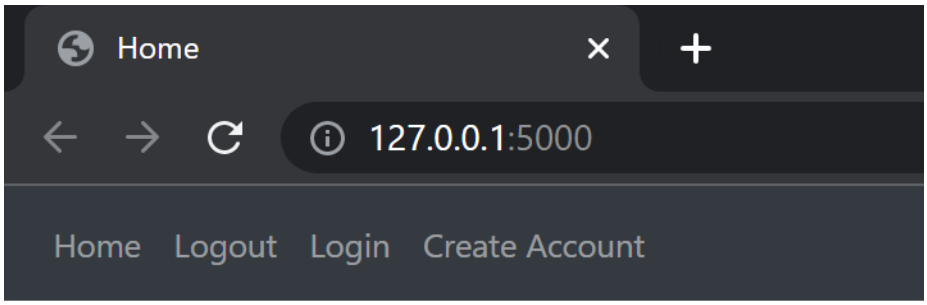
Justification

A navbar is used instead of a page with the main menu, it allows users to go directly to a page instead of needing to use a medium before doing so. This reduces the time and clicks needed to use the website.
Navbars are also a modern approach to websites now, and are commonly used by the most popular websites. Users are likely to use a navbar everyday or have used one before, so using something most users are comfortable with ensures an intuitive program.

home.html

The home file is created to be the first screen the user sees when loading the website. This is set to be information behind the company, AETR laws, and other information surround hauling

```
{% extends "base.html" %} {% block title %}Home{% endblock %}
```



As shown the navbar with the options to go to different pages of the website
The title name has been changed to Home' to show that the blocks are working correctly and are able to be overwritten

create_account.html

In this file, similar to the home file, I had to extend the base file to maintain the theme of the website

```
% extends "base.html" %} {% block title %}Create Account{% endblock %}
```

The first block overwritten was the title block which was changed to Create Account

```
% extends "base.html" %} {% block title %}Create Account{% endblock %}
```

Next I had to build the HTML form, which is where the user will enter the relevant information to create an account. This is an example of a POST request. This method sends data to the server.

```
<form method="POST">
    <h3 align="center"> Create Account</h3>
```

After the method is declared, a heading for the page was added to help with the surfing of the website using a h3 tag

Next input boxes were created for the user to enter their First Name, Second Name, email and password.

This is a bootstrap class for the design of the boxes

```
<div class = "form-group">
```

Next the fields of the form were added, this is needed so that the correct information can be recorded and saved to the correct variable in the database. To help the user fill this form out correctly, place holders were also used to describe what is to be entered in each box.

The validation of this form lies within having an extra field that the user has to fill. This is the confirm password field. This input will be checked against the password field to make sure the password was entered correctly twice, reducing the chance of typos.
All the fields had the same code with different variable names, placeholders and identifiers, however the email also had a different 'type' from the rest. The type here was email while the others were text.

```
20  <div class = "form-group">
21          <label for="second_name">Second Name</label>
22          <input
23              type="text"
24              class="form-control"
25              id ="second_name"
26              name="second_name"
27              placeholder="Enter Second Name"
28          />
29  </div>
30
31  <div class = "form-group">
32      <label for="email">email</label>
33      <input
34          type="email"
35          class="form-control"
36          id ="email"
37          name="email"
38          placeholder="Enter Email Address"
39      />
40  </div>
```

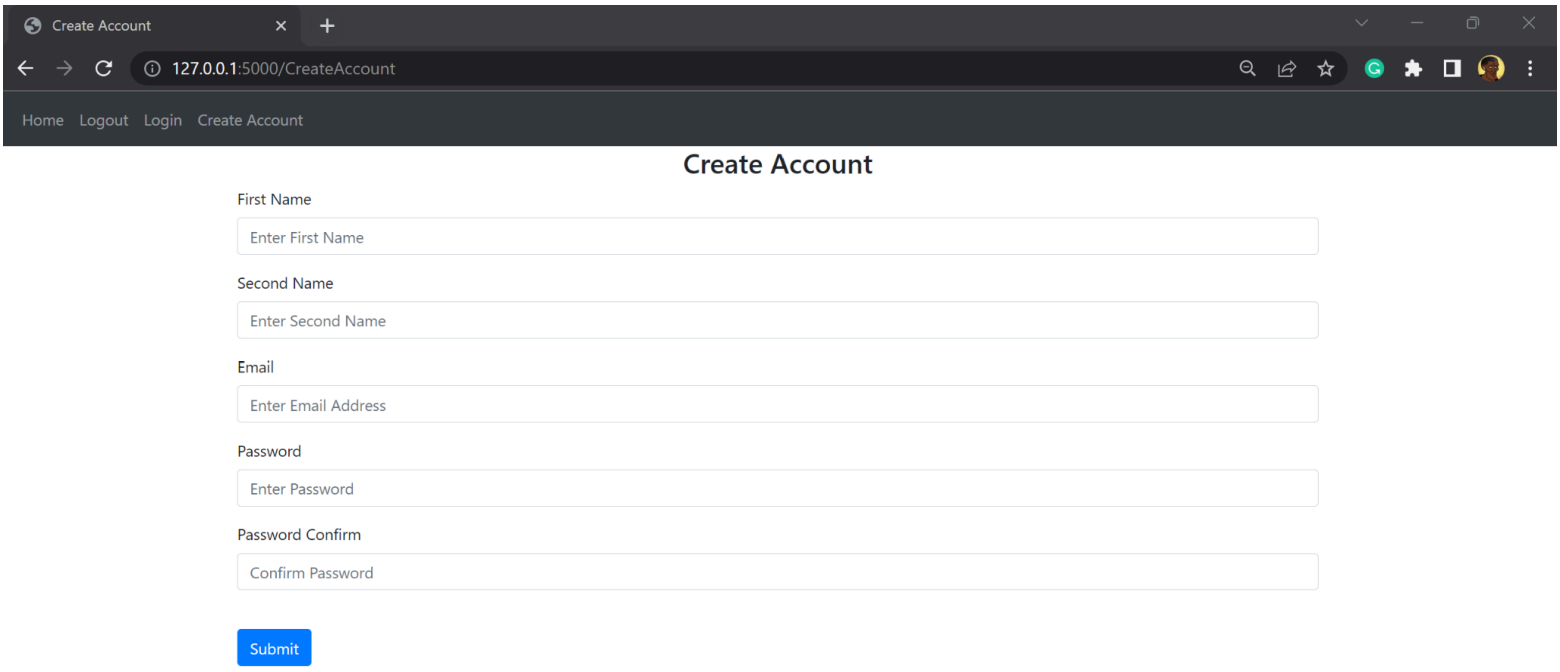
On line 21 or 32, this line is responsible to setting the label to the class used for the input field

On line 26 or 37, the value equal to 'name' is what the attribute is called when passing the information in this field to the backend

Similar code in the one sampled is written for the first name and password fields and can be found in the code dump at the end of this document.

At the end of the file to complete the form, a button is created which would be clicked to send off the information entered to the server

```
<button type="submit" class="btn btn-primary">Submit</button>
```



This is the outcome of the text boxes to enter information to create an account. The placeholders are self explanatory to reduce confusion of what each box is for.

Justification

This file is important as it displays a web page for the user to interact with. Each user needs an account to access the main website, so this page is important to gain access. This file is responsible for displaying a form that is then used to add each user to the database

login.html

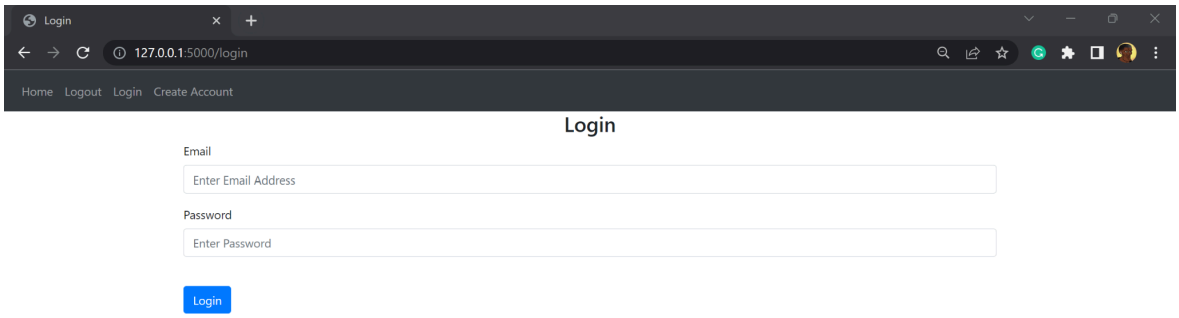
```html
<form method="POST">
    <h3 align="center"> Login</h3>

    <div class = "form-group">
        <label for="email">email</label>
        <input
            type="email"
            class="form-control"
            id ="email"
            name="email"
            placeholder="Enter Email Address"
        />
    </div>

    <div class = "form-group">
        <label for="password">Password</label>
        <input
            type="text"
            class="form-control"
            id ="password"
            name="password"
            placeholder="Enter Password"
        />
    </div>
    <br />
    <button type="submit" class="btn btn-primary">Login</button>

</form>
```

This file is very similar to the create account file, the difference is that there are only two input boxes needed for this page to sign the user in. They will need to enter their email address and their password. This is taken in using the POST method which sends the data to the server to validate its authenticity.

A label is added so that the user knows what page they are on, this reads Login. It has also been centred to help on the eyes of the user instead of having it to one side in line with the input boxes.

At the end of the file to complete the form, a button is created which would be clicked to send off the information entered to the server

Underneath is the outcome of this code. It makes two text box input fields for the user to enter their email and password and submit as a POST request:



Justification

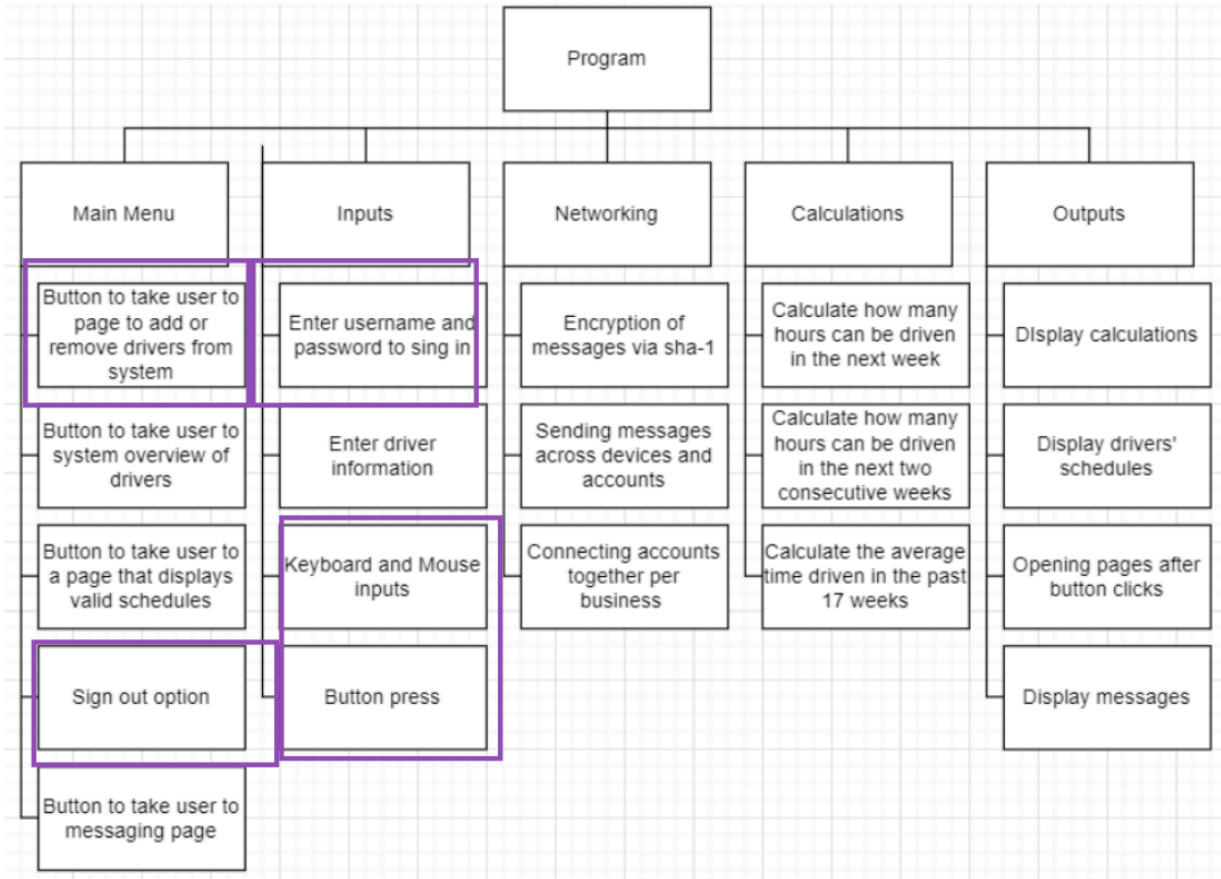
This is needed to allow the user to login to the website and access information pertained to them on the database. The login results in different information being displayed per user so it is important they are logged in to see their particular information such as their schedule.

Validation

```
<div class = "form-group">
    <label for="password">Password</label>
    <input
        type="text"
        class="form-control"
        id ="password"
        name="password"
        placeholder="Enter Password"
    />
</div>

<div class = "form-group">
    <label for="password_confirm">Password Confirm</label>
    <input
        type="text"
        class="form-control"
        id ="password_confirm"
        name="password_confirm"
        placeholder="Confirm Password"
    />
</div>
```

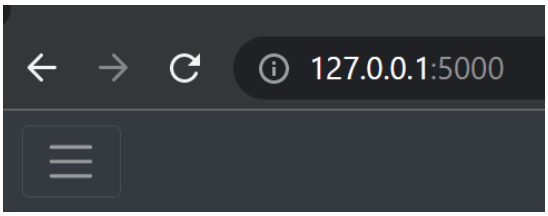
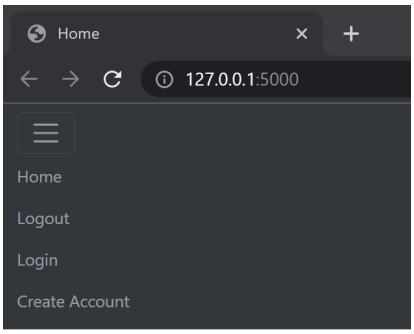
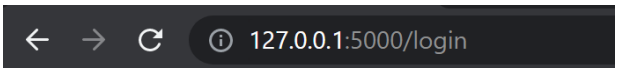
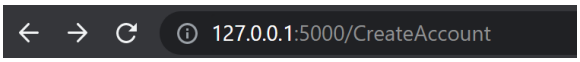
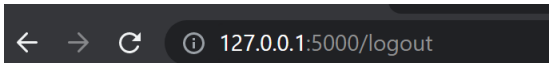
Validation is used in this module by having the user enter their password twice when creating an account, limiting a typo in their password as they must match for them to continue on creating an account.

Relation to top down modular design



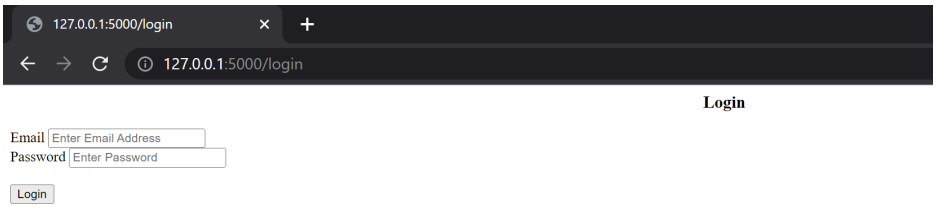
The user is able to create an account and log into the account through a HTML form
The log out features relates to the top down modular design created in the design section, as it falls under the sign out option for the Main Menu branch

Testing

Test No.	Explanation and justification	Expected outcome / criteria	Actual outcome	Pass or fail
1	This will check if a navbar is created and has the toggle button to collapse and expand the rest of the buttons. This test is important for the completion of this program because the program may be accessed on a phone which would be unable to display all the buttons on the top of the page. The navbar is essential to allow for multiple device use with varied screen sizes. To carry out this test, I will run the program and click the button and assess how it functions.	The button should collapse and expand when clicked showing other buttons	The button collapsed and expanded showing	pass
	Evidence:  			
2	This test will check the buttons will redirect the user to another page, corresponding with the button clicked This is important for the completion and meeting the user requirements because this will act as the menu and is what is used to navigate the website, to get to its different features. To carry out this test I will click each button.	When any button is clicked I am taken to a new page, and the URL will change with the path similar to the button name	Redirected to a new page and the path names appropriate to the button pressed	Pass
	Evidence:   			

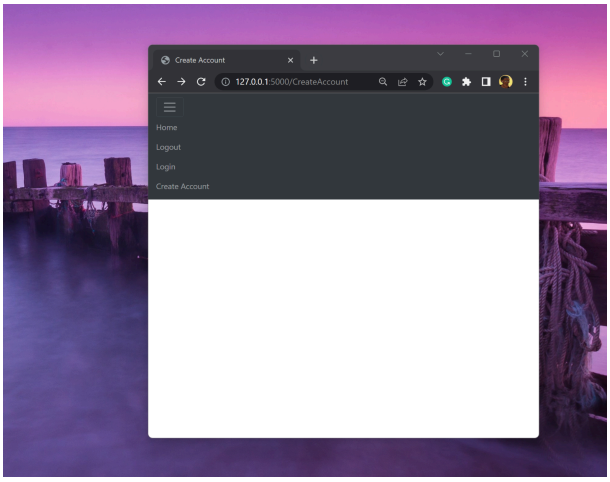
3	This test will check that the right information is displayed on the screen for the create account and login page This is important to meet the requirements of the stakeholder, because they requested functional login systems to manage employees. These pages take in the information and send it to the server which the manager can then view.	When the login button is clicked, redirected to a form with two fields and a label that says Login, using the linked bootstrap and stylesheet. When I press the Create Account button I am redirected to a page that has 5 input boxes and a label that says create account using the linked bootstrap and stylesheet.	The correct boxes were displayed and could be written in. Incorrect bootstrap formatting theme.	Bad pass

Evidence:



Explanation: The correct boxes and fields were displayed however the formatting was the HTML default and not the formatting from the linked bootstrap and CSS stylesheet

Explanation: This is the outcome of clicking the Create Account button. The URL changed to the correct one, however no fields were displayed on the screen.

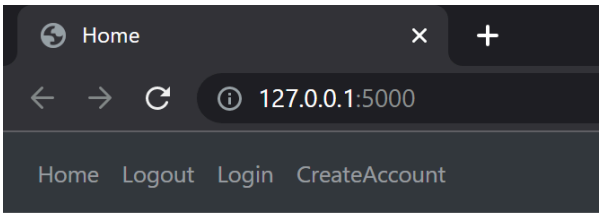


Prototype 2

The error in test 3 was based on some missing code. To fix these errors and pass the tests I added some lines of code to the login.html file and the create_account.html file.

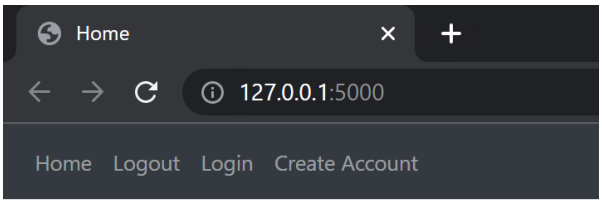
A few spelling mistakes were made during the coding of this module, which were fixed after the tests, such as:

Error:
CreateAccount button should have a space




```
<a class="nav-item nav-link" id="CreateAccount" href="/CreateAccount">CreateAccount</a>
```

Remedial Action:




```
<a class="nav-item nav-link" id="CreateAccount" href="/CreateAccount">Create Account</a>
```

login.html

```
{% extends "base.html" %} {% block title %}Login{% endblock %} {% block content%}

<form method="POST">
    <h3 align="center"> Login</h3>

    <div class = "form-group">
        <label for="email">Email</label>
        <input
            type="email"
            class="form-control"
            id ="email"
            name="email"
            placeholder="Enter Email Address"
        />
    </div>

    <div class = "form-group">
        <label for="password">Password</label>
        <input
            type="text"
            class="form-control"
            id ="password"
            name="password"
            placeholder="Enter Password"
        />
    </div>
    <br />
    <button type="submit" class="btn btn-primary">Login</button>

</form>
{%endblock%}
```

The sections circled in red are the lines of code added for my remedial action.
I had to extend the base file and overwrite the two blocks, title and content - this will then add the lines of code as input fields on the display whilst maintaining the theme of the website.

The reason the code displayed but had no theme was because it did not extend the base file. Because it didn't extend the base file, it didn't have to overwrite any blocks for the code to be displayed, unlike the create_account.html file

create_account.html

```
{% extends "base.html" %} {% block title %}Create Account{% endblock %} {% block content %}
```

At the beginning of this file, the code circled in red was added - this is the code to overwrite the content block from the base file which was extended. The reason there was nothing shown on the screen was because the code body of the files code wasn't inside any block for it to overwrite anything

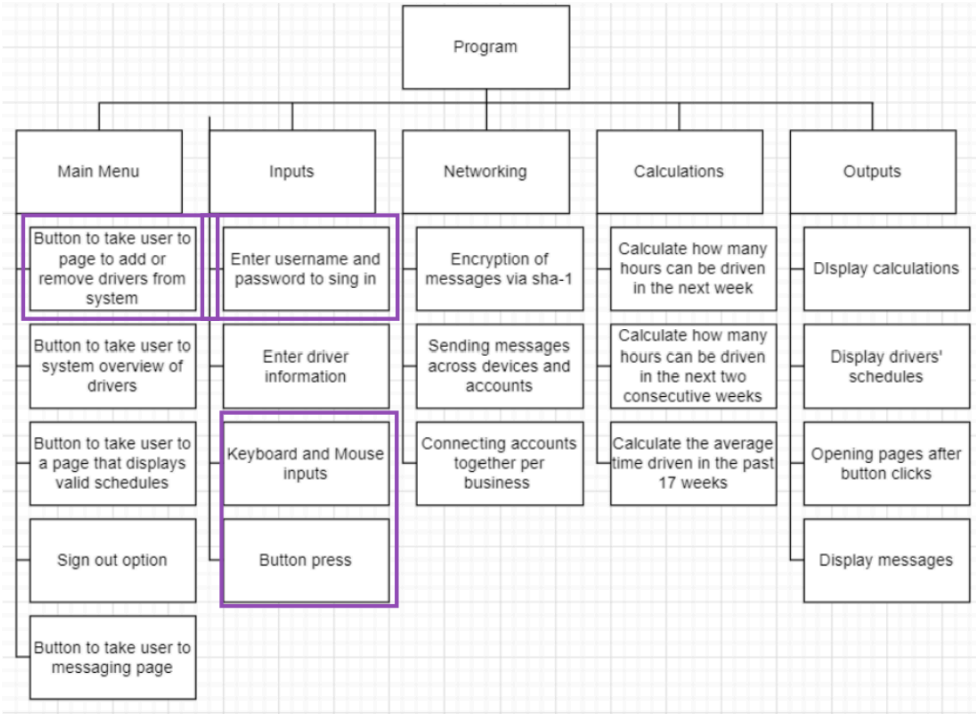
```
{% endblock %}
```

This was added at the end of the file to end the block so that the code was inside a block to overwrite a specific section of the base file.

Justification

So that the page maintains the theme of the program, its code must be encased by the block content.

Relation to top down modular design

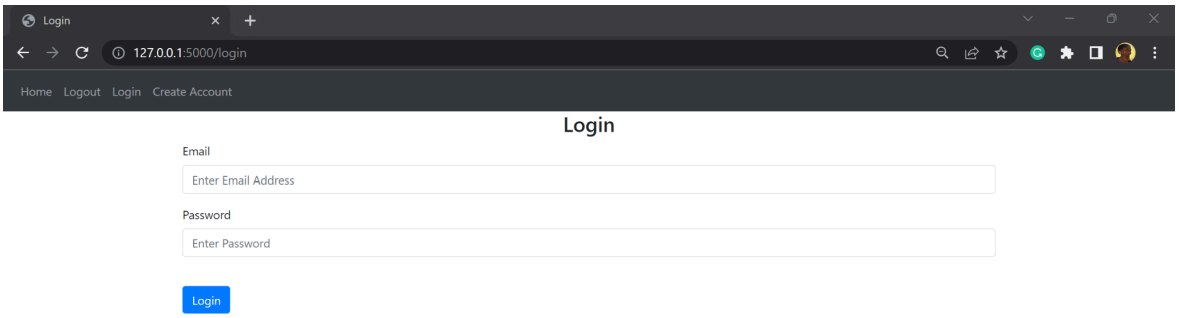


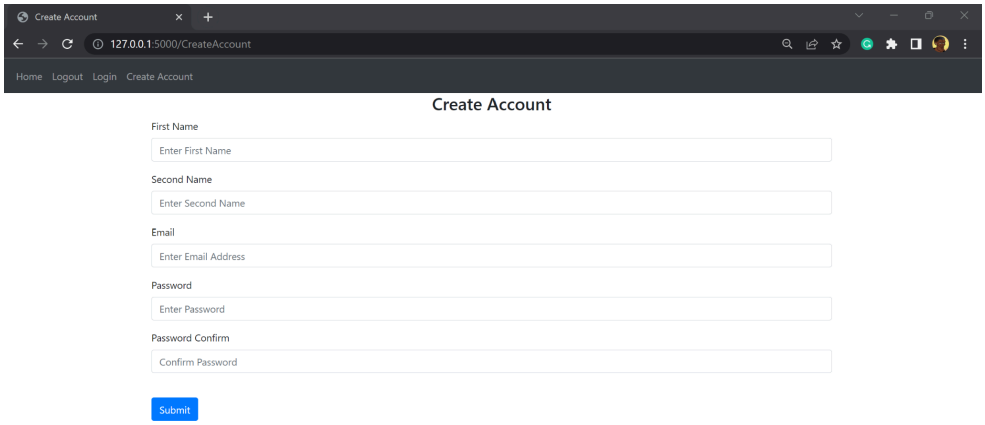
The user is able to create an account and log into the account through a HTML form

Testing

Test No.	Explanation and justification	Expected outcome / criteria	Actual outcome	Pass or fail
1	This test will check that the right information is displayed on the screen for the create account and login page This is important to meet the requirements of the stakeholder, because they requested functional login systems to manage employees. These pages take in the information and send it to the server which the manager can then view.	When the login button is clicked, redirected to a form with two fields and a label that says Login, using the linked bootstrap and stylesheet. When I press the Create Account button I am redirected to a page that has 5 input boxes and a label that says create account using the linked bootstrap and stylesheet.	The correct boxes were displayed and could be written in. They also followed the theme of the website.	Pass

Evidence:





Conclusion of testing
Even though the tests in prototype 2 passed - the passwords are visible when entered.

Stakeholder feedback:
- “The passwords should not be visible when entered, please fix this”
- “Can you make it so that the user has to have a capital letter for their first name and second name please”

Prototype 3

To satisfy stakeholder feedback the aim of this prototype is to mask the password when entered and to follow english language rules for nouns by capitalising the first letter of the name and making everything else lower case

backend.py

Remedial action:

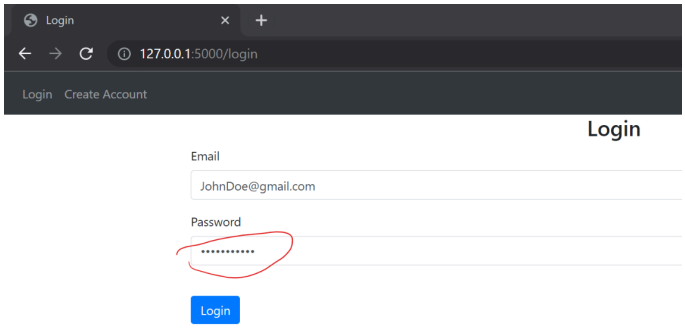
```
@backend.route('/CreateAccount', methods=['GET', 'POST']) #creating a decorator, when the URL is /CreateAccount the function below is invoked
def create_account():
    if request.method == 'POST': # differentiate between GET and POST request
        first_name = request.form.get('first_name').capitalize() # .get is used to get a specific attribute from the form
        second_name = request.form.get('second_name').capitalize() # capitalize function used to make first letter a capital
        email = request.form.get('email')
        password = request.form.get('password')
        password_confirm = request.form.get('password_confirm')
```

This function saves the user input for their first_name and second_name in the database with the first character in the string upper case and the rest of the characters in the string lower case

login.html, create_account.html

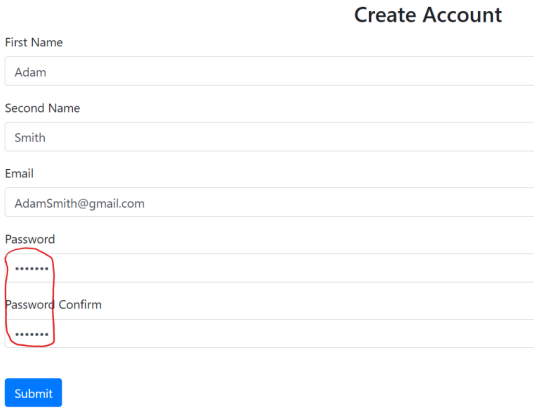
Remedial action:

```
<div class = "form-group">
    <label for="password">Password</label>
    <input
        type="password"
        class="form-control"
        id ="password"
        name="password"
        placeholder="Enter Password"
    />
</div>
```



Here I have made changes to the lines of code highlighted: the change made was the type is now equal to password and no longer text. This automatically replaces the characters entered by the user to stars

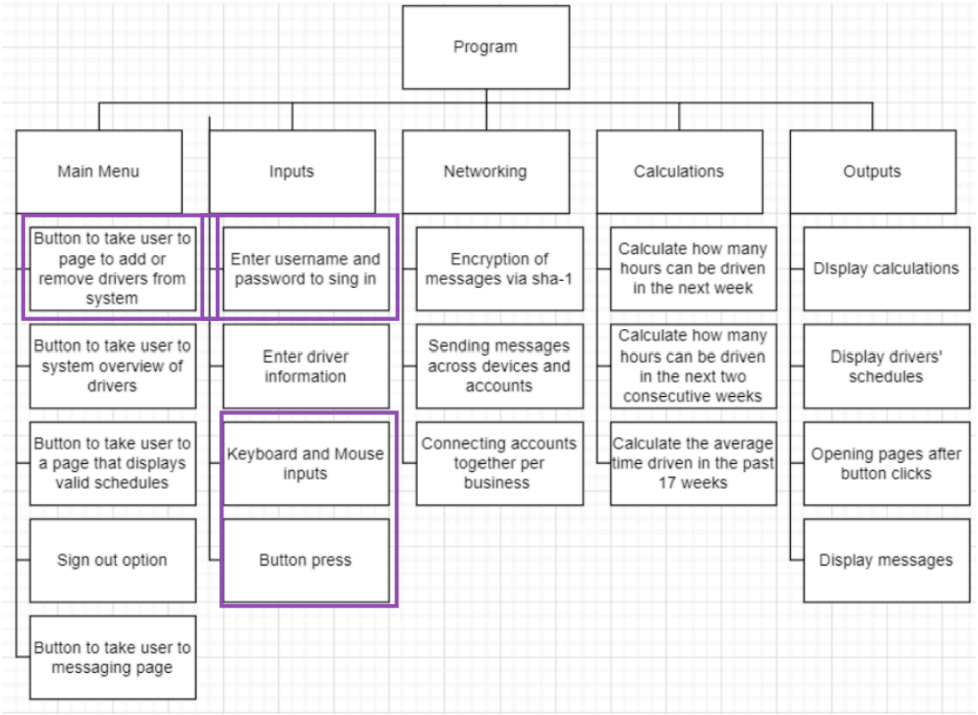
```
<div class = "form-group">
    <label for="password">Password</label>
    <input
        type="password"
        class="form-control"
        id ="password"
        name="password"
        placeholder="Enter Password"
    />
</div>

<div class = "form-group">
    <label for="password_confirm">Password Confirm</label>
    <input
        type="password"
        class="form-control"
        id ="password_confirm"
        name="password_confirm"
        placeholder="Confirm Password"
    />
</div>
```



Justification

This feature provides an extra layer of security as the program handles personal information. Compromisation of this information can result in a breach of the Data Protection Act with the website holder at fault. This feature reduces the risk of social engineering methods such as shouldering.

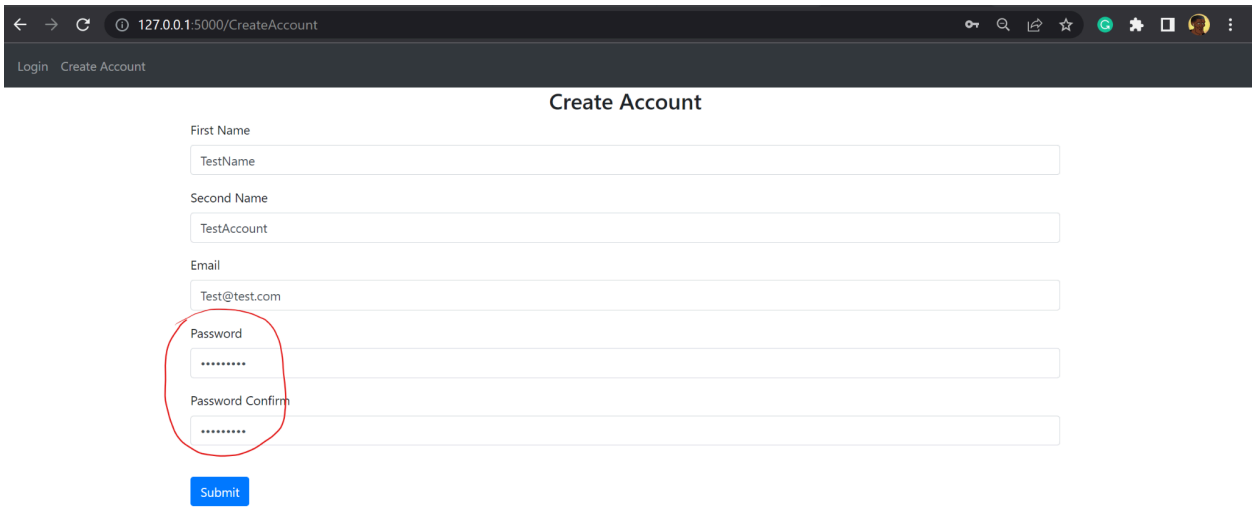
Relation to top down modular design



The user is able to create an account and log into the account through a HTML form

Testing

Test No.	Explanation and justification	Expected outcome / criteria	Actual outcome	Pass or fail
1	This is a new test due to stakeholder feedback, this ensures their requirements are met. It will test that the password upon entry on the signup page is replaced with stars In the password field, the password entered for testing will be ‘123456789’ This is a needed feature to ensure the risk of shouldering is reduced and provides an extra layer of security to its users, as this website handles sensitive information.	Password field to show equal number of stars to the characters entered. Should show 9 stars	9 stars appeared instead of the actual password entered	Pass



Prototype 4

This prototype adds more web pages accessible to users. This prototype focuses on the pages that allows the manager to add to the schedules of each driver, enter the number of hours per weekday driven by each driver, and output the schedules of drivers.

edit_schedules.html

This html file displays its content when the URL is /EditSchedules

To begin with the first line of code extends from the base file to keep correct formatting and consistency throughout the program

```
{% extends "base.html" %} {% block title %}Edit Schedules{% endblock %} {% block content %}
```

```
{% endblock %}
```

This is a form for the manager to select the driver they'd wish to edit the schedule of. All drivers who have created an account are displayed.

In html using {%%} allows me to type lines of python code, and is used in this file to loop through the drivers and weekdays. This is because these are template tags which allow me to do programming on the server before sending HTML

```
{% for driver in drivers %}
    <input type = "radio" id ="driver" name="driver" value = "{{driver}}">
    <label for="driver"> {{driver}}</label>  
{% endfor %}
```

In html using {} allows me to output variables I had declared in the respective function found in Module 2. These are syntax for template variables

The driver names are queried from the User table in backend.py and the variable drivers are passed through the function.
The drivers are shown using radio buttons.

Justification

This file displays a web page that is responsible for the creation of a schedule to be manipulated and thus complete the user requirements such as calculating the mean hours worked in the last 17 weeks.

schedule_days.html

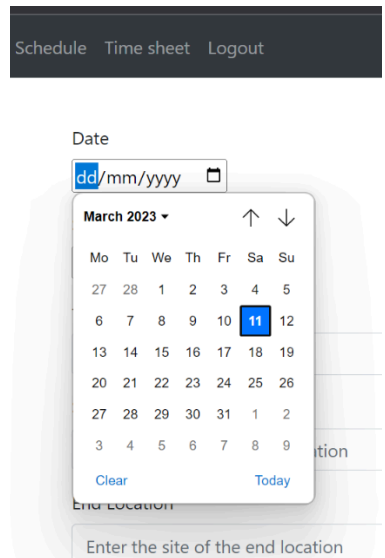
The purpose of this html file is to allow the manager to add information to the jobs for the users.
A html form is used to POST the information to the server to update the Jobs table

The code is formatted this way to increase readability by separating the lines and attributes of this form class.

The first two fields are to get the date and time of the job to be added.

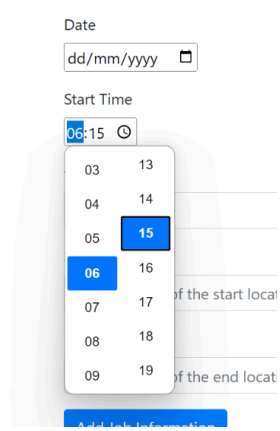
```
<div class = "form-group">
    <label for="date">Date</label> <br>
    <input
        type="date"
        id ="date"
        name="date"
    />
</div>
```

The input type for the first field is date which provides a calendar




```
<div class = "form-group">
    <label for="time">Start Time</label> <br>
    <input
        type="time"
        id ="time"
        name="time"
    />
</div>
```

The input type for the second field is time which provides a scroll of hours and minutes



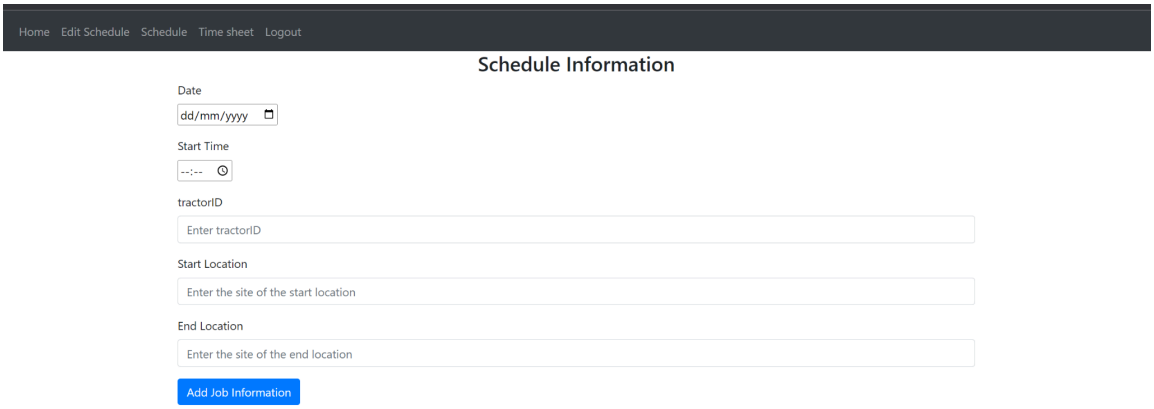
```html
<div class = "form-group">
    <label for="tractorID">tractorID</label>
    <input
        type="text"
        class="form-control"
        id ="tractorID"
        name="tractorID"
        placeholder="Enter tractorID"
    />
</div>

<div class = "form-group">
    <label for="start_location">Start Location</label>
    <input
        type="text"
        class="form-control"
        id ="start_location"
        name="start_location"
        placeholder="Enter the site of the start location"
    />
</div>

<div class = "form-group">
    <label for="end_location">End Location</label>
    <input
        type="text"
        class="form-control"
        id ="end_location"
        name="end_location"
        placeholder="Enter the site of the end location"
    />
</div>
```

These are 3 similar fields. The tractordID, the start and end location. These inputs are all text

They use the class form-control which makes the boxes look a certain way with equal length

```html
<button type="submit" class="btn btn-primary">Add Job Information</button>
```

This button submits the form to the server and when pressed results in the user being redirected to another page




```jinja
{% extends "base.html" %} {% block title %}Add Job{% endblock %} {% block content %}
```

This is the form the user sees. It continues the theme of the website as it extends the base html file.

Justification

This allows the manager to add schedules for each driver. It only asks for the necessary information to not further complicate understanding the schedule.

view_schedule.html

This file displays the drivers schedules. However due to the nature of the business where managers and drivers have different roles, the manager will need to be able to see all the schedules of the drivers, whereas the drivers only need to see their own personal schedule. This results in the use of the manager column in the table User.

This page relies on an if statement to separate the page a manager and driver see

To begin with outside of this if statement:

```
<h6 align="right"><b>Today's date:</b> {{day}}</h6>
<h6 align="right"><b>Weekday: </b> {{weekday}}</h6>
```

These two lines of code output the date and the weekday at the top right of the program. This is done by using the variable names in the python file by encasing them in two braces. This is displayed on this page as it shows the user the current date so they can compare it to the schedules also outputted on the page.

The output:

Today's date: 2023-03-12

Weekday: Sunday

Next is the if statement for the manager role:

```
{% if user.manager == 1 %}
    <ul class="list-group list-group-flush" id="Job"></ul>
    {% for job in jobs %}
    <li class="list-group-item">
        <b>Date:</b> {{job.date}}  
        <b>Time:</b> {{job.time}}  
        <b>TractorID:</b> {{job.tractorID}}  
        <b>Start Location:</b> {{job.start_location}}  
        <b>End Location:</b> {{job.end_location}}  
        <b> Driver:</b> {{job.user}}
    </li>
    {% endfor %}
```

user is the variable passed into the html file. It uses flask's current_user function to determine whether the user currently signed in is a manager or not.

In the User table the manager column is True if the user is a manager and is Null or False if the user is not a manager and therefore a driver.

To determine if this column is set to True, 1 is used.

Once the current user is verified as a manager then the schedules are loaded for all users by looping through all the jobs

Next is the else clause if the current user is not a manager

```
{% else %}
    <ul class="list-group list-group-flush" id="Job"></ul>
    {% for job in user.jobs %}
    <li class="list-group-item">
        <b>Date:</b> {{job.date}}  
        <b>Time:</b> {{job.time}}  
        <b>TractorID:</b> {{job.tractorID}}  
        <b>Start Location:</b> {{job.start_location}}  
        <b>End Location:</b> {{job.end_location}}  
        <b> Driver:</b> {{job.user}}
    </li>
    {% endfor %}
{% endif %}
```

If the user is not a manager then only their schedule will be displayed.

user is the variable passed from the backend.py file. It uses an imported library from flask to validate the user logged in. When the user is logged in, their account information such as their records in the database are remembered. As a result the current users records can be queried automatically

```
return render_template("view_schedule.html", user = current_user, jobs = job, drivers=driver, day=day, weekday=weekday)
```

jobs is a variable passed by the view_schedules function in the backend.py file. It is a query of the Jobs table. This allows each job in the users schedule to be displayed

Justification

This file is needed to display the schedules only when the manager is signed in. This feature protects the manager from possibly breaking GDPR regulations. It also prevents other drivers knowing the schedules of other drivers, preventing an increased nuisance of disgruntled employees.
The function of the program is to view schedules so this directly fulfils requirements from the design section
Drivers also are unlikely to need to see other drivers schedules and could cause even more confusion with understanding their schedule

specific_schedule.html

```
<h6 align="right"><b>Today's date:</b> {{day}}</h6>
<h6 align="right"><b>Weekday: </b> {{weekday}}</h6>
```

These two lines of code output the date and the weekday at the top right of the program. Identical to the view_schedule.html file above

The output:

Today's date: 2023-03-12

Weekday: Sunday

specificDriver is a variable passed to this html file. It is the name of the driver the manager has selected to view the schedule of.

```
<center>

    <br>

    <b> Schedule for {{specificDriver}} </b>
    <br>

    <ul class="list-group list-group-flush" id="Job"></ul>
    {% for job in jobs %}
        <li class="list-group-item">
            <b>Date:</b> {{job.date}}   <b>Time:</b> {{job.time}}   <b>TractorID:</b> {{job.tractorID}}   <b>Start Location:</
        </li>
    {% endfor %}

</center>
```

Next the program loops through the jobs of the driver to display each job from the table Job.
  highlighted, is html syntax that displays white space, I used this because html gets rid of any white space in the code which groups all the information to close together making it hard to read. So this tells the program to create space in between each set of information

The output:

Date: 2023-02-09 **Time:** 20:04 **TractorID:** DT37 HCS **Start Location:** Northampton **End Location:** Bristol **Driver:** Devin

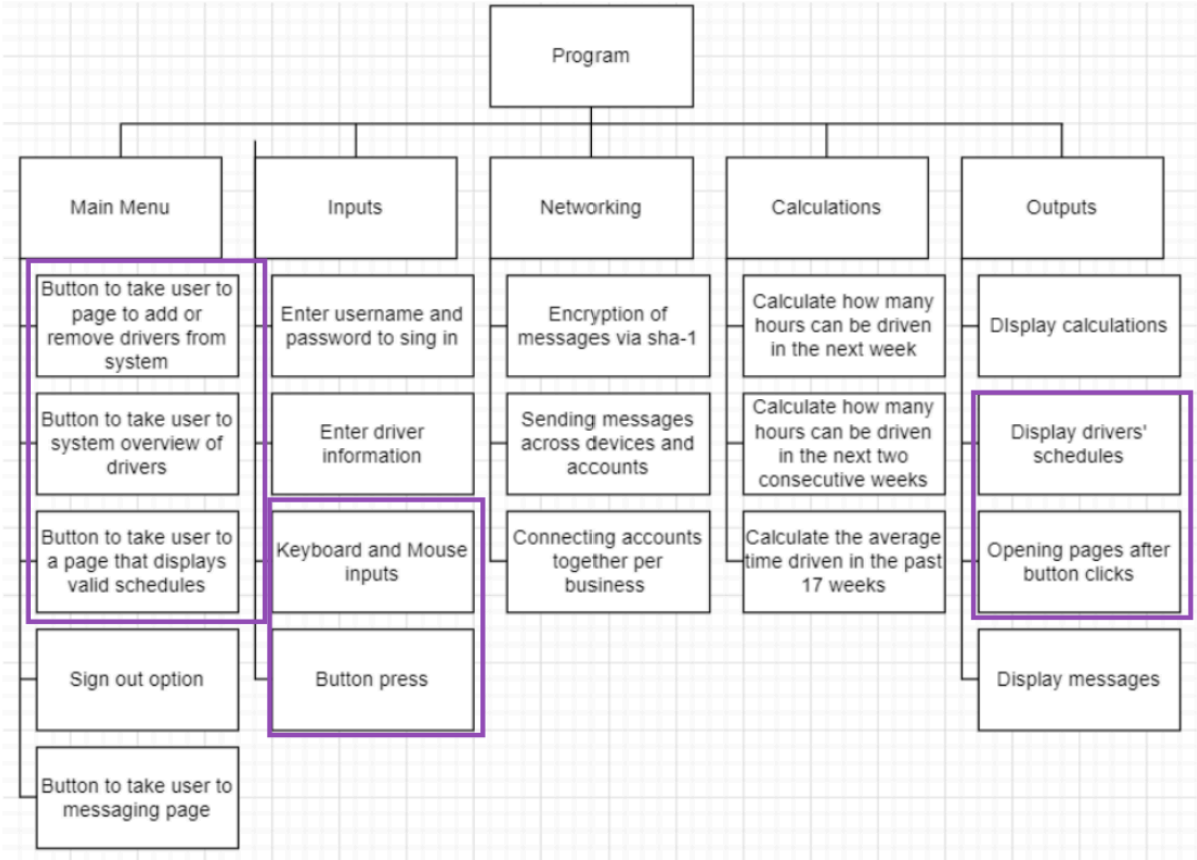
Date: 2023-02-19 **Time:** 17:54 **TractorID:** TER3 TYX **Start Location:** London **End Location:** Heathrow **Driver:** Devin

Date: 2023-02-23 **Time:** 07:00 **TractorID:** KL02 HRM **Start Location:** London **End Location:** Leeds **Driver:** John

Justification

To maintain scalability of a business, the manager would need to be able to filter jobs by important attributes, such as the date or the driver name. This results in a narrower scope of jobs in a schedule, making it easier to read specific jobs. This function provides this

Relation to top down modular design

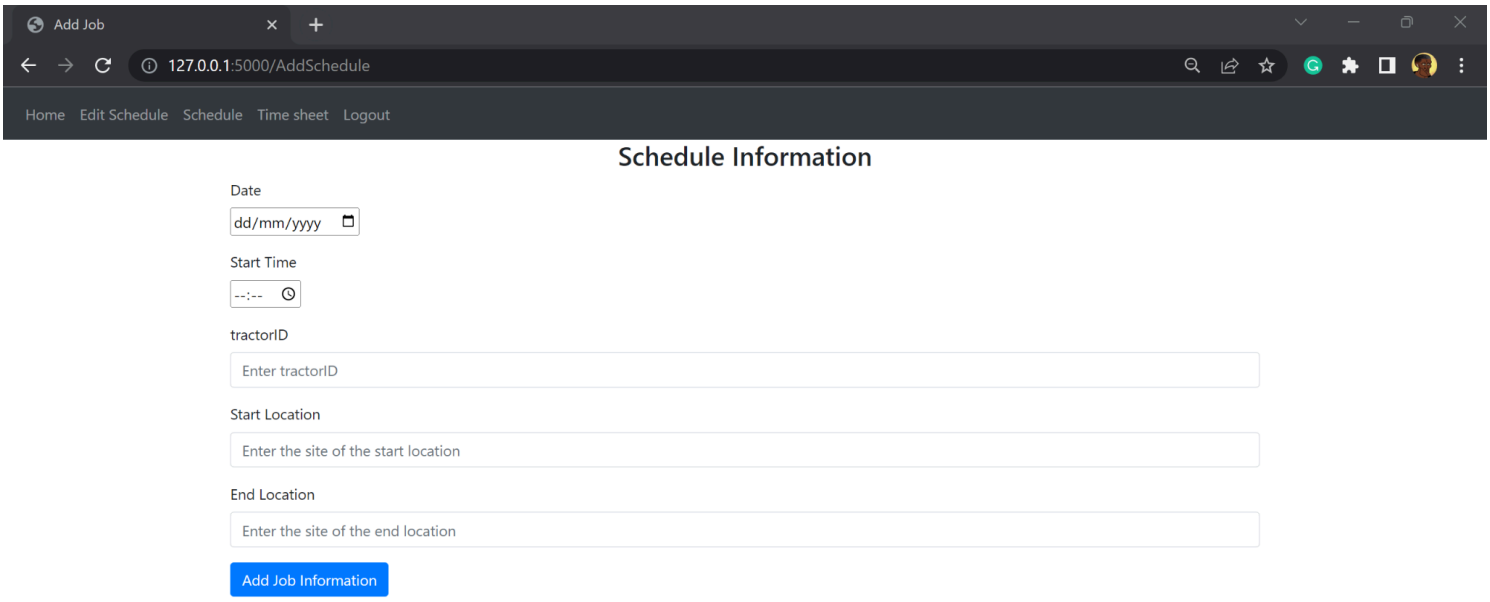


This relates to the diagram because the buttons take the user to different pages

- Can view schedules after button press

Testing

Test No.	Explanation and justification	Expected outcome / criteria	Actual outcome	Pass or fail
1	This test is to make sure the users are displayed on the web page if the user is the manager This test will be carried out by clicking the Edit Schedule button and recording what is displayed This is important test because the manager must be able to see its drivers and select the one they want to assign a job for	The names of all drivers in the User table	The table name and identifier to represent each record in the table	Fail
	Evidence: 			
2	This test is to display the weekdays in a drop down menu when the web page is loaded This is needed to make sure the manager can filter through the schedules by weekday	A drop down menu of all 7 weekdays	7 drop down menus with each only displaying one option that says Weekdays	Fail
	Evidence: 			
3	This test is to make sure the page is redirected to the page that allows the user to add information for the schedule after submitting the driver wanted to add a schedule for. To do this test, I will click the edit schedule button on the navbar, pick the driver Devin and submit it. This test is important to ensure that the correct pages can be loaded, allowing for the functionality of the program to be	After selecting Devin, the page redirects to the Add schedule page	After selecting Devin, the page redirects to the Add schedule page	pass

	correct.			
	Evidence:  			
4	This test is to make sure the jobs added, are displayed properly with the correct driver from the previous submission This test will be carried out by completing the form on the add schedule page. Completion of this should automatically redirect the web page to the schedules page displaying all the schedules as the user signed in will be the manager. The job added will be for Devin The date of the job will be August 9th, at 20:15 The tractor ID added will be AB12 CDE The start location is Portsmouth The end location is Bournemouth	A list of all the jobs from the Job table The last job should have the details stated in the explanation box	The page redirected to the schedules page. The test data added was displayed. The date showed: 2023-08-09 The time showed: 20:15 The tractor ID showed: AB12 CDE The start location showed: Portsmouth	pass

			The end location showed: Bournemouth	
	Evidence: The test data added for this test: Date 09/08/2023 Start Time 20:15 tractorID AB12 CDE Start Location Portsmouth End Location Bournemouth Add Job Information **Date:** 2023-08-09 **Time:** 20:15 **TractorID:** AB12 CDE **Start Location:** Portsmouth **End Location:** Bournemouth **Driver:** Devin			
5	Display the schedules row by row when the Schedules button is pressed. The program has two user access levels, manager and not manager. Not managers are drivers and therefore only their schedule should be displayed Managers can have access to all drivers schedules I will sign into two accounts for this test: John Doe (driver) and Jae Crowder (manager) When signed in a manager account, schedules of all the drivers will be shown When signed in a driver account, only the schedules of the current user is shown	When signed in as Jae Crowder, all the schedules will be shown When signed in as John Doe only jobs for this driver is shown	When signed in as Jae Crowder, all the schedules is shown When signed in as John Doe only jobs for this driver is shown	pass
	Evidence: **Jae Crowder account:**			

Schedule

Date: 2023-02-09	**Time:** 20:04	**TractorID:** DT37 HCS	**Start Location:** Northampton	**End Location:** Bristol	**Driver:** Devin
Date: 2023-02-19	**Time:** 17:54	**TractorID:** TER3 TYX	**Start Location:** London	**End Location:** Heathrow	**Driver:** Devin
Date: 2023-02-23	**Time:** 07:00	**TractorID:** KL02 HRM	**Start Location:** London	**End Location:** Leeds	**Driver:** John
Date: 2023-02-23	**Time:** 15:02	**TractorID:** GL57 BND	**Start Location:** Bedford	**End Location:** Stratford	**Driver:** Austin
Date: 2023-02-24	**Time:** 09:00	**TractorID:** AS24 KYX	**Start Location:** Luton	**End Location:** Nottingham	**Driver:** John
Date: 2023-02-24	**Time:** 09:00	**TractorID:** LR76 RGM	**Start Location:** Lancaster	**End Location:** Portsmouth	**Driver:** Martin
Date: 2023-02-25	**Time:** 10:00	**TractorID:** RL67 OPB	**Start Location:** Newcastle	**End Location:** Sheffield	**Driver:** Martin
Date: 2023-02-25	**Time:** 13:00	**TractorID:** FK78 HGX	**Start Location:** Essex	**End Location:** Croydon	**Driver:** John
Date: 2023-02-26	**Time:** 13:00	**TractorID:** WE23 KBC	**Start Location:** Northampton	**End Location:** Arenson	**Driver:** Austin
Date: 2023-02-28	**Time:** 17:30	**TractorID:** NK22ZWG	**Start Location:** Arenson	**End Location:** Arenson	**Driver:** John
Date: 2023-03-09	**Time:** 17:15	**TractorID:** GH12 KLP	**Start Location:** Canterbury	**End Location:** Newcastle	**Driver:** Chris
Date: 2023-03-16	**Time:** 18:00	**TractorID:** PL14 KOP	**Start Location:** Leeds	**End Location:** London	**Driver:** Kevin
Date: 2023-03-29	**Time:** 08:30	**TractorID:** TE23 YKS	**Start Location:** Leeds	**End Location:** Hemel hempstead	**Driver:** Martin

John Doe account:

Schedule

Date: 2023-02-23	**Time:** 07:00	**TractorID:** KL02 HRM	**Start Location:** London	**End Location:** Leeds	**Driver:** John
Date: 2023-02-24	**Time:** 09:00	**TractorID:** AS24 KYX	**Start Location:** Luton	**End Location:** Nottingham	**Driver:** John
Date: 2023-02-25	**Time:** 13:00	**TractorID:** FK78 HGX	**Start Location:** Essex	**End Location:** Croydon	**Driver:** John
Date: 2023-02-28	**Time:** 17:30	**TractorID:** NK22ZWG	**Start Location:** Arenson	**End Location:** Arenson	**Driver:** John

6	This test is to only display the jobs of the driver selected. This is a manager account feature. The account signed in will be Jae Crowder. The test will be done by selecting the driver Devin to show the schedule of. The passing of this test is important for the scalability of the	Option on the schedules page to select a driver and submit. Schedule for Devin shown	Option on the schedules page to select a driver and submit. Schedule for Devin shown	pass

	haulage company, to allow them to filter all the jobs to allow better access to schedules			

Evidence:

Schedule for Devin

Date: 2023-02-09	**Time:** 20:04	**TractorID:** DT37 HCS	**Start Location:** Northampton	**End Location:** Bristol	**Driver:** Devin
Date: 2023-02-19	**Time:** 17:54	**TractorID:** TER3 TYX	**Start Location:** London	**End Location:** Heathrow	**Driver:** Devin

Conclusion of testing of prototype 4:

The users are hard to identify as their first names are not displayed resulting in a fail of the test. This issue with the weekdays display is that the program iterates through the weekday list which has 7 items. It iterates this number of times and for each item it displays its own drop down menu with one value which is weekday instead of the days of the week. The fail is due to not passing in the correct value, and using incorrect syntax

Stakeholder feedback:

- Requested for users to be shown in a drop down menu instead of using radio buttons.

Prototype 5

This prototype focuses on minor details and remedial action such as how information is displayed.

- Fixes the drivers names displayed
- Fixes the weekdays displayed

edit_schedules.html

Remedial action:

As per stakeholder request, the drivers are now shown through a drop down menu, this is possible through the select tag for a html form

```
{% if user.manager == 1 %}
    <h5>Driver</h5>
    <select name="drivers">
        {% for driver in drivers %}
            {% if driver.manager == 0 %}
                <option value="{{driver}}" SELECTED>{{driver.first_name}}</option>
            {% endif %}
        {% endfor %}
    </select>
     <button type="submit" class="btn btn-primary">Submit</button>
```

The program iterates through each driver in the User table, using the query; ‘drivers’

The error in prototype 4 was because the first name column of the records displayed was not called to be displayed. Here this column is used to display the drivers. first_name is the column in the User’s table

As the manager is a part of the User table, but is not a driver, logically their name shouldn’t be shown in the drop down menu. To prevent this logical error, only the names of the users who are drivers are displayed

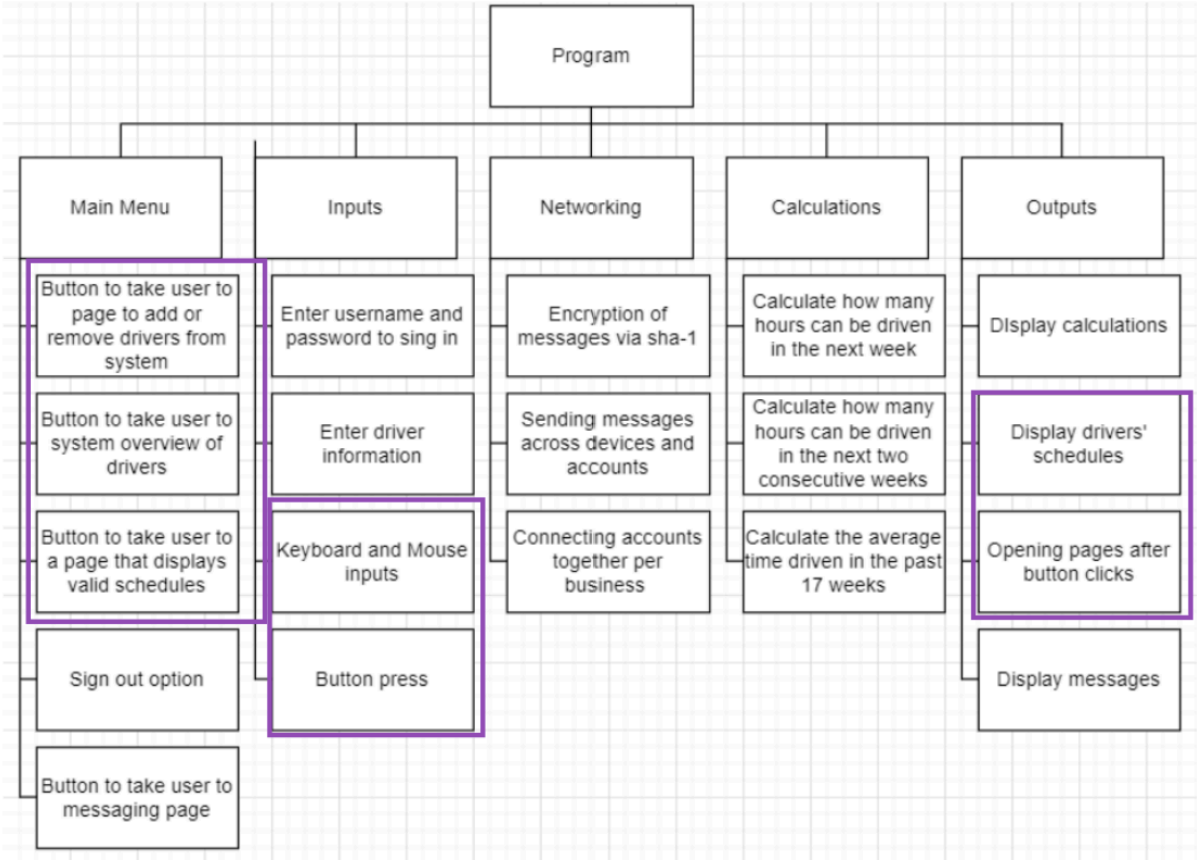
```
<h5>Day</h5>
<select name="weekdays" method="POST" action="EditSchedules">
    {% for weekday in weekdays %}
        <option value="{{weekday}}" SELECTED>{{weekday}}</option>"
    {% endfor %}
</select>
```

In the backend.py file, I created a list including all the weekdays, this list is passed through the function and is made accessible in the html file.
The weekdays are displayed in a drop down menu by looping through the list

Justification

This remedial action is needed because the stakeholder was unsatisfied with the design and layout of the program before. Here the changes meet the stakeholders requirement and also changes some logical aspects of the program such as only displaying the drivers and not the managers when looking for a schedule or editing a schedule.

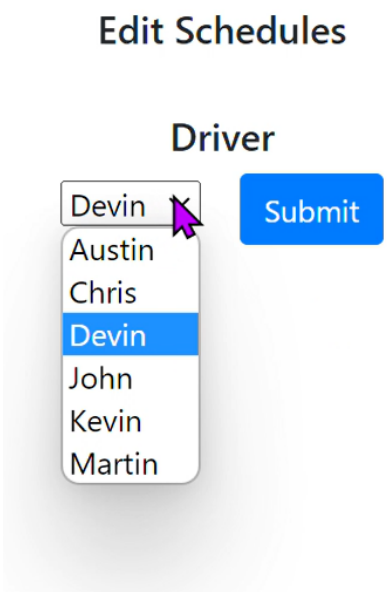
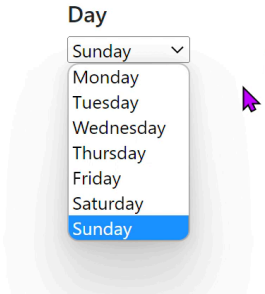
Relation to top down modular design



This relates to the diagram because the buttons take the user to different pages

- Can view schedules after button press

Testing

Test No.	Explanation and justification	Expected outcome / criteria	Actual outcome	Pass or fail
1	The drivers are shown in a drop down menu This test will be carried out by clicking the edit schedules button and expecting to see a drop down menu of all the drivers This is important it passes to allow the managers to add schedules of its drivers so they can complete business	Display drivers names in a drop down menu	Display drivers names in a drop down menu	Pass
	Evidence:  			
2	This test is to make sure the weekdays are shown in a drop down menu correctly It is important it passes to allow for the manager to filter through jobs depending on the day	Displays a drop down menu of all weekdays	Displays a drop down menu of all weekdays correctly	Pass
	Evidence:  			

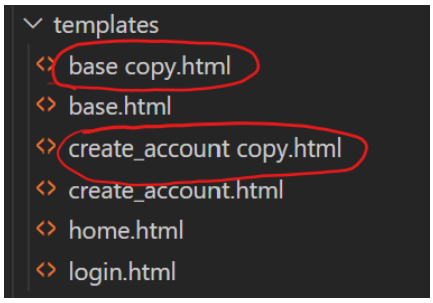
Conclusion of module 3 testing

This module passed all tests on allowing me to move onto the next module. The basic structure and theme of the website has been developed which readily enables scalability of this website. DIfferent areas of code highlighted, makes the program accessible on more a devices

Comments for maintenance

As stated earlier, this module is different from the rest, as it is coded primarily in html, this meant that commenting was inaccessible on the files being used in the program as they disrupted the code, as they were parsed in the compilation stages when run. As a result copies of the html files were made, with comments in green around the files explaining different lines of code.

The snippets shown come from the copy of the files:



Comments can be found in these highlighted files

create_account copy.html

```
<!-- Building an HTML form -->
<form method="POST"> <!-- When submitting the form it will send a POST request -->
    <h3 align="center"> Create Account</h3> <!-- A header for the from aligned in the middle -->

    <div class = "form-group"> <!-- This is a bootstrap class -->
    <!-- The fields of this form -->
        <label for="first_name">First Name</label> <!-- Setting it to the class used for the input field-->
        <input
            type="text"
            class="form-control"
            id ="first_name"
            name="first_name"
            placeholder="Enter First Name"
        />
            <!-- the name is what the attribute is called when passing the information in this field to backend -->
    </div>
```

A comment at the top of the screenshot to describe what the following block of code does
Line by line commenting which then describes what each respective line does

base copy.html

```
<head>
  <meta charset="utf-8" /> <!-- This defines the character set used for this document -->
  <meta name="viewport" content="width=device-width, initial-scale=1" /> <!-- Makes the page take up the entire width of the screen, intial-scale=1 sets the zoom level -->

  <!-- Bootstrap and stylesheet -->
  <link
    rel="stylesheet"
    href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css"
    integrity="sha384-Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh"
    crossorigin="anonymous"
  />
  <link
    rel="stylesheet"
    href="https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css"
    crossorigin="anonymous"
  />

  <title>{% block title %}Home{% endblock %}</title> <!-- Defining a block for the title -->
</head>
```

Line above a block of code to describe what that section does - here in this example it is the information for the bootstrap

```
<!-- defining a navbar -->
<nav class="navbar navbar-expand-lg navbar-dark bg-dark"> <!-- Bootstrap classes found on bootstrap website-->
  <button
    class="navbar-toggler"
    type="button"
    data-toggle="collapse"
    data-target="#navbar"
  >
    <span class="navbar-toggler-icon"></span>
  </button>
  <div class="collapse navbar-collapse" id="navbar">
    <div class="navbar-nav">
      {% if user.is_authenticated %}
      <a class="nav-item nav-link" id="home" href="/">Home</a> <!-- nav-link means this links to something, the href is where it redirects the button to -->
      <a class="nav-item nav-link" id="logout" href="/logout">Logout</a>
      {% else %}
      <a class="nav-item nav-link" id="login" href="/login">Login</a>
      <a class="nav-item nav-link" id="signUp" href="/sign-up">Create Account</a>
      {% endif %}
    </div>
  </div>
</nav>
```

Line above block of code to describe what that block does - in this example it is the code for the navbar

```
<div class="container">{% block content %} {% endblock %}</div> <!-- A block to be overwritten to add content for a page -->
```

Describing what block is and how to define a block

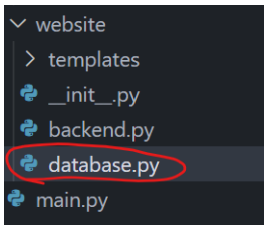
Module 4: Database

The database consists of multiple SQL tables. The different tables will be coded at different points of the creation of this program due to the nature of some tables relying on other modules to be made, such as the HTML templates for the scheduling table and messaging table.
The nature of other tables, such as the account table means it has to be created first as other modules rely on its presence to work.

In this section, I will focus on making the table specific to account creation as the next modules rely on calling on this table to test its functionality.

Prototype 1

database.py




```
from . import db # importing from the package db which is in the init file
from flask_login import UserMixin # custom class to inherit giving user object specific things to login
```

To begin with, in database.py, I had to import db from the init file and UserMixin from the flask_login library.

```
#Creating a table for user information using a class object
class User (db.Model, UserMixin): #UserMixin used for implementations of this objects properties
```

Next, I created a table. This table is called User because it handles all user information received when an account is created.

```
id = db.Column(db.Integer, primary_key=True)
email = db.Column(db.String(150), unique=True)
password = db.Column(db.String(150)) #Creates a
first_name = db.Column(db.String(150)) #Sets fi
second_name = db.Column(db.String(150))
manager = db.Column(db.Boolean) #True when acco
```

The names of the columns are the same as the variable names used to receive the information on the forms. Columns are created in the table, and the name is assigned.

```
id = db.Column(db.Integer, primary_key=True)
email = db.Column(db.String(150), unique=True)
password = db.Column(db.String(150)) #Creates a
first_name = db.Column(db.String(150)) #Sets fi
second_name = db.Column(db.String(150))
manager = db.Column(db.Boolean) #True when acco
```

The types of columns are the first arguments written. The strings have a maximum length of 150 characters.

```
id = db.Column(db.Integer, primary_key=True)
```

The primary key is id, which will be used to distinguish between the different user accounts.

Reference

TechWithTim
Github: https://github.com/techwithtim/Flask-Web-App-Tutorial/blob/main/website/models.py

__init__.py

The code added to this file is responsible for creating a database if one is not found
To begin with, I had to import path from OS and User from the database.

```
db = SQLAlchemy()
```

db is defined in this file and is SQLALchemy

```
DB_NAME = 'database.db' # returns the name of a specified database
```

DB_NAME is a function that returns the name of the specified database: database.db

```
app.config['SQLALCHEMY_DATABASE_URI'] = f'sqlite:///{DB_NAME}' #SQLAlchemy is stored in this location
db.init_app(app) # initialise the database by giving it the flask app
```

This line is responsible for storing the database in the website folder, inside the directory of where this init file is

```
from .database import User
```

Every time the serves is ran, the program needs to make sure the imported classes from the database are found when before initialising or creating the database - To do this the class is imported from the database file

```
#checking if the database exists already to determing whether one needs to be made or not
def create_database(app):
    if not path.exists('website/' + DB_NAME):
        db.create_all(app=app)
        print ('Created Database')
```

Next a new function responsible for checking if the database exists or not was coded. It determines whether a database needs to be made or not.

- A database will be made if one isn't found
- No database will be made if one is found

```
from os import path
```

path is an environmental variable that contains a list of paths to folders

```
if not path.exists('website/' + DB_NAME):
```

This line checks to see if there isn't an existing path or an open file descriptor. If this is the case then the code that follows will create a database

```python
# Creating an instance of the Flask class
def create_app():
    app = Flask(__name__) # initialising the app, __name__ used so that Flask knows where to look for templates etc...
    app.config['SECRET_KEY'] = 'sfibdjkfhb hdfjbs' # This is to sign session cookies for protection

    app.config['SQLALCHEMY_DATABASE_URI'] = f'sqlite:///{DB_NAME}' #SQLAlchemy is stored in this location - stores databas
    db.init_app(app) # initialise the database by giving it the flask app
```

```python
    create_database(app)

    return app
```

For this function to work it needs to be invoked, so it is invoked in the crearte_app function mentioned in **Module 1**

backend.py

In this file, I had to import more libraries: db, User, generate_password_hash, check_password_hash

- db and User are my defined modules to be imported
- generate_password_hash and check_password_hash were imported from werkzeug.security: This will be used to hash a password

```python
from flask import Blueprint, render_template, request, flash, redirect, url_for
from . database import User
from . import db
    #help hash a password from flask_login - hashing a password
from werkzeug.security import generate_password_hash, check_password_hash
```

```python
#creating a new user
new_user = User(first_name=first_name, second_name=second_name, email=email, password=generate_password_hash(password, method='sha256')) # Defining a user
db.session.add(new_user) # add user to database
db.session.commit() # update the database
flash('Account Created', category = 'success')
return redirect(url_for('views.home')) # redirect user to homepage of website
```

Next the information entered in the form needs to be validated against some rules before being added to the database.

Then each user's new account that passes the checks needs to be added to the database. To start off with a variable is created which will be used to add the data to the database. Then the table it is going to be added to is next, here it takes in the arguments first_name, second_name, email and password.

- These are assigned to the column variables of the table which are circled in purple.
- Circled in orange are the variable names which were gotten using the request.get() function for the forms post method

```python
new_user = User(first_name=first_name, second_name=second_name, email=email, password=generate_password_hash(password, method='sha256'))
```

```python
db.session.add(new_user) # add user to database
db.session.commit() # update the database
```

The variable new_user needs is added to the database

Validation

backend.py

```
#validation

user = User.query.filter_by(email=email).first()
if user:
    flash('Email already exists.', category='error')

# category allows you to display the message in a different colour
if len(first_name) < 2:
    flash('First name must be greater than 1 characters', category='error')
elif len(email)<4:
    flash('Email must be greater than 3 characters.', category='error')
elif len(password) < 7:
    flash('Password must be greater than 6 characters.', category='error')
elif password != password_confirm:
    flash('Passwords do not match.', category='error')
```

```
user = User.query.filter_by(email=email).first()
```

This line of code defines the variable ‘user’ and queries the User database - it searches for an email that is the same as the one entered on the Create Account form

```
if user:
    flash('Email already exists.', category='error')
```

Next the program will check if a user exists. If it does then a display error is displayed. If it doesn’t then this line isn't parsed

Different messages are flashed throughout the program. I have used two categories: error and success

- Errors display a red message
- Success’ display a green message

When the length of the first name is less than 2 then an error message is displayed

```
if len(first_name) < 2:
    flash('First name must be greater than 1 characters', category='error')
```

When the length of the email is less than 4 then an error message is displayed

```
elif len(email)<4:
    flash('Email must be greater than 3 characters.', category='error')
```

When the length of the password is less than 7 then an error message is displayed

```
elif len(password) < 7:
    flash('Password must be greater than 6 characters.', category='error')
```

When the password and confirm password are not the same, then an error message is displayed

```
elif password != password_confirm:
    flash('Passwords do not match.', category='error')
```

When the category is error, the code below is ran

This is what gives the alert the possibility to be dismissed

This is what makes the message red - it is a predefined class from the linked botstrap

```
{% with messages = get_flashed_messages(with_categories=true) %}
{% if messages %} {% for category, message in messages %}
{% if category =='error' %}
<div class="alert alert-danger alter-dismissable fade show" role="alert">
  {{ message }}
  <button type="button" class="close" data-dismiss="alert">
    <span aria-hidden="true">&times;</span>
  </button>
</div>
```

This is the message displayed - the message is written in the python file which invoked a flash

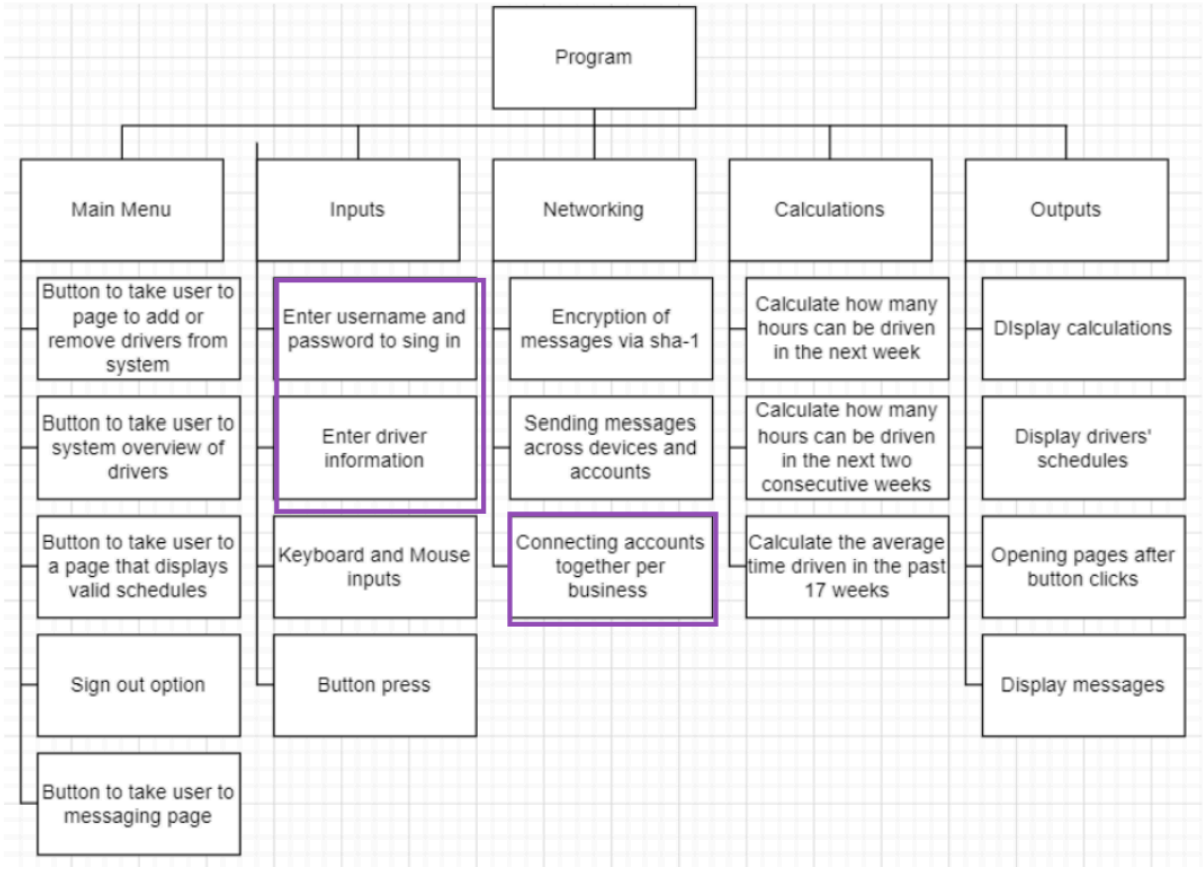
This makes the button to dismiss the message a small x icon

This creates the button that dismisses the alert

```
@backend.route('/logout') #creating a decorator, when the URL is /logout the function below is invoked
@login_required #decorator - makes sure, this rout is inaccesible unless user is logged in
def logout():
    logout_user() #logs out current user
    return redirect(url_for('backend.login')) #Brings user back to login page when they've signed out
```

This logs out the user. Once the user is logged out, they are redirected to the login page

Relation to top down modular design



This relates to the top down modular design because the users accounts are stored in the same table and are queried together and separately at different parts of the program. This allows managers to interact with driver accounts

- The tables are also a place information is added to

Testing

Test No.	Explanation and justification	Expected outcome / criteria	Actual outcome	Pass or fail
1	The test will check if a database is created, and the correct columns are added. This is important for the completion of the program so that, user information is stored and accessible for different functions	A new file called, database.db should be made When the file is opened it should show the column names	TypeError in the console	fail
	Evidence: The result in the console `TypeError: SQLAlchemy.create_all() got an unexpected keyword argument 'app'`			
2	This test will test that on account creating, users information is stored in the table For this test I downloaded sqlitebrowser: this allows me to see the information for each sql table used. I will use this to query specific data and check its output To do this test I will create an account, with the fields entered as: First name ← John Second name ← Doe Email ← JohnDoe@gmail.com Password ← 123456789 Confirmed Password ← 123456789 Then in the external app DB browser for sqlite, I will run the line: select * from User where email = 'JohnDoe@gmail.com'; This is important for the completion of the program so that the user information is stored safely and correctly	Outputted password will be hashed and different from entered password The other fields will be the same as the ones I entered	N/A - Relies on test 1 to work	N/A
3	This is testing the validation in place to make sure it is functional This will test whether a user can enter a 1 letter username - I will enter the letter J for this test	Red error message saying 'First name must be greater than 1 characters'	N/A - Relies on test 1 to work	N/A
4	This is testing the validation in place to make sure it is functional	Red error message saying 'Email must be	N/A - Relies on test 1 to work	N/A

	This will test whether a user can enter an email that is less than 4 characters - I will leave it blank	greater than 3 characters'		
5	This is testing the validation in place to make sure it is functional This will test whether a user can enter a password that is less than 7 characters - I will enter: 12345	Red error message saying 'password must be greater than 6 characters'	N/A - Relies on test 1 to work	N/A
6	This is testing the validation in place to make sure it is functional This will test whether a user can enter different passwords in the password field and confirm password field - I will enter - Password: TestTest - Confirm password: DifferentPass	Red error message saying 'passwords do not match'	N/A - Relies on test 1 to work	N/A
7	This test is to make sure users can log out	Redirect to login page	N/A - Relies on test 1 to work	N/A

Prototype 2

Prototype 1 failed overall because the first test failed, preventing the file from running and continuing with other tests: This leads to remedial action being needed.
The error in test no. 1 is due to outdated software. The database frame I'm using is flasks SQL and it no longer accepts ‘app’ as an argument. It now requires an active Flask application context.

Remedial action

In the __init__.py file, I deleted the function create_database() and removed the line of code that imports the tables from the database file. As I have removed this from my code, I could remove the line of code at the beginning of the file that imports path from the os library.

```
from os import path
```

```
from .database import User
create_database(app)
```

```
def create_database(app):
    if not path.exists('website/' + DB_NAME):
        db.create_all(app=app)
        print('Created Database!')
```

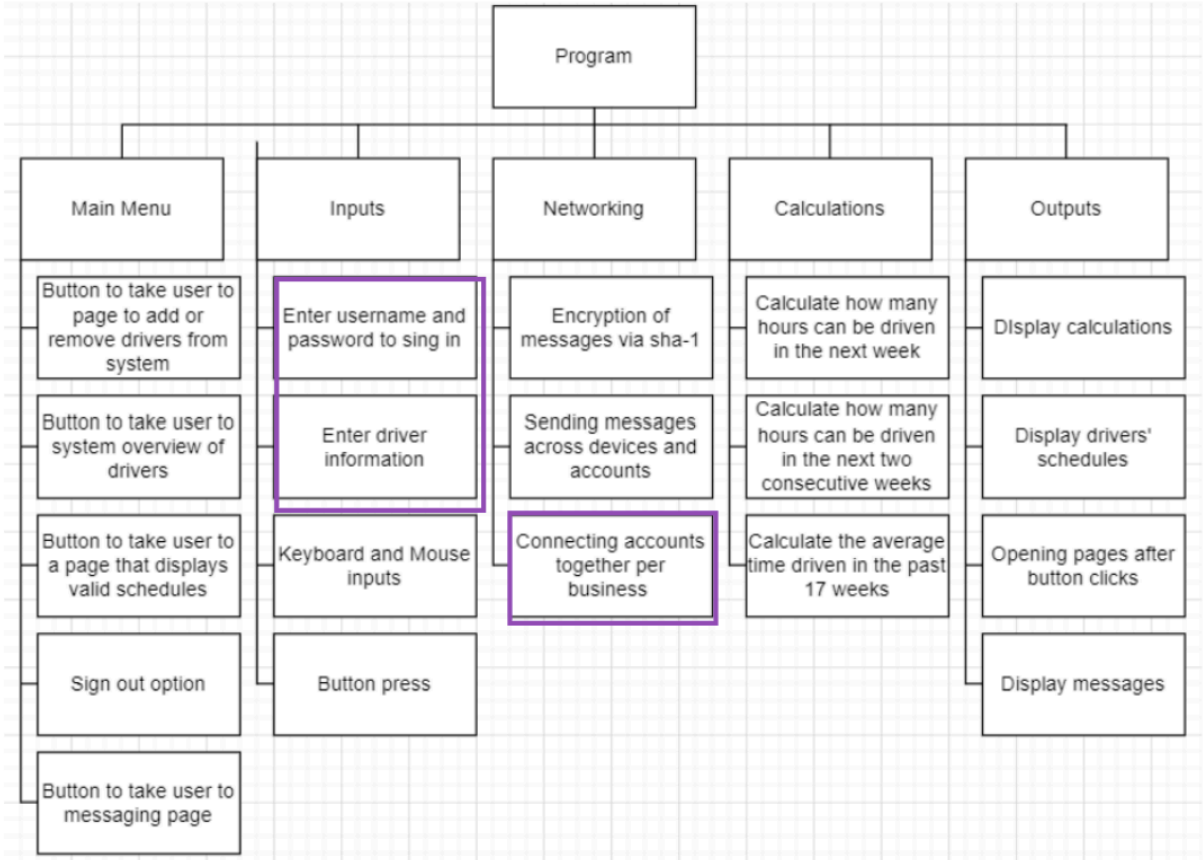
This has been replaced with two lines of code - this works because SQL will automatically not overwrite an existing file, with the only time it created a database when an error is created

```
#creating a database - sql now automatically does this and eliminates need for the create_database function
with app.app_context():
    db.create_all()
```

Reference:

The solution to this problem was found using stack overflow:
https://stackoverflow.com/questions/73968584/flask-sqlalchemy-db-create-all-got-an-unexpected-keyword-argument-app

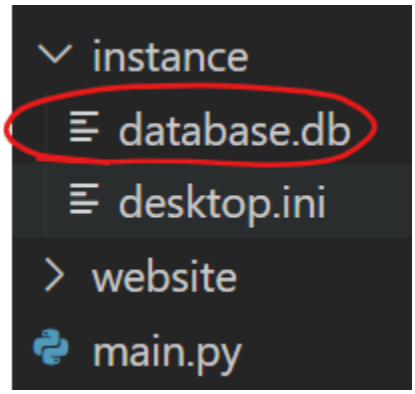
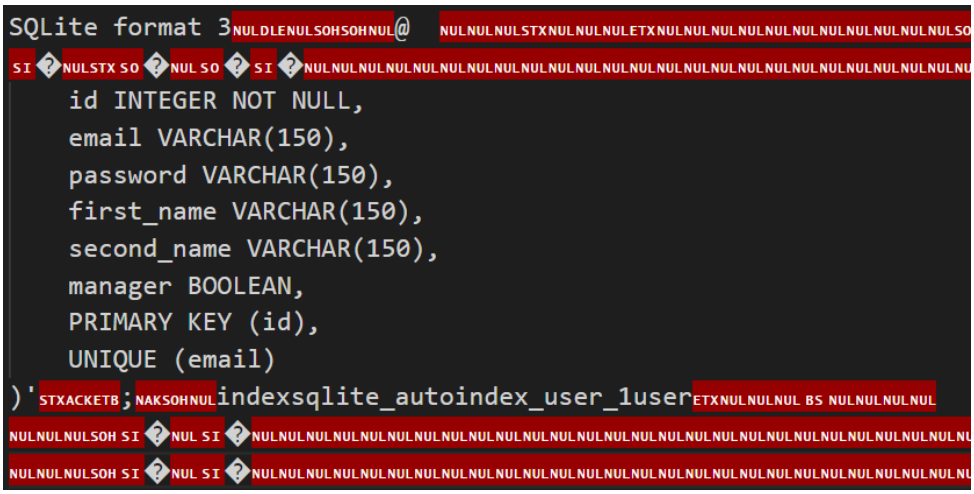
Relation to top down modular design



This relates to the top down modular design because the users accounts are stored in the same table and are queried together and separately at different parts of the program. This allows managers to interact with driver accounts

- The tables are also a place information is added to

Testing

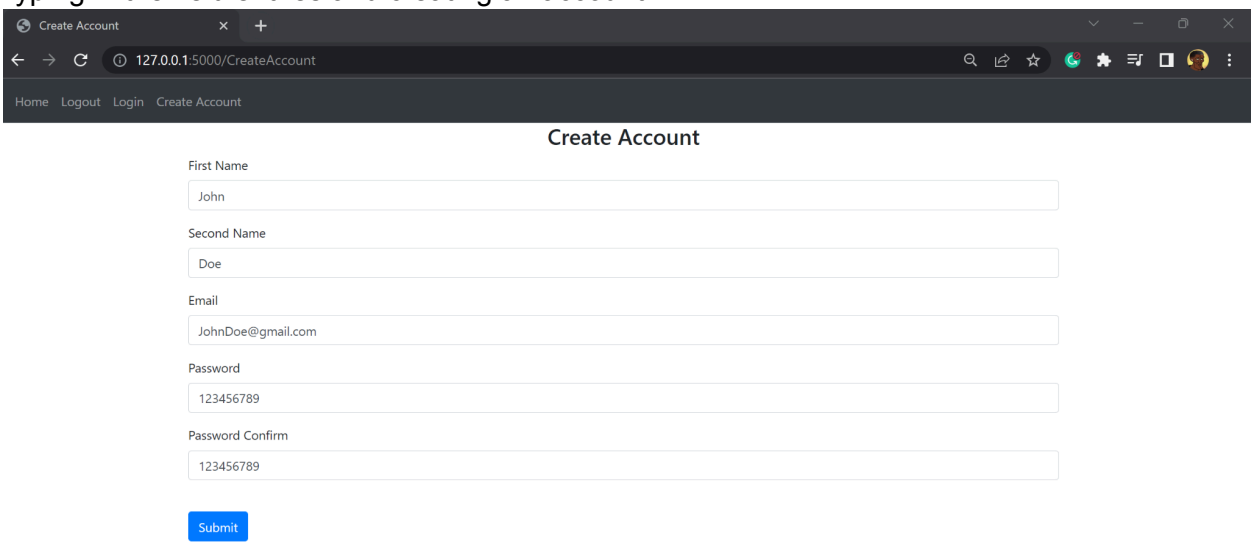
Test No.	Explanation and justification	Expected outcome / criteria	Actual outcome	Pass or fail
1	The test will check if a database is created, and the correct columns are added. This is important for the completion of the program so that, user information is stored and accessible for different functions	A new file called, database.db should be made When the file is opened it should show the column names	New file created called database.db Column names visible: Id,email. first_name, password, second_name, manager	pass
	Evidence:   			
2	This test will test that on account creating, users information is stored in the table For this test I downloaded sqlitebrowser: this allows me to see the information for each sql table used. I will use this to query specific data and check its output To do this test I will create an account, with the fields entered as: First name ← John Second name ← Doe Email ← JohnDoe@gmail.com Password ← 123456789 Confirmed Password ← 123456789 Then in the external app DB browser for sqlite, I will run the	Outputted password will be hashed and different from entered password The other fields will be the same as the ones I entered	Password was hashed, columns and the information added is found	pass

line:
select * from User where email = 'JohnDoe@gmail.com';

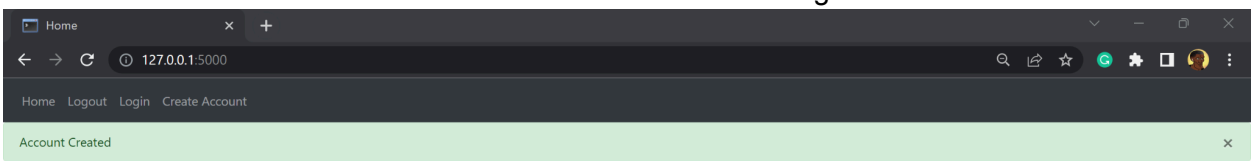
This is important for the completion of the program so that the user information is stored safely and correctly

Evidence:

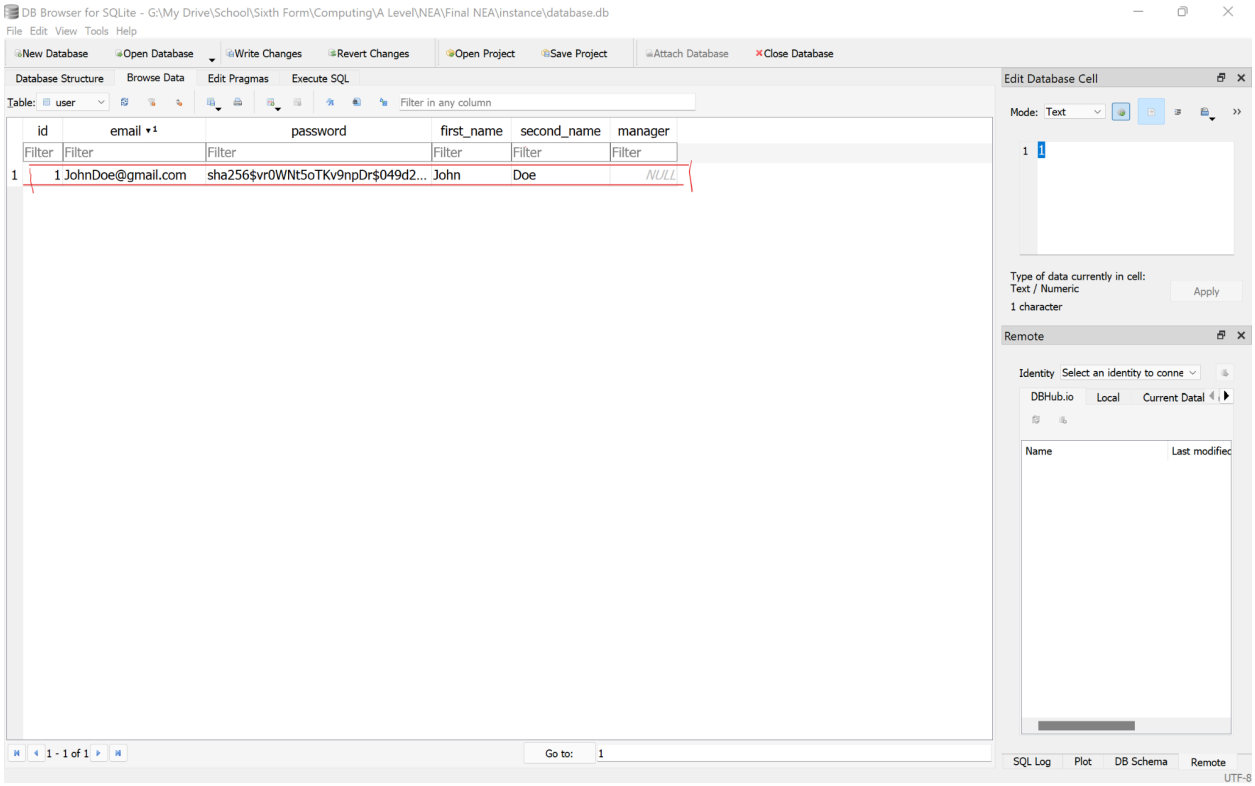
Typing in the field entries and creating an account:



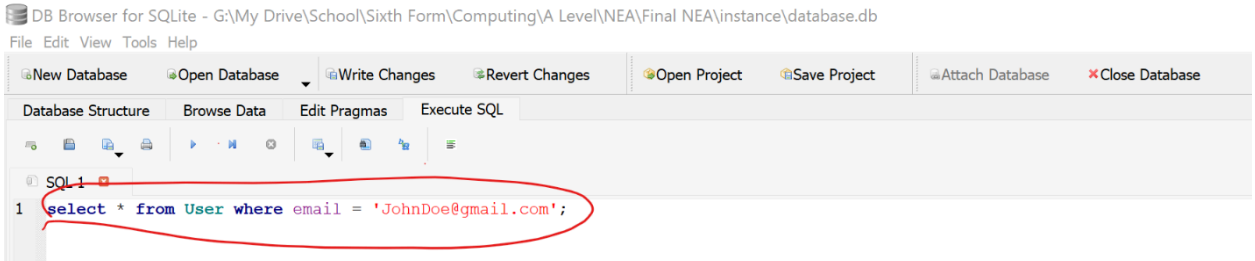
This is the confirmation that the account has been created using the flash function



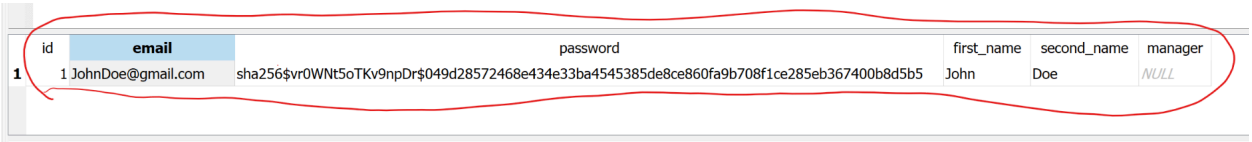
This is the table and all the current users accounts stored in the table



This is the query to output all the data in a record when email is JohnDoe@gmail.com

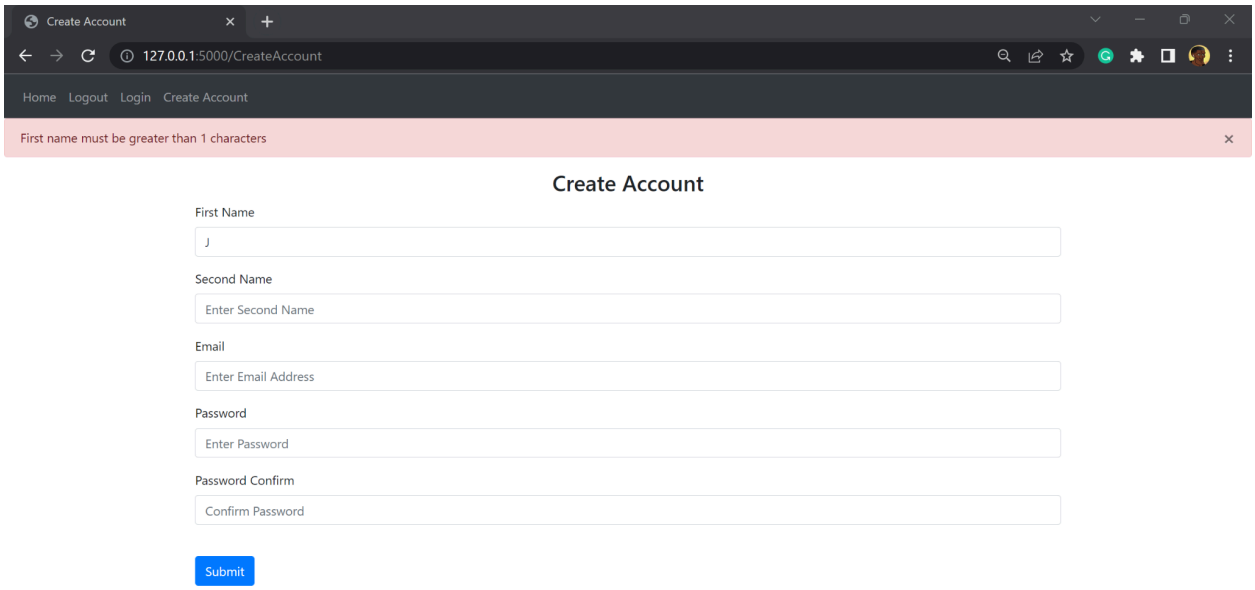
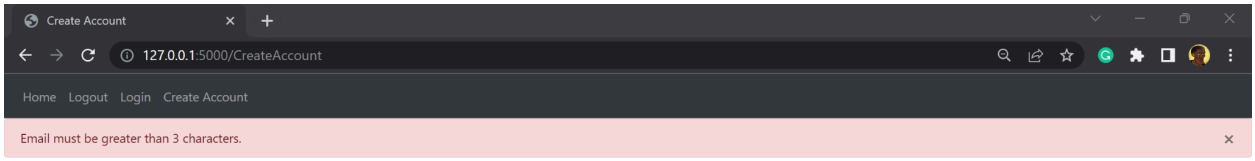
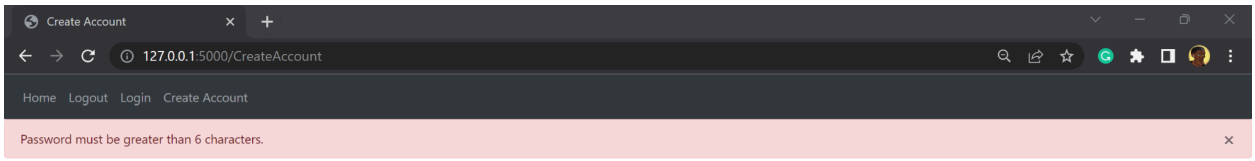


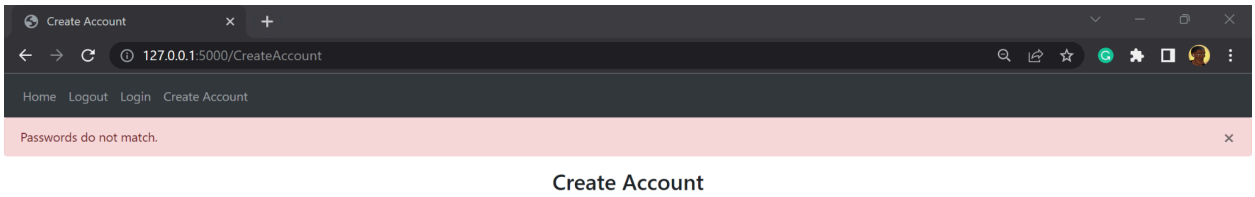
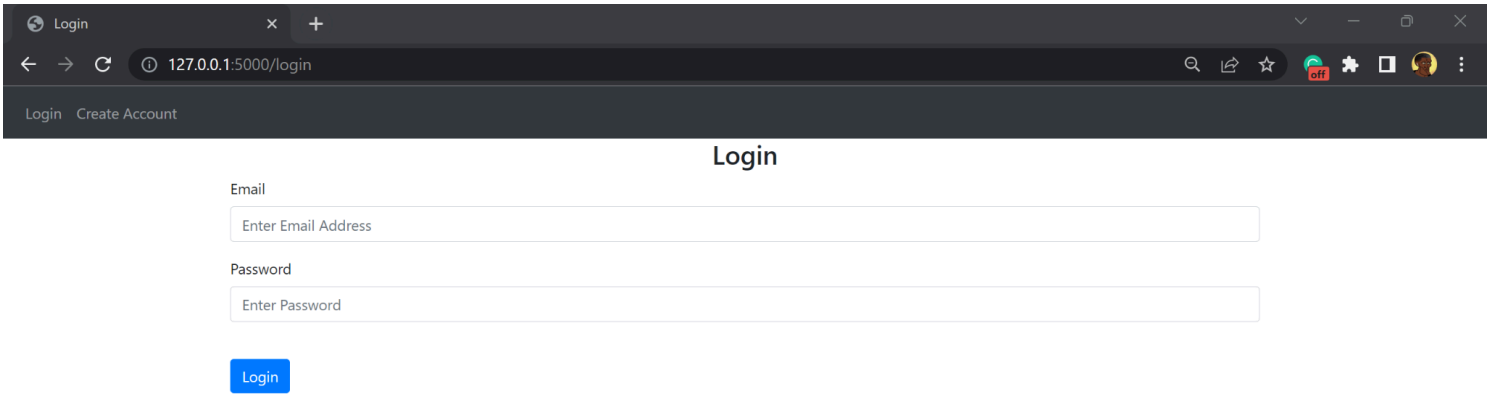
This is the output when the query is ran: the password has been stored as a hash



	id	email	password	first_name	second_name	manager
1	1	JohnDoe@gmail.com	sha256$vr0WNt5oTKv9npDr$049d28572468e434e33ba4545385de8ce860fa9b708f1ce285eb367400b8d5b5	John	Doe	NULL

3	This is testing the validation in place to make sure it is functional This will test whether a user can enter a 1 letter username - I will enter the letter J for this test	Red error message saying 'First name must be greater than 1 characters'	Red error message saying 'First name must be greater than 1 characters'	pass

	Evidence: 			
4	This is testing the validation in place to make sure it is functional This will test whether a user can enter an email that is less than 4 characters - I will leave it blank	Red error message saying 'Email must be greater than 3 characters'	Red error message saying 'Email must be greater than 3 characters'	pass
	Evidence: 			
5	This is testing the validation in place to make sure it is functional This will test whether a user can enter a password that is less than 7 characters - I will enter: 12345	Red error message saying 'password must be greater than 6 characters'	Red error message saying 'password must be greater than 6 characters'	pass
	Evidence: 			

6	This is testing the validation in place to make sure it is functional This will test whether a user can enter different passwords in the password field and confirm password field - I will enter - Password: TestTest - Confirm password: DifferentPass	Red error message saying 'passwords do not match'	Red error message saying 'passwords do not match'	pass
	Evidence:  			
7	This test is to make sure users can log out	Redirect to login page	N/A - Relies on test 1 to work	Pass
	Evidence:  			

Prototype 3

This program requires more tables to reach the final solution. This prototype focuses on the table that records the jobs and schedules of the drivers. The table has a foreign key to link to the User table created throughout prototype 1-3.

database.py, Job

```
#Creating a table for the schedules
class Job(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    date = db.Column(db.String)
    time = db.Column(db.String)
    tractorID = db.Column(db.String(20))
    start_location = db.Column(db.String(150))
    end_location = db.Column(db.String(150))
    user = db.Column(db.String(150))
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'))
```

The foreign key to link the table to the User table is user_id.

The table is called Job because each driver is allocated a job by the manager. The records of Job make up a drivers schedule

The column 'date' will store the information sent by a html form that uses the 'date' function type in html.
The column 'time' takes in information that is sent using the 'time' function type.

tractorID, start_location and end_location are also columns and their fields come from the html form.
The user column is the first name of the driver the manager adds the job for, so that the jobs are assigned to drivers.

```
jobs = db.relationship('Job')
```

Due to the foreign key in this table, a relationship needs to be defined with the User table. This is done using the keyword 'relationship'

database.py, Days

```
class Days(db.Model):
    id=db.Column(db.Integer, primary_key=True)
    week = db.Column(db.Integer)
    monday = db.Column(db.Integer)
    tuesday = db.Column(db.Integer)
    wednesday = db.Column(db.Integer)
    thursday = db.Column(db.Integer)
    friday = db.Column(db.Integer)
    saturday = db.Column(db.Integer)
    sunday = db.Column(db.Integer)
    user = db.Column(db.String(150))
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'))
```

This table is called Days, and it records the hours driven by each driver per week.

This is saved in a table because it will be called on when doing calculations to help the manager complete their schedules.

user is a column and is used to save the first name of the driver selected.

user_id is the foreign key to link the table with the User table.

```
hours = db.relationship('Days')
```

This line of code is in the Users table to link the Days table

database.py, Week

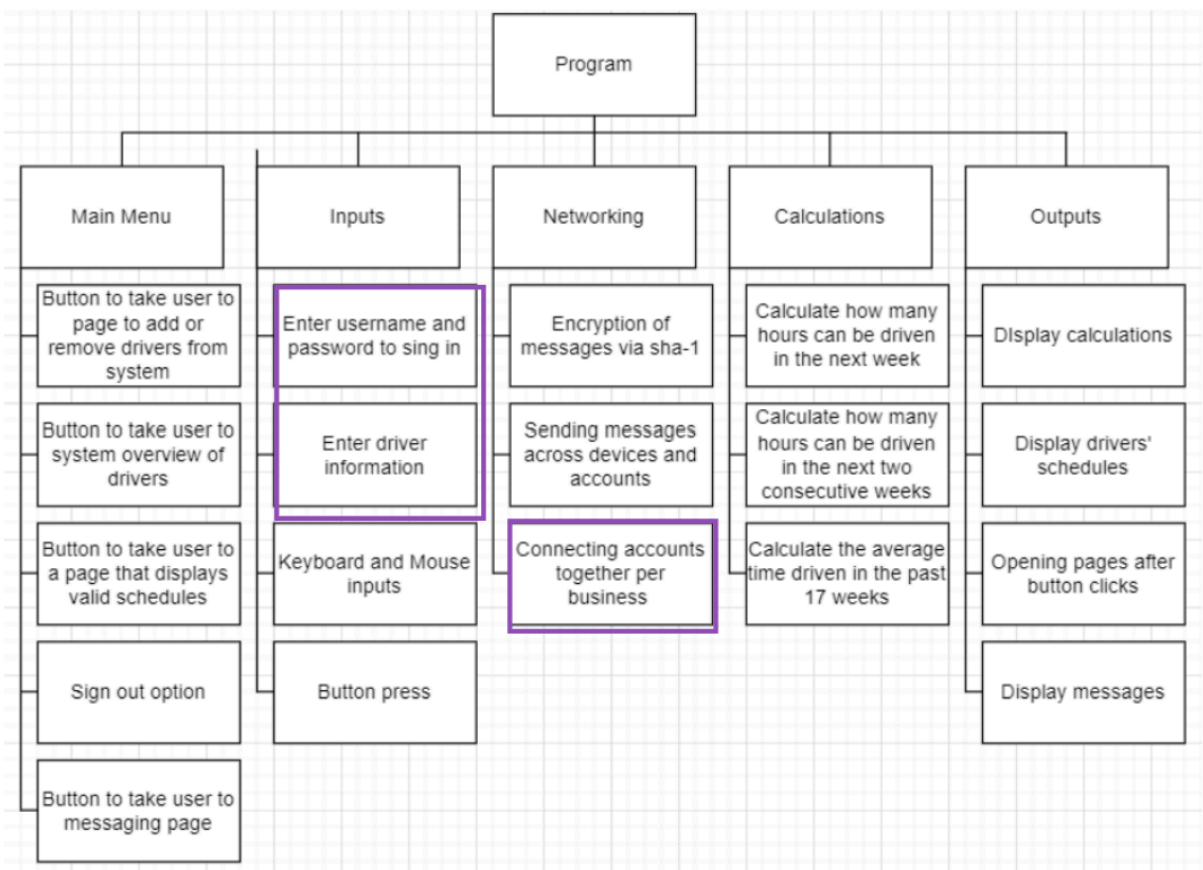
```
# Creating a table for the schedule days
class Week(db.Model):
    id=db.Column(db.Integer, primary_key=True)
    week_no = db.Column(db.Integer)
    monday_date = db.Column(db.String)
```

This table will be used to help calculate the mean hours driven by drivers in multiple time periods.

This table will call on the specified last weeks in the table. The Days table will use the value stored in week_no for cross reference and data manipulation to help with outputting and formatting of information.

monday_date will be used for validation.

Relation to top down modular design



This relates to the top down modular design because the users accounts are stored in the same table and are queried together and separately at different parts of the program. This

allows managers to interact with driver accounts

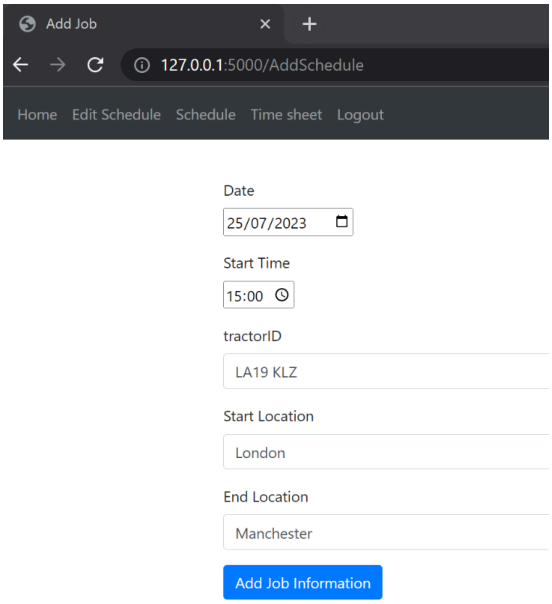
- The tables are also a place information is added to

Testing

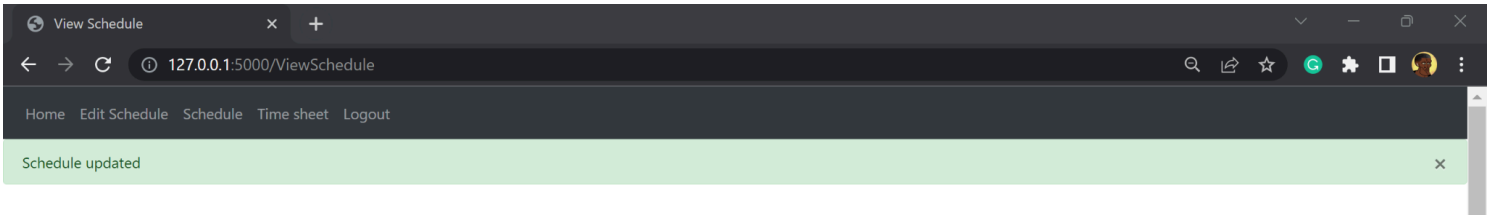
Test No.	Explanation and justification	Expected outcome / criteria	Actual outcome	Pass or fail
1	This is a test that information is correctly stored in the Job table This will be carried out by adding a job as normal on the web page. A message should appear saying job added The information added will be July 25, at 15:00, LA19 KLZ, London, Manchester, John Can view the table using DB Browser (SQLite) app	A green pop up message flashed The record added showing in the table	A green pop up message flashed The record added showing in the table	pass

Evidence:

Test data added:



Outcome:



Date: 2023-07-25 **Time:** 15:00 **TractorID:** LA19 KLZ **Start Location:** London **End Location:** Manchester **Driver:** John

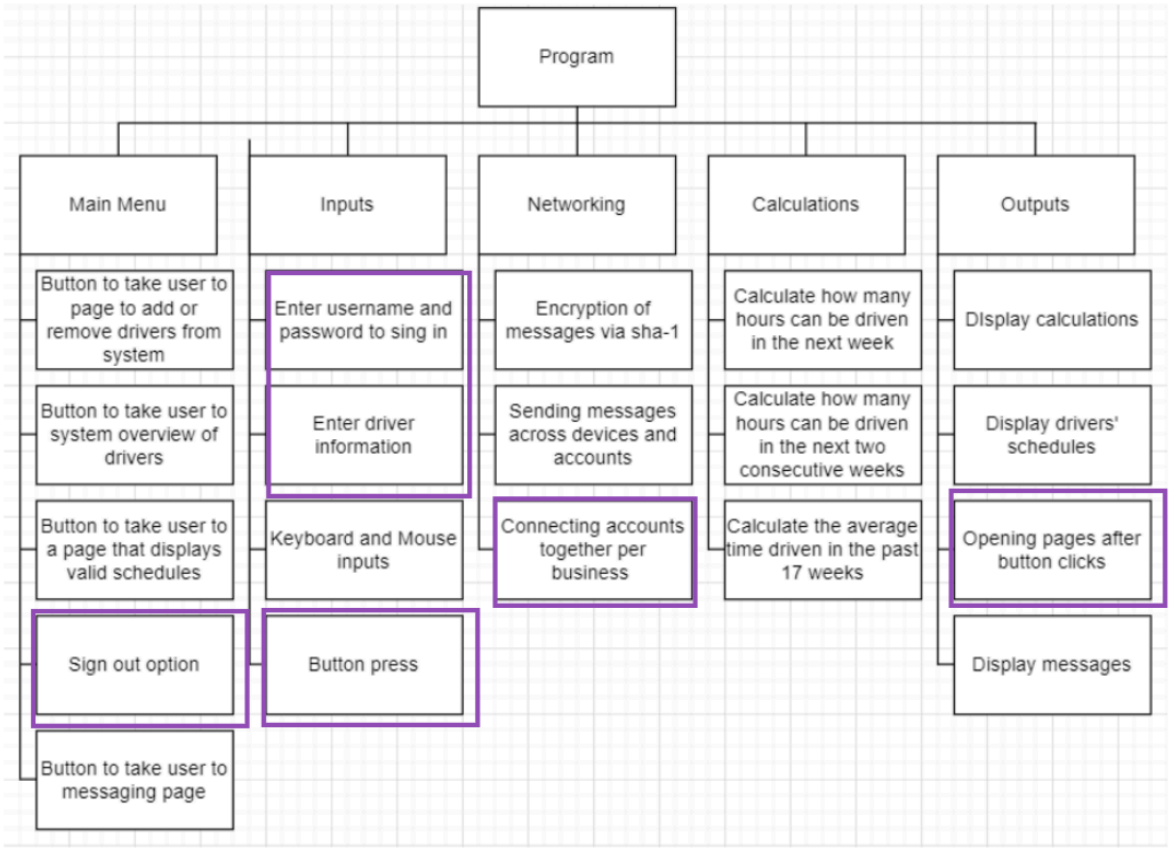
DB Browser (SQLite) app result for Job table:

15	15	2023-07-25	15:00	LA19 KLZ	London	Manchester	John	1

Relation to top down modular design

This relates to the top down modular design from the design section because buttons take the user to the respective pages and also allows users to submit information to the database to create or log into accounts

Button press is used for submission of information



The logout button signs out the user

Comments for maintenance

database.py

```
from . import db # importing from the package db which is in the init file
from flask_login import UserMixin # custom class to inherit giving user object specific things to login

#Creating a table for user information using a class object
class User (db.Model, UserMixin): #UserMixin used for implementations of this objects properties
    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(150), unique=True) #Identifies this column as the primary key
    password = db.Column(db.String(150)) #Creates a column for password that accepts strings
    first_name = db.Column(db.String(150)) #Sets first_name character limit to 150
    second_name = db.Column(db.String(150))
    manager = db.Column(db.Boolean) #True when account level is manager False when it is not
```

Comments in grey to describe what each line does
Comments as to why each import is imported and their use

backend.py

```
db.session.add(new_user) # add user to database
db.session.commit() # update the database
```

The comments describing what these SQL lines do and stating that these directly affect the database

```
#validation
```

Comment to state where the validation for the code is and where to add further validation in future to help with traversing the code for errors

```
# category allows you to display the message in a different colour
```

Explaining what some syntax means that may be new to programmers

```
else:
    #creating a new user
    new_user = User(first_name=first_name, second_name=second_name, email=email, password=generate_password_hash(password, method='sh
    db.session.add(new_user) # add user to database
    db.session.commit() # update the database
    flash('Account Created', category = 'success')
    login_user(new_user, remember=True)
    return redirect(url_for('backend.home')) # redirect user to homepage of website, home is the function - backend is the name of bl

urn render_template("create_account.html", user=current_user)
```

Comment above block of code to describe what the lines do when working together

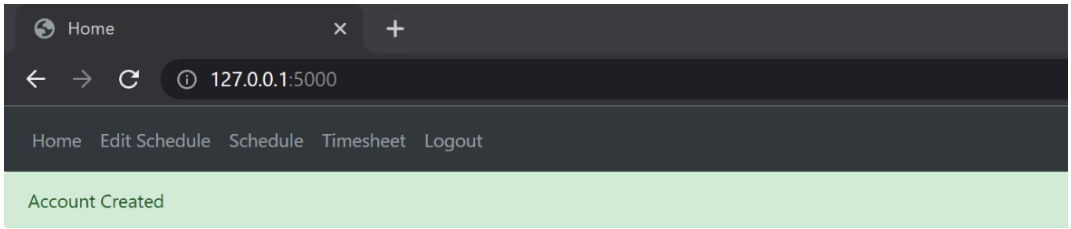
Testing and Evaluation

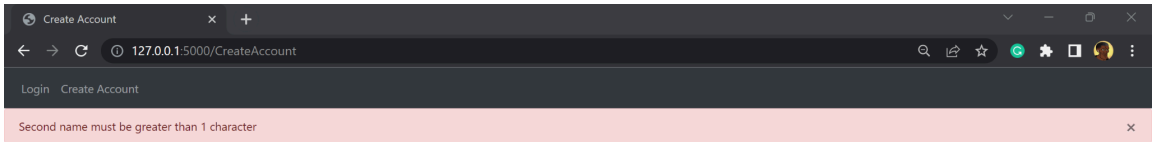
4.1 Post development testing for function and robustness

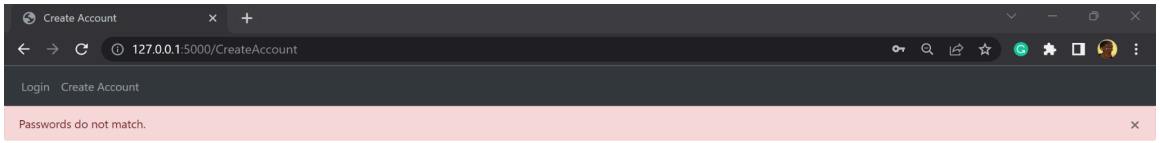
For this section the functionality of each function is tested using erroneous, boundary, extreme and normal inputs to try and replicate multiple real uses of the program.
The program can be split into two main sections for testing such as authentication and non-authentication views

4.1.1 Authentication:

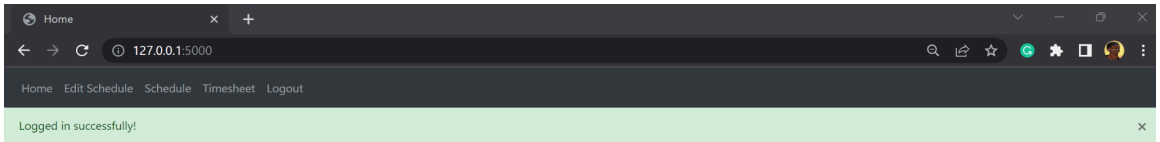
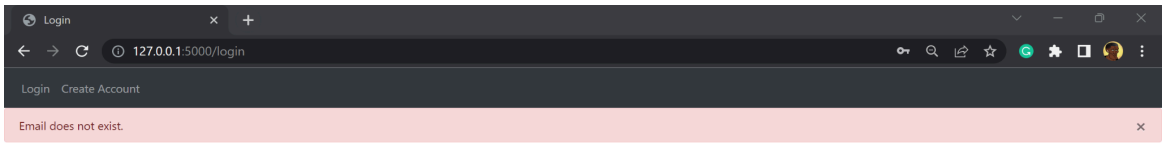
Account Creation

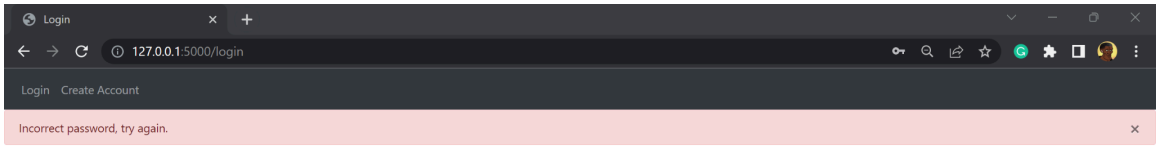
Account creation				
Function being tested	**Input**	**Expected result**	**Actual results**	**Met: Fully/partially/Not**
Creating an account The create account function is tested by entering realistic information likely to be entered by a user. **This test will validate success criteria 5 from 1.6.3 Success Criteria**	The input here is Normal First Name Paul Second Name George Email PG@gmail.com Password ••••••••• Password Confirm •••••••••	Redirect to the home page with a green message flashing on the screen saying the account has been created	Redirect to the home page with a green message flashing on the screen saying the account has been created	Fully
	Evidence:  			
Creating an account validation The create account function is tested by	The test data input is erroneous	A red error message saying that the second name needs to be entered	An error message saying that the second name must be more than 1 character	Fully

entering incorrectly formatted information. **This test will validate success criteria 5 from 1.6.3 Success Criteria**	First Name Alex Second Name Enter Second Name Email Alex@gmail.com Password •••••••• Password Confirm •••••••••			
	Evidence:  			
Creating an account validation The create account function is tested by entering mis matching passwords **This test will validate success criteria 5 from 1.6.3 Success Criteria**	The test data input entered is erroneous First Name Forest Second Name Gump Email Gump@me.com Password •••••••••• Password Confirm ••••••••••••••••••••••	Red error message saying that the passwords don't match	Red error message saying that the passwords don't match	Fully
	Evidence:			

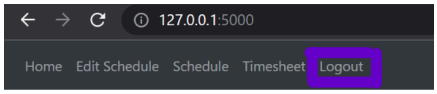
	

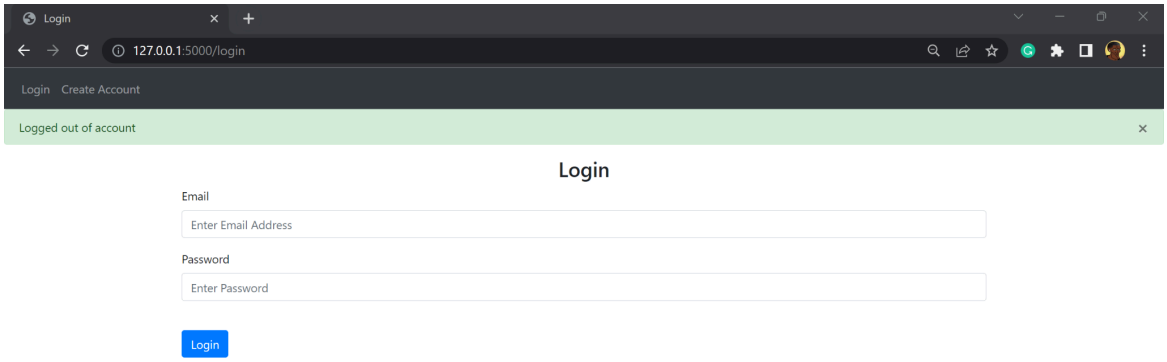
Account login

Account login				
Function being tested	**Input**	**Expected result**	**Actual results**	**Met: Fully/partially/Not**
Logging into existing account The test simulates a real user signing into their actual account **This test will validate success criteria 1 from 1.6.3 Success Criteria**	The test data entered is normal Email PG@gmail.com Password •••••••••	Redirect to the home page and display a green flash message saying logged in successfully	Redirect to the home page and display a green flash message saying logged in successfully	Fully
	Evidence: 			
Logging into account that doesn't exist - validation The test simulates a user trying to sign into an account in a situation where they confuse their emails and use the wrong one that doesn't exist on the database **This test will validate success**	The test data entered is erroneous Email FakeEmail@gmail.com Password •••••••••••	Red error message saying that the account doesn't exist	Red error message saying that the account doesn't exist	Fully
	Evidence: 			

criteria 1 from 1.6.3 Success Criteria				
Logging into account that exists but with the wrong email - validation The test simulates a scenario where the user enters the wrong password to an existing account and can't gain access **This test will validate success criteria 1 from 1.6.3 Success Criteria**	The test data entered is erroneous Email PG@gmail.com Password •••••••••••••	A red error message saying incorrect password is displayed	A red error message saying incorrect password is displayed	Fully
	Evidence:  			

Account logout

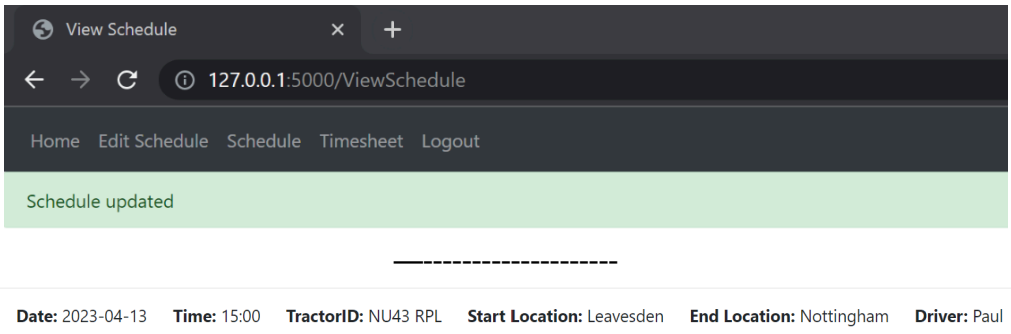
Account log out				
Function being tested	**Input**	**Expected result**	**Actual results**	**Met: Fully/ partially/ Not**
Logging out of an account The user can only log out of an account when they are signed into one as the option doesn't show until signed in	The test data entered is normal The method of input is through button press  	Redirect to the login page and display a green flash message saying logged out successfully	Redirect to the login page and display a green flash message saying logged out successfully	Fully
	Evidence:			

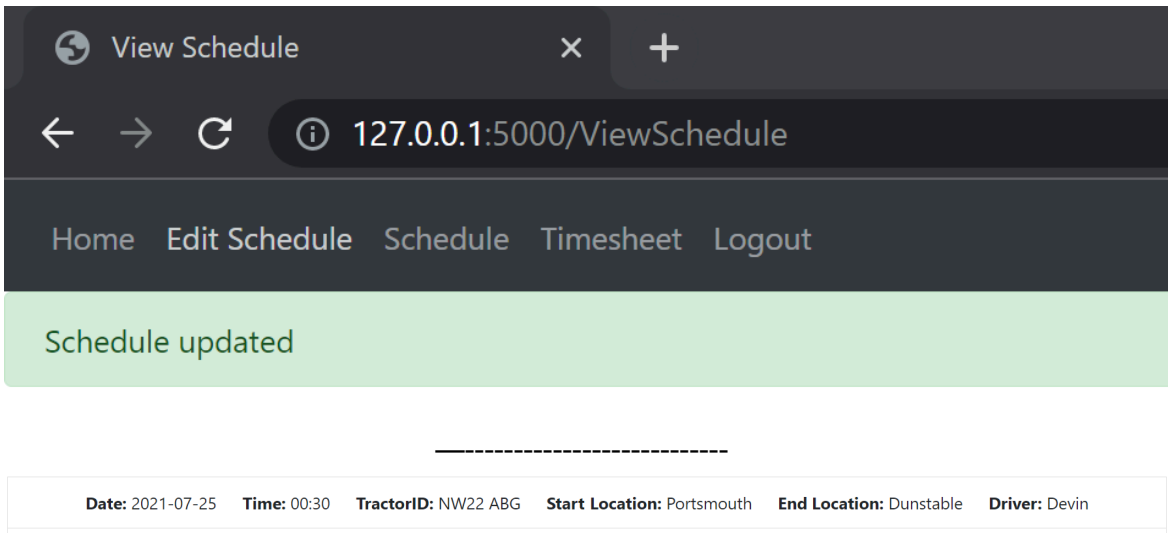
4.1.2 Views:

Edit schedules

Edit schedules				
Function being tested	**Input**	**Expected result**	**Actual results**	**Met: Fully/partially/Not**
Adding a job to the schedule This test is for the manager account only, as they are the only ones with access to this **This test will validate success criteria 4 from 1.6.3 Success Criteria** **This test will validate success criteria 8 from 1.6.4.2 Input usability success criteria** **This test will validate success**	Selecting a Paul, and filling out the form The test data is Normal Edit Schedules Driver Paul Submit ----------	After selecting the driver Paul, the page redirects to a form to fill out the Date, start time, TractorID and locations of job Once this submitted the user is redirected to the schedules page of all the drivers and the added job can be found displayed there	After selecting the driver Paul, the page redirects to a form to fill out the Date, start time, TractorID and locations of job Once this submitted the user is redirected to the schedules page of all the drivers and the added job can be found displayed there	Fully

criteria 12 from 1.6.4.3 Output usability success criteria	Date 13/04/2023 Start Time 15:00 tractorID NU43 RPL Start Location Leavesden End Location Nottingham			
	Evidence:  			

Adding a job to the schedule **This test will validate success criteria 4 from 1.6.3 Success Criteria** **This test will validate success criteria 8 from 1.6.4.2 Input usability success criteria** **This test will validate success criteria 12 from 1.6.4.3 Output usability success criteria**	Selecting Devin, and filling out the form, the date will be set to a day before the current day The test data is erroneous Edit Schedules Driver Devin Submit ———------- Date 25/07/2021 Start Time 00:30 tractorID NW22 ABG Start Location Portsmouth End Location Dunstable	Red error message saying that the data has to be after the current date	A green successful message saying that the schedule has been updated, and the new job showing up as a job	Fail The job should not be added because the date entered is before the current date
	Evidence:			

<table>
<tr><td></td><td colspan="4"> 
</td></tr>
<tr><td rowspan="2">Adding a job to the schedule

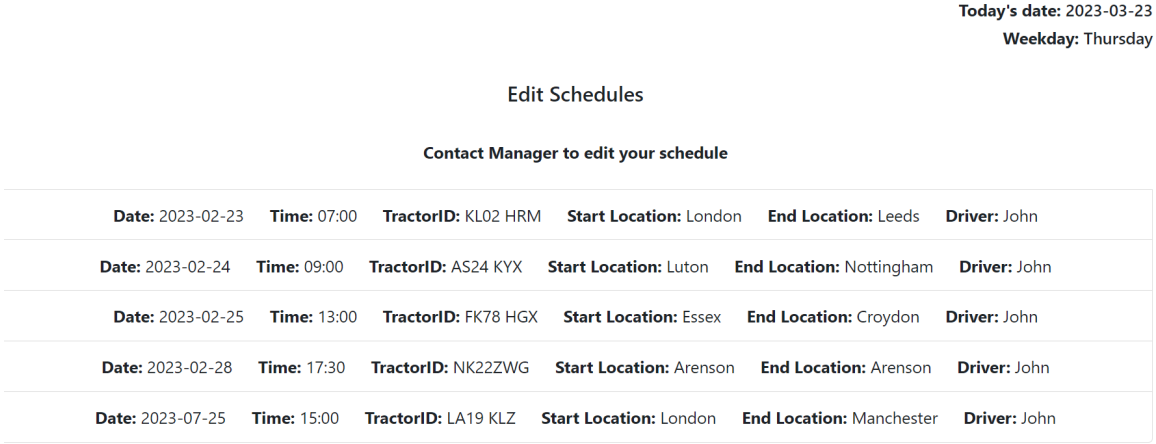
This test is done on a driver's account who does not have manager roles

This test will validate success criteria 4 from 1.6.3 Success Criteria

This test will validate success criteria 11 from 1.6.4.3 Output usability success criteria

This test will validate success criteria 12 from 1.6.4.3 Output usability success criteria

This test will validate success criteria 13 from</td><td>On button press on edit schedule on the navbar</td><td>The user can only see their schedule and is told to contact their manager if they wish to edit their schedule</td><td>The user is told to contact manager to edit their schedule They are then presented with their schedule</td><td>Partially

The user shouldn’t have the option to edit their schedule on the navbar</td></tr>
<tr><td colspan="4">Evidence:
 
</td></tr>
</table>

1.6.4.3 Output usability success criteria	

View schedule

Edit schedules				
Function being tested	**Input**	**Expected result**	**Actual results**	**Met: Fully/partially/Not**
Viewing the schedule The manager account has access to all drivers schedules **This test will validate success criteria 12 from 1.6.4.3 Output usability success criteria** **This test will validate success criteria 13 from 1.6.4.3 Output usability success criteria**	The input for the test is a button press on the navbar menu Schedule	A list of all jobs in the schedule ordered by date	A list of all jobs in the schedule ordered by date	Fully

Evidence:

Schedule

Date: 2021-07-25 **Time:** 00:30 **TractorID:** NW22 ABG **Start Location:** Portsmouth **End Location:** Dunstable **Driver:** Devin

Date: 2023-02-09 **Time:** 20:04 **TractorID:** DT37 HCS **Start Location:** Northampton **End Location:** Bristol **Driver:** Devin

Date: 2023-02-19 **Time:** 17:54 **TractorID:** TER3 TYX **Start Location:** London **End Location:** Heathrow **Driver:** Devin

Date: 2023-02-23 **Time:** 07:00 **TractorID:** KL02 HRM **Start Location:** London **End Location:** Leeds **Driver:** John

Date: 2023-02-23 **Time:** 15:02 **TractorID:** GL57 BND **Start Location:** Bedford **End Location:** Stratford **Driver:** Austin

Date: 2023-02-24 **Time:** 09:00 **TractorID:** AS24 KYX **Start Location:** Luton **End Location:** Nottingham **Driver:** John

Date: 2023-02-24 **Time:** 09:00 **TractorID:** LR76 RGM **Start Location:** Lancaster **End Location:** Portsmouth **Driver:** Martin

Date: 2023-02-25 **Time:** 10:00 **TractorID:** RL67 OPB **Start Location:** Newcastle **End Location:** Sheffield **Driver:** Martin

Date: 2023-02-25 **Time:** 13:00 **TractorID:** FK78 HGX **Start Location:** Essex **End Location:** Croydon **Driver:** John

Date: 2023-02-26 **Time:** 13:00 **TractorID:** WE23 KBC **Start Location:** Northampton **End Location:** Arenson **Driver:** Austin

Date: 2023-02-28 **Time:** 17:30 **TractorID:** NK22ZWG **Start Location:** Arenson **End Location:** Arenson **Driver:** John

Date: 2023-03-09 **Time:** 17:15 **TractorID:** GH12 KLP **Start Location:** Canterbury **End Location:** Newcastle **Driver:** Chris

Date: 2023-03-16 **Time:** 18:00 **TractorID:** PL14 KOP **Start Location:** Leeds **End Location:** London **Driver:** Kevin

Date: 2023-03-29 **Time:** 08:30 **TractorID:** TE23 YKS **Start Location:** Leeds **End Location:** Hemel hempstead **Driver:** Martin

Date: 2023-04-13 **Time:** 15:00 **TractorID:** NU43 RPL **Start Location:** Leavesden **End Location:** Nottingham **Driver:** Paul

Date: 2023-07-25 **Time:** 15:00 **TractorID:** LA19 KLZ **Start Location:** London **End Location:** Manchester **Driver:** John

Date: 2023-08-09 **Time:** 20:15 **TractorID:** AB12 CDE **Start Location:** Portsmouth **End Location:** Bournemouth **Driver:** Devin

<table>
<tr><td></td><td colspan="4">—----------
Show driver schedule
Paul Submit</td></tr>
<tr><td rowspan="2">Viewing the schedule

The manager may wish to filter the schedule by driver

This test will validate success criteria 12 from 1.6.4.3 Output usability success criteria

This test will validate success criteria 13 from 1.6.4.3 Output usability success criteria</td><td>This test data type is normal

Chris
Show driver schedule
Chris Submit</td><td>Chris’ schedule is displayed</td><td>The schedule of Chris is displayed</td><td>Fully</td></tr>
<tr><td colspan="4">Evidence:

Schedule for Chris
Date: 2023-03-09 Time: 17:15 TractorID: GH12 KLP Start Location: Canterbury End Location: Newcastle Driver: Chris</td></tr>
<tr><td rowspan="2">Viewing the schedule

The driver doesn’t have manager access roles

Currently schedule related things should be read only as driver

Signed in as John Doe</td><td>Button press on navbar menu for schedule</td><td>The schedule of John Doe and him only is displayed</td><td>The schedule of John Doe and him only is displayed</td><td>Fully</td></tr>
<tr><td colspan="4">Evidence:

Schedule

Date: 2023-02-23 Time: 07:00 TractorID: KL02 HRM Start Location: London End Location: Leeds Driver: John
Date: 2023-02-24 Time: 09:00 TractorID: AS24 KYX Start Location: Luton End Location: Nottingham Driver: John
Date: 2023-02-25 Time: 13:00 TractorID: FK78 HGX Start Location: Essex End Location: Croydon Driver: John
Date: 2023-02-28 Time: 17:30 TractorID: NK22ZWG Start Location: Arenson End Location: Arenson Driver: John
Date: 2023-07-25 Time: 15:00 TractorID: LA19 KLZ Start Location: London End Location: Manchester Driver: John</td></tr>
</table>

4.2 Post development testing for usability

Function	**User feedback**	**Met: Fully/partially/not**
Login	The login page has been kept minimalistic as asked for and follows the mock up plan in the design section agreed on	Fully
Create account	The create account page has been kept minimalistic as asked for and only prompts for business required information	Fully
Home page	The home page is a nice added feature which helps remind the manager and drivers the rules of haulage, however doesn't provide any other purpose and acts as a low profile page	Partially
Menu	The menu is accessible across all pages and buttons are only viewable when signed in or not. Reduces the amount of pages needed to visit to get to end page, however it would have been better to have the log out button on the right by itself	Partially
Text forms	The text forms are good and minimalist but also have a nice style to them, however the text boxes seem unrealistically long where people are unlikely to ever need that much space. But as it fits to screen when the screen size changes it isn't an issue	Fully
Button forms	The drivers are displayed in a drop down menu which allows the user to type in the drivers names and for them to appear	Fully

4.3 Success against objectives of functional components against success criteria

1. Must be able to sign into account
    - The user can log into their account and are shown information only for them, without the interruption of others accounts.
    - There are also multiple validation checks in place to guarantee referential integrity in the database and data integrity when updating account information
2. Must have a intuitive menu
    - The menu is intuitive because it is always visible and limits the pages users can visit at various points of the program, preventing mistakenly accessing the wrong page.
    - As navbars are commonly used in other websites this gives users familiar friendly feel
3. Must display hours worked
    - This fails as the code has been partially finished
4. Must be able to enter schedule
    - The managers have more permissions than the drivers, and is evident here as only they can add jobs to the schedule. This feature provides the system with a more robust solution.
5. Add/remove drivers
    - Currently as many drivers can join the current system and there is no way to remove the drivers from the database
6. Web page fits multiple screen sizes
    - The navbar has a collapse feature on smaller screens to accommodate smaller screens. The website also scales the display up and down to fit the screens perfectly
7. The program should have a minimalist feel
    - The stakeholder found it easy to navigate and use the website without guidance and says its very easy to use
8. Sending a message across networks from different computer systems
    - This fails as the function hasn't been coded yet
9. Must display schedule
    - The schedules have had extra features added for them, such as being able to filter the schedules and sorting them by date
    - The schedules are much easier to read than the previous solution
10. Must display drivers
    - The drivers are displayed when choosing who to add the jobs for

Criteria	1	2	3	4	5	6	7	8	9	10
Pass/Fail	Pass	Pass	Fail	Pass	Fail	Pass	Pass	Fail	Pass	Pass

4.4 Addressing unmet/partially met tests of the system in the future

Displaying the hours and its relevant calculations code was started however not finished. Currently it only takes in the hours worked per week, with validation making it so that this information can only be added on the monday. This feature however is not necessary for the program to work and just acts as an extra handy tool.

The removal of drivers by the manager in the event a driver no longer works for the business, they should be removed from the main table in the database and added to the table for deleted users where all their information is stored for up to 1 year as their past jobs are still needed in the businesses calculations.

Sending messages over networks may have been a far fetched feature to add as it could be its own project to begin with.

4.5 Evaluation of maintenance of the solution

The developed solution is easy to maintain as there is an abundance of comments throughout the program, with copies of files added just for comments.
The program was developed using html and python, and the html that would be updated has already been extended from the base file resulting in very little html language knowledge to make each webpage.
Python is also a commonly used language and is the top 3 most used programming languages, so a lot of solutions to problems can be found easily and quickly online.
The database used is a database used specifically for the imported library, allowing pre made functions to be used and called using its keyword.

The program will not need to be changed or updated a lot other than editing the database manually. The only need for updates to the program are changing the AETR rules on the home page if there is a change in them by the government.

The naming conventions used for the program consisted of

- File naming: lower case words and words separated by underscores
- Variable naming: camelCasing
- Functions and classes: lower case words and words separated by underscores

4.6 Limitations of the current program

One limitation of my program is that the manager must inform the programmer of their role so that the programmer is able to forcefully update the table through an external app. This leads to the program needing someone to maintain it constantly until a solution is found.
However a solution is to have the user declare their role in the business on the sign up page. This route of programming would lead to a very unprofessional website as people may not act rational and claim something they are not.

Another limitation of the program is that it cannot have multiple businesses using it to manage their own business separately as all names of the database show up.
However a solution to this is that the program is stored locally on each company system so that each version of the program is independent from each other

Users cannot change their information once they create an account, however the user can delete their account and make a new one to update their details. This however would result in a loss of all their jobs and the deleting function isn't coded and working

Reflection
Overall I think this program is a success as it is a clear winner in solutions for the stakeholder as all the information in their multiple spreadsheets is automatically stored and sorted onto one app for them.

Bibliography

Module 1, prototype 1
Tim Ruscica
techwithtim
https://github.com/techwithtim/Flask-Web-App-Tutorial/blob/main/website/__init__.py
https://www.youtube.com/watch?v=dam0GPOAvVI&t=7354s

Module 2, prototype 2, backend.py, home:
Last Record in Column, SQLALCHEMY
https://stackoverflow.com/questions/49607031/last-record-in-column-sqlalchemy

Module 4, prototype 1:
Tim Ruscica
techwithtim
https://github.com/techwithtim/Flask-Web-App-Tutorial/blob/main/website/models.py

Module 4, prototype 2:
Flask-SQLAlchemy db.create_all() got an unexpected keyword argument 'app'
https://stackoverflow.com/questions/73968584/flask-sqlalchemy-db-create-all-got-an-unexpected-keyword-argument-app

Appendix

main.py

```
#This file is where the program is to be ran from

from website import create_app #importing website package from __init.py__ to
create an application and run it
# Website folder is a python package so when imported it will run all files in
folder

app = create_app() # Function in the __init__.py file

# This runs a flask application
if __name__ == '__main__': # Only when this file is ran the following line is
executed
    app.run(debug=True) # Starts a web server
                        # debug=True change in code will auto rerun web server
```

__init__.py

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
from flask_login import LoginManager

# the name of this file makes the folder 'website' a python package

db = SQLAlchemy() # defining a new database - this line initialises it - an
object
DB_NAME = 'database.db' # returns the name of a specified database


# Creating an instance of the Flask class
def create_app():
    app = Flask(__name__) # initialising the app, __name__ used so that Flask
knows where to look for templates etc...
    app.config['SECRET_KEY'] = 'sfibdjkfhb hdfjbs' # This is to sign session
cookies for protection

    app.config['SQLALCHEMY_DATABASE_URI'] = f'sqlite:///{DB_NAME}' #SQLAlchemy
is stored in this location
    db.init_app(app) # initialise the database by giving it the flask app


    # registering blueprints in this file and showing where the different URLs
for the application are
    from .backend import backend #importing the blueprint backend from the file
backend.py

    # Shows how to access all the URLs from the corresponding blueprints
    app.register_blueprint(backend, url_prefix='/') #sets the prefix of a URL
to /: when creating different routes don't need to add / every time before the
rest of the URL


    from .database import User

    #creating a database - sql now automatically does this and eliminates need
for the create_database function
    with app.app_context():
        db.create_all()

    login_manager = LoginManager()
```

```
    login_manager.login_view = 'backend.login' #where flask should redirect
page when user is not logged in
    login_manager.init_app(app) #tells LoginManager which app is being used

    # Tells flask how to load a user - what user is being looked for, by
referencing their id
    @login_manager.user_loader
    def load_user(id):
        return User.query.get(int(id)) # Going to look for primary key and
check it is equal to what is passed

    return app
```

database.py

```
from . import db # importing from the package db which is in the init file
from flask_login import UserMixin # custom class to inherit giving user
object specific things to login

# Creating a table for the schedule days
class Week(db.Model):
    id=db.Column(db.Integer, primary_key=True)
    week_no = db.Column(db.Integer)
    monday_date = db.Column(db.String)


class Days(db.Model):
    id=db.Column(db.Integer, primary_key=True)
    week = db.Column(db.Integer)
    monday = db.Column(db.Integer)
    tuesday = db.Column(db.Integer)
    wednesday = db.Column(db.Integer)
    thursday = db.Column(db.Integer)
    friday = db.Column(db.Integer)
    saturday = db.Column(db.Integer)
    sunday = db.Column(db.Integer)
    user = db.Column(db.String(150))
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'))


#Creating a table for the schedules
class Job(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    date = db.Column(db.String)
    time = db.Column(db.String)
    tractorID = db.Column(db.String(20))
    start_location = db.Column(db.String(150))
    end_location = db.Column(db.String(150))
    user = db.Column(db.String(150))
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'))


#Creating a table for user information using a class object
```

```
class User (db.Model, UserMixin): #UserMixin used for implementations of
this objects properties
    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(150), unique=True) #Identifies this column
as the primary key
    password = db.Column(db.String(150)) #Creates a column for password
that accepts strings
    first_name = db.Column(db.String(150)) #Sets first_name character
limit to 150
    second_name = db.Column(db.String(150))
    manager = db.Column(db.Boolean) #True when account level is manager
False when it is not
    jobs = db.relationship('Job')
    hours = db.relationship('Days')
```

backend.py

```
from flask import Blueprint, render_template, request, flash, redirect,
url_for
from . database import User, Job, Days, Week
from . import db
from werkzeug.security import generate_password_hash, check_password_hash
#help hash a password from flask_login - hashing a password
from flask_login import login_user, login_required, logout_user,
current_user
from datetime import date, datetime


weekdays = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
"Saturday", "Sunday"]
day = date.today()
dt = datetime.now()
weekdayNumber = dt.weekday()
weekday = weekdays[weekdayNumber]



# This file is a blueprint of the program, containing multiple routes
inside it

backend = Blueprint('backend',__name__) #Defining the name of the
blueprint

    ###  AUTHENTICATION ROUTES
-----------------------------------------------------------------------
-

@backend.route('/login', methods=['GET', 'POST']) #creating a decorator,
when the URL is /login the function below is invoked
def login():
    if request.method == 'POST':
        email = request.form.get('email') # request variable when accessed
in a route will have information of a request sent to access this route -
URL, method, information sent
        password = request.form.get('password')
```

```
        # validation that user is in database
        user = User.query.filter_by(email=email).first()  # query database
and searching for something in database
        if user:
            if check_password_hash(user.password, password): #hashes
password entered and compares it to stored hasehd password
                flash('Logged in successfully!', category ='success')
                login_user(user, remember=True) #remebers user is logged
in until flask session over or logs out
                return redirect(url_for('backend.home'))
            else:
                flash('Incorrect password, try again.', category='error')
        else:
            flash('Email does not exist.', category='error')

    return render_template("login.html",user=current_user)


@backend.route('/logout') #creating a decorator, when the URL is /logout
the function below is invoked
@login_required #decorator - makes sure, this rout is inaccesible unless
user is logged in
def logout():
    logout_user() #logs out current user
    flash('Logged out of account', category='success')
    return redirect(url_for('backend.login')) #Brings user back to login
page when they've signed out


@backend.route('/CreateAccount', methods=['GET', 'POST']) #creating a
decorator, when the URL is /CreateAccount the function below is invoked
def create_account():
    if request.method == 'POST': # differentiate between GET and POST
request
        first_name = request.form.get('first_name').capitalize() # .get is
used to get a specific attribute from the form
        second_name = request.form.get('second_name').capitalize() #
capitalize function used to make first letter a capital
        email = request.form.get('email')
        password = request.form.get('password')
```

```
        password_confirm = request.form.get('password_confirm')

        #validation

        user = User.query.filter_by(email=email).first()
        if user:
            flash('Email already exists.', category='error')

        # category allows you to display the message in a different colour
        if len(first_name) < 2:
            flash('First name must be greater than 1 characters',
category='error')
        elif len(second_name) <2:
            flash ('Second name must be greater than 1 character',
category='error')
        elif len(email)<4:
            flash('Email must be greater than 3 characters.',
category='error')
        elif len(password) < 7:
            flash('Password must be greater than 6 characters.',
category='error')
        elif password != password_confirm:
            flash('Passwords do not match.', category='error')
        else:
            #creating a new user
            new_user = User(first_name=first_name,
second_name=second_name, email=email,
password=generate_password_hash(password, method='sha256'), manager=False)
# Defining a user and defining the password with an ireverisble hash
instead of plain text - sha256 is a hashing algorithm
            db.session.add(new_user) # add user to database
            db.session.commit() # update the database
            flash('Account Created', category = 'success')
            login_user(new_user, remember=True)
            return redirect(url_for('backend.home')) # redirect user to
homepage of website, home is the function - backend is the name of
blueprint, home is the function - do it in this way incase we change the
URL path in that function

    return render_template("create_account.html", user=current_user)
```

```
    ### OTHER ROUTES
-------------------------------------------------------------------------
--------------


@backend.route('/') # defining the homepage route - this is a decorator (A
decorator is a function that takes in another function as a parameter and
then returns a function)
@login_required # decorator - cannot get to the home page unless logged in
def home(): #this function runs when the URL is /

    #Have to get last week_no from Week database and increment that number
by 1 every time it is a new monday
    weekQuery = Week.query.order_by(Week.id.desc()).first()
    lastMonday = weekQuery.monday_date

    if str(day) != str(lastMonday) and weekday=="Monday":
        week = weekQuery.week_no
        week += 1
        new_week = Week(week_no=week, monday_date=date.today())
        db.session.add(new_week)
        db.session.commit()

    return render_template("home.html", user = current_user) #will be able
to reference the current user and check if it is authentic


@backend.route('/EditSchedules', methods=['GET','POST']) # defining the
route for the manager to edit the drivers schedules
@login_required # decorator so that you cannot access schedules without
being logged in
def edit_schedules():#when the URL path is /EditSchedules then this
function is ran

    driver = User.query.order_by(User.first_name)    # sort the drivers
into alphabetical order, more professional than before
    if request.method == 'POST':
        # allow the field information from the form to be accessed by
other functions
```

```
        global driverEdit
        global driverEditID

        ### Before the driver submitted would show as < User 7 >
        ### used to record the user selected in the form from
'edit_schedules.html' file to save in database
        driverSelect = request.form.get('drivers') #saves the user input
from webpage
        sliceDriverEdit = driverSelect [6:] # slices the variable value
        sliceDriverEdit = sliceDriverEdit[:1] #slices the variable again
        driverEdit = User.query.filter_by(id=sliceDriverEdit).first()
#queries for the user with the correct id
        driverEditID = driverEdit.id
        driverEdit = driverEdit.first_name

        return redirect(url_for('backend.schedules'))

    return render_template("edit_schedules.html", user=current_user,
drivers=driver, day=day, weekday=weekday, weekdays = weekdays) #will be
able to reference the current user and check if it is authentic



@backend.route('/AddSchedule', methods=['GET', 'POST'])
@login_required
def schedules():
    if request.method == 'POST':
        date = request.form.get('date')
        time = request.form.get('time') # .get is used to get a specific
attribute from the form
        tractorID = request.form.get('tractorID')
        start_location = request.form.get('start_location').capitalize() #
capitalize function used to make first letter a capital
        end_location = request.form.get('end_location').capitalize() #
capitalize function used to make first letter a capital

        ### Validation
        if date == "":
            flash('Must enter a date', category='error')
        elif time == "":
```

```
            flash('Must enter a time', category='error')
        elif tractorID == "":
            flash('Must enter a tractorID', category='error')
        elif start_location == "":
            flash('Must enter the start location', category='error')
        elif end_location ==="":
            flash('Must enter the end location', category='error')
        else:
            new_job = Job(date=date, time=time, tractorID=tractorID,
                          start_location=start_location,
end_location=end_location,
                          user = driverEdit, user_id = driverEditID)
            db.session.add(new_job) # add user to database
            db.session.commit() # update the database
            flash('Schedule updated', category = 'success')
            return redirect(url_for('backend.view_schedule'))

    return render_template('schedule_days.html',  user = current_user)


@backend.route('/ViewSchedule', methods=['GET', 'POST'])
@login_required
def view_schedule():

    job = Job.query.order_by(Job.date).all()
    driver = User.query.all()

    if request.method == 'POST':
        global specificDriver
        specificDriver = request.form.get('drivers')
        sliceSpecificDriver = specificDriver [6:] # slices the variable
value
        sliceSpecificDriver = sliceSpecificDriver[:1] #slices the variable
again
        specificDriver =
User.query.filter_by(id=sliceSpecificDriver).first()
        specificDriver = specificDriver.first_name
        return redirect(url_for('backend.specific_schedule',
specificDriver=specificDriver))
```

```
    return render_template("view_schedule.html", user = current_user, jobs
= job, drivers=driver, day=day, weekday=weekday)


@backend.route('/<specificDriver>_schedule', methods=['GET', 'POST'])
@login_required
def specific_schedule(specificDriver):

    job =
Job.query.order_by(Job.date).filter_by(user=specificDriver).all()

    return render_template("specific_schedule.html", user=current_user,
jobs = job, specificDriver = specificDriver, day=day, weekday=weekday)


@backend.route('/ChooseDriverForHours', methods = ['GET','POST'])
@login_required
def hours_driver():

    weekQuery = Week.query.order_by(Week.id.desc()).first()
    weekNumber = weekQuery.week_no
    driver = User.query.all()
    if request.method == 'POST':
        global timeSheetDriver
        global timeSheetDriverID

        timeSheetDriver = request.form.get('drivers')
        sliceTimeSheetDriver = timeSheetDriver [6:] # slices the variable
value
        sliceTimeSheetDriver = sliceTimeSheetDriver[:1] #slices the
variable again
        timeSheetDriver =
User.query.filter_by(id=sliceTimeSheetDriver).first()
        timeSheetDriverID = timeSheetDriver.id
        timeSheetDriver = timeSheetDriver.first_name


        return redirect(url_for('backend.hours_worked'))
```

```
    return render_template('hours_driver.html',  user=current_user,
drivers=driver, day=day, weekday=weekday, weekNumber = weekNumber)


@backend.route('/HoursWorked', methods = ['GET', 'POST'])
@login_required
def hours_worked():

    weekNumber = Week.query.order_by(Week.id.desc()).first()
    weekNumber = weekNumber.week_no


    if request.method == 'POST':
        monday = request.form.get('monday')
        tuesday = request.form.get('tuesday')
        wednesday = request.form.get('wednesday')
        thursday = request.form.get('thursday')
        friday = request.form.get('friday')
        saturday = request.form.get('saturday')
        sunday = request.form.get('sunday')


        if monday == "":
            monday = 0

        if tuesday == "":
            tuesday = 0

        if wednesday == "":
            wednesday = 0

        if thursday == "":
            thursday = 0

        if friday == "":
            friday = 0

        if saturday == "":
            saturday = 0
```

```
        if sunday == "":
            sunday = 0

        #updating the schedule
        new_hours = Days(week=weekNumber, monday=monday, tuesday=tuesday,
wednesday=wednesday, thursday=thursday, friday=friday, saturday=saturday,
sunday=sunday, user=timeSheetDriver, user_id = timeSheetDriverID) #
Defining a user and defining the password with an ireverisble hash instead
of plain text - sha256 is a hashing algorithm
        db.session.add(new_hours) # add user to database
        db.session.commit() # update the database

    return render_template("hours_worked.html", user=current_user, weekday
= weekday, day=day, weekNumber=weekNumber)


# def calculateHours():
#     pass
```

base.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <link
      rel="stylesheet"

href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min
.css"

integrity="sha384-Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf
23Q9Ifjh"
      crossorigin="anonymous"
    />
    <link
      rel="stylesheet"

href="https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/font-aweso
me.min.css"
      crossorigin="anonymous"
    />

    <title>{% block title %}Home{% endblock %}</title>
  </head>

  <body>
    <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
      <button
        class="navbar-toggler"
        type="button"
        data-toggle="collapse"
        data-target="#navbar"
      >
        <span class="navbar-toggler-icon"></span>
      </button>
      <div class="collapse navbar-collapse" id="navbar">
        <div class="navbar-nav">
          {% if user.is_authenticated %}
```

```
            <a class="nav-item nav-link" id="home" href="/">Home</a>
            <a class="nav-item nav-link" id="Editchedule"
href="/EditSchedules">Edit Schedule</a>
            <a class="nav-item nav-link" id="schedule"
href="/ViewSchedule">Schedule</a>
            <a class="nav-item nav-link" id="timesheet"
href="/ChooseDriverForHours">Timesheet</a>
            <a class="nav-item nav-link" id="logout"
href="/logout">Logout</a>

          {% else %}
          <a class="nav-item nav-link" id="login" href="/login">Login</a>
          <a class="nav-item nav-link" id="CreateAccount"
href="/CreateAccount">Create Account</a>
          {% endif %}
        </div>
      </div>
    </nav>

    {% with messages = get_flashed_messages(with_categories=true) %}
    {% if messages %} {% for category, message in messages %}
    {% if category =='error' %}
    <div class="alert alert-danger alter-dismissable fade show"
role="alert">
      {{ message }}
      <button type="button" class="close" data-dismiss="alert">
        <span aria-hidden="true">&times;</span>
      </button>
    </div>
    {% else %}
    <div class="alert alert-success alter-dismissable fade show"
role="alert">
      {{ message }}
      <button type="button" class="close" data-dismiss="alert">
        <span aria-hidden="true">&times;</span>
      </button>
    </div>
    {% endif %} {% endfor %} {% endif %} {% endwith %}

    <div class="container">{% block content %} {% endblock %}</div>
```

```
    <script
      src="https://code.jquery.com/jquery-3.2.1.slim.min.js"

integrity="sha384-KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93
hXpG5KkN"
      crossorigin="anonymous"
    ></script>
    <script

src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.mi
n.js"

integrity="sha384-ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXu
svfa0b4Q"
      crossorigin="anonymous"
    ></script>
    <script

src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js"

integrity="sha384-JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+
76PVCmYl"
      crossorigin="anonymous"
    ></script>

    <script
      type="text/javascript"
      src="{{ url_for('static', filename='index.js') }}"
    ></script>
  </body>
</html>
```

create_account.html

```
{% extends "base.html" %} {% block title %}Create Account{% endblock %} {%
block content %}

<form method="POST">
    <h3 align="center"> Create Account</h3>

    <div class = "form-group">
        <label for="first_name">First Name</label>
        <input
            type="text"
            class="form-control"
            id ="first_name"
            name="first_name"
            placeholder="Enter First Name"
        />
    </div>

    <div class = "form-group">
            <label for="second_name">Second Name</label>
            <input
                type="text"
                class="form-control"
                id ="second_name"
                name="second_name"
                placeholder="Enter Second Name"
            />
    </div>

    <div class = "form-group">
        <label for="email">Email</label>
        <input
            type="email"
            class="form-control"
            id ="email"
            name="email"
            placeholder="Enter Email Address"
        />
    </div>
```

```
    <div class = "form-group">
        <label for="password">Password</label>
        <input
            type="password"
            class="form-control"
            id ="password"
            name="password"
            placeholder="Enter Password"
        />
    </div>

    <div class = "form-group">
        <label for="password_confirm">Password Confirm</label>
        <input
            type="password"
            class="form-control"
            id ="password_confirm"
            name="password_confirm"
            placeholder="Confirm Password"
        />
    </div>
    <br>
    <button type="submit" class="btn btn-primary">Submit</button>

</form>
{% endblock %}
```

edit_schedules.html

```
{% extends "base.html" %} {% block title %}Edit Schedules{% endblock %} {%
block content %}



<br>
<h6 align="right"><b>Today's date:</b> {{day}}</h6>
<h6 align="right"><b>Weekday: </b> {{weekday}}</h6>

<form method="POST">
    <center>
        <br>
        <h5> Edit Schedules</h5>
        <br>

        {% if user.manager == 1 %}
            <h5>Driver</h5>
            <select name="drivers">
                {% for driver in drivers %}
                    {% if driver.manager == 0 %}
                        <option value="{{driver}}"
SELECTED>{{driver.first_name}}</option>
                    {% endif %}
                {% endfor %}
            </select>
             <button type="submit" class="btn
btn-primary">Submit</button>

        {% else %}
            <b>Contact Manager to edit your schedule</b>
            <br>
            <br>
            <ul class="list-group list-group-flush" id="Job"></ul>
            {% for job in user.jobs %}
            <li class="list-group-item">
                <b>Date:</b> {{job.date}}   <b>Time:</b> {{job.time}}
  <b>TractorID:</b> {{job.tractorID}}   <b>Start Location:</b>
{{job.start_location}}   <b>End Location:</b> {{job.end_location}}
  <b> Driver:</b> {{job.user}}
            </li>
```

```
            {% endfor %}
        {% endif %}


    </center>


{% endblock %}
```

home.html

```
{% extends "base.html" %} {% block title %}Home{% endblock %} {% block
content %}

<h1 align="center">AETR Rules</h1>
<br>


<h5 align="center">Driving Hours</h5>
<br>

<ul style="list-style-type:circle">
    <li>Musn't drive more than 9 hours in a day</li>
    <ul style="list-style-type:circle">
        <li>Can extend this to 10 hours twice a week</li>
    </ul>
    <li>Musn't drive more than 56 hours in a week</li>
    <li>Musn't drive more than 90 hours in any 2 consecutive weeks</li>
</ul>
<p>Must record all driving you do under EU rules on a tachograph </p>
<br>

<h5 align="center">Breaks and Rest</h5>
<br>

<ul style="list-style-type:circle">
    <li>Must have at least 11 hours rest every day</li>
    <ul style="list-style-type:circle">
        <li>Can reduce this to 9 hours rest 3 times between any 2 weekly
rest periods</li>
    </ul>
    <li>Must take an unbroken rest period of 45 hours every week</li>
    <ul style="list-style-type:circle">
        <li>Can reduce this to 24 hours every other week</li>
    </ul>
    <li>Must take a break or breaks totalling at least 45 minutes after no
more than 4 hours 30 minutes driving</li>
    <li>Must take a weekly rest 6 consecutive 24-hour periods of working,
starting from the end of the last weekly rest period taken</li>
</ul>
```

```
{% endblock %}
```

hours_driver.html

```
{% extends "base.html" %} {% block title %}Select driver{% endblock %} {%
block content %}

<h6 align ="right"> WeekNumber: {{weekNumber}} </h6>

<form method = 'POST'>
    <center>
        {% if user.manager == 1 %}
            <h5>Select driver to add to their time sheet</h5>
            <select name="drivers">
                {% for driver in drivers %}
                    {% if driver.manager == 0 %}
                        <option value="{{driver}}"
SELECTED>{{driver.first_name}}</option>
                    {% endif %}
                {% endfor %}
            </select>
             <button type="submit" class="btn
btn-primary">Submit</button>
        {% else %}
            <h5>Manager access only!</h5>
        {% endif %}
    </center>
</form>

{% endblock %}
```

hours_worked.html

```
{% extends "base.html" %} {% block title %}Hours worked{% endblock %} {%
block content %}

<style>
    input[type='number']{
        width: 80px;
    }
</style>

<br>
<h6 align="right"> <b>Today's date:</b> {{day}}</h6>
<h6 align="right"> <b>Weekday: </b> {{weekday}}</h6>
<h6 align="right"> <b>Week Number: </b> {{weekNumber}}</h6>
<center>
    <br>

    {% if weekday == 'Sunday'%}
        <form method = 'POST'>
            <h5>Hours driven by {{driver}} this week</h5><br>

            <label for="monday">Monday:
        </label>
            <input type="number" id="monday" name="monday" min = "0" max =
"10"><br><br>

            <label for="tuesday">Tuesday:
        </label>
            <input type="number" id="tuesday" name="tuesday" min="0" max =
"10"><br><br>

            <label for="wednesday">Wednesday:      </label>
            <input type="number" id="wednesday" name="wednesday" min ="0"
max = "10"><br><br>

            <label for="thursday">Thursday:     
</label>
            <input type="number" id="thursday" name="thursday" min = "0"
max ="10"><br><br>
```

```
            <label for="friday">Friday:
             </label>
            <input type="number" id="friday" name="friday" min="0" max =
10><br><br>

            <label for="saturday">Saturday:      
</label>
            <input type="number" id="saturday" name="saturday" min = "0"
max = "10"><br><br>

            <label for="sunday">Sunday:
         </label>
            <input type="number" id="sunday" name="sunday" min = "0" max =
"10"><br><br>

            <button type="submit" class="btn btn-primary">Submit</button>
        </form>

    {% else %}
        <h5>You can only add the hours worked on Sunday</h5>
        <h5>Please come back on Sunday!</h5>
    {% endif %}


</center>


{% endblock %}
```

login.html

```
{% extends "base.html" %} {% block title %}Login{% endblock %} {% block
content%}

<form method="POST">
    <h3 align="center"> Login</h3>

    <div class = "form-group">
        <label for="email">Email</label>
        <input
            type="email"
            class="form-control"
            id ="email"
            name="email"
            placeholder="Enter Email Address"
        />
    </div>

    <div class = "form-group">
        <label for="password">Password</label>
        <input
            type="password"
            class="form-control"
            id ="password"
            name="password"
            placeholder="Enter Password"
        />
    </div>
    <br />
    <button type="submit" class="btn btn-primary">Login</button>

</form>
{%endblock%}
```

schedule_days.html

```
{% extends "base.html" %} {% block title %}Add Job{% endblock %} {% block
content %}

<form method="POST">
    <h3 align="center">Schedule Information</h3>

    <div class = "form-group">
        <label for="date">Date</label> <br>
        <input
            type="date"
            id ="date"
            name="date"
        />
    </div>


    <div class = "form-group">
        <label for="time">Start Time</label> <br>
        <input
            type="time"
            id ="time"
            name="time"
        />
    </div>

    <div class = "form-group">
        <label for="tractorID">tractorID</label>
        <input
            type="text"
            class="form-control"
            id ="tractorID"
            name="tractorID"
            placeholder="Enter tractorID"
        />
    </div>

    <div class = "form-group">
        <label for="start_location">Start Location</label>
        <input
```

```
            type="text"
            class="form-control"
            id ="start_location"
            name="start_location"
            placeholder="Enter the site of the start location"
        />
    </div>

    <div class = "form-group">
        <label for="end_location">End Location</label>
        <input
            type="text"
            class="form-control"
            id ="end_location"
            name="end_location"
            placeholder="Enter the site of the end location"
        />
    </div>
        <button type="submit" class="btn btn-primary">Add Job
Information</button>
</form>

{% endblock %}
```

specific_schedule.html

```
{% extends "base.html" %} {% block title %}View Schedule{% endblock %} {%
block content %}

<br>
<h6 align="right"><b>Today's date:</b> {{day}}</h6>
<h6 align="right"><b>Weekday: </b> {{weekday}}</h6>

<center>

    <br>

    <b> Schedule for {{specificDriver}} </b>
    <br>

    <ul class="list-group list-group-flush" id="Job"></ul>
    {% for job in jobs %}
        <li class="list-group-item">
            <b>Date:</b> {{job.date}}   <b>Time:</b> {{job.time}}
  <b>TractorID:</b> {{job.tractorID}}   <b>Start Location:</b>
{{job.start_location}}   <b>End Location:</b> {{job.end_location}}
  <b> Driver:</b> {{job.user}}
        </li>
    {% endfor %}

</center>

{% endblock %}
```

view_schedule.html

```
{% extends "base.html" %} {% block title %}View Schedule{% endblock %} {%
block content %}

<br>
<h6 align="right"><b>Today's date:</b> {{day}}</h6>
<h6 align="right"><b>Weekday: </b> {{weekday}}</h6>

<center>
    <br>
    <h5>Schedule</h5>
    <br>

    {% if user.manager == 1 %}
        <ul class="list-group list-group-flush" id="Job"></ul>
        {% for job in jobs %}
        <li class="list-group-item">
            <b>Date:</b> {{job.date}}  
            <b>Time:</b> {{job.time}}  
            <b>TractorID:</b> {{job.tractorID}}  
            <b>Start Location:</b> {{job.start_location}}  
            <b>End Location:</b> {{job.end_location}}  
            <b>Driver:</b> {{job.user}}
        </li>
        {% endfor %}

        <br>
        <br>

        <form method="POST">

            <h5>Show driver schedule</h5>

            <select name="drivers">
                {% for driver in drivers %}
                    {% if driver.manager == 0 %}
                        <option value="{{driver}}"
SELECTED>{{driver.first_name}}</option>
                    {% endif %}
                {% endfor %}
```

```
            </select>
             <button type="submit" class="btn
btn-primary">Submit</button>
        </form>

    {% else %}
        <ul class="list-group list-group-flush" id="Job"></ul>
        {% for job in user.jobs %}
        <li class="list-group-item">
            <b>Date:</b> {{job.date}}  
            <b>Time:</b> {{job.time}}  
            <b>TractorID:</b> {{job.tractorID}}  
            <b>Start Location:</b> {{job.start_location}}  
            <b>End Location:</b> {{job.end_location}}  
            <b> Driver:</b> {{job.user}}
        </li>
        {% endfor %}
    {% endif %}

</center>

<!-- {{driverEdit}}     -->



{% endblock %}
```