A SOLUTION TO CASUALLY CONTRACTED LIFEGUARD SCHEDULING

Aiden Gourley

CONTENTS

Problem Identification 5-11

Stakeholders 7-13

Research 9

Current Solutions on the Market 9

Software Limitations/ Constraints 9

Hardware Limitations/ Constraints 9

The Existing Solution 10

Specifics of the Proposed Solution 11

Success Criteria 12

Database 12

Graphical User Interface 12-19

Core Algorithm 14

Development and Design Plan 16

Database Design and Features 17-28

Validation and Security 21-28

User Interface and Usability Features 24

Security 24

User Interface Design 25-35

Core Algorithm Design and Features 31

flowchart 32-39

Pseudocode 35-41

Algorithm's Key Variables and Validation 37-44

Hand Trace- Viability of Proposed Solution Testing 40-58

Structure Chart (Algorithm) 54

Structure Chart (GUI) 55

Structure Chart (Database) 56

Data Flow Diagram (DFD) 57-66

Level 0 Data Flow Diagram (DFD) 61

Level 1 Data Flow Diagram (DFD) 62

Introduction 64

Initial Django SETUP 65

Database Development 66-74

Full Database Visualisation 68

GUI Development 69-115

GUI Navigation Bar and URL Development Testing 76-84

GUI Development User Authentication Testing 88-95

GUI Development Rota Generation Rendering and Shift Input Testing 99-110

GUI Development Rota Generation Rendering and Shift Input Testing 107

Core Algorithm Development 110-120

Core Algorithm Development Testing 112-120

Development and Testing Conclusion 115

Description of the Final Product 117

User Feedback 117

User Feedback Results 118-128

User Manual 123-132

Evaluation Against Success Criteria 127-139

Database 127

Graphical User Interface 128-138

Core Algorithm 133

Evaluative Testing 134-144

Success of the Solution 139

Suggestions for Further Development 139-146

Maintenance 140

Evaluative Testing Screenshots 141-151

References 146

PROBLEM ANALYSIS

Aiden Gourley

PROBLEM IDENTIFICATION

A very common time-consuming exercise for management, particularly within corporations that operate shift-based working hours, is the scheduling that takes place, usually by hand. To free up time and ensure that reference and changes can be accommodated for both management and employees quickly and easily, there are many pieces of paid or free software available. Different types of organisations require specific features and criteria so that the software saves time and doesn't just create different issues. In this case, a scheduling application will be developed for the leisure industry, particularly for managing a lifeguard team.

This particular problem is suitable for a computational solution due to a variety of factors. As a human attempts to schedule staff, they will look at each constraint and attempt to figure out who is able to fit where. However, many managers will attempt brute forcing the problem without realising that this is highly inefficient and time consuming by hand. In fact, it is possible to add a mathematical visualisation to the number of schedule combinations. We shall take an example of 3 fillable shifts, given 3 employees named: A, B, and C.

The possible combinations can be displayed as the following table:

SHIFT 1	**SHIFT 2**	**SHIFT 3**
A	B	C
A	C	B
B	A	C
B	C	A
C	A	B
C	B	A

This totals 6 different solutions, and mathematically we can formulate this as n! where n is the number of shifts. Although this may not seem too complicated or time consuming at the moment, this example is using roughly 10x less shifts and employees, without any constraints to consider. Therefore, if the client were to brute force his schedule without any consideration of the constraints, he would need to consider 30! or 2.65252859812e+32 which is over 265,000 nonillion combinations. Even using computational power, this is a huge amount of work to do in order to find an optimal schedule for 30 employees. Therefore, other methods should be considered in order to reduce the time and space complexity when an algorithmic solution is developed. Scheduling, as a general problem is naturally abstracted by humans when we discuss solutions, however the complexity lies in constraints and other factors and features the client requires. Abstraction will most likely have to be used in order to remove the unnecessary detail surrounding the constraints while formulating them in a computational manner.

PROBLEM IDENTIFICATION (CONT.)

When displaying results, the abstracted detail can then be displayed to the user so that it makes user interpretation easier. This is common practice and takes place in nearly every program that has some form of user interface and backend. Depending on how the algorithm is designed, it could potentially implement some form of backtracking in order to prevent solutions that clearly do not work from using processing power unnecessarily. In a sense, it is an optimisation that relies on early intervention as soon as a problem with the current solution arises. If this is the case, the algorithm will take a step back, and continue with a different input on the next attempt. However, this does not mean that in the worst-case scenario it will not take just as long as brute forcing. Again, this will depend on how the algorithm is designed, however it highlights the different computational methods that lend themselves to this type of scheduling problem. Caching can also be linked to backtracking, because in order to speed up the process, the data not being lost could be stored in a cache. Since many of these computational methods are still very time consuming and unrealistic when attempting to schedule by hand, the management will spend much less time on such a mundane task if they had a technologic solution.

A key decision that must be made before beginning the design process is of what form the solution will take place. The client has stated a concern that it takes a long time for the staff to let him know what dates they are available, therefore making it difficult to start the scheduling process. There are two solutions that I shall be considering- a desktop application, or a web application. It is now no longer the case that desktop applications are limited in terms of communication with other computers over an internal network, or the World Wide Web. However, there are few frameworks available that allow for this type of structure that offer security features and ease of development. There are also the key disadvantages of the program requiring a sound distribution method, and installation by all users. The client has highlighted the fact that many of his staff are not computer literate and will find it difficult to conduct any setup themselves. To overcome these concerns, the program will be developed as a web application. The distinct advantages include that despite the cost of hosting a server, there will be little set-up on the client side. The staff should also experience less incompatibility issues while running different operating systems as the only software they will require is a web browser. The centralised structure removes the need for client's to be running computers that meet a specific spec, as most of the processing shall be completed on the host's server. This structure will allow the staff to have as much of an input in the scheduling process as the client needs, while also allowing the managers to make changes or add constraints to the algorithm before it produces a solution. This ensures that not only will the algorithm produce an optimal solution, but allows for unforeseeable situations to be catered for, such as staff sickness.

STAKEHOLDERS

This project was requested by Wentworth Club, a private company that run an exclusive golf club and health and fitness resort in Virginia Water, Surrey. Although the club is well known for hosting professional golf tournaments such as the BMW PGA Championship, it has an extensive health and fitness facility that cost £9 million. Such a facility contains is responsible for managing many staff members, and the employee structure is further split into different departments within the facility. The 'Pools' department has requested this project in order to manage their lifeguard team, as it is one of the most complex teams to schedule. Notable stakeholders in this project will include:

Stakeholder	**User needs**	**User skills**	**Impact on solution**
Head of IT & IT Technicians *(These users will be responsible for the maintenance of the system following completion of the development. These users will have access to the root account with full administrative permissions*)	Technical Documentation. Although I shall not be responsible, these users will need to either purchase hardware to host the application and data or outsource the hosting and data management/ security. The technicians will also require the ability to access all different account privilege levels in order to fix bugs that might occur within just one user group.	The user will have to understand how the system works in case bugs are found after deployment. They should also have a deep understanding of how to use the software, in case a change of management occurs, and they need to teach new staff how to use the system. Skills in data management and security are also required in order to comply with data protection law. The user should also at least have a basic understanding of the framework and programming language used to develop this application.	The solution will have to come with adequate technical documentation and code that is easy to follow and understand. To keep data management costs low, the database should be normalised and not contain any redundancies. The technicians will also require a separate account on each user privilege level.

STAKEHOLDERS (CONT.)

Stakeholder	User needs	User skills	Impact on solution
Duty Management (DM) (*These users are responsible for informing other stakeholders of software changes and helping with the swift introduction of the technology. After the implementation, they will have higher level accounts with extra permissions allowing them to edit and manage the rotas*)	The users will require basic login and logout functionality, the ability to manage other accounts, access to the administrative management pages where changes to the existing schedule can be made, and new ones generated. They will also require the ability to view the full current rota, remove existing employee accounts, add new employee accounts, set specific constraints when generating a new schedule, manually change the existing rota	The users will need to be fully familiar with the system before it is implemented to make for a smooth transition from the current paper-based system. They will also require basic IT knowledge of data management regarding input form fields.	All features should be simple to understand and annotated as much as possible. The user group will require a separate privilege level that allows access to a management page. The Management user interface should be kept as simple to use as possible, in a way that requires little training.
Employees (*These users are responsible for inputting the required data in a timely fashion so that the algorithm has enough data to be used when generating a schedule the following month*)	These users will require access to a web browser, login details, the ability to request changes to the current rota, ability to add or remove their available working hours, view the current rota, login and logout functionality.	The users will require the least technical skills, and simply need a basic understanding of web browsers and pages.	All features should be easily comprehensible and accessible. This group will also require a separate user account permission level. Only simple training should be required.

RESEARCH

CURRENT SOLUTIONS ON THE MARKET

Research has shown that there are many different types of shift scheduling software on the market now including 'Humanity' and 'WhenToWork'. These applications are unsuitable for the client, as they are full of unnecessary and confusing features. The main reason for this is because they attempt to cater for more than one industry, and although they offer a small level of customisations, the client found the interface difficult to navigate when using the demo versions. The price is also unacceptable at around £130/ month for 'Humanity'. This would lead to annual costs of £1560, with the client mentioning that it would be cheaper to hire someone to create the schedules by hand. The use of technology should lead to cost reductions and efficiency savings, whereas the current solutions out there are not bespoke and do not reduce costs. In fact, the lifeguard wage is about £9 per hour, which makes the annual cost of 'Humanity' equivalent to about 173 hours of work. Therefore, the current solutions available are not bespoke, and are not economically viable solutions. However, many of the current solutions available also come in the form of a web application which indicates that other organisations see the solution is best developed this way.

SOFTWARE LIMITATIONS/ CONSTRAINTS

The decision to develop the solution as a web application was made early on during the problem identification stage. The front-end development will use HTML as the formatting language, and CSS to style the web pages. It is important to the client that the website is aesthetically pleasing and easy to navigate and use. Therefore, the front-end development will play a big part in this project. However, regarding the backend development, I will be using python as the programming language as it is one that I am very familiar with. Since I have a strong knowledge of python, it makes sense for me to use a language I already know, with a framework that I do not, thereby limiting my knowledge in only one area rather than using both a framework and language I am unfamiliar with.

HARDWARE LIMITATIONS/ CONSTRAINTS

There are a few constraints such as the requirement for specific hardware in order to host the web application, however since the organisation's IT department already has methods of outsourcing the hosting of their website, there should not be many issues or restrictions when it comes to hosting the final version regarding limited resources. I have also made it clear that for ease of development, I shall be using the sqlite3 database format. However, in practice it is said that SQLite should not be used in production. There are a few reasons for this, mainly the fact that it does not support concurrency (writing to the database can only happen one at a time). There is a slim chance of this being the case for a small number of users, however if the team grows in the future to the point where a new database needs to be implemented, the migration process will be difficult.

HARDWARE LIMITATIONS/ CONSTRAINTS (CONT.)

It will also be the case that both types of users (managers and employees) will have the ability to write to the database, making errors more likely. Therefore, to support this application long-term, it is best to implement a more advanced database such as PostgreSQL. The database management and security will have to be managed by the client after the release, and they have agreed to hire a security specialist to ensure that there is no way confidential data can be leaked by the application. This therefore lets me store sensitive data without worrying too much about the security precautions. However, the framework (Django) already has many security features meaning that there should be very few if any vulnerabilities.

THE EXISTING SOLUTION

An existing solution to scheduling problems similar to this is to linearly loop through a list of employees and their availability, using a first come first served approach. This has two distinct disadvantages. One being that it is commonly done by hand, meaning that the process takes a long time to complete. The other is that there is no guarantee of a fair solution in terms of attempting to balance the solution when assigning number of hours to each employee. An example is the following:

Employee 1 is available for 5 shifts/ week.
Employee 2 is available for 1 shift/ week.

However, Employee 2's shift is one that Employee 1 is also available for. Since the existing solution would be first come first served, Employee 1 would be given all 5 shifts, leaving Employee 2 without any. This is by far a solution that is not optimal, and a concern to the client, because he frequently must compensate for this when scheduling by hand.

In order to optimise the solution above, we must ask ourselves how managers overcome the issue stated above, and whether it can be turned into an algorithmic optimization. The answer is that they do this 'by eye', which is an unhelpful answer when attempting to implement it as a set of instructions because it is unspecific. However, despite humans taking longer to process than computers, the brain is very good at spotting patterns, therefore 'by eye' could be considered an answer to the manager simply using the brain's capability at pattern recognition to help with a complex computational problem.

However, by looking at the current solution, we can take away some important information that will help us designing an alternative algorithm. We can take away the following:

- The objective of the solution should be to 'fairly assign number of shifts' to each employee.
- The 'by eye' technique that optimizes the current solution is unlikely to be replicated in a simple manor with limited computational power and time frame, therefore a different approach may need to be considered.
- The current solution may have a good success rate; however, it does not make it optimal due to the unfair first come first served aspect.

SPECIFICS OF THE PROPOSED SOLUTION

The programming language of choice, Python, also happens to be the basis of one of the more advanced web application frameworks called Django. This web application framework will be used because of its major advantages regarding providing features that help with both the front end and back end development. Django provides tools that can deal with data formatting and display, a templating language and authentication. It also has an object relational mapping tool to manipulate the database, and form management to help with user input and validation. However, I have very little experience using the framework. This is without much concern to me due to the seemingly endless amount of documentation and the large community online that use Django. I will use SQLite3 as the development database, however the client will change over to PostgreSQL after the project's completion, due to multi-client concurrency concerns. Due to undoubtedly requiring a lot of time manipulation within the logic of this program, be that in the GUI, database, or core algorithm, I shall be using Python's DateTime module to format the DateTime objects that will be stored in the database.

SUCCESS CRITERIA

DATABASE

- The database should only include data that is relevant and used within the program in order to keep the database size to a minimum.
- The database tables should not allow for data redundancy and should contain features that maintain data integrity such as Django 'Cascading delete' feature.
- The data should be protected from malicious users damaging the database through security flaws such as SQL injections. (Validate all user inputs)
- If a many-to-many relationship is required, Django's 'Many-To-Many' field feature should be called so that a linking table can be automatically generated and managed.
- The database used in this development project should be easily migrated to a more secure type of object relational database system such as PostgreSQL.

GRAPHICAL USER INTERFACE

Management Specific Features:

- The program should allow for managers to add employees to the database
    - It should display this feature as a forum, with fields for store data regarding: Employee's first name, last name, password, email and phone number. There should be an option to add this user with the manager account privilege.
- The program should allow for managers to delete employees and all related records from the database.
    - It should display all the current employee level accounts in a drop-down menu, in which the user is able to select one and submit their choice.
- The program should allow managers to add shifts they want to be filled for the current or following month
    - It should be a forum, displayed with compulsory options to set the start and finish time of the shift, the month and year the shift will be implemented for, and a field to select which days of the month the shift will be implemented for.
- The program should accommodate for deleting shifts for the following month.
    - The feature should display all the shifts that have been set for next month's rota as a drop-down menu as choices of shifts to delete, with the user then able to submit once selected. The shift will then be removed from the database.
- The program should include a button to lock in next month's rota manually, however this should only happen after a certain number of days into the current month (yet undecided).
    - Once the schedule is locked in, changes will no longer be made regarding completely re-generating the schedule.
- The program should include a button on the management page to go directly to view the next month's rota, this is for convenience.
- The program should allow the managers to manually change the generated schedules in case of unforeseen circumstances such as staff sickness

SUCCESS CRITERIA (CONT.)

GRAPHICAL USER INTERFACE (CONT.)

Management Specific Features:

- The program should display a button to generate a new schedule, which is a different colour to the other buttons so to highlight the importance of the feature and prevent miss-clicks.
    - The button should call a function that runs the core scheduling algorithm.

Employee Specific Features:

- The program should allow the users to set their availability for the following month.
    - This should be displayed as a form with one choice field that allows multiple shifts to be selected by the user.
- The program should allow users to cancel their availability for the following month.
    - This should be displayed as a form with one choice field that allows multiple shifts to be selected by the user.
- The program should display the shifts that the logged in user is available for, in a list that is easily read.

All Account Holder Features:

- When the user has logged in, they should be greeted by a welcome page that makes the user experience more customised by displaying their name.
- The users should be able to view the rota of the current month
    - The rota should be displayed in a timetable format, that is easily followed and read.
- The users should be able to view the rota of the following month
    - The rota should be displayed in a timetable format, that is easily followed and read.

Non-Account Holder Features:

- The user should not have access to the management pages, or access to the rota pages.
- The user should have access to the login page via the navigation bar.

All Account Privilege Level Features:

- All pages should contain the navigation bar.
    - The navigation bar should contain links to all relevant pages such as the management page, rota pages and home page.
    - The navigation bar should also display a customised 'Welcome, **USERNAME**' message.
    - The navigation bar should also contain a 'Logout' button, which when pressed logs out the user and redirects them to the homepage.

SUCCESS CRITERIA (CONT.)

GRAPHICAL USER INTERFACE (CONT.)

All Account Privilege Level Features:

- The user should be able to view the homepage.
    - The homepage should be aesthetically pleasing and contain a little information regarding the program.

- The GUI design should be deemed contemporary, intuitive, and easily navigated.

CORE ALGORITHM

- The core algorithm should consider all available employees and produce an acceptable schedule. An acceptable schedule is one that results in > 90% of available employees getting at least one shift per week, with 100% of the required shifts filled by the algorithm, given that an employee is available to work it.
- The core algorithm should result in a fair schedule in terms of the number of shifts each employee is to work.
- The algorithm's runtime is not an issue, and so the only constraint here is that it is generated in under 5 minutes for a full set of test data that represents that of normal operational use. However, the worst-case time complexity of the project should not be equal to or exceed exponential time O(2^n).
- The algorithm should allow room for the team to grow in numbers, and therefore be developed in such a way so that it is able to work with an unlimited number of shifts and employees.

SOLUTION DESIGN

Aiden Gourley

DEVELOPMENT AND DESIGN PLAN

1. Set up new Django project
   - Create the default project files and app
   - Change the settings file to accommodate for the user authentication. (i.e. Login and Logout redirection URLs)
   - Check that the default database file has been created
   - Create a superuser account called 'root'
2. Design and create the models (database tables)
3. Design and create all HTML pages
   - Create a new View definition for each page
   - Create a new URL link for each page to each view in the URLs file
4. Build each feature for each page (e.g. a user creation forum)
   - Create each form in the Forms file
   - Link the form to the View definition and handle the data input, validation, database read/ writing, and output
   - Append the HTML file, adding the required features.
5. Create a method of rendering the rota
   - Create a separate file that will be imported as a module. This file will get data from the database regarding the shifts and reformat it in a way that can be easily read and rendered into Django's templating language and HTML
   - Get the returned data and enter it as context to the HTML page
   - Use Django tags and HTML to display the data within the context in a tabular format
6. Create an algorithm
   - Create a separate file that will be imported as a module. This file will not return any data; however, it will be called from the View any time the appropriate button is pressed by the user.
   - Read the shift and employee data form the database and bring it together into a readable data structure to be used by the main algorithm
   - Assign shifts according to the algorithm
7. Fully test all features, ensuring security and functionality by implementing realistic test data.
8. Refactor the code ensuring there are no redundancies, it is commented, efficient, and easily read and maintainable by ensuring the CamelCase convention.

DATABASE DESIGN AND FEATURES

Django's ORM (Object Relational Mapping) means that items stored in the database will be accessed like they are objects which are generally more commonly used in application code. This feature does not in any way benefit the user, however it makes the development much easier as little to no SQL will be required because the Django translates framework specific phrases into SQL when compiled. This therefore means that the 'Classes' section will relate very closely to the database design.

A core decision will also be to render the database in real time each time the page is loaded, therefore the database will not have to store the shifts in any sort of two dimensional array, and instead the data will be grabbed and sorted into a renderable format upon each refresh. In the future, optimisations regarding GUI responsiveness could be put in place such as caching, although since the process does not handle too much data, it should be fairly quick and only have a small impact on GUI responsiveness. A benefit to doing this is that it makes the database design an easier process as it will include fewer complex data structures. It allows the shifts to simply be stored in their own 'Shifts' table.

The design process started by simply questioning what data will need to be stored. This led to the following list of items that may need storage:

- Employee Data
  - FirstName
  - Surname
  - EmployeeID
  - Phone Number
  - Email Address
  - Password
  - Date Joined
  - Total Hours Worked
  - Manager (Boolean)
  - Availability
- Each Day of the Months Data
  - ShiftID
  - EmployeeID
- Shifts
  - Start
  - End
  - ShiftID
  - Shift Notes
  - Cover Requests

DATABASE DESIGN AND FEATURES (CONT.)

As I was researching Django Models, I learnt about how when defining the models and fields, there are options to specify relationships between different models. I was interested in the many-to-many and one-to-one fields. Django also automatically handles these situations by creating linking tables in the background, in order to keep a normalised database and ease of use for the developer. This means that instead of keeping the 'Each Day' category which existed as a linking table between the 'Shifts' and 'Employee Data' categories, relationships between the fields could be defined instead, letting Django deal with the back-end normalisation.

Django also provides built in databases to handle its own built in functions such as user authentication (which we shall be making use of). However, it means that every user that has an account on the site will have a default object in the 'Auth_User' table. Upon further investigation, this table contains much of the information that the program requires to be stored about the employee. For example, it stores the user's first name, second name, email address, password, and date joined. This led to further investigation on how to extend the User model, so that I could include data such as availability, phone number, and the manager Boolean. After long deliberation, I decided that the easiest way to store this data would be in a separate model aptly named 'Employees' that contains a one-to-one relationship with the Auth_User model.

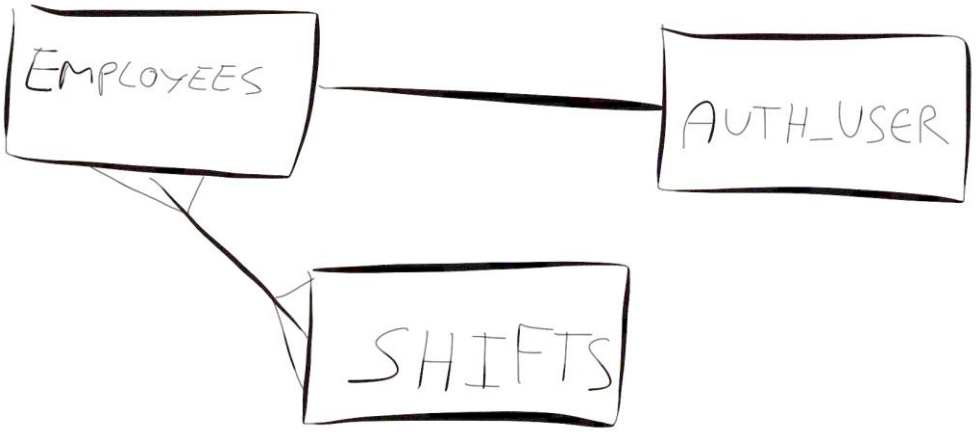
Although 'Total Hours Worked' would be a nice inclusion, it would be an estimate and require a lot more work and time to implement such a feature, however it may be a nice idea for future development. Therefore, the user-defined models will look something like the following:

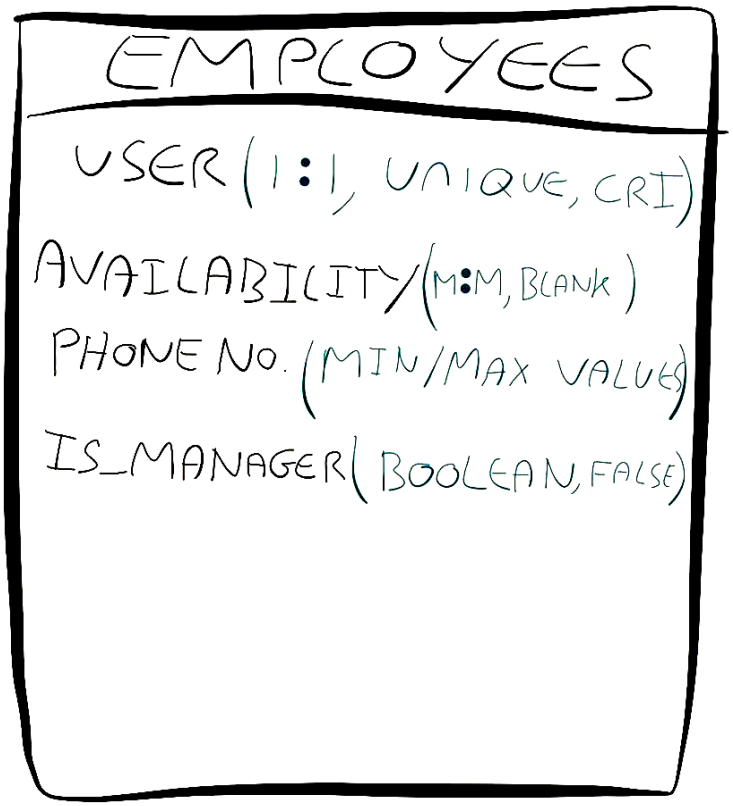
- Employees
  - Primary Key
  - Phone Number
  - User (1:1 With Auth_User)
  - Manager (Boolean)
  - Availability (M:M with Shifts)
- Shifts
  - Primary Key
  - Start (DateTime field)
  - End (DateTime field)
  - Employees (1:1 With Employees)
  - Shift Notes
  - Cover Requests

A key decision was one to assign a One-To-One relationship between the Shifts and Employees Model. This means that one employee can be assigned to many different shifts, however one shift can only be assigned to one employee. Multiple shift objects can be created with the same shift times, meaning that this is not a limitation to the day to day use of the program if more than one lifeguard is needed on duty at the same time. This could also potentially make the design of the algorithm much simpler.

DATABASE DESIGN AND FEATURES (CONT.)

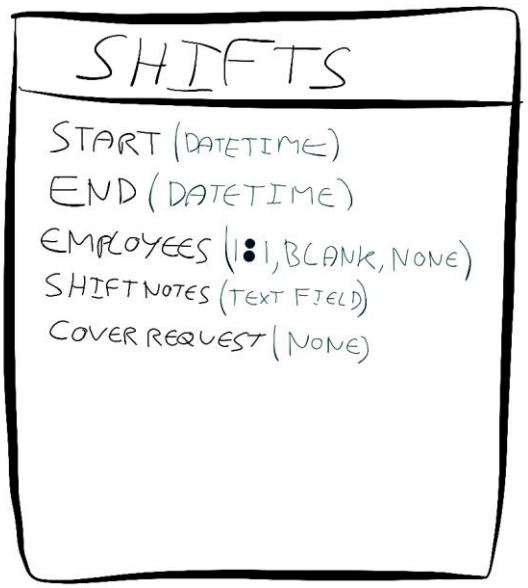
The Employees model will act as an extension to the default Auth_User, which could cause a few complications when creating new users, however it is doable according to many online forums. So far, the design regarding the models includes:



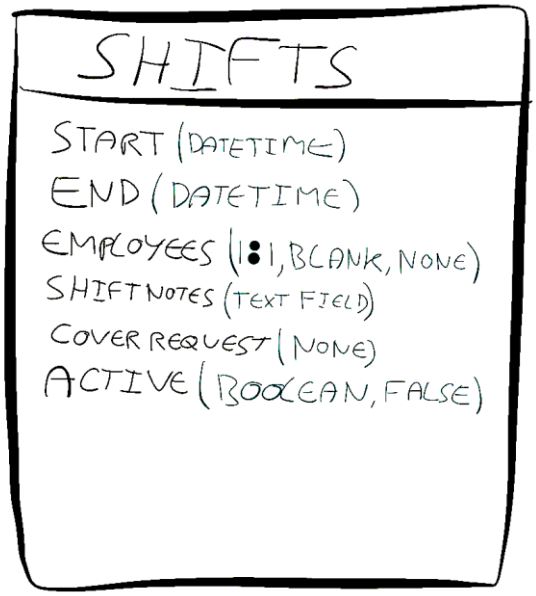


DATABASE DESIGN AND FEATURES (CONT.)

Initially, the Shift's model had the following attributes:



However, I later thought about how the models need to store all the shifts for both the current month's rota, and the next month's rota. This led me to question how the program will know whether any particular shift is for the current month or the next month. My first judgement was that the start time could be queried each time to check what month it is, however, this would be time consuming as the developer, and therefore a much simpler solution would be to add an extra Boolean field to query whether the shift is active or not. Therefore, the field will be named 'Active'.



DATABASE DESIGN AND FEATURES (CONT.)

VALIDATION AND SECURITY

When each field is defined within the model class, it can be given parameters which let Django know how to validate the field inputs. The validation takes place when any changes are saved to the database, be that through user-submitted forums or within the program itself. Django performs other sorts of validation on form inputs in order to prevent the user from submitting malicious inputs that could harm the website or database. This process is commonly known as 'cleaning data'. Django has two main functions that are used called clean() and is_valid(). is_valid() checks the input types are valid as the model field defines within the parameters. An example is checking that the user input is in a valid DateTime format. The purpose of the clean() function is to ensure that inputs such as SQL are rejected. This prevents the user from injecting code into the website forums that could be used to retrieve data from the database that should not be accessible by non-authorised users. An example of an attack like this happening in the past is Talk Talk's October 2015 SQL hack. The clean() function is automatically called when is_valid() is called. This makes it easier for the developers as it is one less thing to remember. Above, as shown in the class cards are the different validation that will be assigned to each field. Bellow there will be an Input and validation table which will clearly display examples of validation that takes place if a user submits a model form:

EMPLOYEES

Model Field	Description	Data Type	Validation
User	A 1:1 field to Auth_User, will make the User model accessible through the Employees model.	Object	The field will have to be unique, and more importantly have cascading referential integrity, meaning that if the Employee is deleted, the Auth_User object must be deleted too, and vice-versa.
Availability	A m:m field to Shifts, meaning that many employees can be available for many shifts.	QuerySet	The query set will only have the validation that it can be set to blank-meaning that it can have no value.

DATABASE DESIGN AND FEATURES (CONT.)

VALIDATION AND SECURITY (CONT.)

EMPLOYEES (CONT.)

Model Field	Description	Data Type	Validation
PhoneNo	An integer field used to store an employee's phone number.	Integer	The phone number will have to be within the range of 7000000000 and 7999999999.
IsManager	A Boolean field which will be used to query whether the employee is a manager or not. It will be used for user privileges.	Boolean	The field will have to be a Boolean value, however it will have the default parameter of FALSE.

SHIFTS

Model Field	Description	Data Type	Validation
Start	This field stores the start time of the shift.	DateTime	The field will have to be submitted in a DateTime format and is a required field.
End	This field stores the end time of the shift.	DateTime	The field will have to be submitted in a DateTime format and is a required field.
Employees	The field will be the basis of the 1:1 relationship between Shifts and Employees. It means that one shift can be assigned to one employee.	Object	The field will have an employee object stored in it; however it can also be left blank.

SHIFTS (CONT.)

Model Field	Description	Data Type	Validation
ShiftNotes	This is an optional feature that could store additional information about the specific shift (e.g. a pre-planned pool party is taking place)	String	This is a text field, meaning that the data will be stored as a string. It can also be left blank as it is an optional field.
CoverRequest	This again is an optional feature that may store other employee usernames that are able to cover said shift.	String	The validation of this field will be that the data stored is in fact an actual username in a string format.
Active	This is a Boolean field that stores whether the shift is currently locked in as the current month's rota or if it can be changed.	Boolean	The only validation is the data type, it can either be set to TRUE or FALSE.

USER INTERFACE AND USABILITY FEATURES

The main objective for the user interface is to keep it simple and intuitive. Due to many of the managers not being the most technically minded, the client has asked for a simple administrative interface, and a simple employee interface so that not a lot of training time has to be spent when implementing the application. Django's built in user interface is very powerful, however, for someone to use it effectively, they must understand how each table in the database is linked and where to enter specific pieces of data in order to add to, remove from, or maintain the records. If a mistake is made, it could cause inconsistencies or a loss of data that may end up corrupting the database itself (this is unlikely because Django has many built in methods to prevent this) or removing data that they did not realise the system required. Tasks that should be simple for the user would take much longer due to having to navigate an administrative interface with many features that are useless to this stakeholder. It is for this reason that instead of giving the managing account Django-administrative permissions, I shall put a Boolean option within every employee's record called IsManager. Each time the user attempts to access the management page, if it is true, the user will be redirected to the admin manage page. If, however it is false, the user will be redirected to the staff management page. These pages will contain very different features, applicable to each stakeholder needs. There will also be one account with superuser privileges called 'root' that will be used during the development process and will be used by the client's IT department for managing the implementation and maintenance of the system. The superuser account will be able to access every aspect of the site.

The website will include the following pages:

- Home
- Login
- Login Success
- Current Rota
- Potential Rota
- Management page for 'Managers'
- Management page for 'Employees'
- Django's administration console

All the website's features have been previously covered, however each feature shall be mapped onto a specific page in the future when it comes to designing the page layout and creating data flow diagrams.

SECURITY

There are a few security features that should be mentioned within the design of the user interface. The main feature that has already been touched on is the user account privileges. There will be 3 account levels:

1. Superuser (root)
2. Manager
3. Employee

USER INTERFACE AND USABILITY FEATURES (CONT.)

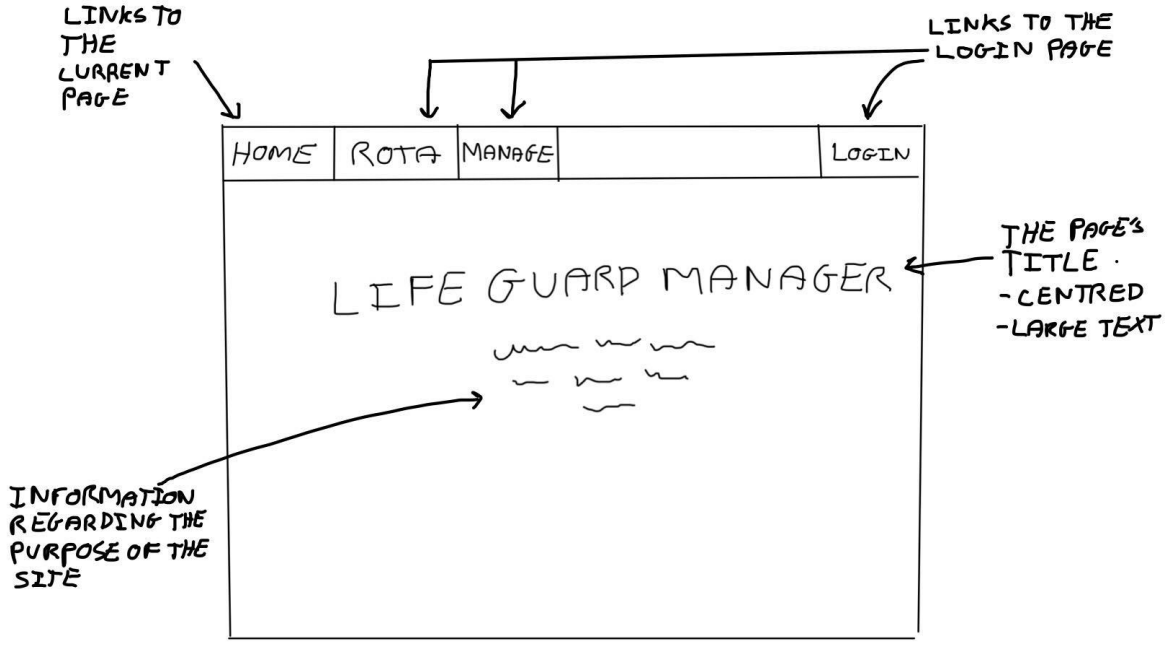
SECURITY (CONT.)

Each different user type will have its own unique access privileges to different areas which will contain different features. Each account type will have access to the features outlined within the success criteria, with one exception being the superuser account that was not mentioned. This account will have access to all the management features + the Django administrator console.

Only manager's will be able to register users. This is since hiring new employees does not happen frequently enough to warrant a dedicated sign up page with a separate feature for managers to accept the registration, thereby confirming it is an employee's account. It is much easier for a manager to sign an employee up and then relay login details to said employee once an account has been created.

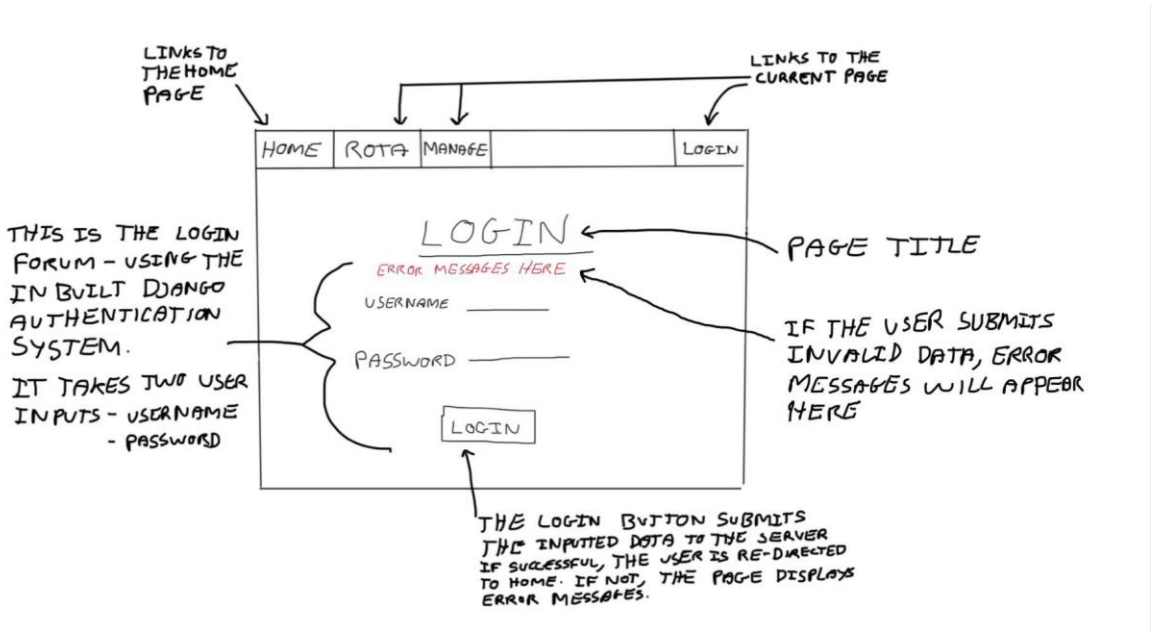
There are other security features that will be implemented; however, these are slightly more technical and will be covered in the 'Development' section.

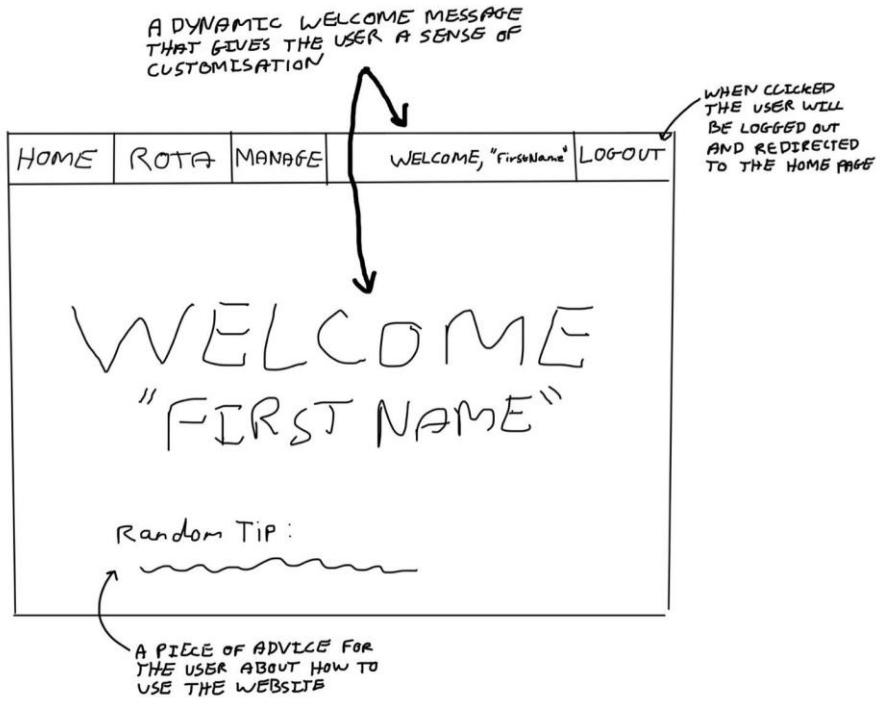
USER INTERFACE DESIGN



USER INTERFACE AND USABILITY FEATURES (CONT.)

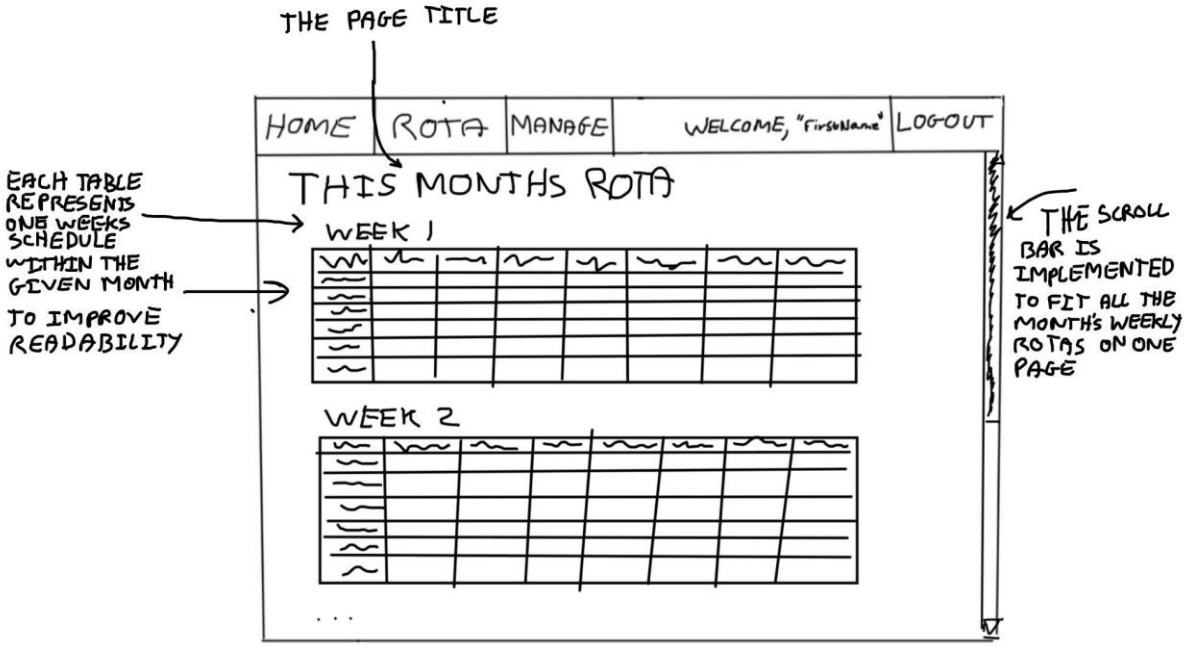
USER INTERFACE DESIGN (CONT.)

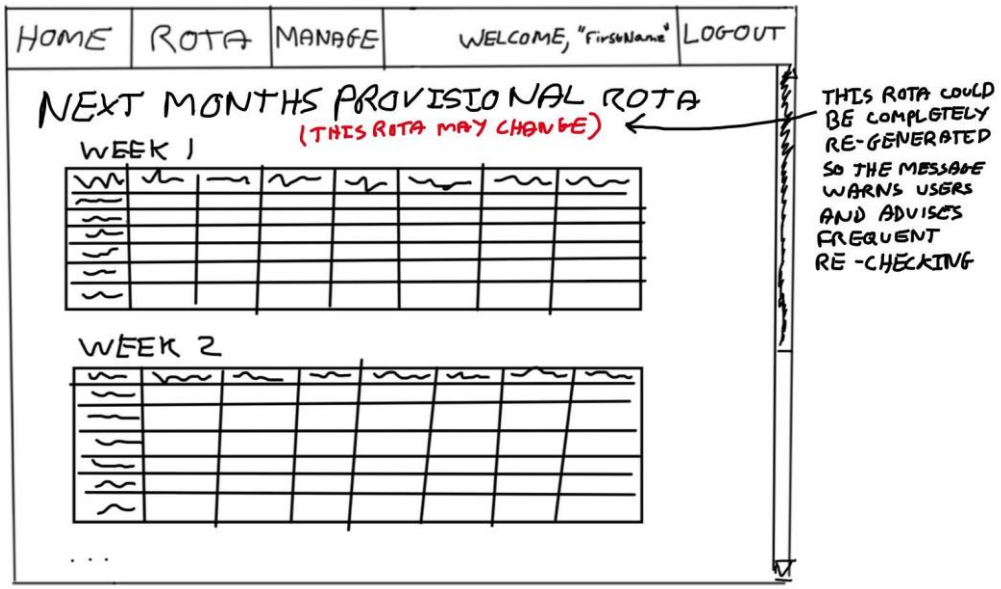




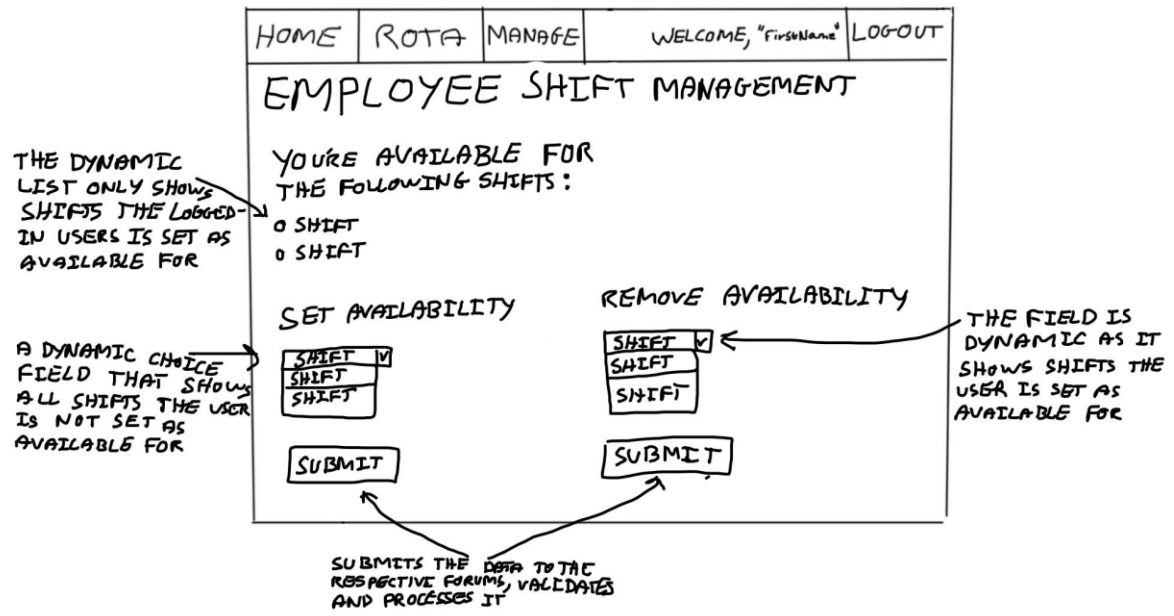
USER INTERFACE AND USABILITY FEATURES (CONT.)

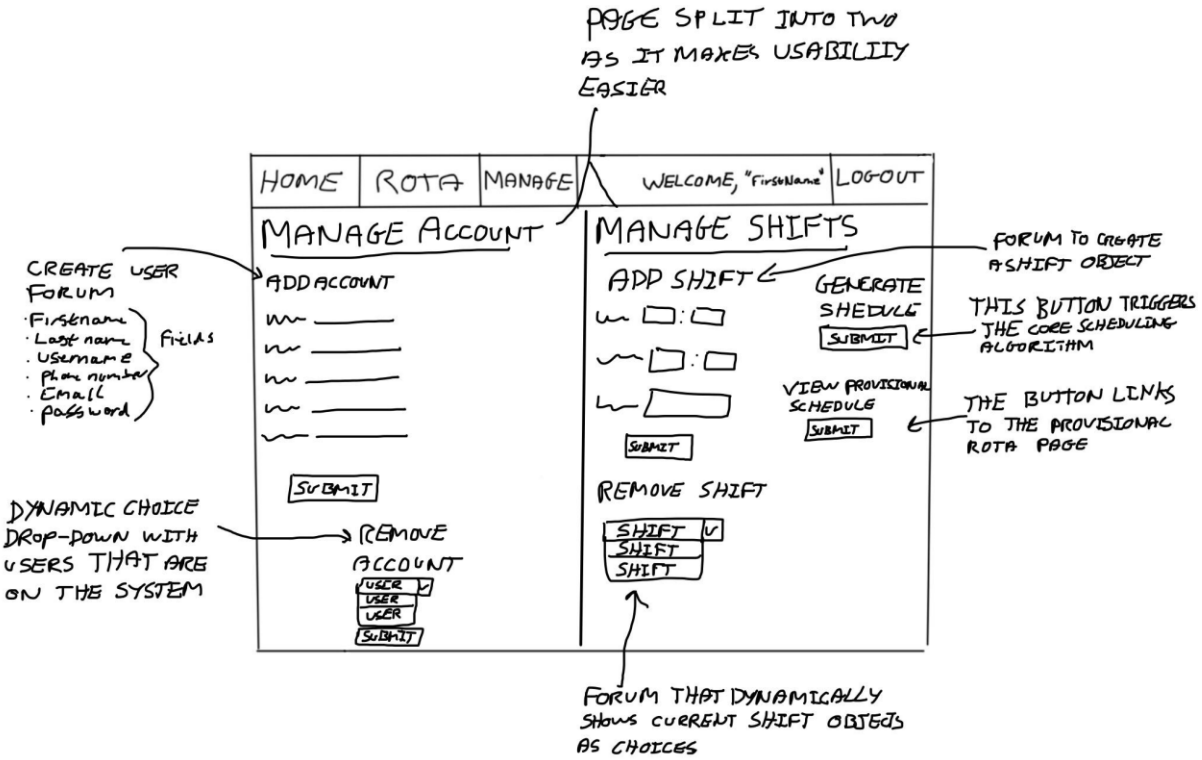
USER INTERFACE DESIGN (CONT.)





USER INTERFACE DESIGN (CONT.)





USER INTERFACE AND USABILITY FEATURES (CONT.)

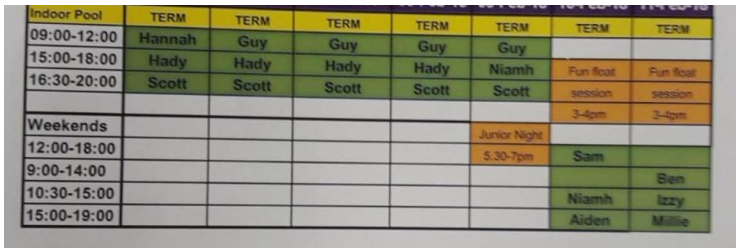
USER INTERFACE DESIGN (CONT.)

Figure 1
The home screen will be kept very simple and professional as it will be the first screen anyone sees when they visit the site. Due to the nature of the site, there are relatively few pages, and therefore most of the navigation to these key areas can take place through a navigation bar. Aside form the navigation bar, the home page will simply contain a title, and a little information about the website purpose.

Figure 2
The login page will simply consist of the default Django user authentication login forum, which will be re-styled to match the website theme using CSS. If the user submits the forum incorrectly, an error message will appear as shown in the page design above. When the user submits the form with valid data and correct credentials, the user will be logged in and redirected to a 'login success' page.

Figure 3
The login success page does not contain any forums for the user, however its purpose is to welcome the user onto the platform and provide some helpful advice regarding how to use the website. The page will make use of dynamically getting and displaying the user's first name when addressing them.

Figure 4
The rota page has the sole purpose of displaying the current months shifts in a readable and intuitive manor. My client is happy with the current format of his rota, and has requested that it is similar to the following so that staff don't have a hard time reading it, minimising misunderstandings:



Indoor Pool	TERM	TERM	TERM	TERM	TERM	TERM	TERM
09:00-12:00	Hannah	Guy	Guy	Guy	Guy		
15:00-18:00	Hady	Hady	Hady	Hady	Niamh	Fun float	Fun float
16:30-20:00	Scott	Scott	Scott	Scott	Scott	session	session
						3-4pm	3-4pm
Weekends					Junior Night		
12:00-18:00					5.30-7pm	Sam	
9:00-14:00							Ben
10:30-15:00						Niamh	Izzy
15:00-19:00						Aiden	Millie

Figure 5
The provisional rota page is very similar to the rota page in the sense that the information is displayed in the exact same way. The shift data differs since it is the shifts that are for the following month and can be changed at the manager's discretion by generating a new schedule.

USER INTERFACE AND USABILITY FEATURES (CONT.)

USER INTERFACE DESIGN (CONT.)

Figure 6

The employee management page is specifically for employee level accounts. Here, they can submit when they are available to work for the next month. When this happens, they will make themselves available for the algorithm to consider for that shift, however there is no guarantee they will get it. There will also be a forum to let them remove themselves from being available for that shift next month.

Figure 7

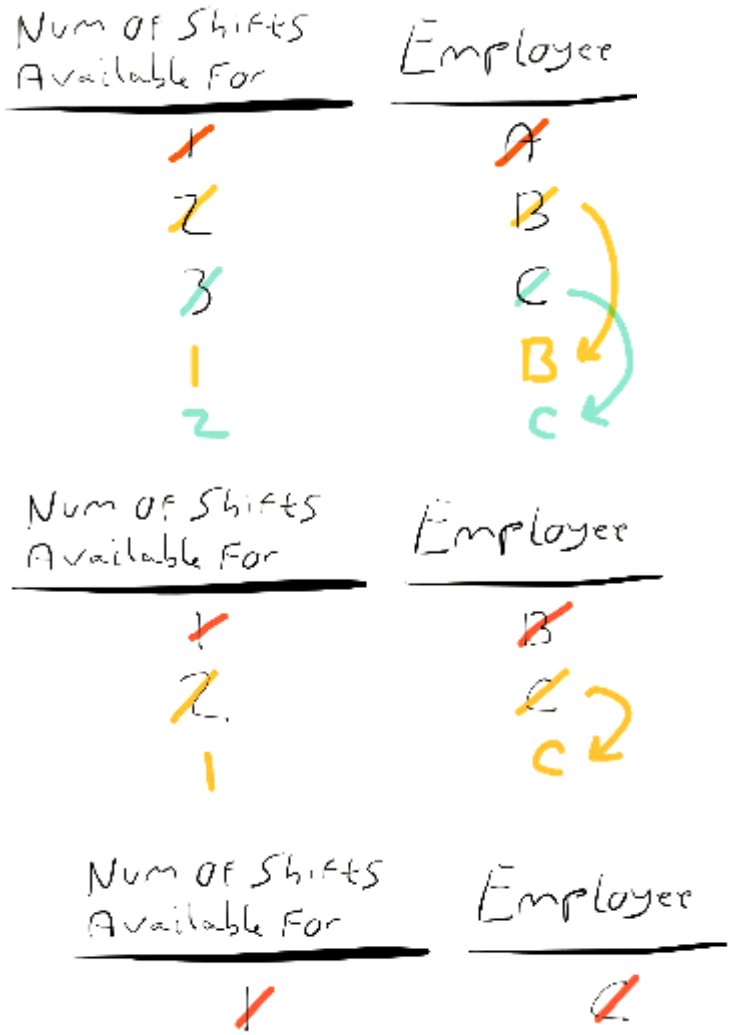
The Manager's Management page consists of many features that are easily categorized into account management and shift management. It is for this reason that the page is split into two- the user will find it easier to use it once they are familiar with the layout as they can quickly navigate to the features they require. The account management features will make use of Django's default user creation forum that the authentication module provides. This is useful as it takes care of password hashing and storage, making security much less of a concern as it has already been thoroughly tested. There will also be a drop-down menu consisting of all the user accounts that do not have the 'manager' status. If one of these accounts are selected, and the forum submitted, the account and all the data in the database that is related to that account will be deleted. This ensures database integrity, and Django has features that can be implemented when defining the database model fields that will ensure this. It is also true that the debugger will display error messages regarding database integrity if the deleting process does not remove every element of related data. The shift management page will include a forum to create a shift object. The fields will include start and end time, and the different days that shift is applicable to. There will also be a remove shift forum that includes all next month's potential shifts- the same ones that the core algorithm will use to fill with employees. A button used to generate a schedule will be available to the manager on this page as well. Once clicked, it will call the core algorithm which will assign the employees to the shift objects created by the manager beforehand. There will also be a button that links to next month's provisional rota page so that the manager has a quicker way to view the rota that the algorithm has generated. The final button will contain a button that removes last month's schedule from displaying on the current month's rota page and replaces it with a locked in version of the provisional rota. The provisional rota will then be made blank ready for the manager to set up again.

CORE ALGORITHM DESIGN AND FEATURES

The objective of this algorithm is that it ensures a full schedule, and where there are conflicts between two employees regarding shifts, it ensures a fair solution based on the number of shifts assigned to each employee that month.

Each employee is coupled with the number of shifts they are available for. For the purpose of simplicity this will be called the weight. There is more chance of an employee winning at least one shift if they are available for more. By taking this fact and the objective into consideration, the algorithm may not always guarantee an optimal solution. Instead, by sorting the odds (weight) into ascending order, when the list is iterated through those whom are least likely to get their shifts of choice can be considered and accommodated for first. An example of an algorithm that uses a system based on similar reasoning would be the First Fit Decreasing Algorithm that is commonly applied to Bin Packing problems.

This therefore means when visualising the above as an algorithmic solution, it may look like the following:



CORE ALGORITHM DESIGN AND FEATURES

FLOWCHART

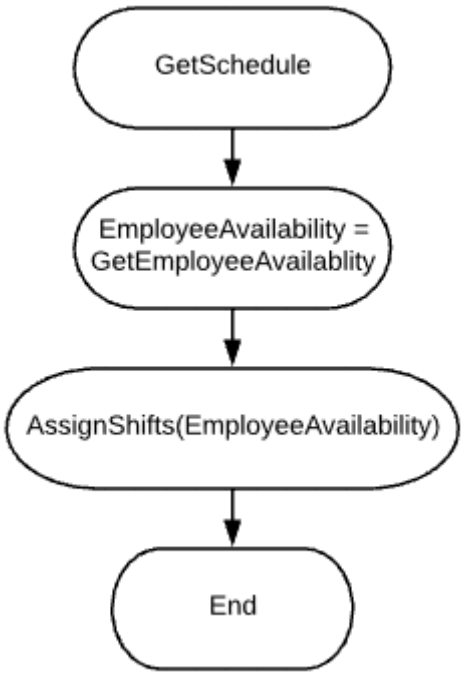
Those over-simplified worked examples help to identify two key processes that will need to take place that can then be brought together as subprocesses in one main procedure. These are getting the employee's availability and assigning the employees to the shifts as per the algorithm.

The flowcharts assume the use of some predefined functions that are built into either Python or Django's framework. Many of the functions will be used to interact with the database, however Django's ORM (Object Relational Mapping) means that items stored in the database will be accessed like they are objects which are generally more commonly used in application code. However, this doesn't mean that the algorithm will not work in different languages, although some methods may have to be elaborated, or adapted to the language in question. These predefined functions include:

- Append()
- Pop()
- get()

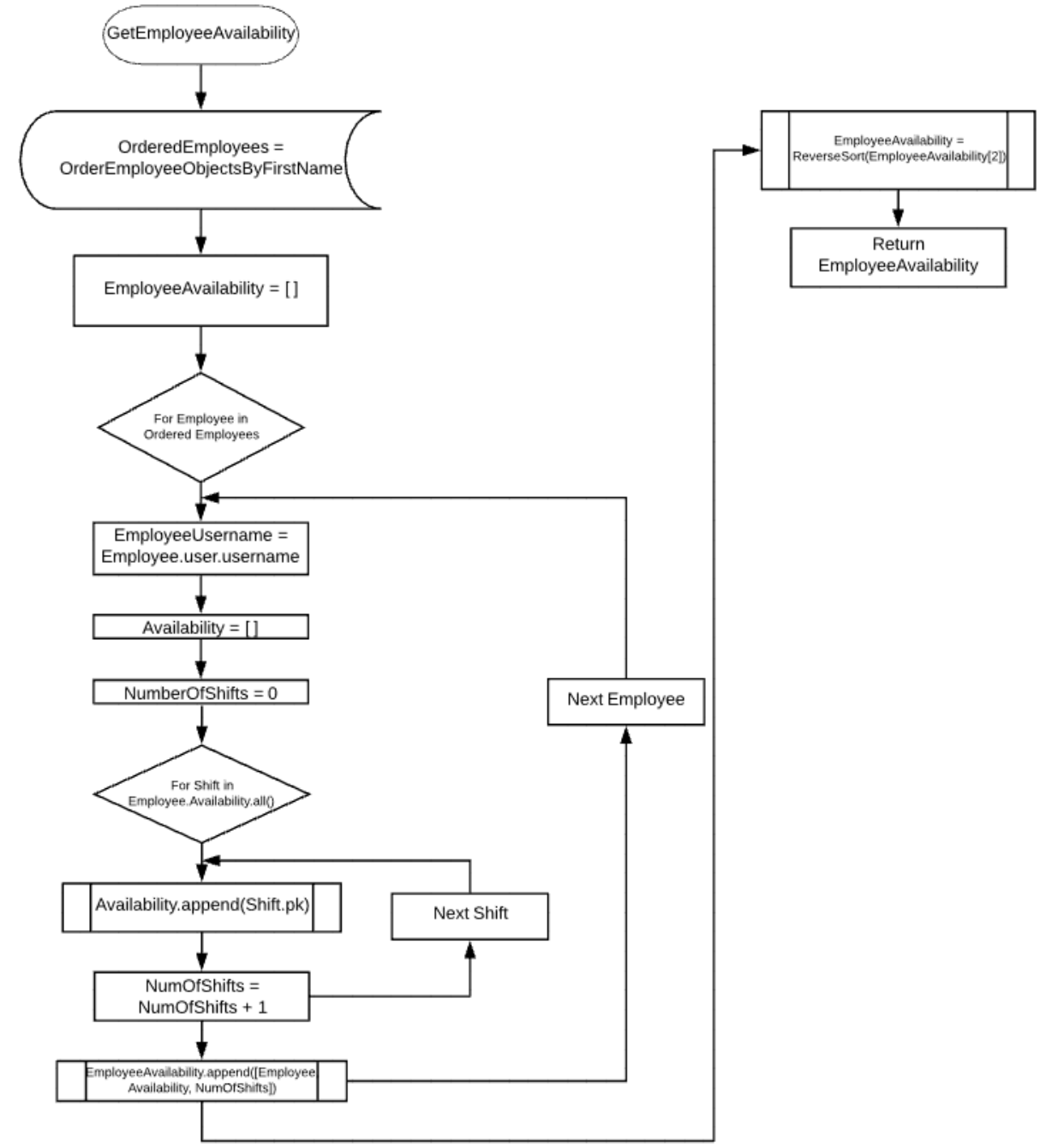
Other process descriptions include 'ReverseSort', however these are not predefined, just require a much longer line of code- for example in this case, it would be:
sorted(*ListInQuestion*, key=lambda x: x[*IndexOfListInQuestion*])
Therefore these are keyword descriptions of methods that may be available in various programming languages, therefore non-specific.

The last assumptions are that 'Employees' and 'Shifts' (the database objects) are imported.



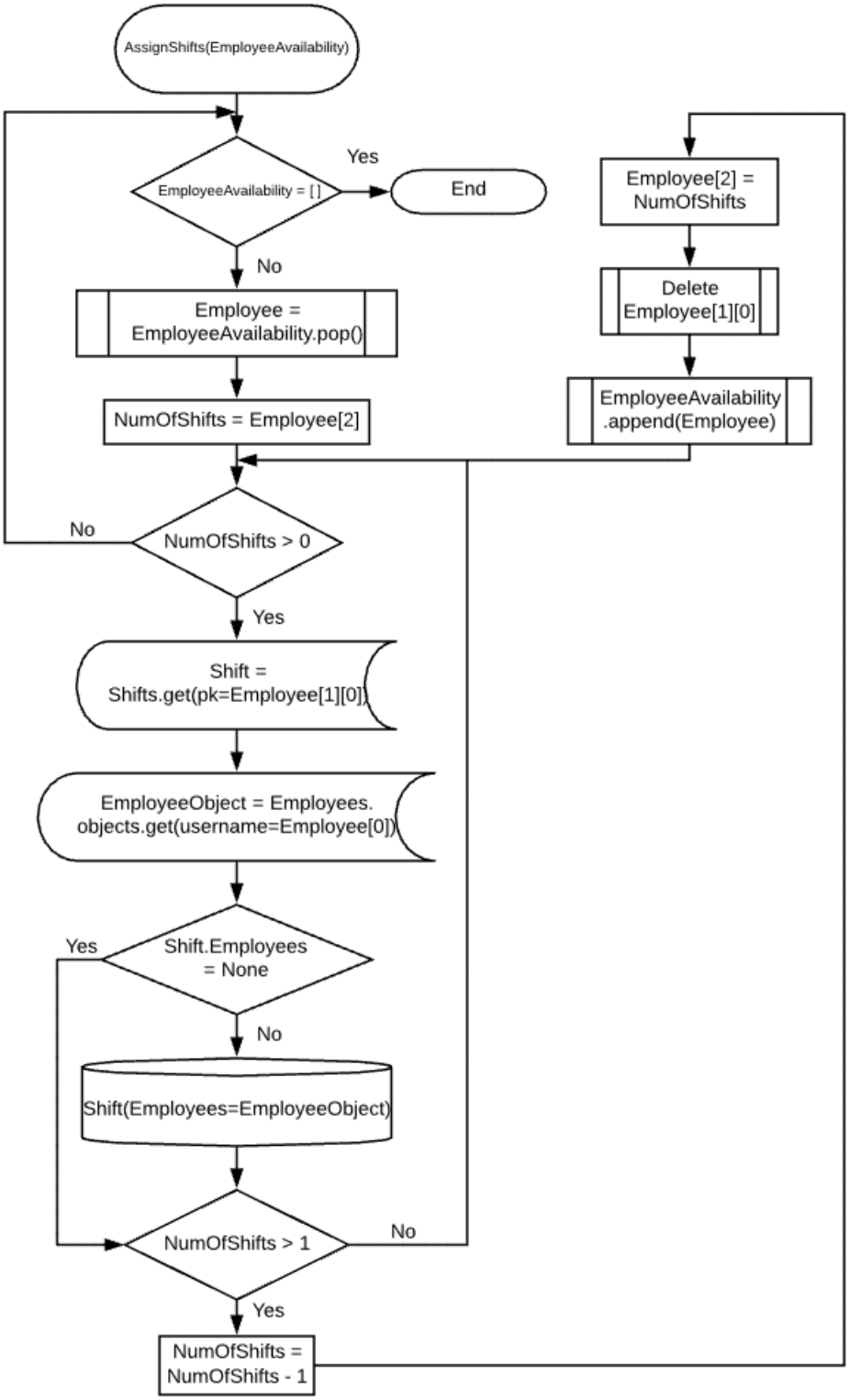
CORE ALGORITHM DESIGN AND FEATURES

FLOW CHART (CONT.)



CORE ALGORITHM DESIGN AND FEATURES

FLOW CHART (CONT.)



CORE ALGORITHM DESIGN AND FEATURES

PSEUDOCODE

```
PROCEDURE GetSchedule()
        EmployeeAvailability ← CALL GetEmployeeAvailability
        CALL AssignShifts(EmployeeAvailability)
END PROCEDURE


FUNCTION GetEmployeeAvailability()
        EmployeeAvailability ← [ ]
        FOR Employee IN Employees DO
                EmployeeUsername ← Employee.user.username
                Availability ← [ ]
                NumberOfShifts ← 0
                FOR Shift IN Employee.Availability.all() DO
                        Availability.Append(Shift.pk)
                        NumberOfShifts ← NumberOfShifts + 1
                END FOR
                AvailabiityList ← [EmployeeUsername, Availability, NumberOfShifts]
                EmployeeAvailability.append(AvailabilityList)
        END FOR
        EmployeeAvailability ← ResverseSort(EmployeeAvailability[2])
        RETURN EmployeeAvailability
END FUNCTION
```

PSEUDOCODE (CONT.)

```
PROCEDURE AssignShifts(EmployeeAvailability)
        WHILE EmployeeAvailability NOT Null DO
                Employee ← EmployeeAvailability.pop(0)
                NumberOfShifts ← Employee[2]
                IF NumberOfShifts > 0 THEN
                        Shift ← Shifts.get(pk=Employee[1][0])
                        EmployeeObject ← Employees.objects.get(username=Employee[0])
                        IF Shift.Employees = None THEN
                                Shift(Employees=EmployeeObject)
                        END IF
                        IF NumOfShifts > 1 THEN
                                NumberOfShifts = NumberOfShifts - 1
                                Employee[2] ← NumberOfShifts
                                Employee[1][0].delete()
                                EmployeeAvailability.append(Employee)
                        END IF
                END IF
        END WHILE
END PROCEDURE
```

CORE ALGORITHM DESIGN AND FEATURES

ALGORITHM'S KEY VARIABLES AND VALIDATION

Since the algorithm is split into 3 processes, there will be a table for each process and its key variables.

GETEMPLOYEEAVAILABILITY()

Variable	Description	Data Type	Validation	Comments
EmployeeAvailability	This is a list that will store an 'AvailabilityList' of each employee.	List	N/A	Since there is not user input, no validation for this variable will be required.
Employee	This variable will store an instance of a User's Employees object.	Object	N/A	Since there is not user input, no validation for this variable will be required. The variable is used to store the output of a database query.
EmployeeUsername	This variable will store the Employee's username as a string value.	String	N/A	This value comes from the Employee variable which originated from the database query, therefore since all data already stored in the database is validated, no validation is required.

ALGORITHM'S KEY VARIABLES AND VALIDATION (CONT.)

GETEMPLOYEEAVAILABILITY()

Variable	Description	Data Type	Validation	Comments
Availability	This variable is a list that will store the primary keys of each shift that the user is available for.	List	N/A	No validation is required as only IDs that exist will be stored within this variable
NumberOfShifts	The variable is used as a counter so that the program can keep track of how many shifts the user is available for.	Integer	N/A	The count is used instead of using inbuilt function to find the length of the 'Availability' list.
Shift	The Shift variable is used within a for loop, and will store one of the shift objects that is returned in the Employee.Availability.all() query.	Object	N/A	This variable holds values from a database query.
AvailabilityList	This variable is a list which will contain the username, the 'Availability' variable, and NumberOfShifts	List	N/A	This list brings together all the queried information into a workable format which will be appended to the EmployeeAvailability list.

ALGORITHM'S KEY VARIABLES AND VALIDATION (CONT.)

ASSIGNSHIFTS()

Variable	Description	Data Type	Validation	Comments
EmployeeAvailability	This variable is a required function input. It is developed by the GetEmployeeAvailability function which is called beforehand.	List	N/A	Although an input, the procedure to get the input will ensure the variable is stored in the required format with validated information.
Employee	This variable is part of a for loop that stores an element of EmployeeAvailability. This element will be a list.	List	N/A	This variable will be in the same format as the 'AvailabilityList' in the previous procedure.
NumberOfShifts	This variable will get and store the value of the element with index [2] in the Employee list.	Integer	N/A	It makes the code more human readable to store it in a separate variable then to constantly access the list index.
Shift	This variable stores an instance of a shift object that has been queried from the database.	Object	N/A	No validation is required as no action has been taken.
EmployeeObject	This variable stores an instance of an employee object that has been queried from the database.	Object	N/A	No validation is required as no action has been taken.

CORE ALGORITHM DESIGN AND FEATURES

HAND TRACE- VIABILITY OF PROPOSED SOLUTION TESTING

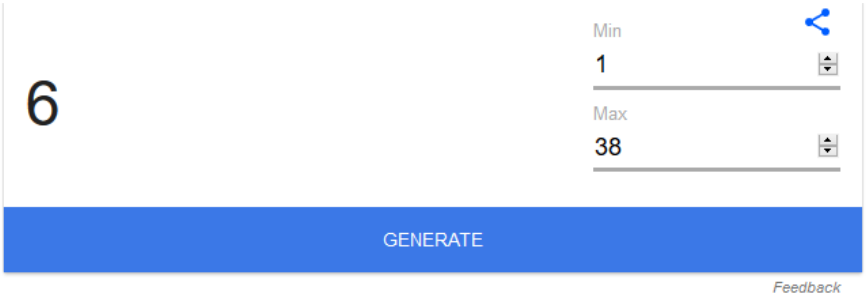
An example of a blank rota for the first week of February 2018

Shift Time	01/02/2018	02/02/2018	03/02/2018	04/02/2018	05/02/2018	06/02/2018	07/02/2018
09:00 - 14:00							
10:30 - 15:00							
12:00 - 18:00							
15:00 - 18:00							
15:00 - 19:00							
16:30 - 20:00							

The same rota, however each white space (or shift) has been allocated a numerical primary key

Shift Time	01/02/2018	02/02/2018	03/02/2018	04/02/2018	05/02/2018	06/02/2018	07/02/2018
09:00 - 14:00	1	2	3	4	5	6	7
10:30 - 15:00	8	9	10	11	12	13	14
12:00 - 18:00	15	16	17	18	19	20	21
15:00 - 18:00	22	23			24	25	26
15:00 - 19:00	27	28	29	30	31	32	33
16:30 - 20:00	34	35			36	37	38

Google's random integer generator will be used to assign available shifts to each employee. We will stop assigning availability once all the shifts have at least one employee assigned. This method of generating test data ensures that there is minimum bias. The number of shifts each employee has will be random, but will include test data for boundary cases such as 0.



CORE ALGORITHM DESIGN AND FEATURES

HAND TRACE- VIABILITY OF PROPOSED SOLUTION TESTING (CONT.)

FirstName	LastName	Username	Email	Availability									
Michael	*Brooks*	*MichaelB*	michaelb@wentworthlifeguards.com										
Hannah	Atkins	HannahA	hannaha@wentworthlifeguards.com	38	19	13							
Hady	Johnson	HadyJ	haddyj@wentworthlifeguards.com	5									
Scott	Samson	ScottS	scotts@wentworthlifeguards.com	6	7	14	9	26	3	20	8	11	18
Guy	Brick	GuyB	guyb@wentworthlifeguards.com	21	29	25	31						
Niamh	Baguette	NiamhB	niamhb@wentworthlifeguards.com	5	2	35	1	22	16				
Sam	Lennon	SamL	saml@wentworthlifeguards.com	33	8	34	17						
Jasmin	Dameon	JasminD	jasmind@wentworthlifeguards.com	36	34	21	4	30	10	14	12		
Aiden	Ukov	AidenU	aidenu@wentworthlifeguards.com	35	11	26							
Ben	Dargon	BenD	bend@wentworthlifeguards.com	32	9	24	15						
Izzy	Circuit	IzzyC	izzyc@wentworthlifeguards.com	14	29	28	27	3					
Millie	Lettice	MillieL	milliel@wentworthlifeguards.com	1	32	23	11	37	19	16			

The white cells indicate repeated availability, while the black cells represent no availability. The rest are orange.

GETEMPLOYEEAVAILABILITY()

Line Number	EmployeeAvailability	Employee	EmployeeUsername	Availability	NumberOfShifts	Shift	AvailabilityList
1	[]						
2		MichaelB (Object)					
3			MichaelB				
4				[ ]			
5					0		
10							[MichaelB, [], 0]
11	[[MichaelB, [], 0]]						

Line Number	EmployeeAvailability	Employee	EmployeeUsername	Availability	NumberOfShifts	Shift	AvailabilityList
2		HannahA (Object)					
3			HannahA				
4				[]			
5					0		
6						38	
7				[38]			
8					1		
6						19	
7				[38,19]			
8					2		
6						13	
7				[38,19,13]			
8					3		
10							[HannahA, [38,19,13], 3]
11	[[MichaelB, [], 0], [HannahA, [38,19,13], 3]],						

CORE ALGORITHM DESIGN AND FEATURES

HAND TRACE- VIABILITY OF PROPOSED SOLUTION TESTING (CONT.)

Line Number	EmployeeAvailability	Employee	EmployeeUsername	Availability	NumberOfShifts	Shift	AvailabilityList
2		HadyJ (Object)					
3			HadyJ				
4				[]			
5					0		
6						5	
7				[5]			
8					1		
10							[HadyJ, [5], 1]
11	[[MichaelB, [], 0], [HannahA, [38,19,13], 3]], [HadyJ, [5], 1]						

CORE ALGORITHM DESIGN AND FEATURES

HAND TRACE- VIABILITY OF PROPOSED SOLUTION TESTING (CONT.)

Line Number	EmployeeAvailability	Employee	EmployeeUsername	Availability	NumberOfShifts	Shift	AvailabilityList
2		ScottS (Object)					
3			ScottS				
4				[]			
5					0		
6						6	
7				[6]			
8					1		
6						7	
7				[6,7]			
8					2		
6						14	
7				[6,7,14]			
8					3		
6						9	
7				[6,7,14,9]			
8					4		
6						26	
7				[6,7,14,9,26]			
8					5		
6						3	
7				[6,7,14,9,26,3]			
8					6		
6						20	
7				[6,7,14,9,26,3,20]			
8					7		
6						8	
7				[6,7,14,9,26,3,20,8]			
8					8		
6						11	
7				[6,7,14,9,26,3,20,8,11]			
8					9		
6						18	
7				[6,7,14,9,26,3,20,8,11,18]			
8					10		
10							[ScottS, [6,7,14,9,26,3,20,8,11, 18], 10]
11	[[MichaelB, [], 0], [HannahA, [38,19,13], 3]], [HadyJ, [5], 1], [ScottS, [6,7,14,9,26,3,20,8,11, 18], 10]						

HAND TRACE- VIABILITY OF PROPOSED SOLUTION TESTING (CONT.)

Line Number	EmployeeAvailability	Employee	EmployeeUsername	Availability	NumberOfShifts	Shift	AvailabilityList
2		GuyB (Object)					
3			GuyB				
4				[]			
5					0		
6						21	
7				[21]			
8					1		
6						29	
7				[21,29]			
8					2		
6						25	
7				[21,29,25]			
8					3		
6						31	
7				[21,29,25,31]			
8					4		
10							[GuyB,[21,29,25,31], 4]
11	[[MichaelB, [], 0], [HannahA, [38,19,13], 3]], [HadyJ, [5], 1], [ScottS, [6,7,14,9,26,3,20,8,11, 18], 10] [GuyB,[21,29,25,31], 4]						

HAND TRACE- VIABILITY OF PROPOSED SOLUTION TESTING (CONT.)

Line Number	EmployeeAvailability	Employee	EmployeeUsername	Availability	NumberOfShifts	Shift	AvailabilityList
2		Niamh (Object)					
3			Niamh				
4				[]			
5					0		
6						5	
7				[5]			
8					1		
6						2	
7				[5,2]			
8					2		
6						35	
7				[5,2,35]			
8					3		
6						1	
7				[5,2,35,1]			
8					4		
6						22	
7				[5,2,35,1,22]			
8					5		
6						16	
7				[5,2,35,1,22,16]			
8					6		
10							[Niamh,[5,2,35,1,22,16], 6]
11	[[MichaelB, [], 0], [HannahA, [38,19,13], 3]], [HadyJ, [5], 1], [ScottS, [6,7,14,9,26,3,20,8,11, 18], 10], [GuyB,[21,29,25,31], 4], [Niamh,[5,2,35,1,22,16], 6]						

CORE ALGORITHM DESIGN AND FEATURES

HAND TRACE- VIABILITY OF PROPOSED SOLUTION TESTING (CONT.)

Line Number	EmployeeAvailability	Employee	EmployeeUsername	Availability	NumberOfShifts	Shift	AvailabilityList
2		SamL (Object)					
3			SamL				
4				[]			
5					0		
6						33	
7				[33]			
8					1		
6						8	
7				[33,8]			
8					2		
6						34	
7				[33,8,34]			
8					3		
6						17	
7				[33,8,34,17]			
8					4		
10							[SamL,[33,8,34,17], 4]
11	[[MichaelB, [], 0], [HannahA, [38,19,13], 3]], [HadyJ, [5], 1], [ScottS, [6,7,14,9,26,3,20,8,11, 18], 10], [GuyB,[21,29,25,31], 4], [Niamh,[5,2,35,1,22,16], 6], [SamL,[33,8,34,17], 4]						

Line Number	EmployeeAvailability	Employee	EmployeeUsername	Availability	NumberOfShifts	Shift	AvailabilityList
2		AidenU (Object)					
3			AidenU				
4				[]			
5					0		
6						35	
7				[35]			
8					1		
6						11	
7				[35,11]			
8					2		
6						26	
7				[35,11,26]			
8					3		
10							[AidenU,[35,11,26], 3]
11	[[MichaelB, [], 0], [HannahA, [38,19,13], 3]], [HadyJ, [5], 1], [ScottS, [6,7,14,9,26,3,20,8,11, 18], 10], [GuyB,[21,29,25,31], 4], [Niamh,[5,2,35,1,22,16], 6], [SamL,[33,8,34,17], 4], [JasminD,[36,34,21,4,30,10,14,12], 8], [AidenU,[35,11,26], 3]						

CORE ALGORITHM DESIGN AND FEATURES

HAND TRACE- VIABILITY OF PROPOSED SOLUTION TESTING (CONT.)

Line Number	EmployeeAvailability	Employee	EmployeeUsername	Availability	NumberOfShifts	Shift	AvailabilityList
2		BenD (Object)					
3			BenD				
4				[]			
5					0		
6						32	
7				[32]			
8					1		
6						9	
7				[32,9]			
8					2		
6						24	
7				[32,9,24]			
8					3		
6						15	
7				[32,9,24,15]			
8					4		
10							[BenD,[32,9,24,15], 4]
11	[[MichaelB, [], 0], [HannahA, [38,19,13], 3]], [HadyJ, [5], 1], [ScottS, [6,7,14,9,26,3,20,8,11, 18], 10], [GuyB,[21,29,25,31], 4], [Niamh,[5,2,35,1,22,16], 6], [SamL,[33,8,34,17], 4], [JasminD,[36,34,21,4,30,10,14,12], 8], [AidenU,[35,11,26], 3], [BenD,[32,9,24,15], 4]						

CORE ALGORITHM DESIGN AND FEATURES

HAND TRACE- VIABILITY OF PROPOSED SOLUTION TESTING (CONT.)

Line Number	EmployeeAvailability	Employee	EmployeeUsername	Availability	NumberOfShifts	Shift	AvailabilityList
2		IzzyC (Object)					
3			IzzyC				
4				[]			
5					0		
6						14	
7				[14]			
8					1		
6						29	
7				[14,29]			
8					2		
6						28	
7				[14,29,28]			
8					3		
6						27	
7				[14,29,28,27]			
8					4		
6						3	
7				[14,29,28, 27,3]			
8					5		
10							[IzzyC,[14,29,28,27,3], 5]
11	[[MichaelB, [], 0], [HannahA, [38,19,13], 3]], [HadyJ, [5], 1], [ScottS, [6,7,14,9,26,3,20,8,11, 18], 10], [GuyB,[21,29,25,31], 4], [Niamh,[5,2,35,1,22,16], 6], [SamL,[33,8,34,17], 4], [JasminD,[36,34,21,4,30,10,14,12], 8], [AidenU,[35,11,26], 3], [BenD,[32,9,24,15], 4], [IzzyC,[14,29,28,27,3], 5]						

CORE ALGORITHM DESIGN AND FEATURES

HAND TRACE- VIABILITY OF PROPOSED SOLUTION TESTING (CONT.)

Line Number	EmployeeAvailability	Employee	EmployeeUsername	Availability	NumberOfShifts	Shift	AvailabilityList
2		MillieL (Object)					
3			MillieL				
4				[]			
5					0		
6						1	
7				[1]			
8					1		
6						32	
7				[1,32]			
8					2		
6						23	
7				[1,32,23]			
8					3		
6						11	
7				[1,32,23,11]			
8					4		
6						37	
7				[1,32,23,11,37]			
8					5		
6						19	
7				[1,32,23,11,37,19]			
8					6		
6						16	
7				[1,32,23,11,37,19,16]			
8					7		
10							[MillieL,[1,32,23,11,37,19,16], 7]
11	[[MichaelB, [], 0], [HannahA, [38,19,13], 3]], [HadyJ, [5], 1], [ScottS, [6,7,14,9,26,3,20,8,11, 18], 10], [GuyB,[21,29,25,31], 4], [Niamh,[5,2,35,1,22,16], 6], [SamL,[33,8,34,17], 4], [JasminD,[36,34,21,4,30,10,14,12], 8], [AidenU,[35,11,26], 3], [BenD,[32,9,24,15], 4], [IzzyC,[14,29,28,27,3], 5], [MillieL,[1,32,23,11,37,19,16], 7]]						

CORE ALGORITHM DESIGN AND FEATURES

HAND TRACE- VIABILITY OF PROPOSED SOLUTION TESTING (CONT.)

Line Number	EmployeeAvailability	Employee	EmployeeUsername	Availability	NumberOfShifts	Shift	AvailabilityList
2		JasminD (Object)					
3			JasminD				
4				[]			
5					0		
6						36	
7				[36]			
8					1		
6						34	
7				[36,34]			
8					2		
6						21	
7				[36,34,21]			
8					3		
6						4	
7				[36,34,21,4]			
8					4		
6						30	
7				[36,34,21,4,30]			
8					5		
6						10	
7				[36,34,21,4,30,10]			
8					6		
6						14	
7				[36,34,21,4,30,10,14]			
8					7		
6						12	
7				[36,34,21,4,30,10,14,12]			
8					8		
10							[JasminD,[36,34,21,4,30,10,14,12], 8]
11	[[MichaelB, [], 0], [HannahA, [38,19,13], 3]], [HadyJ, [5], 1], [ScottS, [6,7,14,9,26,3,20,8,11, 18], 10], [GuyB,[21,29,25,31], 4], [Niamh,[5,2,35,1,22,16], 6], [SamL,[33,8,34,17], 4], [JasminD,[36,34,21,4,30,10,14,12], 8]						

CORE ALGORITHM DESIGN AND FEATURES

HAND TRACE- VIABILITY OF PROPOSED SOLUTION TESTING (CONT.)

Line Number	EmployeeAvailability	Employee	EmployeeUsername	Availability	NumberOfShifts	Shift	AvailabilityList
	[[MichaelB,[], 0], [HadyJ,[5], 1], [HannahA,[38,19,13], 3], [AidenU,[35,11,26], 3], [GuyB,[21,29,25,31], 4], [SamL,[33,8,34,17], 4], [BenD,[32,9,24,15], 4], [IzzyC,[14,29,28,27,3], 5], [Niamh,[5,2,35,1,22,16], 6], [MillieL,[1,32,23,11,37,19,16], 7], [JasminD,[36,34,21,4,30,10,14,12], 8],						
13	[ScottS, [6,7,14,9,26,3,20,8,11, 18], 10]]						
14	**RETURNED**						

ASSIGNSHIFTS()

Line Number	EmployeeAvailability	Employee	NumberOfShifts	Shift	Save Shift: Shift(Employees = x)
	[[MichaelB,[], 0], [HadyJ,[5], 1], [HannahA,[38,19,13], 3], [AidenU,[35,11,26], 3], [GuyB,[21,29,25,31], 4], [SamL,[33,8,34,17], 4], [BenD,[32,9,24,15], 4], [IzzyC,[14,29,28,27,3], 5], [Niamh,[5,2,35,1,22,16], 6], [MillieL,[1,32,23,11,37,19,16], 7], [JasminD,[36,34,21,4,30,10,14,12], 8],				
1	[ScottS, [6,7,14,9,26,3,20,8,11, 18], 10]]				
	[[HadyJ,[5], 1], [HannahA,[38,19,13], 3], [AidenU,[35,11,26], 3], [GuyB,[21,29,25,31], 4], [SamL,[33,8,34,17], 4], [BenD,[32,9,24,15], 4], [IzzyC,[14,29,28,27,3], 5], [Niamh,[5,2,35,1,22,16], 6], [MillieL,[1,32,23,11,37,19,16], 7], [JasminD,[36,34,21,4,30,10,14,12], 8],				
2	[ScottS, [6,7,14,9,26,3,20,8,11, 18], 10]]	[MichaelB,[], 0]			
3			0		

HAND TRACE- VIABILITY OF PROPOSED SOLUTION TESTING (CONT.)

Line Number	EmployeeAvailability	Employee	NumberOfShifts	Shift	Save Shift: Shift(Employees = x)	EmployeeObject
2	[HannahA,[38,19,13], 3], [AidenU,[35,11,26], 3], [GuyB,[21,29,25,31], 4], [SamL,[33,8,34,17], 4], [BenD,[32,9,24,15], 4], [IzzyC,[14,29,28,27,3], 5], [Niamh,[5,2,35,1,22,16], 6], [MillieL,[1,32,23,11,37,19,16], 7], [JasminD,[36,34,21,4,30,10,14,12], 8], [ScottS, [6,7,14,9,26,3,20,8,11, 18], 10]]	[HadyJ,[5], 1]				
3			1			
5				Shift Object (pk=1)		
6						EmployeeObject(HadyJ)
8					EmployeeObject(HadyJ)	

Line Number	EmployeeAvailability	Employee	NumberOfShifts	Shift	Save Shift: Shift(Employees = x)	EmployeeObject
2	[AidenU,[35,11,26], 3], [GuyB,[21,29,25,31], 4], [SamL,[33,8,34,17], 4], [BenD,[32,9,24,15], 4], [IzzyC,[14,29,28,27,3], 5], [Niamh,[5,2,35,1,22,16], 6], [MillieL,[1,32,23,11,37,19,16], 7], [JasminD,[36,34,21,4,30,10,14,12], 8], [ScottS, [6,7,14,9,26,3,20,8,11, 18], 10]]	[HannahA,[38,19,13], 3]				
3			3			
5				Shift Object (pk=38)		
6						EmployeeObject(HannahA)
8					EmployeeObject(HannahA)	
11			2			
12		[HannahA,[38,19,13], 2]				
13		[HannahA,[19,13], 2]				
14	[AidenU,[35,11,26], 3], [GuyB,[21,29,25,31], 4], [SamL,[33,8,34,17], 4], [BenD,[32,9,24,15], 4], [IzzyC,[14,29,28,27,3], 5], [Niamh,[5,2,35,1,22,16], 6], [MillieL,[1,32,23,11,37,19,16], 7], [JasminD,[36,34,21,4,30,10,14,12], 8], [ScottS, [6,7,14,9,26,3,20,8,11, 18], 10]], [HannahA,[19,13], 2]					

Unfortunately, there will be 53 more iterations of this exact procedure making it highly time consuming to fully complete the trace, however the above tests show the program producing expected results, therefore there shouldn't be any logical errors in the development process. A few boundary conditions were shown throughout the assignment procedure such as where an employee is not available for any shifts, where an employee is only available for 1 shift, and a normal case where the employee is available for more than 1 shift. For the sake of completeness, the next example is where the shift that the employee is available for has been filled by another employee already.

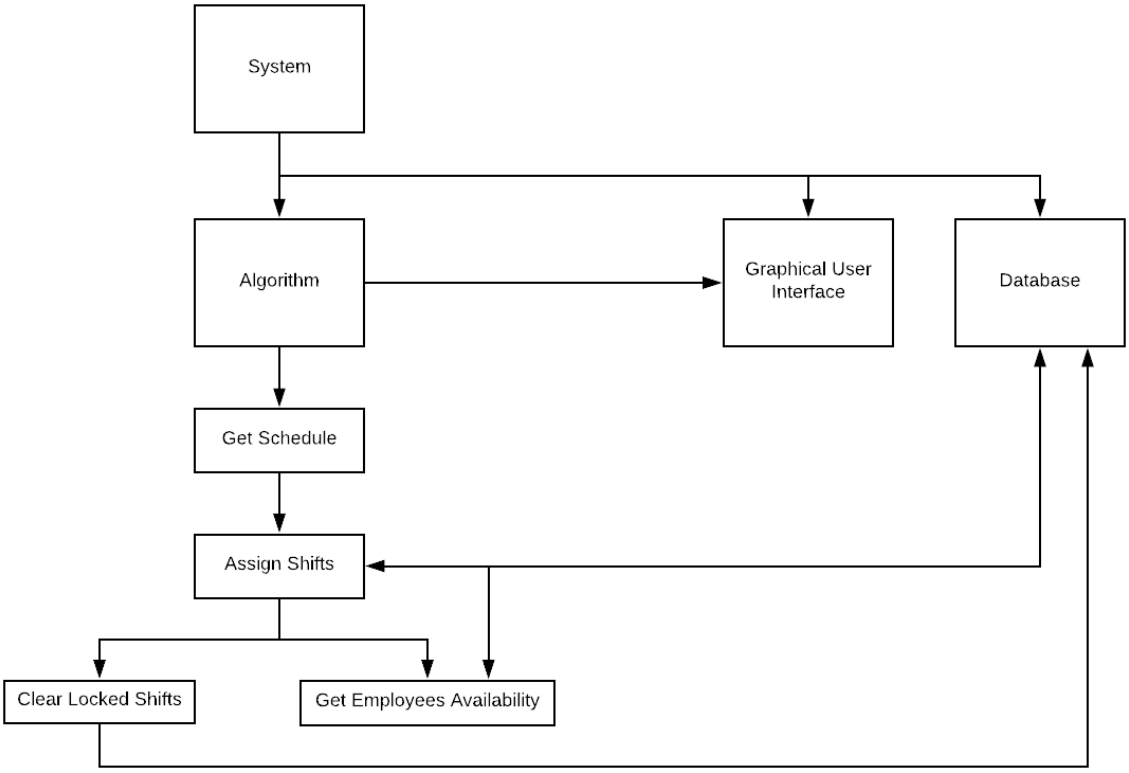
CORE ALGORITHM DESIGN AND FEATURES

HAND TRACE- VIABILITY OF PROPOSED SOLUTION TESTING (CONT.)

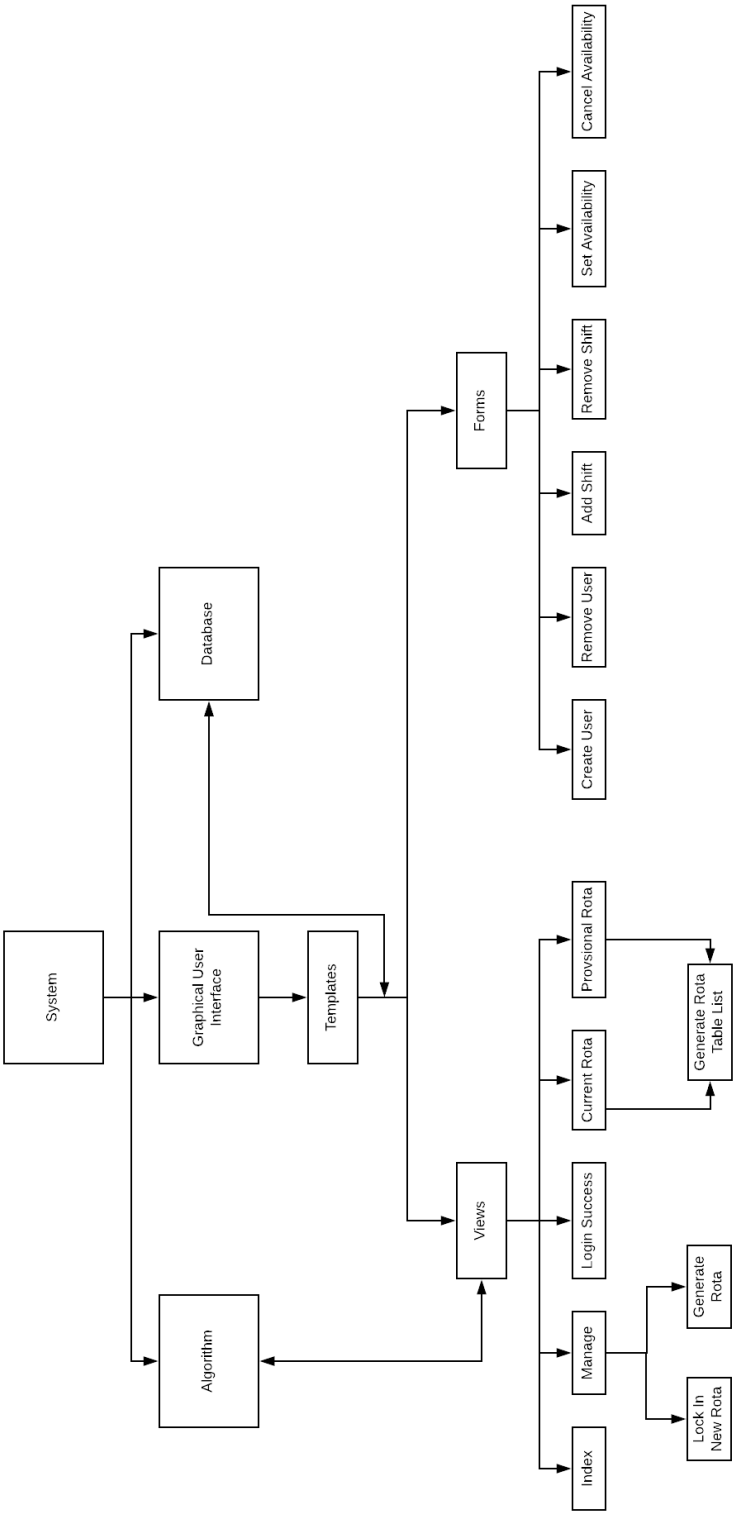
This example only consists of one employee with the assumption that the database already exists with other employees filling the shifts.

Line Number	EmployeeAvailability	Employee	NumberOfShifts	Shift	Save Shift: Shift(Employees = x)	EmployeeObject
2	[JoshM, [20],1]	[JoshM, [20],1]				
3			1			
5				Shift Object (pk=20)		
6						EmployeeObject(JoshM)

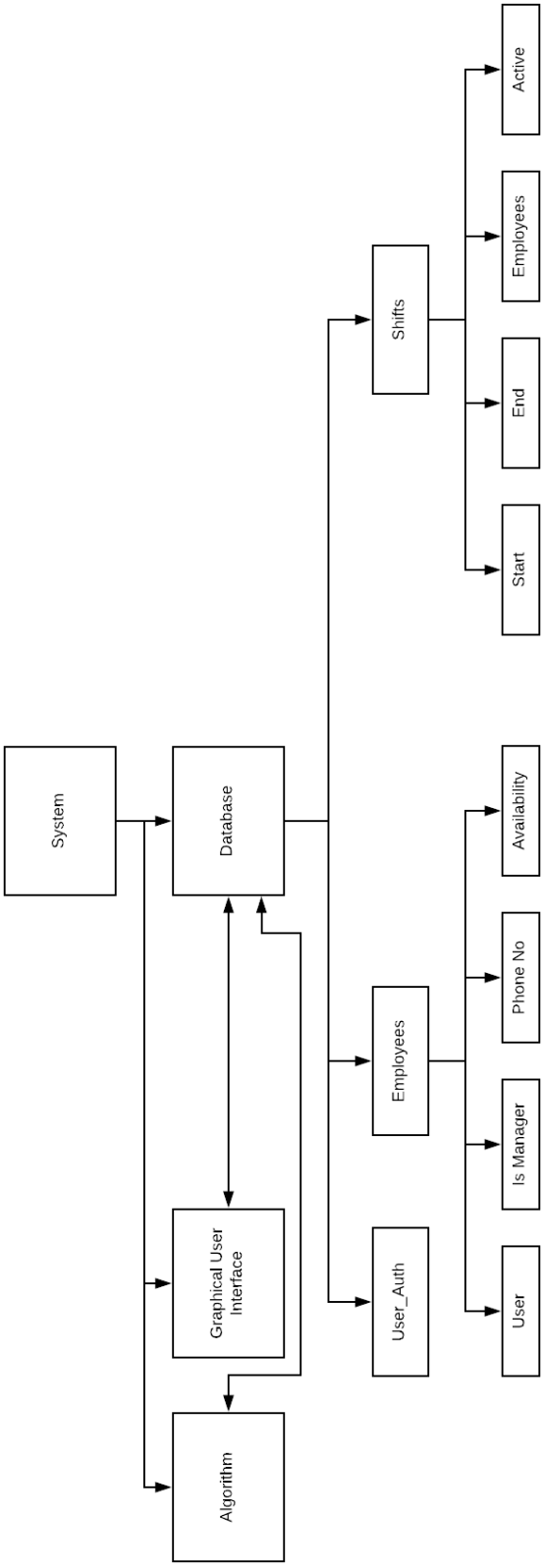
STRUCTURE CHART (ALGORITHM)



STRUCTURE CHART (GUI)



STRUCTURE CHART (DATABASE)



DATA FLOW DIAGRAM (DFD)

The data flow diagram will require many inputs and outputs, therefore a data dictionary will be used to keep the diagram clean. This will only include inputs that transfer data. Therefore other inputs such as the navigation bar will not be included.

DATA NUMBER/ KEY	INPUT/ OUTPUT	NAME OF VARIABLE/ DATA	DATA TYPE	DESCRIPTION
1.0 | LOGIN FORM	INPUT	USERNAME	STRING	The user must input their username into the field so that the data can be processed by Django's authentication.
1.1 | LOGIN FORM	INPUT	PASSWORD	PASSWORD	The user must input their password into the field so that the data can be processed by Django's authentication. This is encrypted.
2.0 | ADD SHIFT FORM	INPUT	START TIME	DATETIME OBJECT	This data is required, then processed to create a shift record in the database with the correct datetime Start and End fields.
2.1 | ADD SHIFT FORM	INPUT	END TIME	DATETIME OBJECT	This data is required, then processed to create a shift record in the database with the correct datetime Start and End fields.
2.2 | ADD SHIFT FORM	INPUT	DAY	INTEGER	This data is required, then processed to create a shift record in the database with the correct datetime Start and End fields.
2.3 | ADD SHIFT FORM	INPUT	MONTH	INTEGER	This data is required, then processed to create a shift record in the database with the correct datetime Start and End fields.

DATA FLOW DIAGRAM (DFD)

DATA NUMBER/ KEY	INPUT/ OUTPUT	NAME OF VARIABLE/ DATA	DATA TYPE	DESCRIPTION
2.4 | ADD SHIFT FORM	INPUT	YEAR	INTEGER	This data is required, then processed to create a shift record in the database with the correct datetime Start and End fields.
2.5 | ADD SHIFT FORM	OUTPUT	ERROR MESSAGES	LIST	A list of error messages should be returned so that the page can display them to the user.
3.0 | DELETE SHIFT FORM	INPUT	SHIFTID	INTEGER	This is a unique key so that the process can recognise which shift needs to be deleted.
3.1 | DELETE SHIFT FORM	OUTPUT	SHIFT START AND END TIME	STRING	This needs to be listed in a dropdown menu so the user can select a particular shift to delete.
4.0 | DELETE ACCOUNT FORM	INPUT	USERID	INTEGER	This will include a unique shift ID which is selected from a dropdown menu by the user, so that the user account corresponding to the ID can be found and deleted.
4.1 | DELETE ACCOUNT FORM	OUTPUT	USERNAME	STRING	This needs to be listed in a dropdown menu so the user can select a particular user to delete.
5.0 | CREATE USER FORM	INPUT	FIRST NAME	STRING	The user needs to be able to enter this detail about the user in order to fully complete the database record.

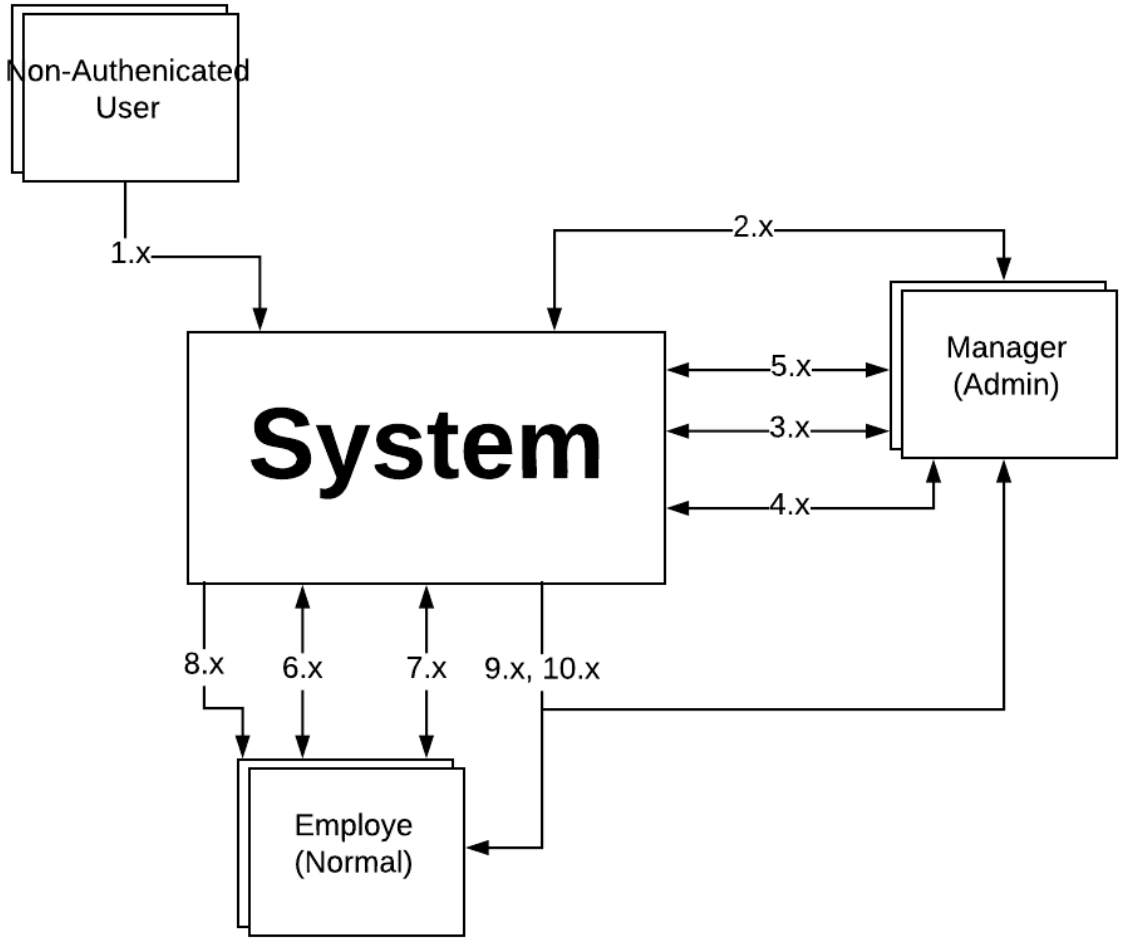
DATA FLOW DIAGRAM (DFD)

DATA NUMBER/ KEY	INPUT/ OUTPUT	NAME OF VARIABLE/ DATA	DATA TYPE	DESCRIPTION
5.1 | CREATE USER FORM	INPUT	LAST NAME	STRING	The user needs to be able to enter this detail about the user in order to fully complete the database record.
5.2 | CREATE USER FORM	INPUT	USERNAME	STRING	The user needs to be able to enter this detail about the user in order to fully complete the database record.
5.3 | CREATE USER FORM	INPUT	PHONE NUMBER	INTEGER	The user needs to be able to enter this detail about the user in order to fully complete the database record.
5.4 | CREATE USER FORM	INPUT	EMAIL	EMAIL	The user needs to be able to enter this detail about the user in order to fully complete the database record.
5.5 | CREATE USER FORM	INPUT	PASSWORD	PASSWORD	The user needs to be able to enter this detail about the user in order to fully complete the database record.
5.6 | CREATE USER FORM	OUTPUT	ERROR MESSAGES	LIST	A list of error messages should be returned so that the page can display them to the user.
6.0 | SET AVAILABILITY FORM	INPUT	SHIFTID	INTEGER	This is a unique key so that the process can recognise which shift needs to be set as available for the user.
6.1 | SET AVAILABILITY FORM	OUTPUT	SHIFT START TIME AND END TIME	STRING	This needs to be listed in a dropdown menu so the user can select a particular shift to set as available for.

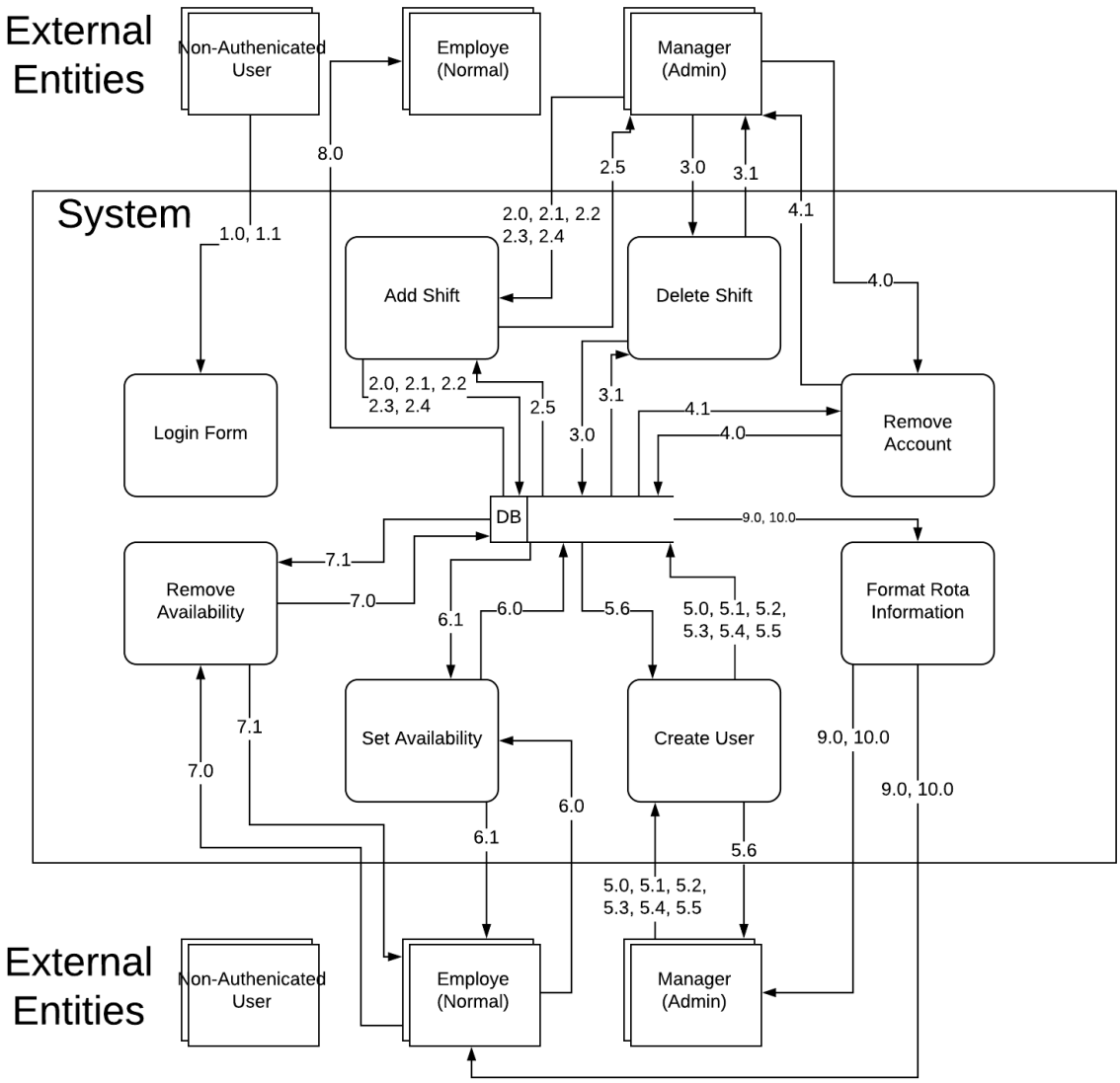
DATA FLOW DIAGRAM (DFD)

DATA NUMBER/ KEY	INPUT/ OUTPUT	NAME OF VARIABLE/ DATA	DATA TYPE	DESCRIPTION
7.0 | REMOVE AVAILABILITY FORM	INPUT	SHIFTID	INTEGER	This is a unique key so that the process can recognise which shift the user needs to be removed as available for.
7.1 | REMOVE AVAILABILITY FORM	OUTPUT	SHIFT START TIME AND END TIME	STRING	This needs to be listed in a dropdown menu so the user can select a particular shift to remove their availability from.
8.0 | USER CURRENT AVAILABILITY LIST	OUTPUT	SHIFTS START TIMES AND END TIMES LIST	LIST	This is a list of strings containing the start and end times of shifts that the current user is available for.
9.0 | CURRENT ROTA DISPLAY	OUTPUT	CURRENT ROTA TABLE	LIST	The data that is required to render the shifts table will most likely be structured into some form of nested list, therefore it will be passed from a separate process and outputted to the user in a human readable format.
10 | PROVISIONAL ROTA DISPLAY	OUTPUT	PROVISIONAL ROTA TABLE	LIST	The data that is required to render the shifts table will most likely be structured into some form of nested list, therefore it will be passed from a separate process and outputted to the user in a human readable format.

LEVEL 0 DATA FLOW DIAGRAM (DFD)



LEVEL 1 DATA FLOW DIAGRAM (DFD)



DEVELOPMENT OF THE SOLUTION WITH TESTING TO INFORM DEVELOPMENT

Aiden Gourley

INTRODUCTION

Just as I approached the design phase of the project, I shall develop the solution in such a way so that I stick to the Database → GUI → Algorithm structure. However, it is also clear, even before we start, that an iterative approach will be required. This means that although most aspects of each section can be completed without reliance on the others, there will undoubtedly be portions of the code that depend on other subsections. An example of this could be when rendering the results of the algorithm into a html table format. This will be very difficult to fully test before a working algorithm is implemented, therefore although the basic code can be developed separately, bugs and errors will have to be dealt with simultaneously, making the debugging a much more difficult process. Nevertheless, since the design outlines much of the development process, there shouldn't be many difficulties, aside from a lot of research regarding the specifics of django and it's syntax.

INITIAL DJANGO SETUP

My IDE (Integrated Development Environment) is PyCharm by JetBrains. It has been especially designed to facilitate the creation of Django web applications, and therefore automatically includes the ability to run, stop and refresh the server, inspect all tables and data within the linked database, and automatically creates all the files that Django requires in order to run.

The most common files that shall be accessed are:

- forms.py
  - Handling forms are highly complex; Django's forms can help the developer to automate common processes and provides numerous security features that developers may otherwise overlook. It can deal with rendering the forms in a HTML format and can process and restructure data so that it is ready for rendering. For a more detailed explanation, see the Django documentation- https://docs.djangoproject.com/en/2.0/topics/forms/
- models.py
  - 'Models' are the name Django gives to tables within a database. Generally, each model is mapped to a single database table, however more complex mappings can be made use of, as we shall see in the database development section. Each attribute within a model refers to a database field. For a more detailed explanation, see the Django documentation- https://docs.djangoproject.com/en/2.0/topics/db/models/
- urls.py
  - The file is used to configure each URL mapping. It maps a URL to a function which is defined in the 'views' file. Therefore, when a URL is entered, the corresponding function shall be called, which will oversee calling other functions that will render templates, process data, or more commonly both. For a more detailed explanation, see the Django documentation- https://docs.djangoproject.com/en/2.0/topics/http/urls/
- views.py
  - A 'View' is, in simple terms, a function that takes a web request (as an input) and returns a web response. Within my application, the responses will mainly include the HTML contents of a webpage. The functions can also be used to process data before it is used in the response. This is very common in dynamic web applications. For a more detailed explanation, see the Django documentation- https://docs.djangoproject.com/en/2.0/topics/http/views/
- settings.py
  - The settings file includes configurations for the server, which are set by defining python variables. An example of a configuration that is stores could be the time zone of the server. For a more detailed explanation, see the Django documentation- https://docs.djangoproject.com/en/2.0/topics/settings/

DATABASE DEVELOPMENT

All the database development will take place within the models.py file, which makes this process the quickest of the three sections.

```
from django.db import models
```

Within the file, the first stage is to import Django's built in database module 'models'. This is what provides Django with the ability to map the models that shall be defined, to the database file.

```
class Employees(models.Model):
    user = models.OneToOneField()
    Availability = models.ManyToManyField()
    PhoneNo = models.IntegerField()
    isManager = models.BooleanField()
```

The first stage includes simply defining each attribute of the model within a new 'Shifts' class. Each attribute requires a defined data type, so therefore I have started with this initially, and then the parameters will be filled in afterwards. The model will need a relationship between the built in 'user' model. The Employees model will be used simply as an extension of the Django authentication built in 'user' model. I researched many ways of extending the user model, however I decided that this is the most quick and simple way of doing it. Other methods included using proxy models or creating a custom user model extending AbstractBaseUser. Availability is used to link the shifts that a particular employee is available for, there is a many to many relationship between these two models because it is required that many employees can set themselves as available for many shifts.

```
from django.contrib.auth.models import User
```

The built in 'user model has to be imported in order to be referenced, and so the imports have to be amended.

```
from django.core.validators import MaxValueValidator, MinValueValidator
```

Through google, I found that Django has many different built in validators such as MaxValue and MinValue, I have made use of them within the Employees model, so that phone numbers lie between the value of 7000000000 and 7999999999 however, it should be noted that this does not take into account of the 0 before the start of each mobile number.

```
class Employees(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE, unique=True, null=True, blank=True)
    Availability = models.ManyToManyField('Shifts', blank=True, default=None)
    PhoneNo = models.IntegerField(validators=[MinValueValidator(7000000000),MaxValueValidator(7999999999)])
    isManager = models.BooleanField(default=False)
```

The whole record will have to be deleted if the linked User is deleted. Therefore, the CASCADE method is required as a parameter to ensure database integrity. The availability field is also allowed to be blank, because an employee may not be available for any shifts, for whatever reason. The isManager field defaults to False so that when a user is created, there is less chance of user error- setting them as a manager incorrectly.

```
class Shifts(models.Model):
    Start = models.DateTimeField()
    End = models.DateTimeField()
    Employees = models.OneToOneField()
    Active = models.BooleanField()
    ShiftNotes = models.TextField()
    CoverRequest = None
```

The same process as I used to develop the other model shall be used here. When defining the link between the Shifts model and the Employees model, I decided that it shall be a 1:1 relationship, because one shift can have only one employee as required by the algorithm. The start and end attributes will be stored as datetimes, the shift notes as a text field, and Active as a Boolean field (as per the design).

```
class Shifts(models.Model):
    Start = models.DateTimeField()
    End = models.DateTimeField()
    Employees = models.OneToOne('Employees', blank=True, default=None, null=True, on_delete=models.SET_DEFAULT)
    Active = models.BooleanField(default=False)
    ShiftNotes = models.TextField(blank=True)
    CoverRequest = None
```

The parameters are set so that the Employees attribute now links to the Employees model, allows it to be left blank, defaults to none, and when an employee instance is deleted the linked shifts will be changed so that they no longer have an employee linked to them. This ensures database integrity, however Django automatically normalises the databases, and so if a linking table is required, it shall be created in the background so that it is invisible to the developer. This helps keep the code clean and easy to read view and read.

I shall also comment out the 'ShiftNotes' and 'CoverRequest' attributes as I am so far unsure how I will utilize them fully, and it is not a priority.

The database requires migrations to take place in order to create these tables that we have defined as models. To do this is a simple task, we simply:

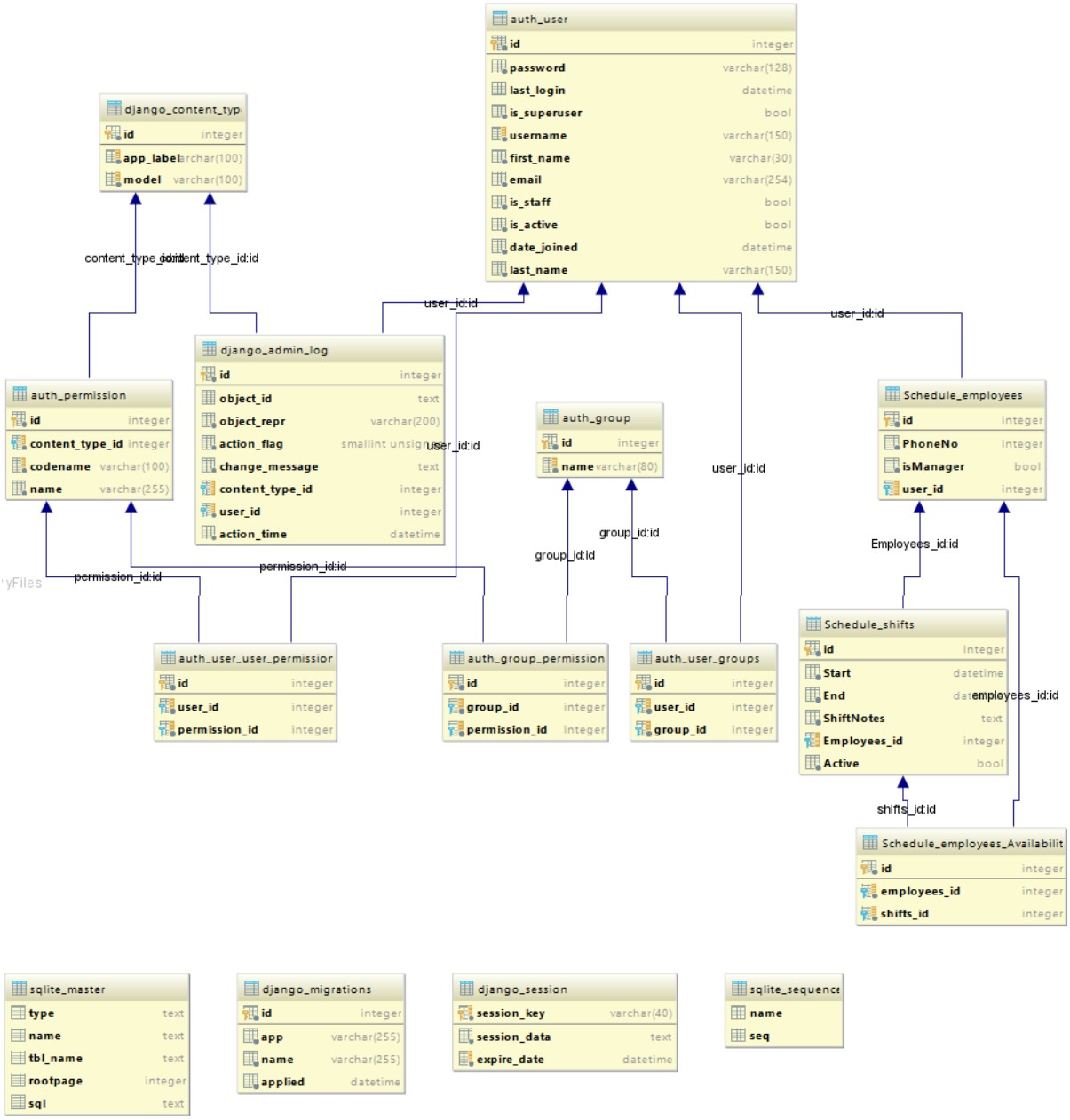
- Open the command line
- Change the directory to the location of the project
- Access the manage.py file by running the command 'python manage.py make migrations' (This creates the migrations which are then ready and waiting to be applied)
- The next command 'python manage.py migrate' applies all of the migrations, changing the database into what we require.

```
C:\Users\Aiden>cd Desktop/WentworthLifeguards

C:\Users\Aiden\Desktop\WentworthLifeguards>python manage.py makemigrations
```

```
C:\Users\Aiden\Desktop\WentworthLifeguards>python manage.py migrate
```

DATABASE DEVELOPMENT (CONT.)

FULL DATABASE VISUALISATION



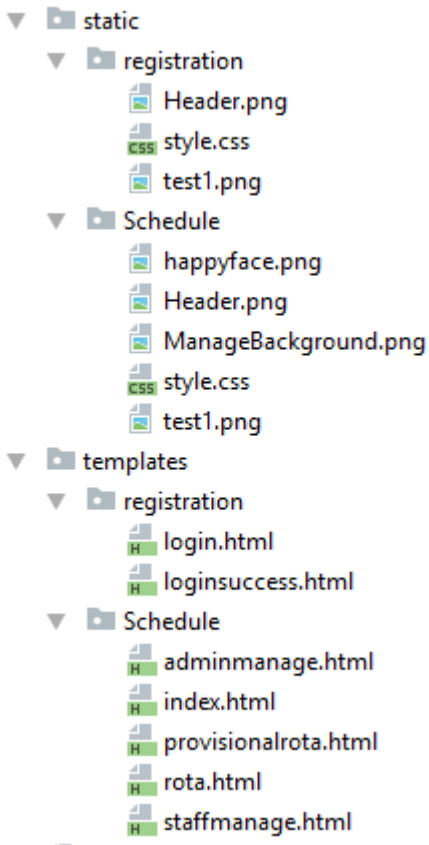
As Django has many other tables within the database that are unseen by the developer, some of this may seem irrelevant, however it displays the linking tables that Django has automatically created in order to ensure the database is normalised. An example is the 'Schedule_employees_Availability'.

GUI DEVELOPMENT (CONT.)

Included in this section of the development will be the logic of rendering data, manipulation and creation of forms and request data, the URLs, and the template creation. Now that the database has been created, it allows us to test each new part of the project (for example each individual forum). It also ensures we know how to syntactically access each model because it's attributes have been defined.

The first stage will be to develop the HTML templates. Each page is referred to as a template, and within, HTML is used to create the static parts such as the navigation bars, backgrounds, and information on the page that does not change. Dynamic data can be inserted through a special syntax called the Django Template Language; however, it is good practice to keep processing in the views and away from the template file. For more information regarding templating, please see the official documentation https://docs.djangoproject.com/en/2.0/topics/templates/ .

These templates are stored in a directory called 'templates' and can be stored in folders within that if it helps organisation. For other elements such as images or CSS files, they can be stored in a separate directory named 'static', this is the case here:



As the design shows, I will require some images in order to make the GUI look as the client and I agreed upon. Therefore, as well as creating the HTML files, I also made/ sourced and edited the images and created blank CSS files. I attempted to split the templates into to different functions within the website, specifically the scheduling and the authentication (or registration).

GUI DEVELOPMENT (CONT.)

For the full code, please see the end of the document- however for the purpose of simplicity, only key elements of the HTML will be discussed. The navigation bar counts as a key element because once developed, it will be used throughout every page within the website.

In simple terms, a website's navigation bar is a reformatted list of links. Therefore, the CSS file which will be linked to each html file includes a lot of manipulation of the list elements. This comprises of background colours, fonts, positioning, padding, text alignment and colour.

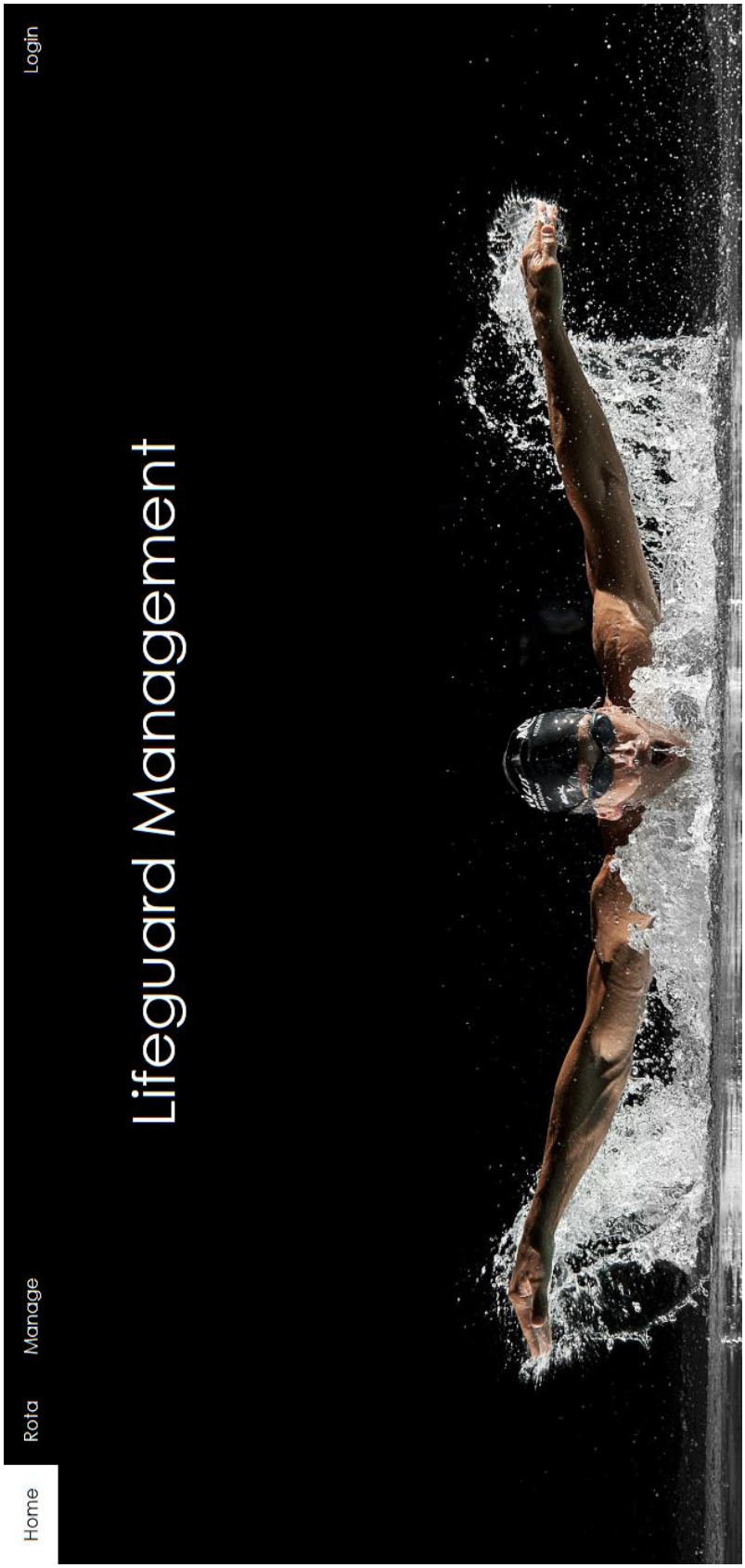
```
<div id="navbar">
    <ul>
        <li><a class="active" href="{% url 'index' %}" style="...">Home</a></li>
        <li><a href="{% url 'rota' %}">Rota</a></li>
        <li><a href="{% url 'manage' %}">Manage</a></li>
        {% if user.is_authenticated %}
            <li style="..."><a href="{% url 'logout' %}">Logout</a></li>
            <li style="...">Welcome, {{ user.first_name }}</li>

        {% else %}
            <li style="..."><a href="{% url 'login' %}">Login</a></li>
        {% endif %}
    </ul>
</div>
```

The Django Template Language (DTL) allows logic such as if else statements to be embedded within the HTML. I have applied this so that the built in user.is_authenticated method can be used to decide whether a use should be shown the login or logout button. It also allows the user to be shown a customised welcoming message at the top of the screen. This was mentioned by the client in the design phase as a feature that he would like as it adds a sense of customisation and could make the user value the software more- making his job of integrating this into the business easier. This is a common technique seen across many web applications across the internet. The links have also been soft coded to an extent. The names reference the names given to the URL pattern which will be later defined in the URLs file. This means that if a specific URL is to be changed, the name can stay as is, with the link only getting changed once within the URLs file. It works a little like external CSS files.

Another very time-consuming element was the animation of the changing text. At first, I had very little idea of how I was going to go about this, however there were many only tutorials that I used to learn how to develop something like this without the use of other web-based languages such as JavaScript.

The final eye-catching element of the home page is the background image. It was originally sourced online. I made changes that included cropping, removing reflections, blacking out the background, changing the image to black and white, but keeping the skin colour. The main reason for these changes was so that it fitted the colour scheme of the website. A black navigation bar and primarily blue picture would look strange. Of Course, no copyright infringement is intended and the rights to the picture remain with the owner. The following pages display the completed HTML page, with screenshot 2 attempting to display the animation effect of help text. The text flies in, pauses in the centre of the page, and exits. Different messages are then displayed, but with the same animation.

SCREENSHOT 1



Home/ Landing Page

SCREENSHOT 2

Dynamic Information and hints flying in, pausing, and flying out across the Home/ Landing page

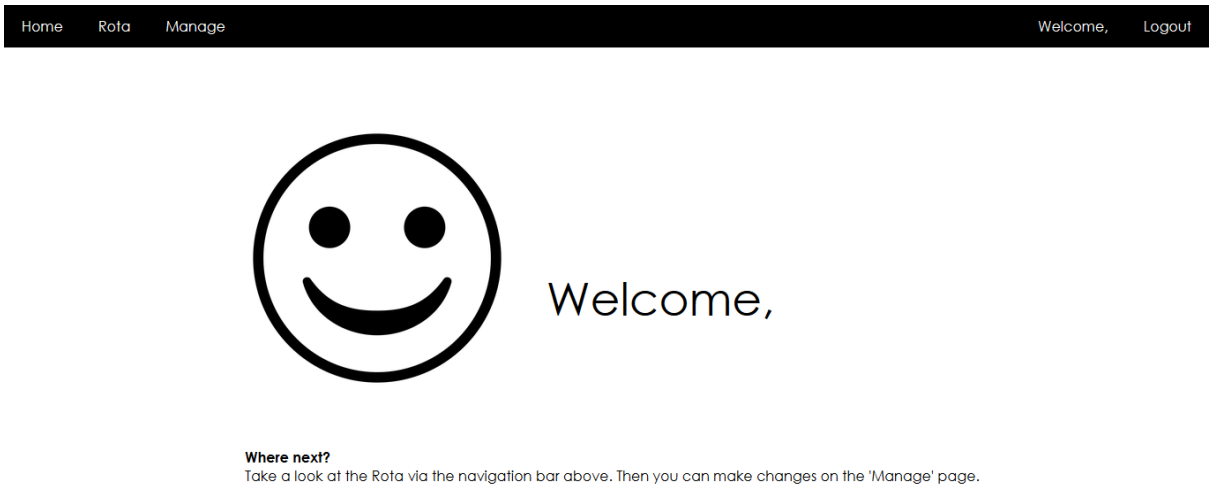
GUI DEVELOPMENT (CONT.)

All other web pages have the same navigation bar, however one aspect that didn't explain before is the 'active' class which is assigned to the link which the current page maps to. For example, if the user is currently on the home page, the 'Home' link will be assigned the 'active' class. Whichever link has the active class, it changes the background of the button to white. This then shows the user which page they are currently looking at.

The titles for each other page have been created, however aside from that the only other page that has any meaningful static content at this stage is the welcome page, which a user will be redirected to after they login. This again offers the user a customised experience, with hints so that the user gets an idea of what to do next.

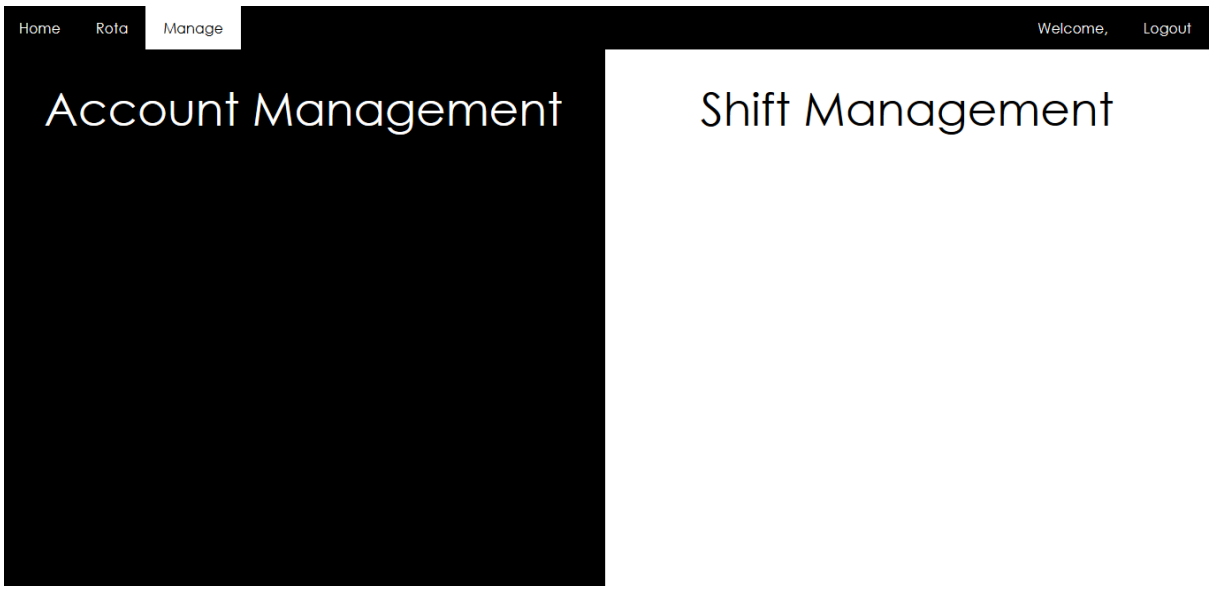
```
<img style="..." src="{% static 'Schedule/happyface.png' %}" height="300" width="300">
<h1 style="...">Welcome, {{ user.first_name }}</h1>

<p style="..."><b>Where next? </b><br>
Take a look at the Rota via the navigation bar above. Then you can make changes on the 'Manage' page.
</p>
```

The same DTL method of 'user.first_name' is used to dynamically get the first name of the user who is currently logged in. This is read from the database each time a user accesses the page. Although some could say that a form of caching could be used to prevent queries being made so frequently, I felt it is not required as the request is very simple and takes next to no time at all, with a simple request putting little stress on the host server even if all users were to be logged in at the same time. The result of this page is the following:



GUI DEVELOPMENT (CONT.)

The final HTML related part is to split the admin manage page into two sections as per the design stage. This was easily done through use of the tags and background colours, giving the following result:



Taking a step away from template creation, the next thing to do is build the URLs file. There are two files- one is website wide, in which you define the 'Level 0' URL patterns. The second is within the Schedule app and can be considered the 'Level 1' URL patterns.

The first URL file will state the following logic:

- If no subdomain is requested after the domain name OR the subdomain includes '/schedule', it will be assumed that they are accessing the 'Schedule' app, therefore the second URLs file will be called.
- If the subdomain includes '/admin', the URL will be handled by Django's built in admin interface.
- If the subdomain includes '/accounts', the URL will be handled by Django's built in authentication.

```
from django.conf.urls import url, include
from django.contrib import admin

urlpatterns = [
    url(r'^', include('Schedule.urls')),
    url(r'^admin/', admin.site.urls),
    url(r'^schedule/', include('Schedule.urls')),
    url('accounts/', include('django.contrib.auth.urls')),
    ]
```

GUI DEVELOPMENT (CONT.)

The second URL file will state the following logic assuming '/schedule/':

- If the subdomain includes nothing it will be mapped to the views function 'index' and the name 'index'
- If the subdomain includes 'rotas' it will be mapped to the views function 'rota' and the name 'rota'
- If the subdomain includes 'provisionalrota' it will be mapped to the views function 'provisionalrota' and the name 'provisionalrota'
- If the subdomain includes 'loginsuccess' it will be mapped to the views function 'loginsuccess' and the name 'loginsuccess'
- If the subdomain includes 'manage' it will be mapped to the views function 'manage' and the name 'manage'

```
from django.conf.urls import url
from . import views

urlpatterns = [
    #/Schedule/ <!-- The Landing Page
    url(r'^$', views.index, name='index'),
    #/Schedule/rota/ <!-- The Rota Display Page
    url(r'^rotas', views.rota, name='rota'),
    #Schedule/staffmanage or adminmanage/ <!-- The Shift Management Page
    url(r'^manage', views.manage, name = 'manage'),
    #/Schedule/loginsuccess/ <!-- The Login Success Notice Page
    url(r'^loginsuccess', views.loginsuccess, name='loginsuccess'),
    #/Schedule/provisionalrota <!-- The Provisional Rota Display Page
    url(r'^provisionalrota', views.provisionalrota, name = 'provisionalrota'),
]
```

For a more detailed explanation of URL configurations and mapping in Django, please see https://docs.djangoproject.com/en/2.0/topics/http/urls/

The next phase is to access the views file and define the functions that the URLs map to. They will only have the basic features that a views function requires in order to work:

```
def index(request):
    context = None
    return render(request, './Schedule/index.html', context)
```

The request parameter will be present in every view that is defined, it includes all the data within the request, and in the future will include the likes of form data and information about the currently logged in user. The context variable includes all the data that can be accessed by the template language, however it must be either None or a dictionary. Finally, the render method enables us to render the initial page with the request data and context. All views in this project will have a very similar structure to the one above. For more information see https://docs.djangoproject.com/en/2.0/topics/http/shortcuts/

GUI DEVELOPMENT (CONT.)

GUI NAVIGATION BAR AND URL DEVELOPMENT TESTING

This in theory means that the navigation bar that we created above will now fully work, because the names we used within the links have been defined in the URLs file and mapped to pages.

There are 7 inputs which require testing, these are as follows:

0.1 Manual change of the subdomain. The inputs are preceded by the server address which is currently: http://127.0.0.1:8000/
0.2 Home Page: NavBar Click
0.3 Provisional Rota Page: NavBar Click
0.4 Rota Page: NavBar Click
0.5 Staff Manage Page: NavBar Click
0.6 Admin Manage Page: NavBar Click
0.7 Login Page: NavBar Click

The inputs will correspond to the dictionary above i.e. TestNumber.InputNumber

Test No.	Type	Description	Input	Expected Output	Actual Output	Comment
1	Normal	Each input type has to pass. If one fails, specific details will be added in the comment section.	/Schedule/ or 'Home' button 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7	1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7 Displays Home page	1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7 Displays Home page	This mapping works as expected. All NavBar links work as expected.

GUI DEVELOPMENT (CONT.)

GUI NAVIGATION BAR AND URL DEVELOPMENT TESTING

Test No.	Type	Description	Input	Expected Output	Actual Output	Comment
2	Normal	Each input type has to pass. If one fails, specific details will be added in the comment section.	/rotas/ or 'Rota' button 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7	2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7 Displays Rota page	2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7 Displays Rota page	This mapping works as expected. All NavBar links work as expected.
3	Normal	Each input type has to pass. If one fails, specific details will be added in the comment section.	/provisionalrota/ 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7	3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7 Displays Provisional Rota page	3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7 Displays Provisional Rota page	This mapping works as expected.
4	Normal	Each input type has to pass. If one fails, specific details will be added in the comment section.	No subdomain input 4.1	4.1 Displays Home Rota page	4.1 Displays Home Rota page	This mapping works as expected.
5	Normal	Each input type has to pass. If one fails, specific details will be added in the comment section.	/manage/ or 'Manage' button 5.1, 5.2, 5.3, 5.4, 5.5, 5.6, 5.7	5.1, 5.2, 5.3, 5.4, 5.5, 5.6, 5.7 Displays Admin Manage Rota page	5.1, 5.2, 5.3, 5.4, 5.5, 5.6, 5.7 Displays Admin Manage page	This mapping works as expected. (Until the user authentication has been setup, the links will redirect to the admin manage page.)

GUI DEVELOPMENT (CONT.)

GUI NAVIGATION BAR AND URL DEVELOPMENT TESTING

Test No.	Type	Description	Input	Expected Output	Actual Output	Comment
6	Normal	Each input type has to pass. If one fails, specific details will be added in the comment section.	/login/ or 'login' button	6.1, 6.2, 6.3, 6.4, 6.5, 6.6, 6.7 Displays Login page	6.1, 6.2, 6.3, 6.4, 6.5, 6.6, 6.7 Displays Login page	This mapping works as expected. All NavBar links work as expected.
7	Normal	Each input type has to pass. If one fails, specific details will be added in the comment section.	/loginsuccess/	7.1, 7.2, 7.3, 7.4, 7.5, 7.6, 7.7 Displays Login page	7.1, 7.2, 7.3, 7.4, 7.5, 7.6, 7.7 Displays Login page	This mapping works as expected. All NavBar links work as expected.

```
LOGIN_REDIRECT_URL = 'loginsuccess'
LOGOUT_REDIRECT_URL = 'index'
```

Now that the basic static HTML templates have been created, it is time to add some functionality to the web app. Since one of the core functions is the login system, this is what shall be developed next. The first step is to open the settings.py file and add a couple of new variables which define the login and logout redirect URL. These can use the name of the url as references instead of the full subdomain because they were previously defined in the URLs file. This is required by Django's login system if the developer wants a redirection after login- although this isn't required. However, in this case it is required because it is a page that the client agreed to having in the design, and which has already been developed.

```
INSTALLED_APPS = [
    'Schedule.apps.ScheduleConfig',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

The next step is to install the user authentication which is done by adding the following line to the installed apps variable.

Within the web applications URLs file, the following line is required for the user authentication, however we have previously already added this in anticipation:

```
urlpatterns = [
    url(r'^', include('Schedule.urls')),
    url(r'^admin/', admin.site.urls),
    url(r'^schedule/', include('Schedule.urls')),
    url('accounts/', include('django.contrib.auth.urls')),
    ]
```

Most of the authentication middleware and installed apps that are required for an authentication system are created by default when a new Django project is setup. This is because it is such a common task, so Django decided to remove even more work for the developer by including it within the default Django setup. For more information regarding the user authentication, please see the official documentation https://docs.djangoproject.com/en/2.0/topics/auth/

To ensure that no extra changes have been made to the database, I will run the migration process again, the same as occurred after the models were created. Once this has been completed, the database should remain unchanged as to prevent risk of integrity errors or corruption- this is especially true when test data is saved within.

GUI DEVELOPMENT (CONT.)

Next on the agenda is to develop the multiple forms that are required for the users to fully interact with the database. I will be making use of ModelForms and the custom Forms. Model Forms are those which are based off a model's attributes. The input type will be determined by the Django default for the data type, which is defined when the model is created. However, with custom forms, this must all be defined manually.

The following includes a list of forms that need to be included within the website:

- Login
- Set Availability
- Remove Availability
- Remove Account
- Add Account
- Add Shift
- Remove Shift

The login form will be handled by Django's built in authentication system, as we have discussed and implemented previously. The process of creating a form is very similar. The form itself needs to be defined in the forms.py file, which is straightforward once the syntax is known. The next stage is to render the form in the corresponding template, making use of the DTL to help with this. The following stage is to process the posted data, and if required, return an appropriate response.

To begin with, the Set Availability form will be a drop-down menu, which is commonly referred to as a 'Multiple Choice Field' within the Django community. A field like this requires a tuple of choices, however for the choices to be a user friendly as possible, there is a requirement to process the data into something that can be easily understood by the user. Therefore, a separate function needs to be defined to create a tuple that contains a programmable choice with a human readable selection.

```
def GetShifts(SetActive):
    ChoicesList = []
    for i in Shifts.objects.all().filter(Active=SetActive):
        String = str(i.Start.strftime('%d/%m/%Y %H:%M') + ' - ' + i.End.strftime('%d/%m/%Y %H:%M'))
        ShiftID = i.pk
        ChoicesList.append((ShiftID, String))
    Choices = tuple(ChoicesList)
    return Choices
```

Each table and record in the database can be accessed by importing the model and calling it as though it were a class with methods and attributes. Django has many built in methods such as 'filter' which can query the dataset according to parameters. As referenced above, the Django model documentation contains all of these methods. The code above simply filters all the shifts in the database according to whether they are active or not. It then creates a tuple for each shift in the query set, with the first element as the shift's primary key, and the second as a string of the start time to the end time (the part which will be presented to the user within the selection). The shift's primary key is used because it can be usefully used in a different part of the program when it comes to processing the user's selection.

This is then returned so that it can be called within the form class which will be defined next.

```
class SetAvailability(forms.Form):
    def __init__(self, *args, **kwargs):
        super(SetAvailability, self).__init__(*args, **kwargs)
        Choices = GetShifts(SetActive=False)
        self.fields['ShiftID'] = forms.MultipleChoiceField(label='', choices=Choices)
```

Simply, a new initialise method is being defined, and within it the GetShifts function is called to return a list of shifts as described above, that are currently stored in the database. This is then displayed as a dropdown menu as the variable named ShiftID. The posted value can then be manipulated in the views file, by referring to the variable 'ShiftID'.

Now that the form has been defined, it is time to implement it within the template.

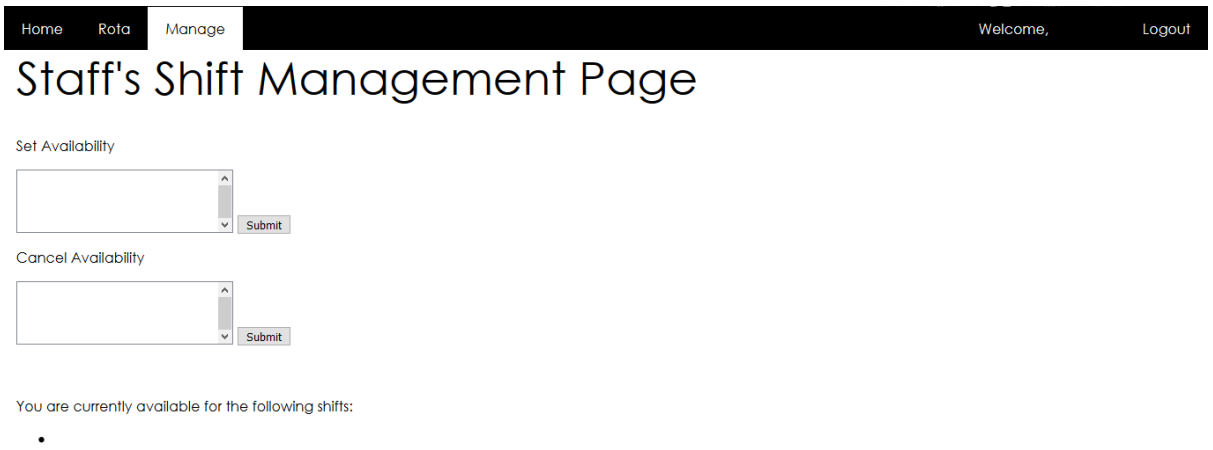
```
<p>Set Availability
    <form action="/manage" method="post">
        {% csrf_token %}
        {{ SetAvailability }}
        <input type="submit" value="Submit" name = "Set Availability">
    </form>
</p>
```

All that had to be done is to utilize the HTML <form> tag, with a post method and an action which will decide where the forum redirects to once the form has been submitted (although this should in most cases be handled by the view's render method return. Each form requires a Cross Site Request Forgery (CSRF) token. This is a security measure which prevents a type of man-in-the-middle attack, whereby a third party attempts to manipulate form data between a client and the host. It ensures the security because only the client and host know the CSRF key, meaning that any other entity attempting to mimic the client, cannot. The form is defined within the 'context' in the views, and therefore can be called as so through the DTL. Then all that is required is the submit button, which should be created manually like so.

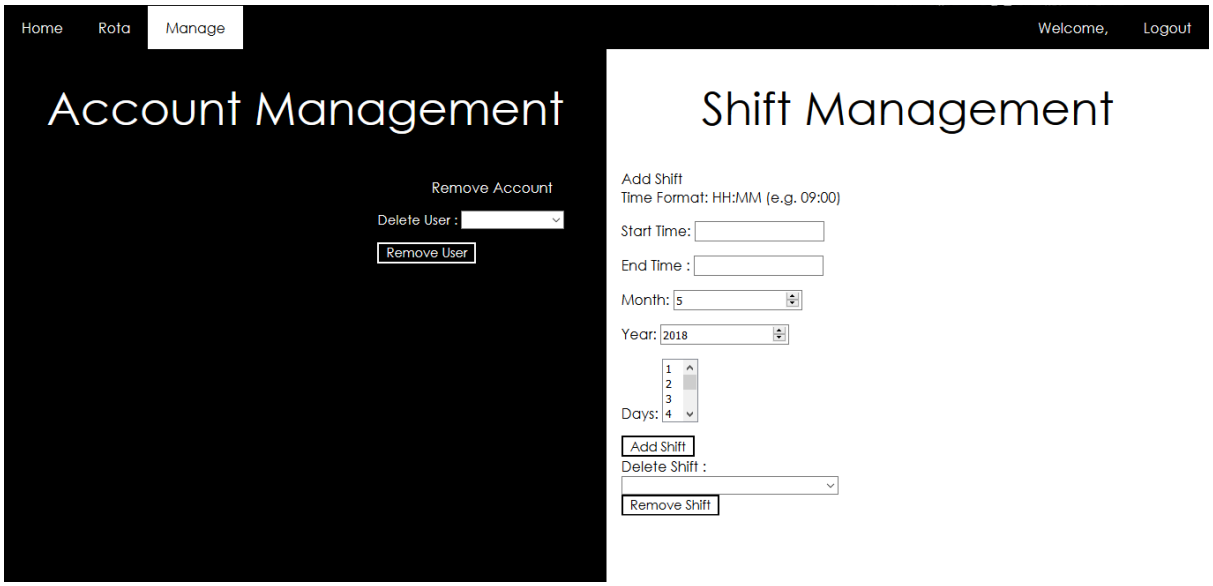
```
if 'Set Availability' in request.POST:
    print('Set Availability Form')
    SetAvailabilityInstance = SetAvailability(request.POST)
    if SetAvailabilityInstance.is_valid():
        ShiftID = SetAvailabilityInstance['ShiftID'].data[0]
        print(ShiftID)
        UserObject = User.objects.get(username = request.user.username)
        FilteredShifts = Shifts.objects.all().filter(Active=False)
        ShiftObject = FilteredShifts.get(pk=ShiftID)
        EmployeeObject = Employees.objects.get(user=UserObject)
        EmployeeObject.Availability.add(ShiftObject)
    page = './Schedule/adminmanage.html'
    context = {'SetAvailability':SetAvailability}
    return render(request, page, context)
```

The views file should contain a 'manage' function as previously defined. Within this, there should be a check for whether the request is POST. If so, there should be following conditional statements testing to check which form has been submitted. (This is currently a slightly simplified method that works now but will require a slightly more complex modification once the user authentication has been fully implemented. This added complexity will only be the case for the 'manage' view, because one link will lead to two different pages depending on user privilege levels.) This can be done by simply checking if the name of the form is contained within the request.POST data. Once the appropriate form is found, the data can be tested for validity using Django's inbuilt validation method is_valid(). This is covered in enough detail within the Database Design section, under Validation and Security. The following logic is simply a set of database queries and an input. In simple terms, the objective is to get the shift object that the user wants, get the user's employee object, and set the employee's availability to the shift object. Once this is complete, the page and context are then set, and the page is re-rendered.

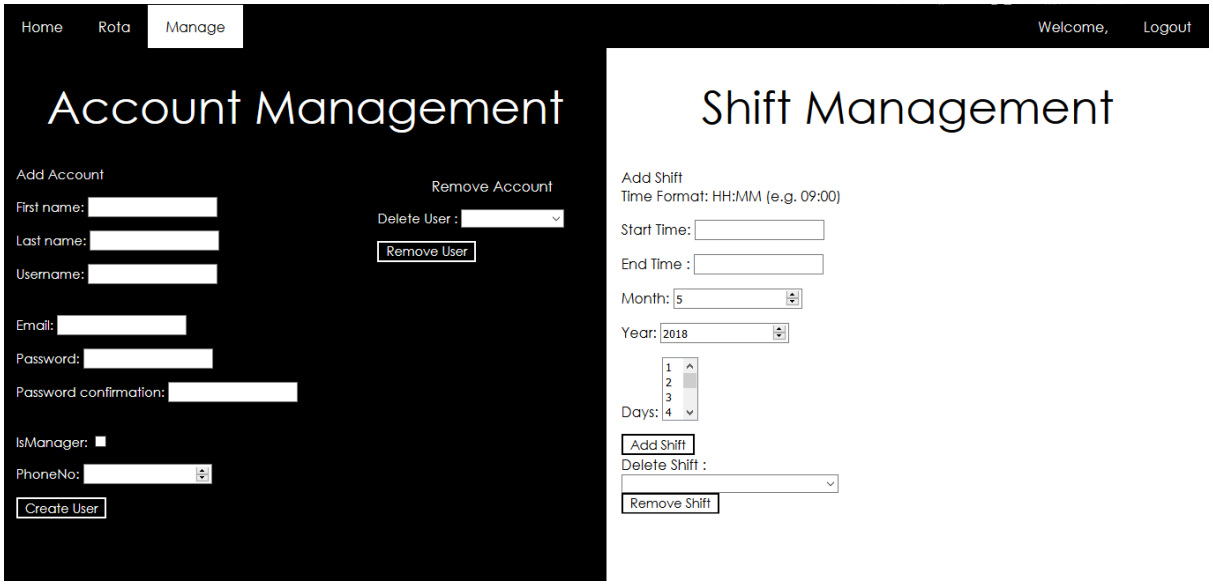
This process is very repetitive, and the only slight differences between each form are the types of input fields, and the logic behind the database queries in the views. There are at least 5 other forms that follow this pattern, they were listed above. For the full code of the other forms, views and templates, please see the end of this document. However, although each form will be created individually through a time consuming and repetitive manner, the luxury of documentation is that I can instantly provide evidence of the forums that render properly and in theory should handle the data as per the design. Please see the screenshots below showing a capture of the development after each form is created.



This screenshot shows the set and cancel availability forms on the Staff Manage page. It also includes a section that lists all the shifts that the logged in user is set as available for.



Here, the admin manage page has been screenshotted, with three separate forms being displayed on the one page so far. It gives the manager the option to add a shift into the database, delete a shift or a user from the database.

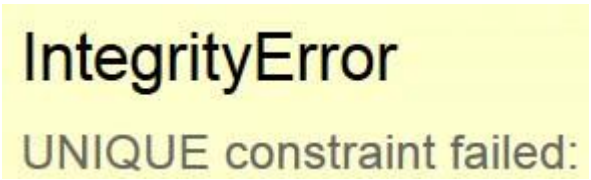


This screenshot shows the admin management page with all the forms completed, now including the option to create a new user account.

This final form is the only one that caused problems. It is quite uncommon to submit two forms at once, through one button. However due to the relationship between the User model and Employee model, ideally, they need to be created at the same point in time. There were a few options to consider:

1. Use 3 separate forms. Form 1: create the user model, form 2: create the employee model, form 3: link the user model to the employee model manually.
2. Use two separate forms and have a drop-down menu within the employee model form to select the user that the employee model will link to.
3. Render both forms within the same divider, and use one submit button for the both. Then in the views, the data is validated separately, but managed in such a way that it can interact.

Option number one and two are not very optimal, considering my client made it clear that many of the users will be computer illiterate. This means that they could forget to link Employee profiles with User profiles, and the database will be left with redundant data. It also means that other sections of the program that depend on the database being fully complete, would have to add validation to check that both elements of the profiles are linked to prevent errors. Within the HTML template itself, the forums are within the same <form> tag. The <submit> tag has a name which can be used to uniquely identify both the forms in the request data, so that they can be validated and the process of creating a user and employee record can take place. Although at first, I did not consider the fact that the user model would have to be created first.



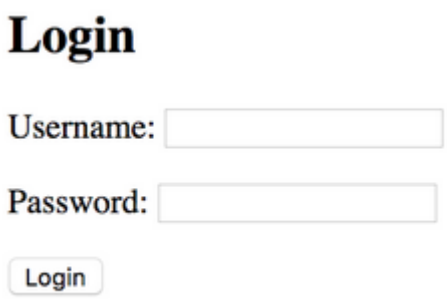
There was a referential integrity error attempting to save the employee model instance first, because the 'user' field is required, however the auth_user model want saved yet. This was easily fixed by saving the user model instance before the employee model instance, although it took a little while for me to realise the reason for the error. The fixed code now looks like the following after the logic error:

```
if "Create User" in request.POST:
    UserReg1 = UserRegistrationPart1(request.POST)
    UserReg2 = UserRegistrationPart2(request.POST)
    print('UserReg Forms 1')
    if UserReg1.is_valid() and UserReg2.is_valid():
        print('UserReg Forms 2')
        Username = UserReg1.cleaned_data['username']
        UserReg1.save()
        EmployeePhoneNumber = UserReg2.data['PhoneNo']
        UserReg2.save()
        EmployeeObject = Employees.objects.get(PhoneNo=EmployeePhoneNumber)
        print(Username)
        print(User.objects.get(username=Username))
        EmployeeObject.user = User.objects.get(username=Username)
        print(EmployeeObject.user)
        EmployeeObject.save()
```

GUI DEVELOPMENT (CONT.)

The login form is created with no difficulty at all; however it does not display exactly as required.

```
<h1>Login</h1>

<form method="post">
  {% csrf_token %}
  {{ form.as_p }}
  <button type="submit">Login</button>
</form>

</body>
</html>
```



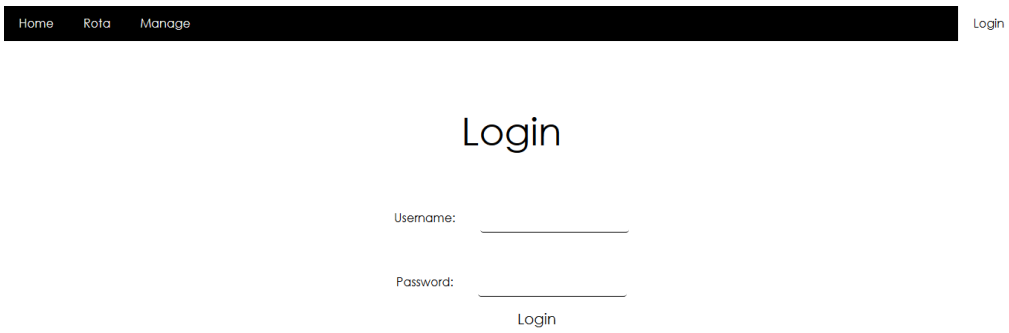
Instead of this bland look, a lot of manipulation of the CSS is required, and a fair bit of time went into researching how the default form was structured so that I could manipulate the html tags.

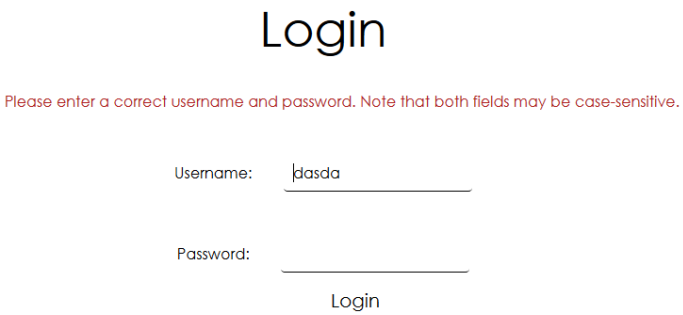
```
<style>
    form {
        text-align: center;

    }

    .errorlist.nonfield {
        padding-top: 0px;
        margin-top: 0px;
        list-style-type: none;
    }
    button {
        position: absolute;
        left: 50%;
        border-radius: 10px;
        border:#000000;
        background-color: #FFFFFF;
        box-sizing: border-box;
        color: #000000;
        font-family: "Century Gothic";
        font-size: 20px;
    }
```

```
input[type="password"], input[type="text"] {
    border: 1px solid #000000;
    border-radius: 4px;
    box-sizing: border-box;
    border-top: hidden;
    border-left: hidden;
    border-right: hidden;
    color: #000000;
    height: 39px;
    margin: 31px 0 0 29px;
    padding-left: 10px;
    width: 200px;
    font-family: "Century Gothic";
    font-size: 16px;
}
h1 {
    margin-top:10%;
    text-align: center;
}
```

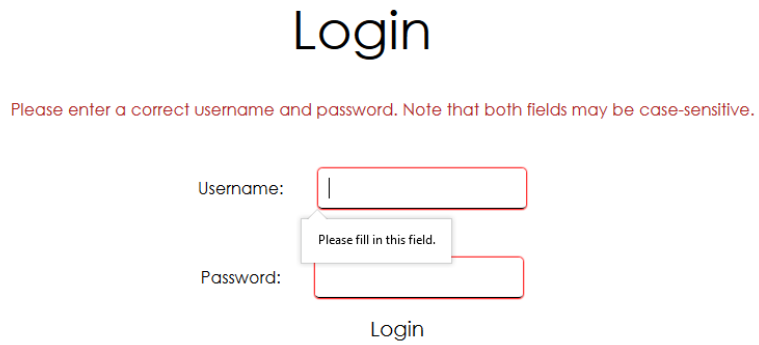
GUI DEVELOPMENT (CONT.)

The following CSS manipulation results in error messages being displayed as per the design, and the overall look of the login form as close to the design specification as possible. The result is the following, although simple:





If an incorrect or invalid password or username combination is entered, the user is provided with an error message. However, if the fields are left blank, they are highlighted by a red box, and promoted with a message inside a speech bubble as displayed by the following screenshot:

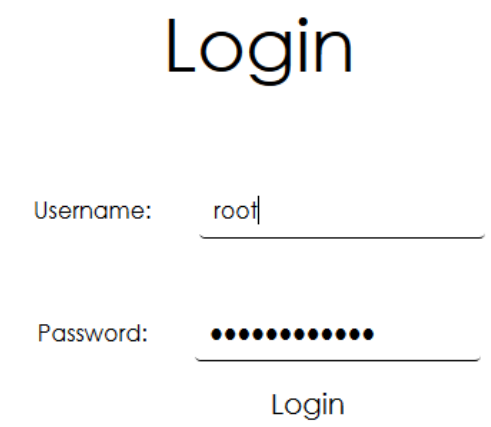


GUI DEVELOPMENT (CONT.)

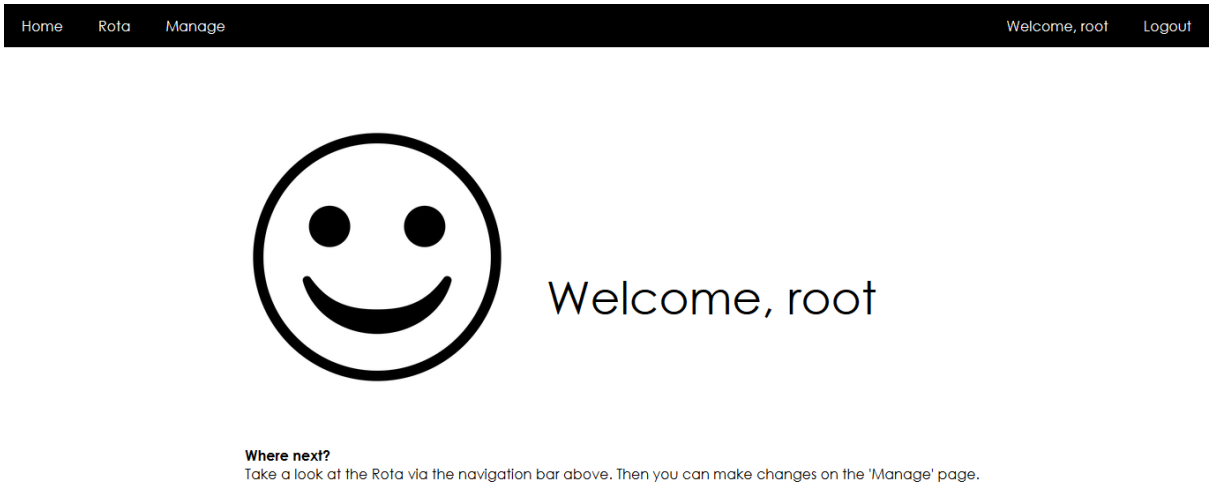
GUI DEVELOPMENT USER AUTHENTICATION TESTING

Since the login system makes use of Django's built in authentication, there is little requirement to heavily test this part of the web app. This is because the function is completely open-source, used by thousands of developers world-wide, and extensively tested by Django itself. Consequently, the tests included shall only be simple functionality tests that ensure a user is able to login and out with ease.

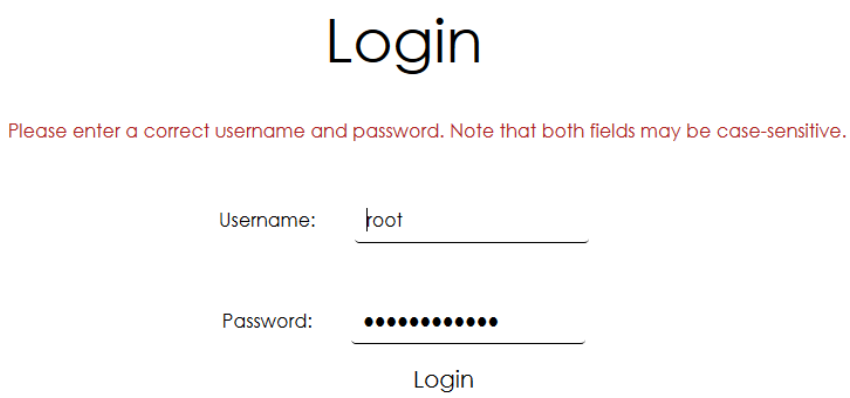
By using command line phrases to interact with the Django website, we can create a new user with superuser privileges. This means that it can be used for testing purposes, interact with the built in Django administrator interface, and can be used in this case to test the login and logout functionality of the form just implemented. This user will be used frequently with a username of 'root'. This shall also be available to the client once the program is distributed so that the maintainability of the software is ensured by the company's IT team.



← Correct details were entered, resulting in a redirection:



GUI DEVELOPMENT USER AUTHENTICATION TESTING



The expected error was raised, and the user was altered when and incorrect password was entered. This basic erroneous data test proves that we have basic functionality, with the previous test proving that the user is authenticated when they enter the correct details. It also keeps the data that was previously entered, which makes the feature much more user friendly. This is a form of usability test as it shows ease of use has been achieved because the user can simply edit their username (if it was entered incorrectly) and retry the login rather than retyping both the password and username.

GUI DEVELOPMENT (CONT.)

Due to the user authentication system having been implemented fully, the way that the views are currently structured will not be enough to implement the different user permission levels.

Django offers a neat little one-line snippet of code which refuses access to a specific page if the user is not logged in. With this implemented, if an unauthenticated user attempts to access a disallowed page (or view), then they are automatically redirected to the login page. The line is as follows, and should be placed above each view that access should be limited to as so:

```
def index(request):...


@login_required(login_url='login')
def manage(request):...


@login_required(login_url='login')
def loginsuccess(request):...


@login_required(login_url='login')
def rota(request):...


@login_required(login_url='login')
def provisionalrota(request):...
```

As per the success criteria, access is limited to only the homepage, or 'index' as the view function is defined. However, there are many 'manage' features that are individual to either employee level accounts, or manager accounts. The simplistic way of going about this would be to have two different pages with corresponding links on the navigation bar and limit which parties can access each page. However, for usability reasons, it could make the lower level account permission holders feel untrusted, which damages staff morale. Consequently, it would be much better for there to be one link to two different pages which changes depending on the account level. This method requires a slightly more complicated development process because one view will be used to manage two pages.

Previously mentioned was that the typical view should return the render function which takes in three parameters 'page', 'context', and the 'request'. To return a different page, the request can always stay the same as it is unique to each authenticated user anyway, the page link will have to differ based on a condition, and the context will also depend on the same condition. This check will be commonly repeated, and so it is obvious that a new function will have to be defined to decide which page and context to deliver to the user.

```
def ReturnPageAndContext(requestUser):
    EmployeeData = Employees.objects.get(user=requestUser)

    if EmployeeData.isManager:
        context = {'UserRegPt1': UserRegistrationPart1, 'UserRegPt2': UserRegistrationPart2,
                   'UserRemove': RemoveUser(requestUser), 'ShiftAdd':AddShifts(), 'ShiftRemove':RemoveShift,
                   'LockRota':''}
        page = './Schedule/adminmanage.html'
        return page, context
    else:
        context = {'SetAvailability':SetAvailability,
                   'ShiftsAvailable':ShiftsAvailableList, 'CancelAvailability':CancelAvailability(user=requestUser)}
        page = './Schedule/staffmanage.html'
        return page, context
```

The context, as mentioned previously is simply a dictionary of items that are required to render the template, in this case they mainly consist of forms. The page is simply the subdomain of the page that should be displayed. These two go together and will both change depending on whether the Employee.isManager condition is True. Therefore, if the user is a manager, the context for the adminmanage page and the adminmanage page URL will be returned and rendered for the user. However, if the user is not a manager, the context for the staffmanage page and the staffmanage page URL will be returned and rendered instead.

If the request method is not POST, then this reusable process will take place straight away

```
page, context = ReturnPageAndContext(request.user)
return render(request, page, context)
```

However, within the view, the if the conditional statement `if request.method == 'POST'` is true, then the following form management process takes place:

The first key part of this process is that each form's HTML submit button in the template is given a unique name e.g. `'Remove Shift'`. Now because the request data will include the name of the form that is submitted, we can distinguish which form the user has submitted, through conditional statements testing each expected form name. An example is the following:

```
    elif 'Remove Shift' in request.POST:
```

If this is true, the other forms can be forgotten about, and this form submission can be handled.

```
elif 'Remove Shift' in request.POST:
    print('Remove Shift Form')
    RemoveShiftInstance = RemoveShift(request.POST)
    FilteredShifts = Shifts.objects.all()
    if RemoveShiftInstance.is_valid():
        print('Remove Shift Is Valid')
        ShiftID = RemoveShiftInstance.cleaned_data['ShiftID']
        ShiftObject = FilteredShifts.get(pk=ShiftID)
        ShiftObject.delete()
        page, context = ReturnPageAndContext(request.user)
        return render(request, page, context)
    else:
        print('Remove Shift Not Valid')
```

The form that is submitted within the request data is then treated as an instance, validity tested and cleaned, then the data submitted can be used to manipulate the database with e.g. saving, querying, deleting etc. In this case the correct record is queried using the data retrieved from the form, and then the record is deleted.

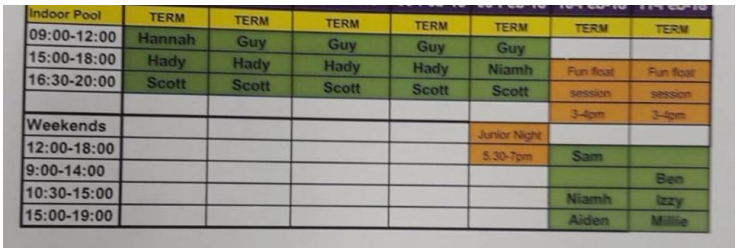
The function that decides which page and context to deliver to the user (which we defined above) is then called, which redirects the user to the page they came from.

This simple methodology is what is used throughout the manage view to handle each form submission.

GUI DEVELOPMENT (CONT.)

The next stage of development is where it will become more difficult because there will be no data to run through the system that is going to render the rota, meaning that if there are any syntax or logic errors like were previously found and corrected earlier (the integrity logic error for example), they will most likely go unnoticed until the algorithm has been developed. Despite this, some planning needs to take place in order to decide the most effective way to store the information that needs to be unpacked and rendered through simple for loops within the Django Template Language.

The aim will be to make looks close to the current rota that my client currently uses. This is a sample:



Indoor Pool	TERM	TERM	TERM	TERM	TERM	TERM	TERM
09:00-12:00	Hannah	Guy	Guy	Guy	Guy		
15:00-18:00	Hady	Hady	Hady	Hady	Niamh	Fun float	Fun float
16:30-20:00	Scott	Scott	Scott	Scott	Scott	session	session
						3-4pm	3-4pm
Weekends					Junior Night		
12:00-18:00					5.30-7pm	Sam	
9:00-14:00							Ben
10:30-15:00						Niamh	Izzy
15:00-19:00						Aiden	Millie

The first thing to consider is the information that needs to be displayed:

- The table headers should include:
  - Days/ Dates
  - 'Shift Times' label above the first column
- After the table headers, there is a common pattern to each row. The first column holds the shift time. The other 7 columns hold names depending on who is down for the shift.
- Each table will need to be made for each week in the month, although of course the table does not require to be a fixed length of 7 (if the month ends)

For simplicity, I will not separate the weekend shift times like the picture above, rather all shift times will be in the column on the left without a divider regardless of whether they are being used on the weekday or not.

In terms of visualising this in pseudocode, it could look something like the following:

Layer 1:

`Month Shift List` ← [Week1ShiftList, Week2ShiftList, Week3ShiftList, Week4ShiftList, Week5ShiftList]

`Week1ShiftList` ← [RowList1, RowList2, RowList3, `RowList4, RowList5, RowList6, RowList7]`

`RowList1` ← [ColumnDetail1, ColumnDetail2, ColumnDetail3, ColumnDetail4, ColumnDetail5, ColumnDetail6, ColumnDetail7]

- Column 1 will always be a shift time string ← Must be human readable.
- Each RowList will consist of several columns, depending on the number of days left in the month. However, the maximum will be 7, to represent 7 days.
- Each WeekList will consist of (the number of different shift times in this case it's 7) rows.
- Each MonthList will consist of a maximum of 5 WeekLists.

Therefore, I require a piece of code that can read all this information and turn it into a data structure like the one described above. Although the structure is quite complicated, once understood, it will make the rendering process within the HTML template itself a much simpler with less reliance on the DTL as it will only require simple iterations.

Since this block of code will likely be very large and require decomposing into multiple functions, I will create a new file rather than putting it all within views (although this would work). This decision was made to ensure the code's maintainability as it is then more easily read than if it weren't to be imported as a module. It is also worth saying that the same sort of code will be reusable when generating the provisional rota, with the only change being that the Shifts that are imported will have to be filtered so that only those with Active= False are used.

```
def getTableHeader(DayNo):
    WeekTH = ['Shift Times']
    OrderedShiftsByStartDate = Shifts.order_by('Start')

    for i in range(DayNo, DayNo+7):
        FilteredShifts = OrderedShiftsByStartDate.filter(Start__day=i)
        if FilteredShifts:
            if FilteredShifts[0].Start.strftime('%a %d %b %Y') not in WeekTH:
                WeekTH.append(FilteredShifts[0].Start.strftime('%a %d %b %Y'))

    return WeekTH


def CreateTableHeadersList():
    Week1TH = getTableHeader(1)
    Week2TH = getTableHeader(8)
    Week3TH = getTableHeader(15)
    Week4TH = getTableHeader(22)
    Week5TH = getTableHeader(29)
    return [Week1TH, Week2TH, Week3TH, Week4TH, Week5TH]
```

The table headers will be returned separately because it removes the need for joining the headings onto the main table's list. Instead, the table headers will all be generated separately, which is how HTML works, as there are separate tags for <th> (table headings). The first function getTableHeader requires an input called DayNo. This variable is used as a pointer for the start of the next 7 days. This can be hardcoded when calling the function (The purpose of the 'CreateTableHeadersList' function), because since the rota is always generated from the 1st of the month, it will always be a case of just adding on 7 more days to get the next week.

Using Python's datetime manipulation, the Start time (Stored as a Datetime object) of the first shift on that day is found. It can then be used to get the full date and turned into a string datatype which can then be appended to the list. The table headers will look like the following:

Sun 01 Apr 2018	Mon 02 Apr 2018	Tue 03 Apr 2018	Wed 04 Apr 2018	Thu 05 Apr 2018	Fri 06 Apr 2018	Sat 07 Apr 2018

This does however depend if there are any shifts on that day. If for example there are no shifts on 'Wed 04 Apr 2018', the day will be skipped out and that table within the rota will contain one less day.

```
def getShiftTimes(DayNo):
    WeekShifts = []
    TempShiftsTimeList = []
    FilteredShifts = Shifts.filter(Start__day__gte=DayNo,
                                   Start__day__lt=DayNo + 8)
    for i in FilteredShifts:
        iStartTime = datetime.strftime(i.Start, '%H:%M')
        iEndTime = datetime.strftime(i.End, '%H:%M')
        iShiftTime = [iStartTime, iEndTime]
        if iShiftTime not in TempShiftsTimeList:
            WeekShifts.append([i.Start, i.End])
            TempShiftsTimeList.append(iShiftTime)
    return WeekShifts


def CreateShiftTimesList():
    Week1Shifts = getShiftTimes(1)
    Week2Shifts = getShiftTimes(8)
    Week3Shifts = getShiftTimes(15)
    Week4Shifts = getShiftTimes(22)
    Week5Shifts = getShiftTimes(29)
    return [Week1Shifts, Week2Shifts, Week3Shifts, Week4Shifts, Week5Shifts]
```

The following piece of code is structured in the same way as the functions that deal with the creation of the table headers list, however this time it is for each Shift (Or row). It gets a list of all the shifts within a 7-day period and linearly iterates through. Once a shift time has been added to the 'WeekShifts' list, it is then also added to the 'TempShiftsTimeList'. The purpose of the second list is so that for each iteration, the shift time in question can be compared to see whether it has already been added to the list. If this is the case, there is no need to add another identical shift time, therefore the next iteration takes place without any further action taken. On the other hand, if the shift doesn't exist within the 'TempShiftsTimeList', then the Shift will be added to said list, and the 'WeekShifts' list. The reason for two lists and not just one, is that the 'TempShiftsTimeList' *only* holds the time, in 24hr format, as strings. The 'WeekShifts' list stores the whole datetime object of the start and end time.

```
def CreateWeekShiftList(WeekNo, TableHeaders, ShiftTimes):
    WeekShiftList = []
    for Shift in ShiftTimes[WeekNo]:
        ShiftStartTimeString = str(datetime.strftime(Shift[0], '%H:%M'))
        ShiftEndTimeString = str(datetime.strftime(Shift[1], '%H:%M'))
        ShiftTimeString = ShiftStartTimeString + ' - ' + ShiftEndTimeString
        WeekShiftListRow = [ShiftTimeString]
        SkipFirstItteration = True
        for Date in TableHeaders[WeekNo]:
            if SkipFirstItteration == True:
                SkipFirstItteration = False
                continue
            Date = datetime.strptime(Date, '%a %d %b %Y')
            Date = datetime.strftime(Date, '%d%m%Y')
            ShiftStartDateTime = datetime.strptime(Date+ShiftStartTimeString, '%d%m%Y%H:%M')
            ShiftEndDateTime = datetime.strptime(Date+ShiftEndTimeString, '%d%m%Y%H:%M')

            try:
                ShiftObject = Shifts.get(Start=ShiftStartDateTime, End=ShiftEndDateTime)
                WeekShiftListRow.append(ShiftObject.Employees.user.username)
            except:
                WeekShiftListRow.append(' ')
        WeekShiftList.append(WeekShiftListRow)
    return WeekShiftList
```

This function brings together all the other function's return values and makes use of them by deciding where the employee names should be located within the table. This is done table by table, therefore a maximum of 7 days are considered, and constructed row by row. The first for loop ensures that the code is repeated for each row (in other words shift time). The next section states that the first column shall contain the shift time, however the code seems slightly more complex as it must deal with reformatting the datetime objects to display only the time in a 24hr format as a string value. The next for loop which is embedded (or nested) within the last, loops through the table headers, which provide information about the date. This information can be coupled with the shift time to create a datetime object. The shifts stored within the database are then queried with the condition that the StartTime must equal the datetime object that was just put together. If this is true, the WeekShiftListRow will be appended with the username of the employee who is associated with that shift. Otherwise, the list is appended with an empty string. The empty string ensures a full table, meaning the rendering process will be simple even with gaps in the schedule.

```
def CreateMonthShiftList():
    ShiftTimes = CreateShiftTimesList()
    TableHeaders = CreateTableHeadersList()
    Week1ShiftList = CreateWeekShiftList(0, TableHeaders, ShiftTimes)
    Week2ShiftList = CreateWeekShiftList(1, TableHeaders, ShiftTimes)
    Week3ShiftList = CreateWeekShiftList(2, TableHeaders, ShiftTimes)
    Week4ShiftList = CreateWeekShiftList(3, TableHeaders, ShiftTimes)
    Week5ShiftList = CreateWeekShiftList(4, TableHeaders, ShiftTimes)
    return [Week1ShiftList, Week2ShiftList, Week3ShiftList, Week4ShiftList, Week5ShiftList], TableHeaders
```

The final function is the one that is called from the 'views', so that it can be returned as context for the Rota template. This simply returns a list of 5 tables, and the table headers separately, after calling the CreateWeekShiftList function 5 times (for each table, as there are usually 4 and a bit weeks in a month- the 'bit' or overflow is entered into the 5th table). However, the CreateWeekShiftList requires all the other functions to run so that they generate its inputs. Therefore, they are called first.

```
MonthShiftList, TableHeaders = CreateMonthShiftList()

context = {
                'Week1List': MonthShiftList[0],'Week2List': MonthShiftList[1],'Week3List': MonthShiftList[2],
                'Week4List': MonthShiftList[3],'Week5List': MonthShiftList[4],
                'Week1TableHeaders': TableHeaders[0], 'Week2TableHeaders': TableHeaders[1],
                'Week3TableHeaders': TableHeaders[2], 'Week4TableHeaders': TableHeaders[3],
                'Week5TableHeaders': TableHeaders[4]
               }
page = './Schedule/rota.html'
return render(request, page, context)
```

Within the Rota view, the context looks as follows. Simply, it is broken down into sections that can then be iterated and brought together as one table within the template. However, the one part that was not shown is that the module had to be imported into the views file before it was called.

```
<table>
    <tr>
        {% for Heading in Week1TableHeaders %}
            <th>{{ Heading }}</th>
        {% endfor %}
    </tr>

    {% for Row in Week1List %}
        <tr>
            {% for Collumn in Row %}
                <td>{{ Collumn }}</td>
            {% endfor %}
        </tr>
    {% endfor %}
</table>
```

Each of the 5 tables are created the same way as this. It is self-explanatory as it is simply unpacking the data structure with for loops to iterate through the lists linearly. The simplicity means that the objective of making the template rendering as simple as possible was achieved.

GUI DEVELOPMENT (CONT.)

The exact same process as described above was used to generate the sample rota, with the only difference that the shifts compared were those with Active=False as seen below:

```
Shifts = Shifts.objects.all().filter(Active=False)
```

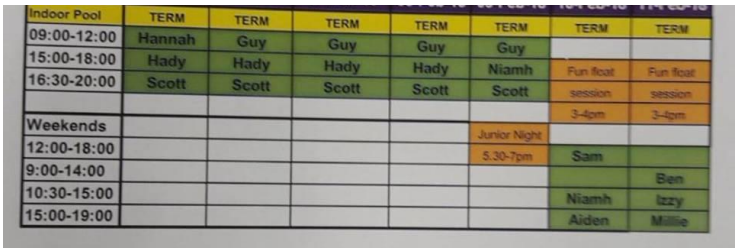
This could also be considered a generalised solution to the problem, although the solution is next to identical aside from the above change.

Before the above features can be fully tested, the requirement for a fully developed algorithm is required. However, it is also possible to run simple tests on the features without that part of the solution. This is done by unconventionally, and manually, entering the test data straight into the database.

GUI DEVELOPMENT (CONT.)

ROTA GENERATION RENDERING AND SHIFT INPUT TESTING

Because the code for the provisional rota rendering and the active rota rendering test is the same, for simplicity I will be using a smaller data set and will be using the provisional rota page to test the generation process. I will also be using software called 'DB Browser for SQLite'. This will help me input test data (assigning shifts to employees) because the algorithm has not been developed yet.



Indoor Pool	TERM	TERM	TERM	TERM	TERM	TERM	TERM
09:00-12:00	Hannah	Guy	Guy	Guy	Guy		
15:00-18:00	Hady	Hady	Hady	Hady	Niamh	Fun float session 3-4pm	Fun float session 3-4pm
16:30-20:00	Scott	Scott	Scott	Scott	Scott		
Weekends					Junior Night 5.30-7pm		
12:00-18:00						Sam	
9:00-14:00							Ben
10:30-15:00						Niamh	Izzy
15:00-19:00						Aiden	Millie

The shift times will be the same as above, but for a different month- April.

Test No.	Type	Description	Input	Expected Output	Actual Output	Comment
1	Normal	This is to test that a normal data input will be accepted, with a new shift object created.	Start Time: 09:00 End Time: 12:00 Day: 1 Month: 5 Year: 2018	The shift should display in the database.	The shift is saved correctly in the database.	See Screenshot No. 1
2	Erroneous	This is to test that an erroneous data input will not be accepted, no new shift object will be created.	Start Time: Nine End Time: Twelve Day: One	The shift should not display anywhere, and the page should refresh.	The shift does not display anywhere, and the page is simply refreshed, ready for a valid input.	See Screenshot No. 1 (Because nothing has changed as expected.)
3	Usability	The overall ease and efficiency of data entry for a new shift.	All the other dates and times for April shifts.	There should be no errors and be efficient.	There are no errors, however the process is inefficient.	See Screenshot No. 2 for a detailed explanation.

GUI DEVELOPMENT (CONT.)

ROTA GENERATION RENDERING AND SHIFT INPUT TESTING

SCREENSHOT 1

Shift Man

Add Shift
Time Format: HH:MM (e.g. 09:00)

Start Time: 09:00

End Time : 12:00

Month: 6

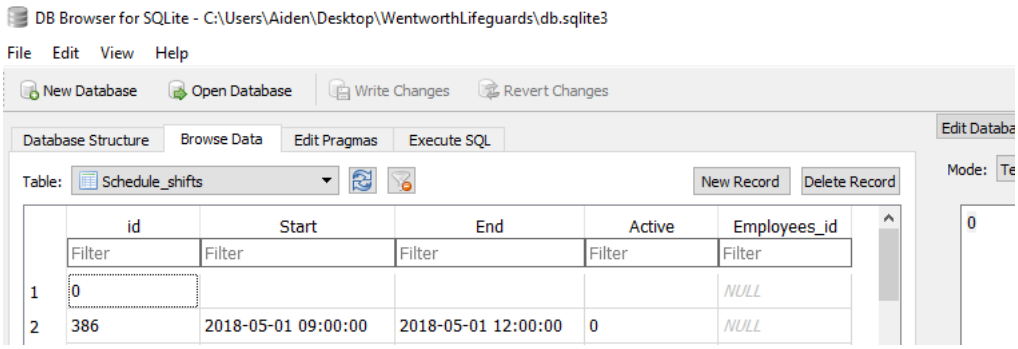
Year: 2018

Days: 1 2 3 4

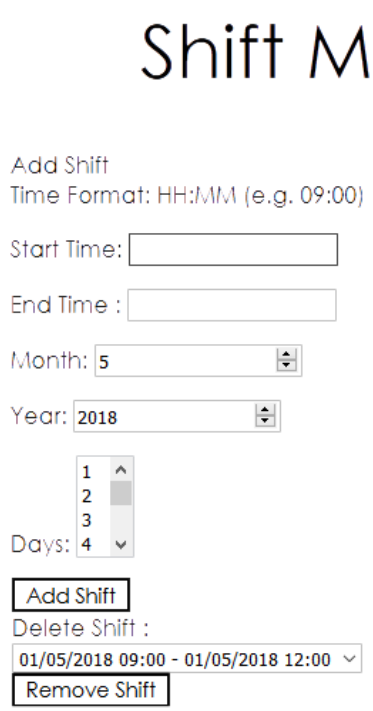
Add Shift

Delete Shift :

Remove Shift



SCREENSHOT 1



As we can see from above, the delete Shift dropdown menu form displays the shift that has just been added. This confirms both that the delete shift form's display is working well, but also that the shift is in the database. To confirm, the previous screenshot shows the record having been saved through the DB manager. This confirmation that the Shift input works with normal data.

Screenshot 2

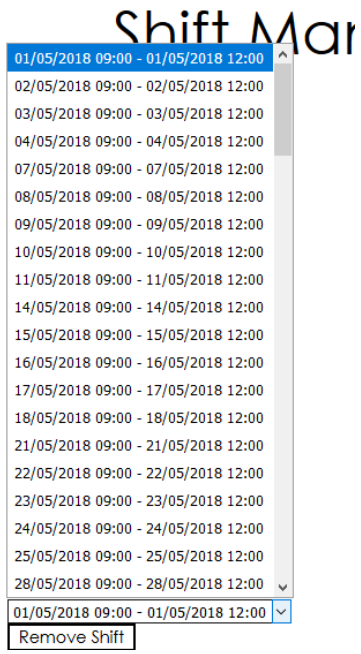
```
Is Bound
<ul class="errorlist"><li>StartTime<ul class="errorlist"><li>Enter a valid time
 .</li></ul></li><li>EndTime<ul class="errorlist"><li>Enter a valid time
 .</li></ul></li></ul>
Add Shift Invalid
```

The terminal output, showing that there was an error entering the form. The page simply refreshed and did not save anything to the database.

GUI DEVELOPMENT (CONT.)

ROTA GENERATION RENDERING AND SHIFT INPUT TESTING

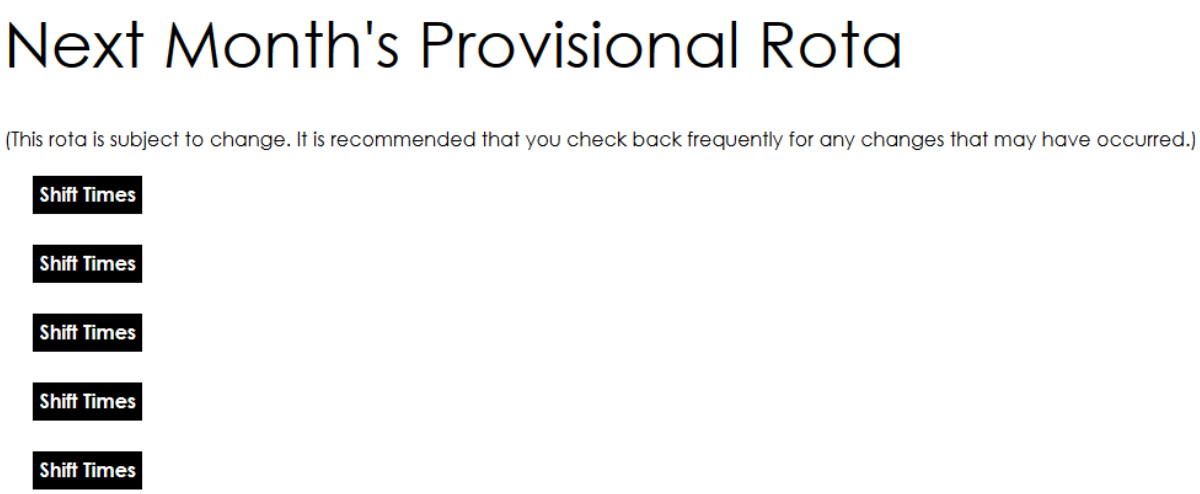
SCREENSHOT 2



The usability test failed because the data entry was unacceptably inefficient. Each shift had to be manually submitted one at a time, with the final shift count totalling 102. Therefore, this same form had to be **submitted 102 times** to create a rota for one month. This was very time consuming and boring, in which time a rota could have probably been created by hand. A method of entering this data will have to be thought of and developed before I, and the client, are happy with the product.

ROTA GENERATION RENDERING AND SHIFT INPUT TESTING

Initially, without any shift data in the database, the rota page displays as follows:



This is because the only cells that contain any data are the hardcoded 'Shift Times' Header. This occurs once in each table, and since there are 5 tables created, there are 5 'Shift Times' showing.
The way this page is displayed is not considered an error.

Test No.	Type	Description	Input	Expected Output	Actual Output	Comment
1	Normal	This is to test that a full dataset will render as expected on the screen.	Full dataset	The shifts should display as 5 separate tables separated after every 7 days.	The shifts are displayed as 5 separate tables which are separated after every 7 days.	Fully working See Screenshot No. 3

Screenshot 3

Next Month's Provisional Rota

(This rota is subject to change. It is recommended that you check back frequently for any changes that may have occurred.)

Shift Times	Tue 01 May 2018	Wed 02 May 2018	Thu 03 May 2018	Fri 04 May 2018	Sat 05 May 2018	Sun 06 May 2018	Mon 07 May 2018
09:00 - 12:00							
15:00 - 18:00							
16:30 - 20:00							
12:00 - 18:00							
09:00 - 14:00							
10:30 - 15:00							
15:00 - 19:00							

Shift Times	Tue 08 May 2018	Wed 09 May 2018	Thu 10 May 2018	Fri 11 May 2018	Sat 12 May 2018	Sun 13 May 2018	Mon 14 May 2018
09:00 - 12:00							
15:00 - 18:00							
16:30 - 20:00							
12:00 - 18:00							
09:00 - 14:00							
10:30 - 15:00							
15:00 - 19:00							

Shift Times	Tue 15 May 2018	Wed 16 May 2018	Thu 17 May 2018	Fri 18 May 2018	Sat 19 May 2018	Sun 20 May 2018	Mon 21 May 2018
09:00 - 12:00							
15:00 - 18:00							
16:30 - 20:00							
12:00 - 18:00							
09:00 - 14:00							
10:30 - 15:00							
15:00 - 19:00							

Shift Times	Tue 22 May 2018	Wed 23 May 2018	Thu 24 May 2018	Fri 25 May 2018	Sat 26 May 2018	Sun 27 May 2018	Mon 28 May 2018
09:00 - 12:00							
15:00 - 18:00							
16:30 - 20:00							
12:00 - 18:00							
09:00 - 14:00							
10:30 - 15:00							
15:00 - 19:00							

Shift Times	Tue 29 May 2018	Wed 30 May 2018	Thu 31 May 2018
09:00 - 12:00			
15:00 - 18:00			
16:30 - 20:00			

GUI DEVELOPMENT (CONT.)

The previous tests revealed that the shift entry was not user friendly due to the sheer number of times the user would have to enter data in order to create a month's rota. A way to fix this would be to amend the form so that it includes an option to apply a certain shift to all weekdays, all weekends, or both. This would be as an alternative option to selecting the days manually. The idea is that it would look like the following:


```
AllWeekDays = forms.BooleanField(required=False, label='All Week Days')
AllWeekendDays = forms.BooleanField(required=False, label='All Weekend Days')
```

It is a simple addition of two Boolean choices to the form. However, when the form is submitted, there will be conditionals to test if the user has selected one of the Boolean options or both. This process will therefore be handled within the views file.
Currently, there is an iteration that states for every day in a list of days selected by the user, a new shift will be created. This is possible because the current day selection is a multiple-choice field, meaning that the days a user chooses are returned in a list.

GUI DEVELOPMENT (CONT.)

```
AllWeekDays = AddShiftsInstance.cleaned_data['AllWeekDays']
AllWeekendDays = AddShiftsInstance.cleaned_data['AllWeekendDays']
DaysInMonth = monthrange(Year, Month)[1]
if AllWeekDays or AllWeekendDays:
    WeekDays = []
    WeekendDays = []
    for Day in range(1,DaysInMonth+1):
        DatetimeStr = str(Day) + str(Month) + str(Year) + str(StartTime.strftime('%H:%M'))
        Datetime = datetime.strptime(DatetimeStr, '%d%m%Y%H:%M')
        if Datetime.weekday() < 5:
            WeekDays.append(Day)
        if Datetime.weekday() >= 5:
            WeekendDays.append(Day)

    if AllWeekDays:
        Days = WeekDays
    elif AllWeekendDays:
        Days = WeekendDays

 elif AllWeekDays and AllWeekendDays:


for Day in Days:
    Day = str(Day).zfill(2)
    StartDatetimeStr = str(Day) + str(Month) + str(Year) + str(StartTime.strftime('%H:%M'))
    print(StartDatetimeStr)
    StartDatetime = datetime.strptime(StartDatetimeStr, '%d%m%Y%H:%M')
    EndDatetimeStr = str(Day) + str(Month) + str(Year) + str(EndTime.strftime('%H:%M'))
    EndDatetime = datetime.strptime(EndDatetimeStr, '%d%m%Y%H:%M')
    Shifts.objects.create(Start=StartDatetime, End=EndDatetime, Employees=None, Active=False)
page, context = ReturnPageAndContext(request.user)
return render(request, page, context)
```

A built-in python module called 'Calendar' has a function called 'monthrange', where the month number and year are entered as parameters, and it returns the number of days in the month. This is a useful feature because it gives us a range to iterate through. Each month day number is the entered as a parameter into the Datetime module's 'weekday()' function. This function returns the number of the week the day number is. For example, if 01/05/2018 was entered, it would return 1 because the 1st of May 2015 is a Tuesday (or the 2 day of the week).
Each day of the month can then be checked whether it is less than 5 (a weekday), or greater than or equal to 5 (a weekend day). A list of the weekdays can then be created, and a list of weekends can also be created. Then, if one of the two Boolean options was chosen, the variable 'Days' can be overwritten to include the new list of day numbers.

All Week Days: ☐

All Weekend Days: ☐

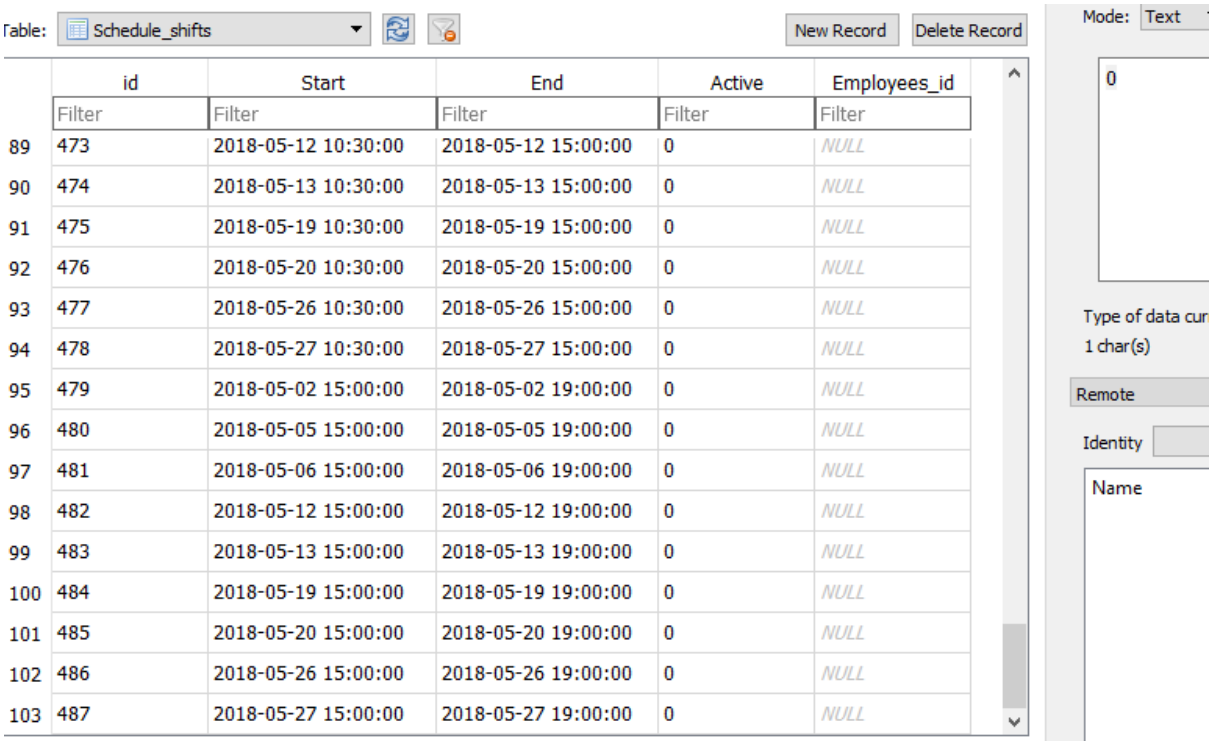
GUI DEVELOPMENT (CONT.)

ROTA GENERATION RENDERING AND SHIFT INPUT TESTING

Now that I have developed a solution to the usability testing, the test shall be re-run so that it can be ticked off as a pass.

Test No.	Type	Description	Input	Expected Output	Actual Output	Comment
3	Usability	The overall ease and efficiency of data entry for a new shift.	All the other dates and times for April shifts.	There should be no errors and be efficient.	There are no errors, and the process is now efficient.	See Screenshot No. 4.

SCREENSHOT 4



	id	Start	End	Active	Employees_id
	Filter	Filter	Filter	Filter	Filter
89	473	2018-05-12 10:30:00	2018-05-12 15:00:00	0	NULL
90	474	2018-05-13 10:30:00	2018-05-13 15:00:00	0	NULL
91	475	2018-05-19 10:30:00	2018-05-19 15:00:00	0	NULL
92	476	2018-05-20 10:30:00	2018-05-20 15:00:00	0	NULL
93	477	2018-05-26 10:30:00	2018-05-26 15:00:00	0	NULL
94	478	2018-05-27 10:30:00	2018-05-27 15:00:00	0	NULL
95	479	2018-05-02 15:00:00	2018-05-02 19:00:00	0	NULL
96	480	2018-05-05 15:00:00	2018-05-05 19:00:00	0	NULL
97	481	2018-05-06 15:00:00	2018-05-06 19:00:00	0	NULL
98	482	2018-05-12 15:00:00	2018-05-12 19:00:00	0	NULL
99	483	2018-05-13 15:00:00	2018-05-13 19:00:00	0	NULL
100	484	2018-05-19 15:00:00	2018-05-19 19:00:00	0	NULL
101	485	2018-05-20 15:00:00	2018-05-20 19:00:00	0	NULL
102	486	2018-05-26 15:00:00	2018-05-26 19:00:00	0	NULL
103	487	2018-05-27 15:00:00	2018-05-27 19:00:00	0	NULL

As can be seen, there is a full set of data, implemented through the form using the added feature. This greatly cut down the time taken to less than a minute, meaning that the test for efficiency and usability now passes. It also meant that the number of form submissions was reduced from a maximum of 102 per month (for this dataset), to a minimum of 7.

GUI DEVELOPMENT (CONT.)

The final stage of developing the graphical user interface, is to include buttons that control the schedule generation. As per the design, this shall be located on the far-right side of the admin management page. The success criteria also state that the important buttons should be coloured, to prevent miss clicks.

The buttons to generate the Rota will work the same as the other form submit buttons. A HTML <submit> button will be created, with a unique name that can be found within the POSTed data in the views. However, the main difference between this structure and the other forms is that no data is sent within the form. The reason for this is because none is required apart from the fact that the button was pressed, meaning that the user requires the matching process to take place.

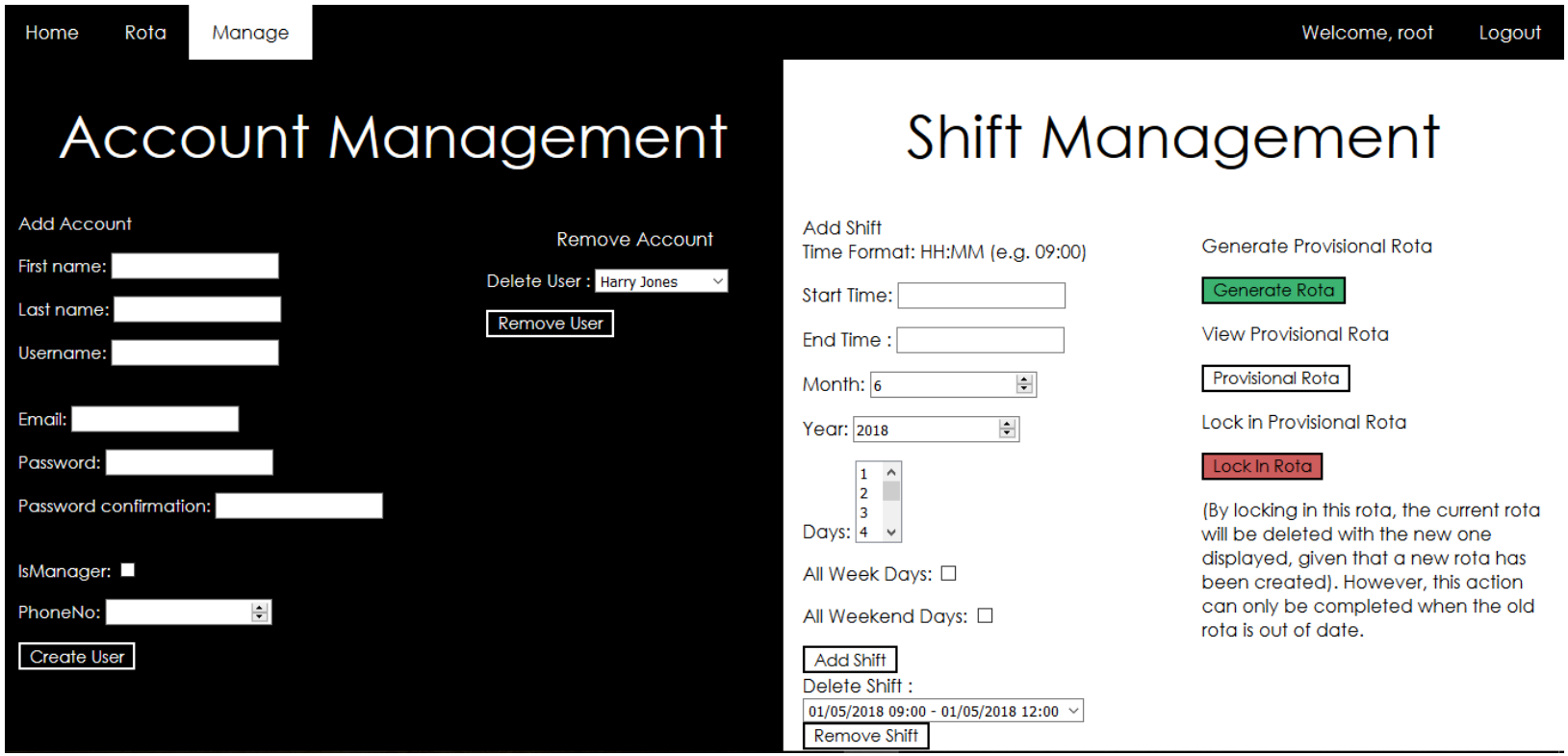
```
<div style="...">
<p>Generate Provisional Rota
<form action="/manage" method="post">
    {% csrf_token %}
    <input type="submit" value="Generate Rota" name = "Generate Rota" style="background-color: mediumseag
</form>
</p>
<p>View Provisional Rota
<form action="/provisionalrota" method="post">
    {% csrf_token %}
    <input type="submit" value="Provisional Rota" name = "Provisional Rota">
</form>
</p>
<p>
Lock in Provisional Rota
<form action="/manage" method="post">
    {% csrf_token %}
    <input type="submit" value="Lock In Rota" name="Lock In Rota" style="background-color: indianred">
</form>
</p>
(By locking in this rota, the current rota will be deleted with the new one displayed, given that a new
 rota has been created). However, this action can only be completed when the old rota is out of date.
</div>
```

There is also a warning message for the user that is located underneath the 'Lock in Provisional Rota' button, as it is a function that cannot be undone. This is for usability so that the user understands the function of the button in question and is therefore less likely to make a mistaken click. Coupled with this, the said button is coloured red to display the importance and act as a warning. The button that will call the generation of a new provisional rota will be coloured green. This is because it is a key feature of the web application, and makes it stand out as an important feature.

The algorithm will be created in a separate file, which can then be imported as a module and called within the view. This is because the algorithm is likely to be long and so the maintainability and readability of the views file is ensured. The views file handling the generation request will simply look like the following after the module is imported:

```
    elif 'Generate Rota' in request.POST:
        print('Generate Rota Form')
        getSchedule()
```

The management page now looks like the following:



CORE ALGORITHM DEVELOPMENT

This section of the development will potentially be the quickest and easiest as the core algorithm has been designed and discussed in depth throughout the design phase. It is a simple process of turning the previously pseudo coded solution into Python and Django code. It is for this reason that there will not be much discussion about how the algorithm is processing the data as done for the other functions of the web application, because it has already been discussed in detail, with worked examples. Please see the 'Core Algorithm Design and Features' section of the development.

There are 3 main functions within the algorithm. The first stage would be to clear the shifts that are currently set as active. This removes the last month's shifts.

```
def clearLockedShifts():
    from .models import Shifts
    for i in Shifts.objects.all().filter(Active=True):
        i.delete()
```

A slightly unconventional local import takes place within this function, because the globally imported 'Shifts' are globally filtered for non-active shifts, so that the filter process only has to occur once and not within any other function. However, the global import wouldn't suffice in the case where we need to access the locked shifts, so it is done locally instead.

```
def getEmployeeAvailability():
    '''
    The function returns a list of lists, that are structured so that:
    [[Employee -(Object)-, Shifts Available To Work (List of shift object primary keys),
    Number of Shifts Available (Integer)]]
    '''

    EmployeeAvailability = []
    for Employee in Employees.objects.all():
        EmployeeUsername =Employee.user.username
        Availability = []
        NumOfShifts = 0
        for Shift in Employee.Availability.all():
            Availability.append(Shift.pk)
            NumOfShifts += 1
        EmployeeAvailability.append([EmployeeUsername, Availability, NumOfShifts])

    EmployeeAvailability = sorted(EmployeeAvailability, key=lambda x: x[2])

    return EmployeeAvailability
```

Python tends to look much like pseudocode, which is a very big advantage when converting between the both. After the main O(n^2) nested loops have finished iterating, a heuristic is applied, whereby the EmployeeAvailability list is sorted by the number of shifts each employee is available for. This prevents the need to individually compare the employee availability throughout the remaining part of the scheduling algorithm, because the following function can assume the inputted list is ordered.

CORE ALGORITHM DEVELOPMENT (CONT.)

The third main function within the core algorithm is where the shifts are assigned based on the algorithm discussed in the design.

```
def assignShifts(EmployeeAvailability):

    while EmployeeAvailability:
        Employee = EmployeeAvailability.pop(0)
        NumOfShifts = Employee[2]
        if NumOfShifts > 0:
            Shift = Shifts.get(pk=Employee[1][0])
            UserObject = User.objects.get(username=Employee[0])
            EmployeeObject = Employees.objects.get(user=UserObject)
            print(EmployeeObject)
            if Shift.Employees == None:
                Shift.Employees = EmployeeObject
                Shift.save()
                print('Unable To Fill Shift')
            if NumOfShifts > 1:
                NumOfShifts -= 1
                Employee[2] = NumOfShifts
                del Employee[1][0]
                EmployeeAvailability.append(Employee)
                print('EmployeeFullyScheduled')
            print('ShiftFilled')
```

Again the code looks very familiar, however I have added print statements at the end of each possible outcome, so that after each iteration if an error is to occur, it is easily debugged. It also makes the code easier to read, which enhances the maintainability.

The final part is to bring both main functions together into a callable function which runs the code.

```
def getSchedule():
    EmployeeAvailability = getEmployeeAvailability()
    assignShifts(EmployeeAvailability)
```

CORE ALGORITHM DEVELOPMENT TESTING

FirstName	LastName	Username	Email	Availability								
Michael	*Brooks*	*MichaelB*	michaelb@wentworthlifeguards.com									
Hannah	Atkins	HannahA	hannaha@wentworthlifeguards.com	19	13	2						
Hady	Johnson	HadyJ	haddyj@wentworthlifeguards.com	5	21	9						
Scott	Samson	ScottS	scotts@wentworthlifeguards.com	6	7	14	9	3	20	8	11	18
Guy	Brick	GuyB	guyb@wentworthlifeguards.com	21								
Niamh	Baguette	NiamhB	niamhb@wentworthlifeguards.com	5	2	1	22	16				
Sam	Lennon	SamL	saml@wentworthlifeguards.com	8	17							
Jasmin	Dameon	JasminD	jasmind@wentworthlifeguards.com	21	4	10	14	12				
Aiden	Ukov	AidenU	aidenu@wentworthlifeguards.com	11								
Ben	Dargon	BenD	bend@wentworthlifeguards.com	9	15							
Izzy	Circuit	IzzyC	izzyc@wentworthlifeguards.com	14	3							
Millie	Lettice	MillieL	milliel@wentworthlifeguards.com	1	23	11	19	16				

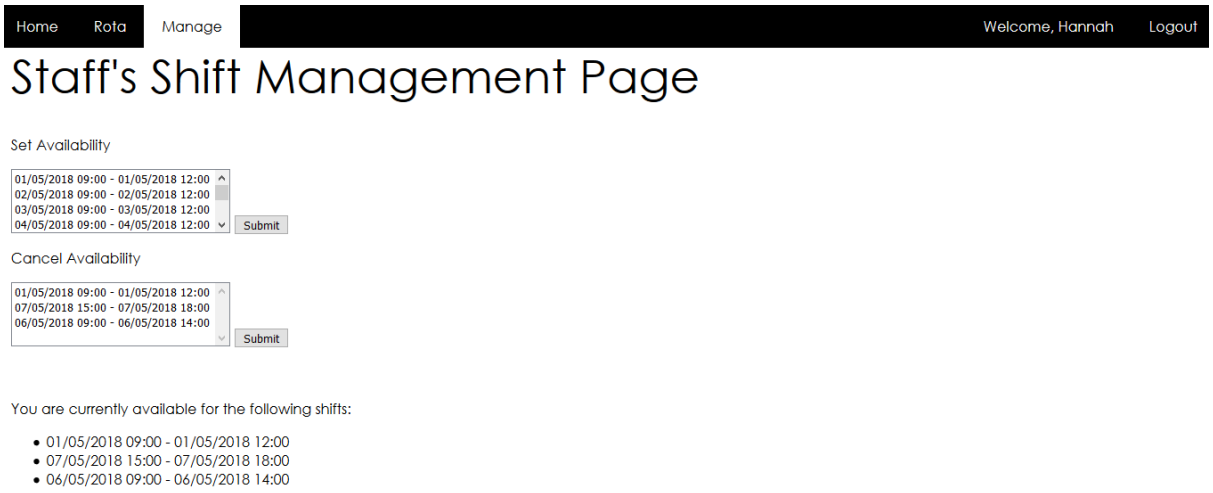
The test data used will be the following, the same you may notice as was used in the design phase. However, the numbers will correspond to a different month's rota. The white numbers represent repeated shifts. The following picture is a key:

Shift Time	01/05/2018	02/05/2018	03/05/2018	04/05/2018	05/05/2018	06/05/2018	07/05/2018
09:00 - 14:00					1	2	
10:30 - 15:00					3	4	
12:00 - 18:00					5	6	
15:00 - 19:00					7	8	
15:00 - 18:00	9	10	11	12			13
16:30 - 20:00	14	15	16	17			18
09:00 - 12:00	19	20	21	22			23

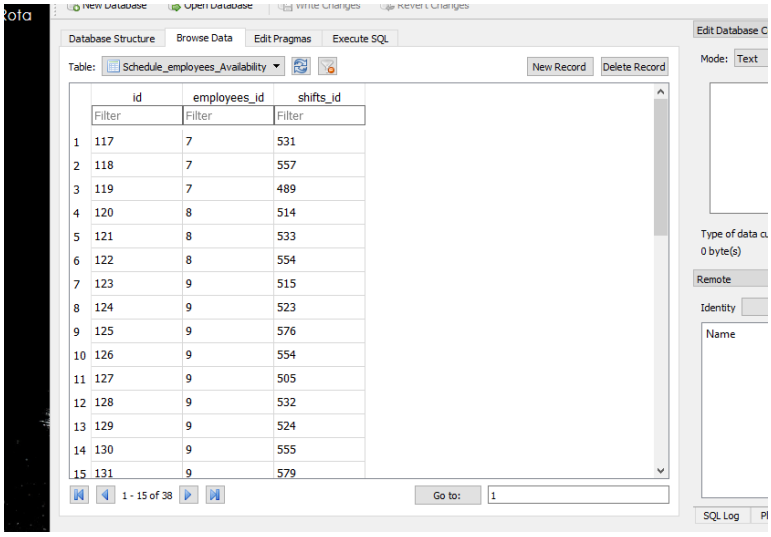
The highlighted dates are weekends, so the shifts vary slightly.

Test No.	Type	Description	Input	Expected Output	Actual Output	Comment
1	Normal	The algorithm runs with no syntactical errors, or critical errors that prevent the code from running.	N/A (No direct inputs, although a lot of data entry will take place to create the dataset.	Each record from the database table Shifts should be assigned an Employee.	Each record from the database table Shifts is assigned an Employee.	See Screenshot No. 1
2	Normal	The algorithm's result should be visualised in a correctly rendered table on the provisional rota page.	N/A	Each shift should be displayed in the table. If an employee is assigned, username should be within the cell.	Each shift is displayed correctly in the table, with corresponding usernames assigned to the correct cells as expected.	See Screenshot No. 2

Screenshots showing the data entry



Setting the availability of HannahA through the account. As the screenshot displays, this process works as expected. This will be the same process for every other account.



All data entry has finished, and there are a total of 23 shifts, with employees available to work a total of 38.

CORE ALGORITHM DEVELOPMENT TESTING (CONT.)

SCREENSHOT 1

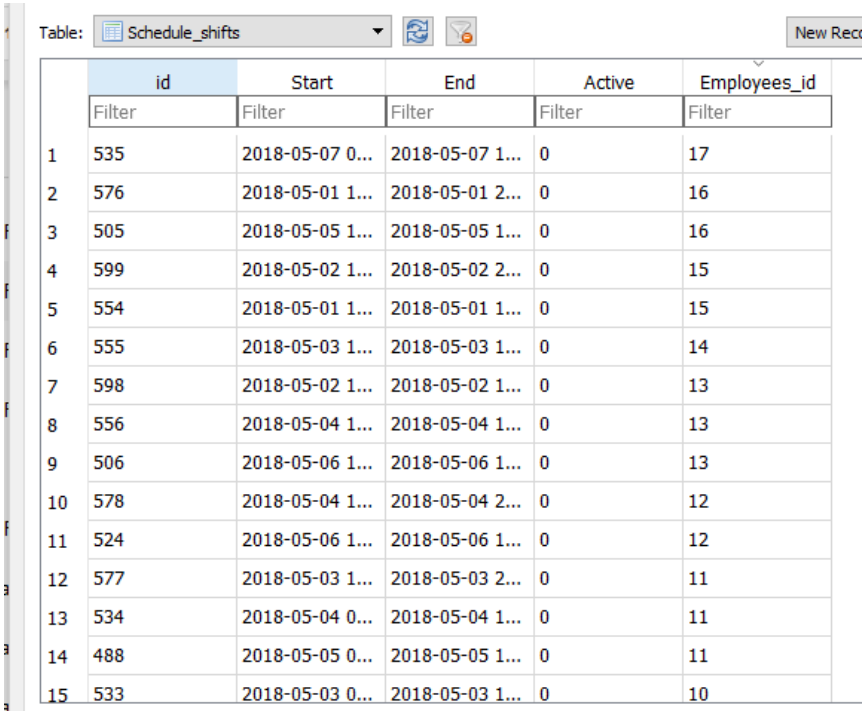
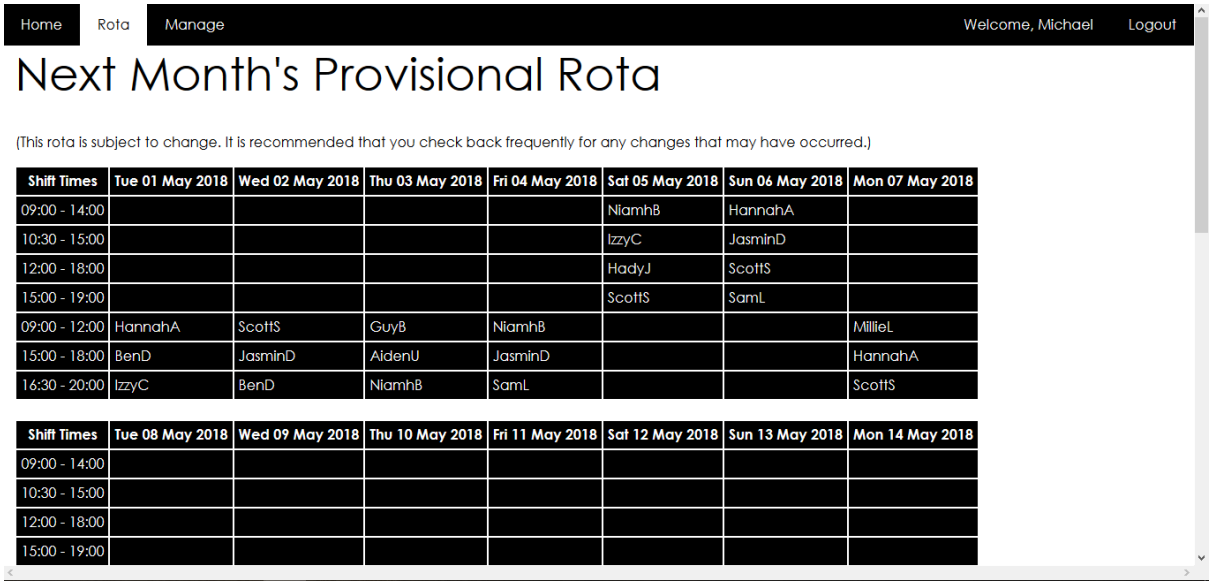


Table: Schedule_shifts

	id	Start	End	Active	Employees_id
	Filter	Filter	Filter	Filter	Filter
1	535	2018-05-07 0...	2018-05-07 1...	0	17
2	576	2018-05-01 1...	2018-05-01 2...	0	16
3	505	2018-05-05 1...	2018-05-05 1...	0	16
4	599	2018-05-02 1...	2018-05-02 2...	0	15
5	554	2018-05-01 1...	2018-05-01 1...	0	15
6	555	2018-05-03 1...	2018-05-03 1...	0	14
7	598	2018-05-02 1...	2018-05-02 1...	0	13
8	556	2018-05-04 1...	2018-05-04 1...	0	13
9	506	2018-05-06 1...	2018-05-06 1...	0	13
10	578	2018-05-04 1...	2018-05-04 2...	0	12
11	524	2018-05-06 1...	2018-05-06 1...	0	12
12	577	2018-05-03 1...	2018-05-03 2...	0	11
13	534	2018-05-04 0...	2018-05-04 1...	0	11
14	488	2018-05-05 0...	2018-05-05 1...	0	11
15	533	2018-05-03 0...	2018-05-03 1...	0	10

This screenshot shows that the shifts that have employees available for them, have all been assigned employees.

SCREENSHOT 2



Home | Rota | Manage | Welcome, Michael | Logout

Next Month's Provisional Rota

(This rota is subject to change. It is recommended that you check back frequently for any changes that may have occurred.)

Shift Times	Tue 01 May 2018	Wed 02 May 2018	Thu 03 May 2018	Fri 04 May 2018	Sat 05 May 2018	Sun 06 May 2018	Mon 07 May 2018
09:00 - 14:00					NiamhB	HannahA	
10:30 - 15:00					IzzyC	JasminD	
12:00 - 18:00					HadyJ	ScottS	
15:00 - 19:00					ScottS	SamL	
09:00 - 12:00	HannahA	ScottS	GuyB	NiamhB			MillieL
15:00 - 18:00	BenD	JasminD	AidenU	JasminD			HannahA
16:30 - 20:00	IzzyC	BenD	NiamhB	SamL			ScottS

Shift Times	Tue 08 May 2018	Wed 09 May 2018	Thu 10 May 2018	Fri 11 May 2018	Sat 12 May 2018	Sun 13 May 2018	Mon 14 May 2018
09:00 - 14:00							
10:30 - 15:00							
12:00 - 18:00							
15:00 - 19:00							

Screenshot 2 shows that the shift rendering worked as expected, and looks very good too.

DEVELOPMENT AND TESTING CONCLUSION

The development process has gone very smoothly considering how much there was to build. However the design section of the project ensured that the development stayed on track and was frequently referred to, much like the success criteria. The product closely resembles the design, and the success criteria that was agreed upon by the client. The code itself is, I believe, highly efficient and written in such a way that is easily understood by someone with a limited experience of Django and Python. Since so many individual features have been included, it is next to impossible to test every single feature in great depth without a development team. However, it is clear through preliminary testing that in most cases, every feature should work as expected. Another slight problem is the fact that refactoring the code will take a very long time, much longer than expected. Therefore, for now, there may be slight inconsistencies regarding aspects such CamelCase convention for variable names etc. Although aside from this, the developed as a whole went well. There were very few syntax errors, and only a few logical errors that were quickly identified and corrected.

Markup Language Used:

HTML5 https://www.w3.org/html/

Styling Language Used:

CSS https://www.w3.org/Style/CSS/Overview.en.html

Programming Language Used:

Python 3.6 https://www.python.org/

Web Framework Used:

Django 2 https://www.djangoproject.com/

Publicly Available Modules that were used:

Calendar https://docs.python.org/3/library/calendar.html

DateTime https://docs.python.org/3/library/datetime.html

EVALUATION

Aiden Gourley

DESCRIPTION OF THE FINAL PRODUCT

The lifeguard management application allows interaction between the management team and employees regarding the scheduling process. The application's goal was to provide a way for lifeguards on casual working contracts to communicate their availability to the management team. However, the application goes a step further, and allows the management team to generate a month's schedule at the click of a button. This is possible using an algorithm which aims to balance the distribution of shifts based on the number of shifts each lifeguard is available to work.

USER FEEDBACK

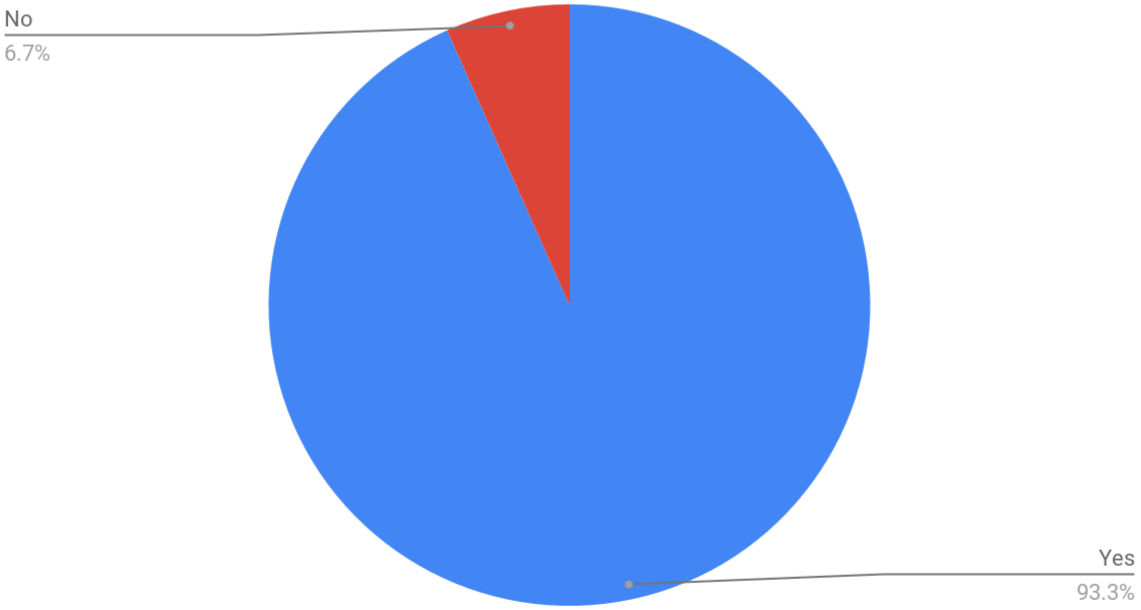
A questionnaire was taken by different stakeholders in order to gauge a reaction to the usability of the program, and the project overall. This was accompanied with a working prototype of the program, so the respondent had first-hand experience of the product when answering these questions. Not all stakeholders were qualified lifeguards, however all of them have had experience working or managing employees on casual contracts. The questionnaire comprised of the following questions:

1. Do you feel the program has a contemporary look/ feel? Y/N
2. Do you feel as though the application was easily and intuitively navigated? Please respond on a scale of 1 (No, not at all) to 10 (Yes, completely):
3. Did you understand every feature and its job either straightaway or after referencing the user manual? Y/N
4. Did you feel your experience using the program was customised? Please respond on a scale of 1 (No, not at all) to 10 (Yes, very):
5. Did you feel there were enough hints/ information blocks within the application? Y/N
6. How easy did you feel it was to enter data and submit forms on the web application. Please respond on a scale of 1 (Not easily) to 10 (Very easily):
7. Please respond stating the most difficult part of using the web application (e.g. data entry for a form):
8. Please respond stating the easiest part of using the web application (e.g. generating a schedule):
9. Did the user interface feel responsive? Y/N
10. Was the schedule generation (the core algorithm) run time acceptable? If not, please explain why you felt this way and an approximate runtime in your case. Y/N

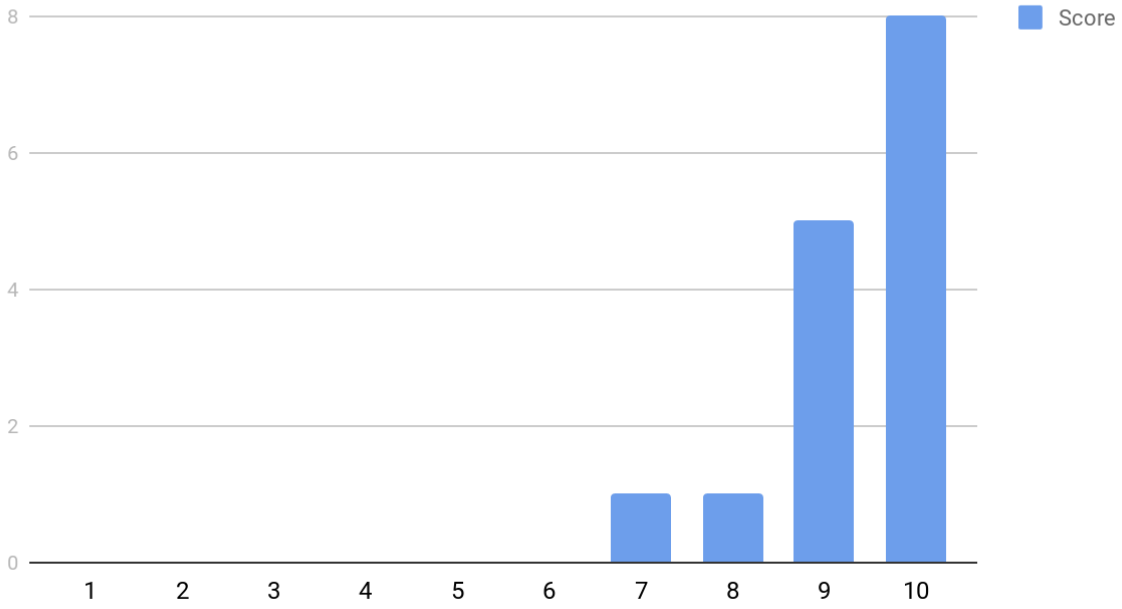
A total of 20 stakeholders were asked to complete the survey, of which 15 obliged. The results are displayed on the following page through various types of graphs.

USER FEEDBACK RESULTS (CONT.)

Question 1

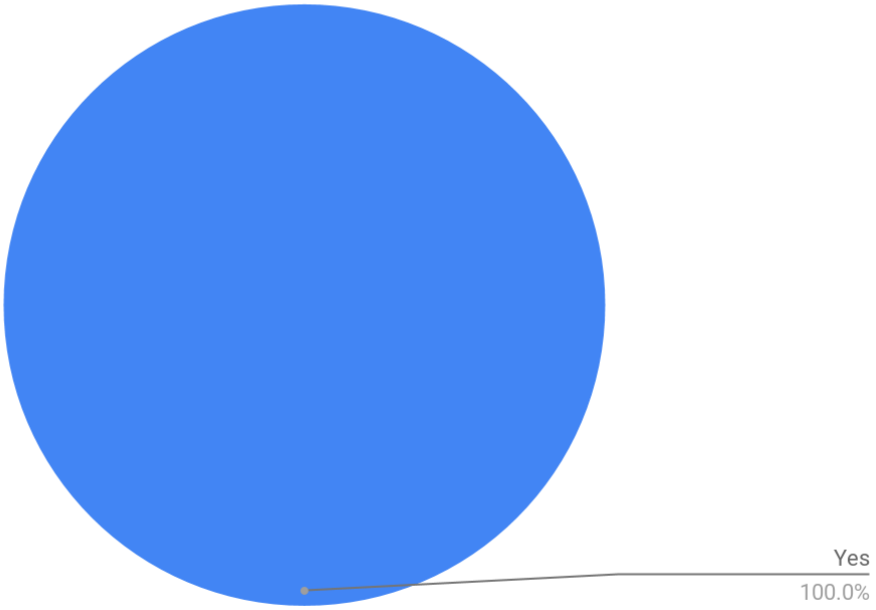


Question 2

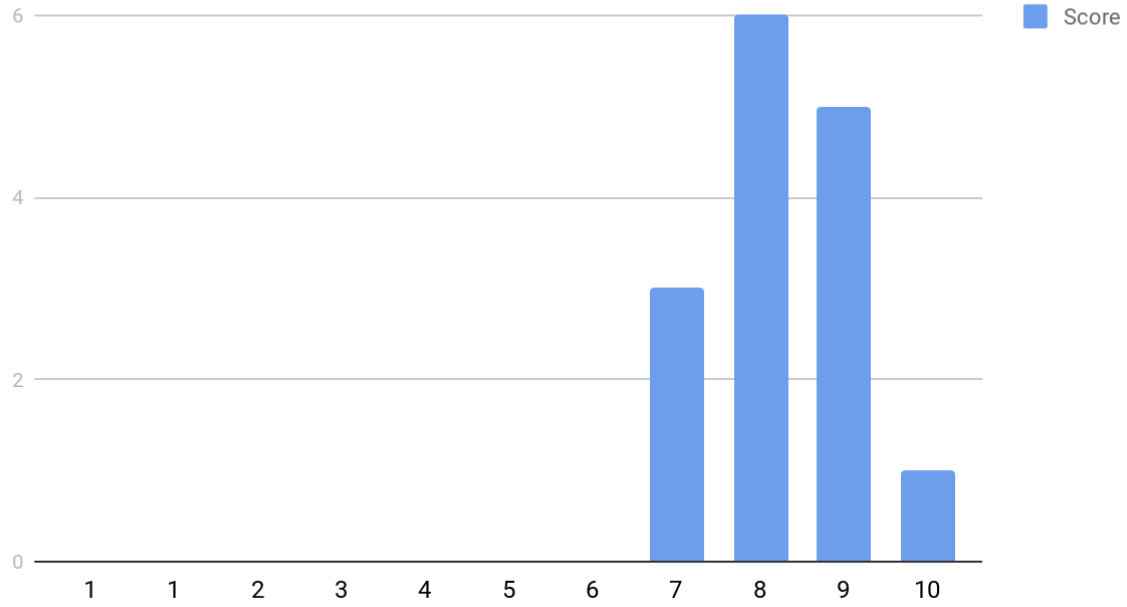


USER FEEDBACK RESULTS (CONT.)

Question 3

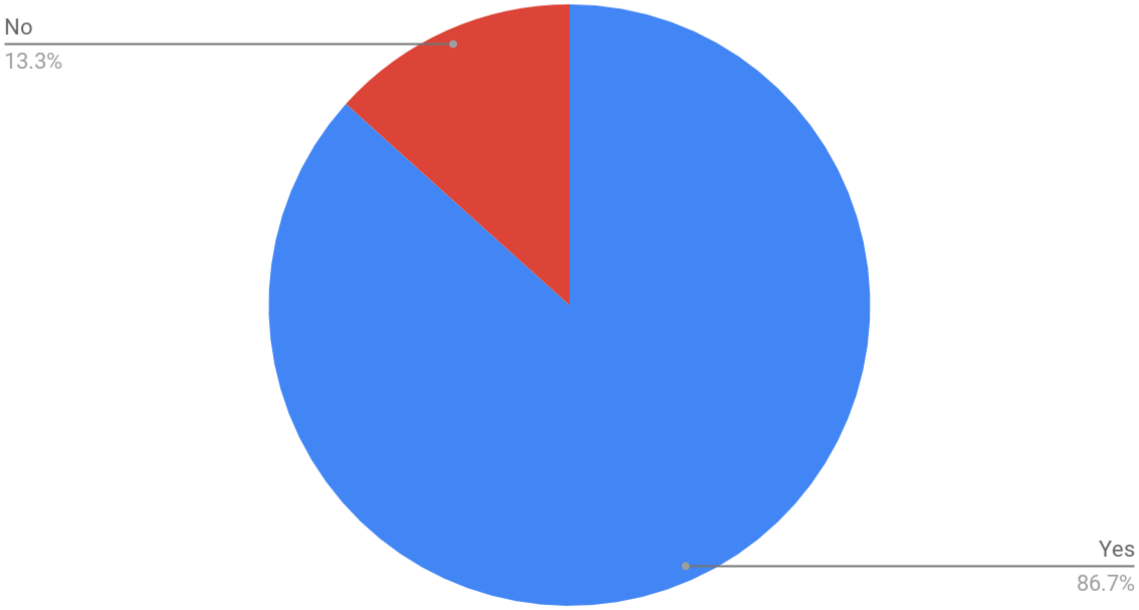


Question 4

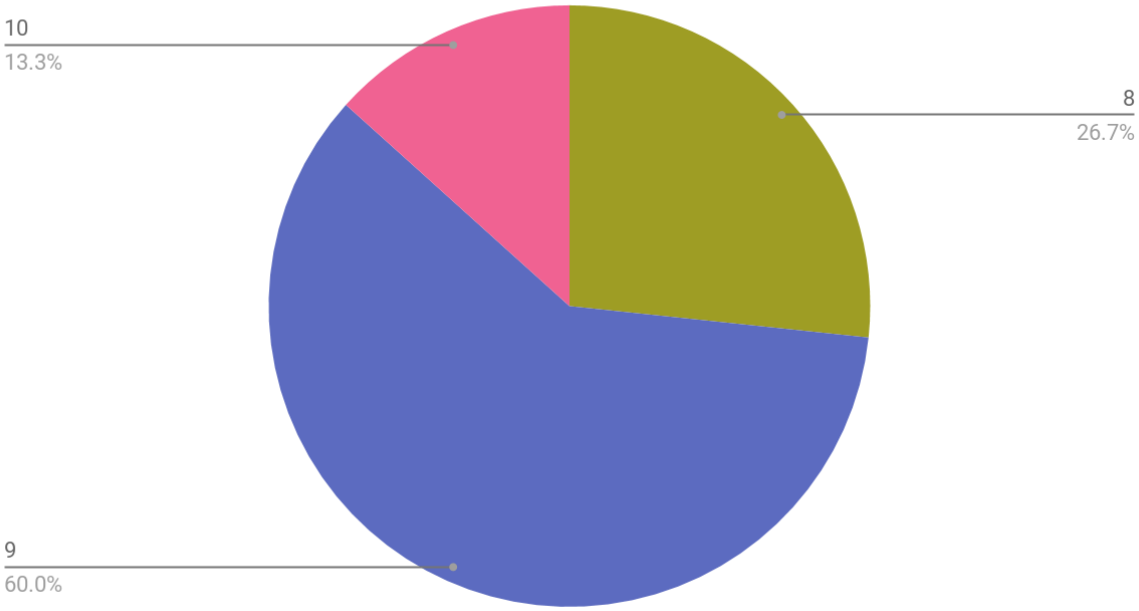


USER FEEDBACK RESULTS (CONT.)

Question 5



Question 6



USER FEEDBACK RESULTS (CONT.)

QUESTION 7

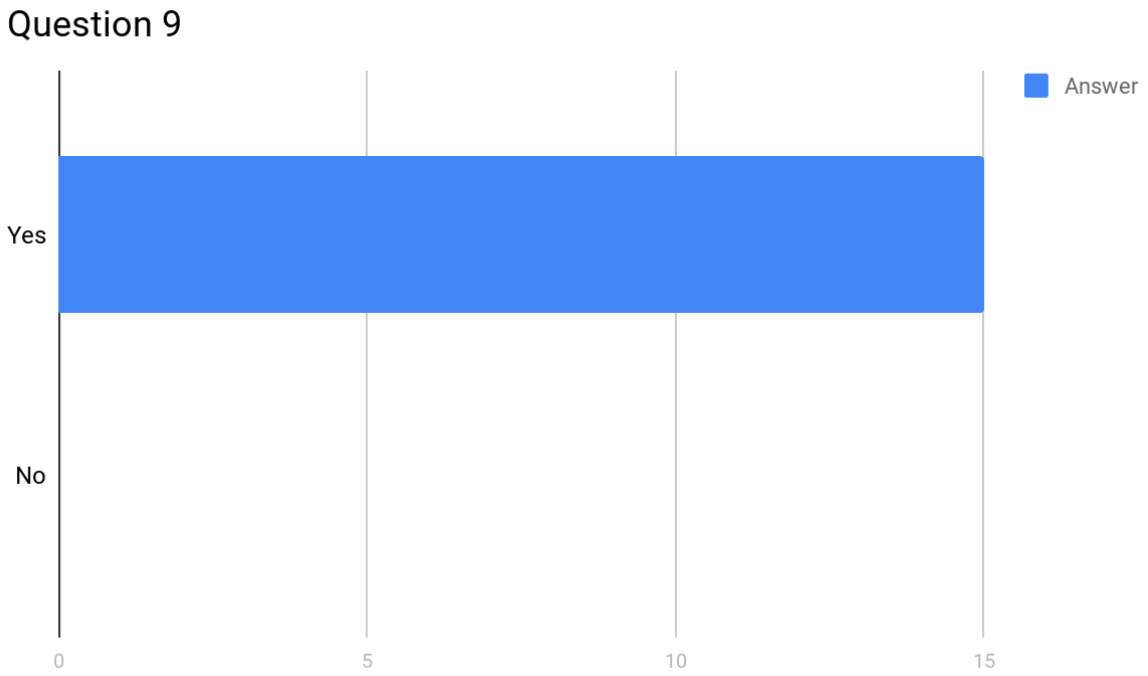
Responses included: Number of Responses:

User Creation Form	7
Selecting Shift Availability	8
Navigating the Web App	2

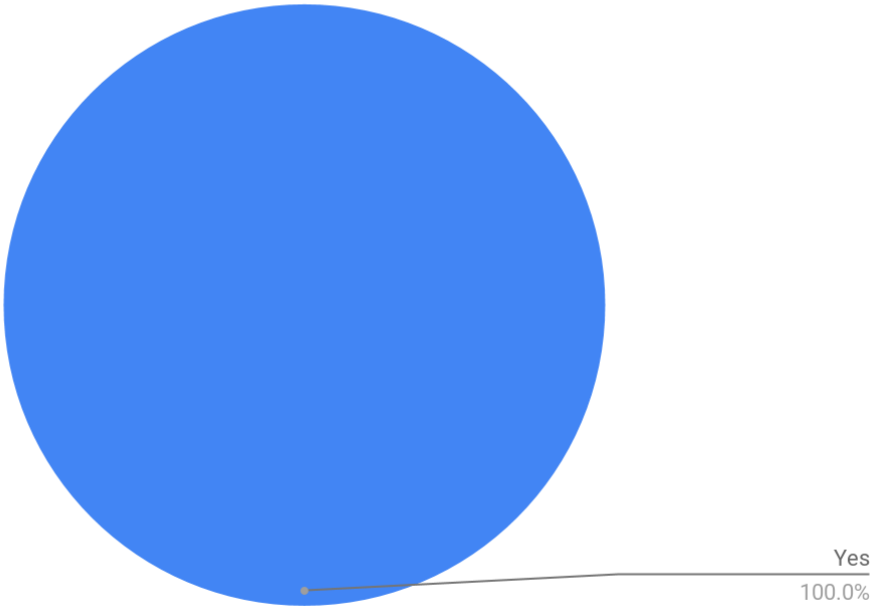
QUESTION 8

Responses included: Number of Responses:

Generating the Schedule	9
Deleting User Accounts	3
Viewing the Current Rota	2
Logging in	1

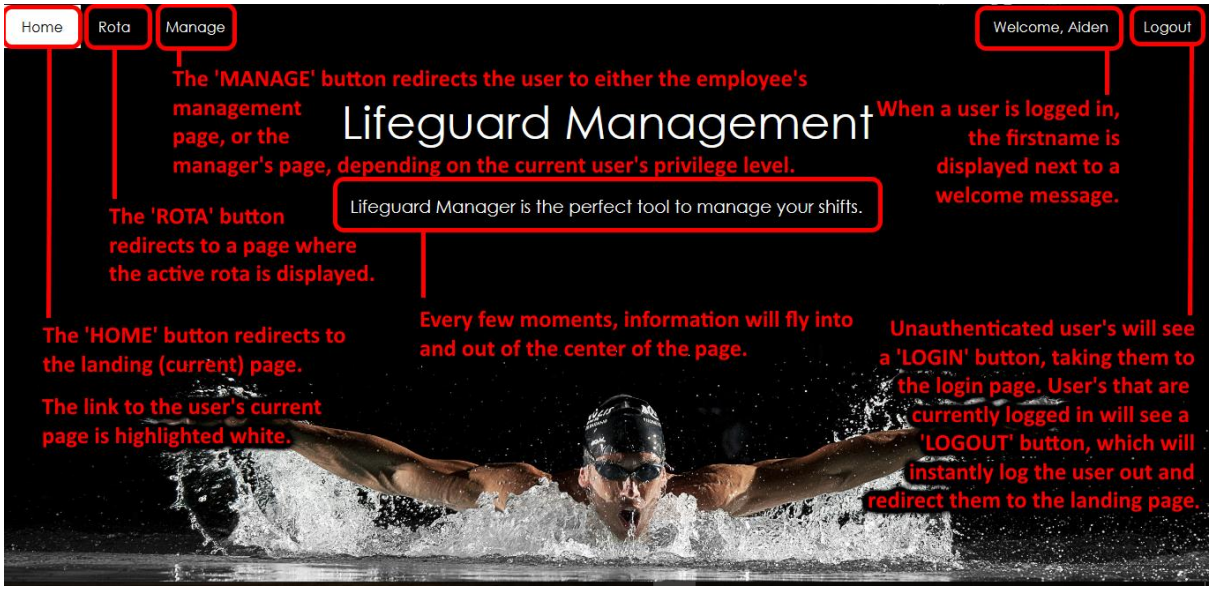


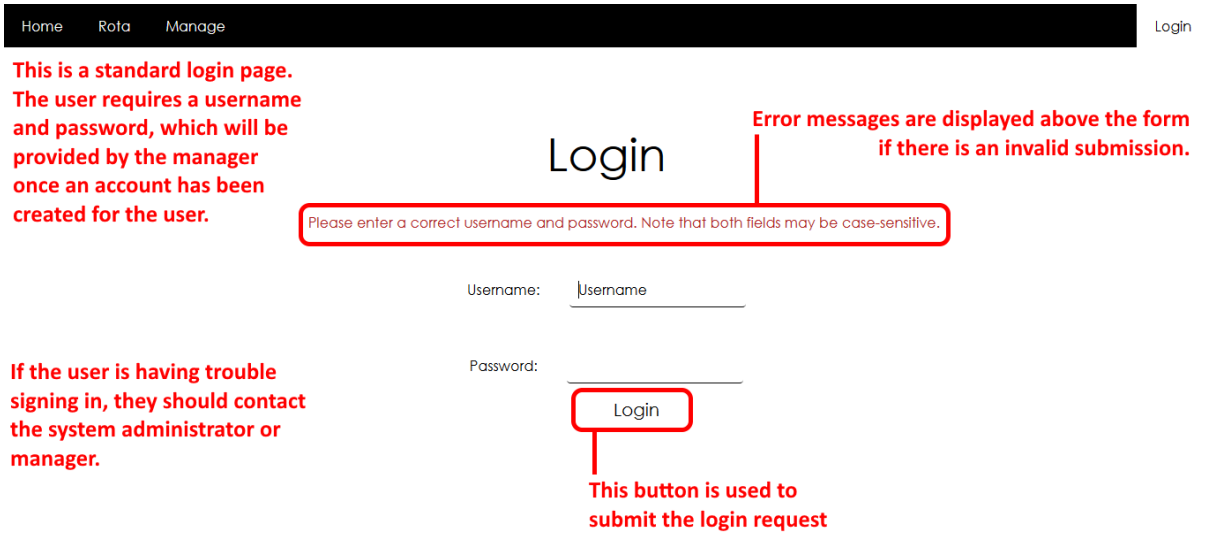
Question 10



The user feedback results show that overall, stakeholders are very pleased with the usability of the web application. No question with a response based on a scale contained a score lower than 7. Only 3 areas of the web app posed any sort of difficulties. These include the user creation form, selecting shift availability, and navigating the web app (although few people found this a problem). The general conversation stated that the question requires an answer even though the tasks many people answered were not actually that difficult. This is a positive result, and therefore the conclusion, based on user feedback can state that the usability features of the web application are more than satisfactory.

USER MANUAL (CONT.)





Home Rota Manage Welcome, Aiden Logout

After a successful login, the user is automatically redirected to this page where they receive an individual welcome message.

Welcome, Aiden

A useful tip is displayed to the user. This is helpful if the user is new to the system.

Where next?
Take a look at the Rota via the navigation bar above. Then you can make changes on the 'Manage' page.

Home Rota Manage Welcome, Aiden Logout

Next Month's Provisional Rota

Shift Times are displayed in 24hr format 'Start - End'

The dates are displayed with an abbreviated day name, day number, month, and year.

The rota is displayed as 5 different tables (one for every 7 days).

(This rota is subject to change. It is recommended that you check back frequently for any changes that may have occurred.)

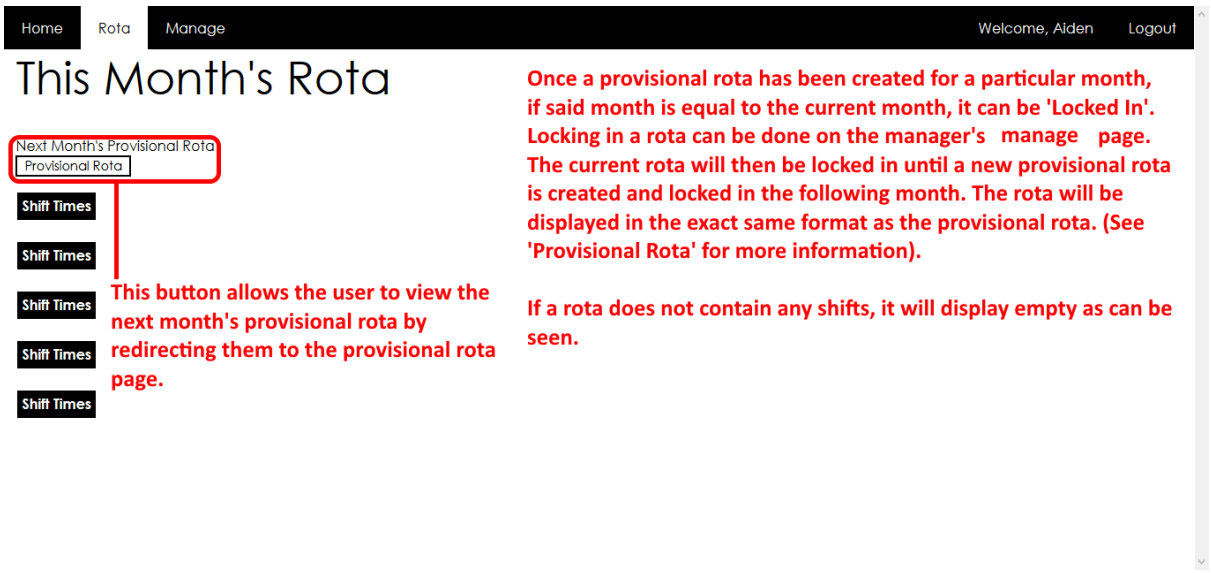
Shift Times	Tue 01 May 2018	Wed 02 May 2018	Thu 03 May 2018	Fri 04 May 2018	Sat 05 May 2018	Sun 06 May 2018	Mon 07 May 2018
09:00 - 14:00					NiamhB	HannahA	
10:30 - 15:00					IzzyC	JasminD	
12:00 - 18:00					HadyJ	ScottS	
15:00 - 19:00					ScottS	SamL	
09:00 - 12:00	HannahA	ScottS	GuyB	NiamhB			MillieL
15:00 - 18:00	BenD	JasminD	AidenU	JasminD			HannahA
16:30 - 20:00	IzzyC	BenD	NiamhB	SamL			ScottS

Employee assigned to the particular shift, username is displayed.

Shift Times	Tue 08 May 2018	Wed 09 May 2018	Thu 10 May 2018	Fri 11 May 2018	Sat 12 May 2018	Sun 13 May 2018	Mon 14 May 2018
09:00 - 14:00							
10:30 - 15:00							
12:00 - 18:00							
15:00 - 19:00							

Provisional rotas can be re-generated with no limit. Therefore, a message explains this fact, urging users to frequently check back.

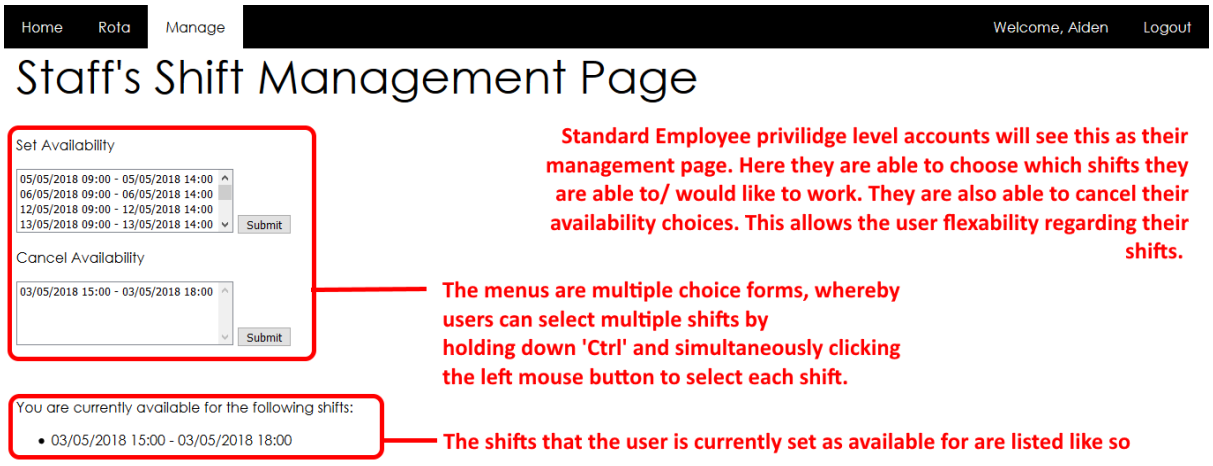
Shifts without employees are left as black cells. In the case where no shift exists, the table is also filled in with black cells.



This button allows the user to view the next month's provisional rota by redirecting them to the provisional rota page.

Once a provisional rota has been created for a particular month, if said month is equal to the current month, it can be 'Locked In'. Locking in a rota can be done on the manager's manage page. The current rota will then be locked in until a new provisional rota is created and locked in the following month. The rota will be displayed in the exact same format as the provisional rota. (See 'Provisional Rota' for more information).

If a rota does not contain any shifts, it will display empty as can be seen.

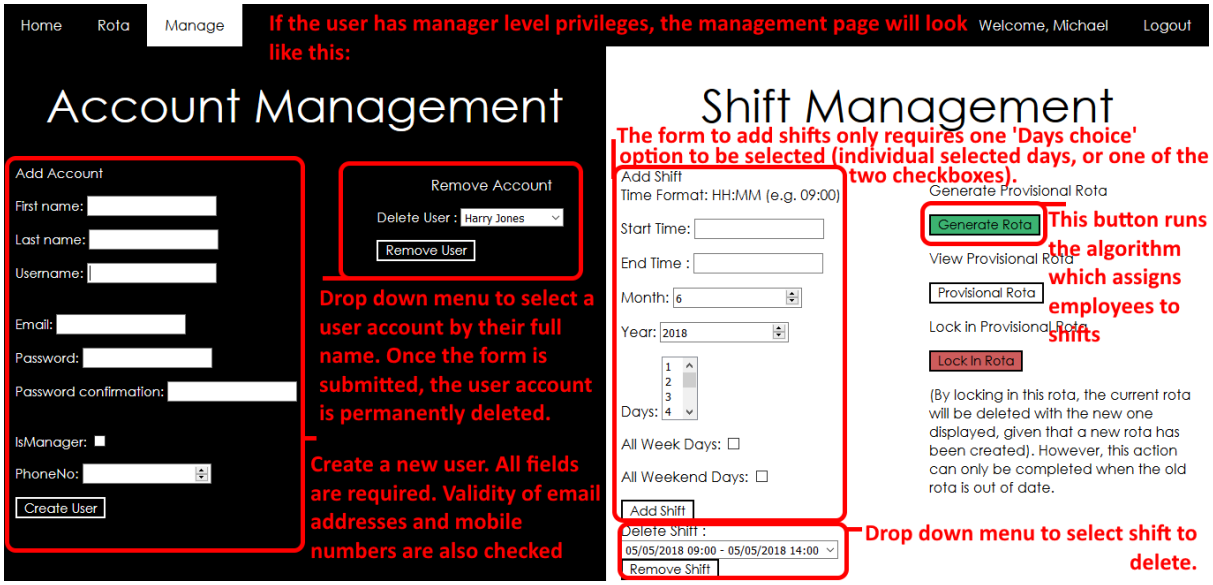


Standard Employee privilidge level accounts will see this as their management page. Here they are able to choose which shifts they are able to/ would like to work. They are also able to cancel their availability choices. This allows the user flexability regarding their shifts.

The menus are multiple choice forms, whereby users can select multiple shifts by holding down 'Ctrl' and simultaneously clicking the left mouse button to select each shift.

The shifts that the user is currently set as available for are listed like so

Users are unable to change their availability for the current provisional rota once it has been locked in and becomes the current rota.


Home
Rota
Manage
If the user has manager level privileges, the management page will look like this:
Welcome, Michael
Logout
Account Management
Add Account
First name:
Last name:
Username:
Email:
Password:
Password confirmation:
IsManager:
PhoneNo:
Create User
Remove Account
Delete User : Harry Jones
Remove User
Drop down menu to select a user account by their full name. Once the form is submitted, the user account is permanently deleted.
Create a new user. All fields are required. Validity of email addresses and mobile numbers are also checked
Shift Management
The form to add shifts only requires one 'Days choice' option to be selected (individual selected days, or one of the two checkboxes).
Add Shift
Time Format: HH:MM (e.g. 09:00)
Start Time:
End Time :
Month: 6
Year: 2018
Days: 1 2 3 4
All Week Days:
All Weekend Days:
Add Shift
Delete Shift :
05/05/2018 09:00 - 05/05/2018 14:00
Remove Shift
Drop down menu to select shift to delete.
Generate Provisional Rota
Generate Rota
This button runs the algorithm which assigns employees to shifts
View Provisional Rota
Provisional Rota
Lock in Provisional Rota
Lock In Rota
(By locking in this rota, the current rota will be deleted with the new one displayed, given that a new rota has been created). However, this action can only be completed when the old rota is out of date.

EVALUATION AGAINST SUCCESS CRITERIA

DATABASE

Status	Success Criterion	Comment
✅	The database should only include data that is relevant and used within the program in order to keep the database size to a minimum.	The database design phase ensured that only relevant data is stored within the database.
✅	The database tables should not allow for data redundancy and should contain features that maintain data integrity such as Django 'Cascading delete' feature.	Data integrity is ensured through Django itself, and linked records are deleted.
✅	The data should be protected from malicious users damaging the database through security flaws such as SQL injections. (Validate all user inputs)	All user inputs are validated through typical Django methodologies such as the use of Cross Site Request Forgery tokens.
✅	If a many-to-many relationship is required, Django's 'Many-To-Many' field feature should be called so that a linking table can be automatically generated and managed.	This is only required once when linking an Employee's availability to the Shifts model.
✅	The database used in this development project should be easily migrated to a more secure type of object relational database system such as PostgreSQL.	Sqlite3 was used in the development of this project, however it is easily migrated to a different database type such as PostgreSQL by following the method suggested in the following post: https://gist.github.com/sirodoht/f598d14e9644e2d3909629a41e3522ad

EVALUATION AGAINST SUCCESS CRITERIA (CONT.)

GRAPHICAL USER INTERFACE

MANAGEMENT SPECIFIC FEATURES:

Status	Success Criterion	Comment
☑	The program should allow for managers to add employees to the database It should display this feature as a form, with fields to store data regarding: Employee's first name, last name, password, email and phone number. There should be an option to add this user with the manager account privilege.	The application contains a form to add new users to the database. The data stored includes Employee's first name, last name, password, email and phone number. There is also a Boolean option to make the new employee a manager.
☑	The program should allow for managers to delete employees and all related records from the database. It should display all the current employee level accounts in a drop-down menu, in which the user is able to select one and submit their choice.	All related records are deleted accordingly to maintain integrity. A drop-down menu is used to select a user's account to delete. Once submitted, the account is deleted.
☑	The program should allow managers to add shifts they want to be filled for the current or following month It should be a forum, displayed with compulsory options to set the start and finish time of the shift, the month and year the shift will be implemented for, and a field to select which days of the month the shift will be implemented for.	Shifts can be added through a user-friendly form in the managers management page. Through usability tests, the ease of data entry scored between 8 and 10, therefore it can be assumed that this form is close to optimal in terms of efficiency of adding shifts.
☑	The program should accommodate for deleting shifts for the following month. The feature should display all the shifts that have been set for next month's rota as a drop-down menu as choices of shifts to delete, with the user then able to submit once selected. The shift will then be removed from the database.	Shifts can be easily deleted through a drop-down menu located underneath the shift creation form. Once the shift is selected and submitted- it is deleted from the database.

EVALUATION AGAINST SUCCESS CRITERIA (CONT.)

GRAPHICAL USER INTERFACE (CONT.)

MANAGEMENT SPECIFIC FEATURES:

Status	Success Criterion	Comment
✓	The program should include a button to lock in next month's rota manually, however this should only happen after a certain number of days into the current month (yet undecided). Once the schedule is locked in, changes will no longer be made regarding completely re-generating the schedule.	This feature was implemented. The decision was made to only allow the provisional rota to be locked once the current month was the same as the one displayed in the provisional rota. This prevents all changes being made to the current rota.
✓	The program should include a button on the management page to go directly to view the next month's rota, this is for convenience and usability.	The button has been fully implemented. This usability feature contributed to a very positive user feedback score because it saved time navigating a different path through the web app.
✗	The program should allow the managers to manually change the generated schedules in case of unforeseen circumstances such as staff sickness.	Although this is a feature that would be useful in the day-to-day running of the application, it was very complicated and time consuming to build and so was not included in the final prototype.
✓	The program should display a button to generate a new schedule, which is a different colour to the other buttons so to highlight the importance of the feature and prevent miss-clicks. The button should call a function that runs the core scheduling algorithm.	This button has been fully implemented, and successfully calls the core scheduling algorithm.

EVALUATION AGAINST SUCCESS CRITERIA (CONT.)

GRAPHICAL USER INTERFACE (CONT.)

EMPLOYEE SPECIFIC FEATURES:

Status	Success Criterion	Comment
✓	The program should allow the users to set their availability for the following month. This should be displayed as a form with one choice field that allows multiple shifts to be selected by the user.	This form has been fully implemented. It uses a built in Django 'multiple choice' form which allows multiple shifts to be selected at once. This increases the usability.
✓	The program should allow users to cancel their availability for the following month. This should be displayed as a form with one choice field that allows multiple shifts to be selected by the user.	This form has been fully implemented. It uses a built in Django 'multiple choice' form which allows multiple shifts to be selected at once. This increases the usability.
✓	The program should display the shifts that the logged in user is available for, in a list that is easily read.	This feature has been included. It lists each shift as a new bullet point on the Employee management page.

ALL ACCOUNT HOLDER FEATURES:

Status	Success Criterion	Comment
✓	When the user has logged in, they should be greeted by a welcome page that makes the user experience more customised by displaying their name.	A login success page was created which all users are redirected to after a successful login. A welcoming message is tailored to the user.
✓	The users should be able to view the rota of the current month. The rota should be displayed in a timetable format, that is easily followed and read.	A specific page displays the current month's rota in a way that closely mirrors the client's current rota layout. This will make the transition easier.

EVALUATION AGAINST SUCCESS CRITERIA (CONT.)

GRAPHICAL USER INTERFACE (CONT.)

ALL ACCOUNT HOLDER FEATURES:

Status	Success Criterion	Comment
✓	The users should be able to view the rota of the following month The rota should be displayed in a timetable format, that is easily followed and read.	A provisional rota is displayed on a designated page. It is displayed in the same format as the current month's rota. This prevents confusion.

NON-ACCOUNT HOLDER FEATURES:

Status	Success Criterion	Comment
✓	The user should not have access to the management pages, or access to the rota pages.	All management pages, rota pages, and administrator pages are locked for non-account holders.
✓	The user should have access to the login page via the navigation bar.	Only a non-authenticated user has access to the login page.

ALL ACCOUNT PRIVILEGE LEVEL FEATURES:

Status	Success Criterion	Comment
✓	All pages should contain the navigation bar. The navigation bar should contain links to all relevant pages such as the management page, current rota page and home page. The navigation bar should also display a customised 'Welcome, **USERNAME**' message. The navigation bar should also contain a 'Logout' button, which when pressed logs out the user and redirects them to the homepage.	Every page contains a navigation bar, the exception being the Django administrator interface. The navigation bar contains links to the management pages, current rota page, and home page. Authenticated users: The navigation bar contains a customised welcome message. It also contains a functional logout button.

EVALUATION AGAINST SUCCESS CRITERIA (CONT.)

GRAPHICAL USER INTERFACE (CONT.)

ALL ACCOUNT PRIVILEGE LEVEL FEATURES:

Status	Success Criterion	Comment
☑	The user should be able to view the homepage. The homepage should contain a little information regarding the program.	Every user has access to the home page. The homepage also contains a feature which displays information regarding the program.
☑	The GUI design should be deemed contemporary, intuitive, and easily navigated.	The interface has been deemed contemporary and easily navigated by a stakeholder survey.

EVALUATION AGAINST SUCCESS CRITERIA (CONT.)

CORE ALGORITHM

Status	Success Criterion	Comment
✅	The core algorithm should consider all available employees and produce an acceptable schedule. An acceptable schedule is one that results in > 90% of available employees getting at least one shift per week, with 100% of the required shifts filled by the algorithm, given that an employee is available to work it.	The initial testing suggests this is the case. The algorithm was designed in such a way that given that an employee is available to work it, each shift will have an employee associated to it.
✅	The core algorithm should result in a fair schedule in terms of the number of shifts each employee is to work.	The algorithm aims to balance the distribution of shifts based on the number of shifts each lifeguard is available to work. This was the main objective and condition of the algorithm.
✅	The algorithm's runtime is not an issue, and so the only constraint here is that it is generated in under 5 minutes for a full set of test data that represents that of normal operational use. However, the worst-case time complexity of the project should not be equal to or exceed exponential time O(2^n).	Question 10's results suggest that the program runs much quicker than the 5-minute limitation. The worst-case time complexity of any function within the web application is O(n^2), which is much better than the exponential limitation.
✅	The algorithm should allow room for the team to grow in numbers, and therefore be developed in such a way so that it is able to work with an unlimited number of shifts and employees.	No software limitation is imposed or exists; therefore the scalability of the product should be maintained. The only limiting factor is the hardware, whereby the server should have enough free storage to handle the database. Although this should not be a problem because the database is fully normalised.

EVALUATIVE TESTING

KEY FEATURE: ADD ACCOUNT FORM

Test No.	Type	Description	Input	Expected Output	Actual Output	Comment
1.0	Normal	'Add Account' form testing a normal data set to check that the employee and linked user account objects are created and saved in the database.	'Harry', 'Jones', 'HJones', 'Hjones@wentworthlifeguards.com', 'Password 1234', 'Password 1234', '07363827364'	A new linked user and employee object should be created and visible in the database.	A new linked user and employee object is created and visible in the database.	Fully working. See screenshot 1.0
1.1	Erroneous	'Add Account' form testing that empty fields are not valid inputs.	Nothing	Empty fields highlighted with a prompting message.	Empty fields are highlighted and display a prompting message.	Fully working. See screenshot 1.1
1.2	Erroneous	'Add Account' form testing that an invalid length phone number prevents the user and employee objects being created and saved in the database.	'John', 'Jones', 'JJones', 'Jjones@wentworthlifeguards.com', 'Password 1234', 'Password 1234', '1'	The page should refresh, clearing the data, with no changes made to the database due to the error.	The page refreshes, clearing the data, with no changes made to the database.	Fully working.

EVALUATIVE TESTING (CONT.)

KEY FEATURE: ADD ACCOUNT FORM

Test No.	Type	Description	Input	Expected Output	Actual Output	Comment
1.3	Boundary	'Add Account' form testing that a valid, but boundary case phone number does not prevent the user and employee objects being created and saved in the database.	'Carl', 'Jones', 'CJones', 'Cjones@wentworthlifeguards.com', 'Password 1234', 'Password 1234', '07999999999'	The account should be created and saved successfully	The account is created and saved successfully	Working as expected. See Screenshot 1.3

KEY FEATURE: REMOVE ACCOUNT FORM

Test No.	Type	Description	Input	Expected Output	Actual Output	Comment
2.0	Normal	'Remove Account' from testing to check that if a user's account is selected from the dropdown menu and the form is submitted, the account and associated records are deleted from the database.	'CJones' is selected from the dropdown menu.	The account is successfully deleted, with no trace of any linked records.	The account is successfully deleted, with no trace of any linked records.	Fully working feature. See Screenshot 2.0

EVALUATIVE TESTING (CONT.)

KEY FEATURE: REMOVE ACCOUNT FORM

Test No.	Type	Description	Input	Expected Output	Actual Output	Comment
2.1	Boundary	'Remove Account' from testing to check that the last user on the list can be successful deleted	'CJones' is selected from the dropdown menu.	The account is successfully deleted, with no trace of any linked records.	The account is successfully deleted, with no trace of any linked records.	Fully working. See Screenshot 2.0

KEY FEATURE: ADD SHIFT FORM

Test No.	Type	Description	Input	Expected Output	Actual Output	Comment
3.0	Normal	'Add Shift' form testing to check that an expected dataset is dealt with successfully and a shift object is saved into the database.	'09:00', '14:00', '5', '2018', '10'	The shift object is created and saved in the database. with the expected datetime object associated with the start and end fields.	The shift object is created and saved in the database. with the expected datetime object associated with the start and end fields.	This feature works as expected, see Screenshot 3.0
3.1	Normal	'Add Shift' form testing to check that an expected dataset is dealt with successfully and multiple shift objects are saved into the database.	'16:30', '20:00', '5', '2018', 'All Weekdays is checked'	For each weekday in the month selected, a shift object is created with the start and end times specified.	For each weekday in the month selected, a shift object is created with the start and end times specified.	This feature works as expected. See Screenshot 3.1

EVALUATIVE TESTING (CONT.)

KEY FEATURE: ADD SHIFT FORM

Test No.	Type	Description	Input	Expected Output	Actual Output	Comment
3.2	Normal	'Add Shift' form testing to check that an expected dataset is dealt with successfully and multiple shift objects are saved into the database.	'09:00', '14:00', '5', '2018', 'All Weekends is checked'	For each weekend day of the month selected, a shift object is created with the start and end times specified.	For each weekend day of the month selected, a shift object is created with the start and end times specified.	This feature works as expected. See Screenshot 3.2
3.3	Erroneous	'Add Shift' form testing to check that an unexpected dataset is dealt with successfully and multiple shift objects are saved into the database.	'09:00', '14:00', '5', '2018', 'All Weekends is checked' 'All Weekdays is checked'	For each weekday of the month selected, a shift object is created with the start and end times specified.	For each weekday of the month selected, a shift object is created with the start and end times specified.	This feature, although not providing optimal usability, deals with the conflict as expected. See screenshot 3.3
3.4	Erroneous	'Add Shift' form testing that an invalid form submission does not result in any changes made in the database.	'Nine O'clock', '14:00', '5', '2018', 'All Weekends is checked'	The page should refresh, clearing the data, with no changes made to the database due to the error.	The page refreshes, clearing the data, with no changes made to the database due to the invalid form submission.	Fully working. See Screenshot 3.4

EVALUATIVE TESTING (CONT.)

KEY FEATURE: ADD SHIFT FORM

Test No.	Type	Description	Input	Expected Output	Actual Output	Comment
3.5	Erroneous	'Add Shift' form testing that an invalid form submission does not result in any changes made in the database.	'09:00', '14:00', '7', '2018', '5'	The form should not allow an input of a month that is greater than or equal to 2 more than the current month. It should alert the user by highlighting the erroneous field in red and display a message when hovering over it.	The form does not allow submission. A message is displayed when the mouse is hovered over the field, and the field is highlighted red.	This validation works well. See screenshot 3.5

KEY FEATURE: GENERATE PROVISIONAL ROTA BUTTON

Test No.	Type	Description	Input	Expected Output	Actual Output	Comment
4.0	Normal	'Generate Provisional Rota Button' testing to ensure that when clicked, the core algorithm runs.	Button pressed 'Generate Rota'	The page should refresh, with each shift assigned to one employee.	The page refreshed, with each shift assigned to one employee. The provisional rota page is used as evidence.	Fully working, please see Screenshot 4.0

SUCCESS OF THE SOLUTION

Based on the evaluation against the success criteria, there is only one area which has not been fully met. This is within the graphical user interface development.

Criterion:
'The program should allow the managers to manually change the generated schedules in case of unforeseen circumstances such as staff sickness.'
Comment:
Although this is a feature that would be useful in the day-to-day running of the application, it was very complicated and time consuming to build and so was not included in the final prototype.

The reason for not meeting this success criteria is simply the fact that the development process, despite running smoothly, took much longer than initially expected. Therefore, the success criteria were prematurely agreed upon before deciding whether including every single feature within the timeframe was feasible. However, it was not a large misjudgement, because every other agreed upon feature was included- making this the only failed criterion.

When coming to a judgement as to the overall success of the project, it is clear from all but one of the success criteria that the project was very successful. All other 27 criteria were met in full, without any compromises. The client will therefore most likely gladly accept this solution.

SUGGESTIONS FOR FURTHER DEVELOPMENT

Based on the user feedback and evaluation, there are few features that should be further developed. However notably, users found the default multiple choice fields difficult to operate. and a slow process because of the number of options e.g. selecting availability of shifts. Based on this information, I would suggest a method for the user to filter shifts by searching for beginning / end times, or dates.

Equally, for many, a mobile app would also be very valuable as guaranteeing access to a desktop/ laptop can sometimes be troublesome. However, most people now own a smartphone, which means that employees and managers would then be able to access the application whenever best suits them, or when they have spare time e.g. waiting for public transport. In theory, accessing the web application on a mobile device is possible, however it would not render properly as the HTML does not consider this type of user as of yet.

The user feedback also suggested that the rota could be colour coded so that it is easy for users to distinguish which shifts still require employees. This is because currently, the table is such that shifts that do not exist are displayed as black cells, so the table renders properly, however this causes confusion to which shifts are still available to fill. The solution to this problem would take some consideration but is a simple objective of filling the cells that do not correspond to a shift with a white background colour.

SUGGESTIONS FOR FURTHER DEVELOPMENT (CONT.)

Other features that could be added in the future include the ability to request cover for active shifts and adding shift notes so that special events can be highlighted to the employees that do the shift. An example would be a note stating that a party will be hosted throughout the shift.

Overall, the core features of the product are in place, however there is still plenty of room for further development should the client require it. Since the project is written in a language and framework that are well known, growing in popularity, and have outstanding documentation, the client will not have a problem outsourcing further development work should it come to it.

MAINTENANCE

The code (see attached at the end of this document) is consistent in terms of conventions.

It follows the pythonic guidelines of:

- Each line containing less than 120 characters,
- The CamelCase naming convention is followed in most cases,
- 2 blank lines separate every function or class definition,
- Indentation is followed precisely (a requirement),
- Trailing Commas are kept to a minimum
- Comments and documentation strings are not overused
- Explicit relative imports are used when dealing with user defined modules (otherwise absolute imports are used). All imports are located at the top of the document (apart from one exceptional circumstance)
- Variable names are kept descriptive but short for ease of use and readability

Of course, the code will contain areas that do not follow the PEP-8 style to the book. The code could be refactored; however, the majority of the code is very readable and therefore at this present time, a complete refactor is welcome but unnecessary.

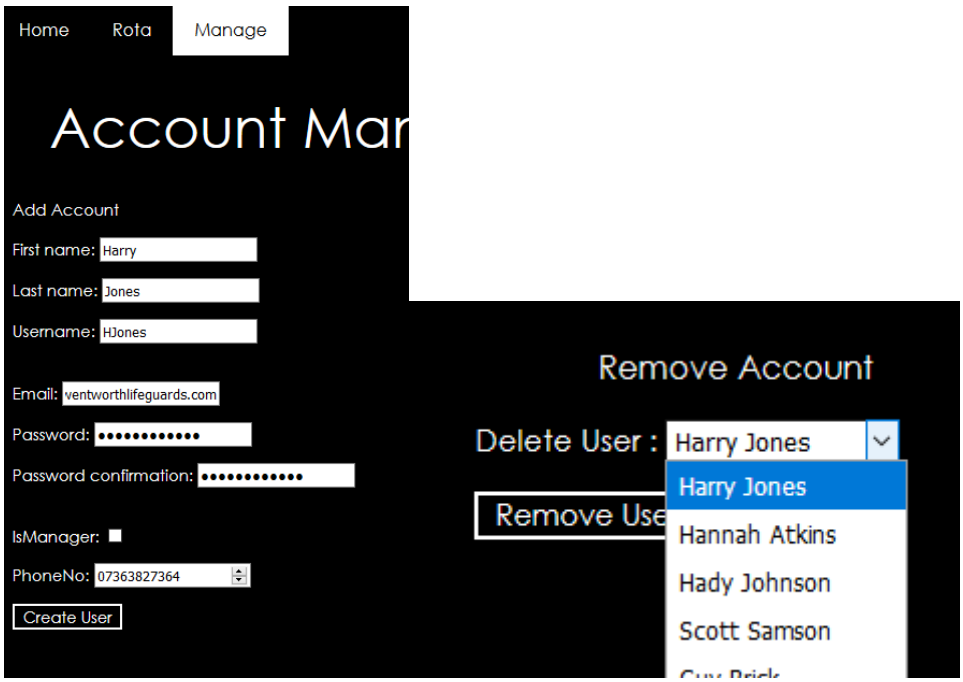
This documentation provides analysis of all key features present within the program. This is very useful information to anyone who takes on the role of maintaining the program because it greatly helps in understanding the processes. It is highly recommended that my client's IT department reads the design and development sections preceding this evaluation.

The code has no repetition of key code, no redundant code, and follows the design which has previously considered O notation of solutions. These are all components required to evaluate whether the code is efficient or not. Since these conditions have been met, the code can be considered efficient. Efficient code is important when regarding maintainability because it means that there are no unnecessary parts of code that require maintenance.

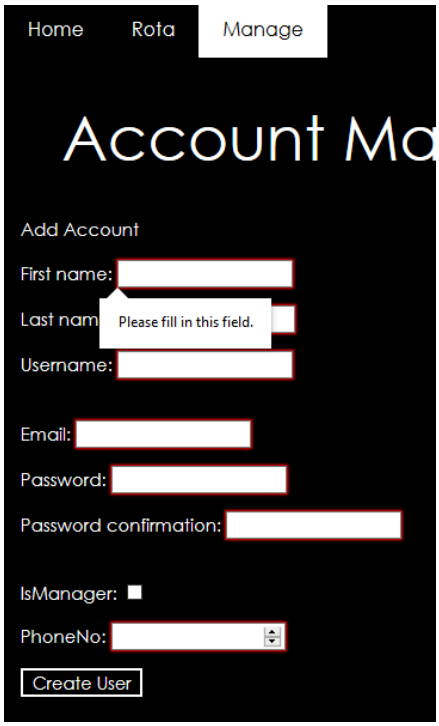
Readability is ensured not only by convention as stated above, however also by following a modular approach- grouping key processes into functions and classes. For example, this includes importing the core algorithm into the views to keep the views file as clean and readable as possible.

EVALUATIVE TESTING SCREENSHOTS

SCREENSHOT 1.0

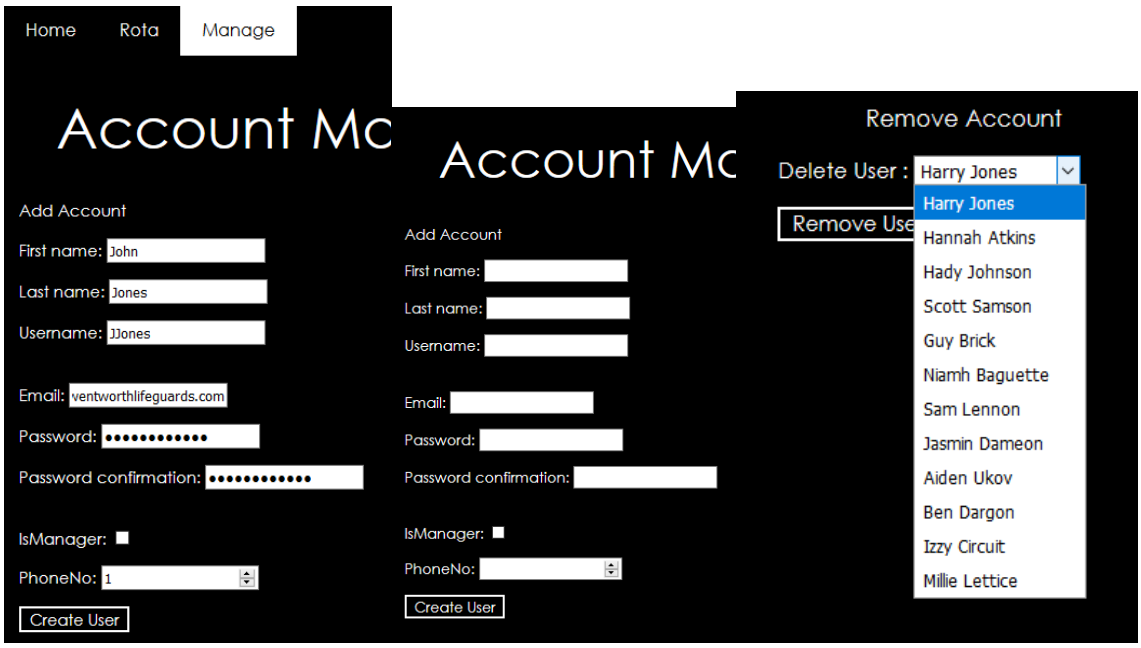


SCREENSHOT 1.1

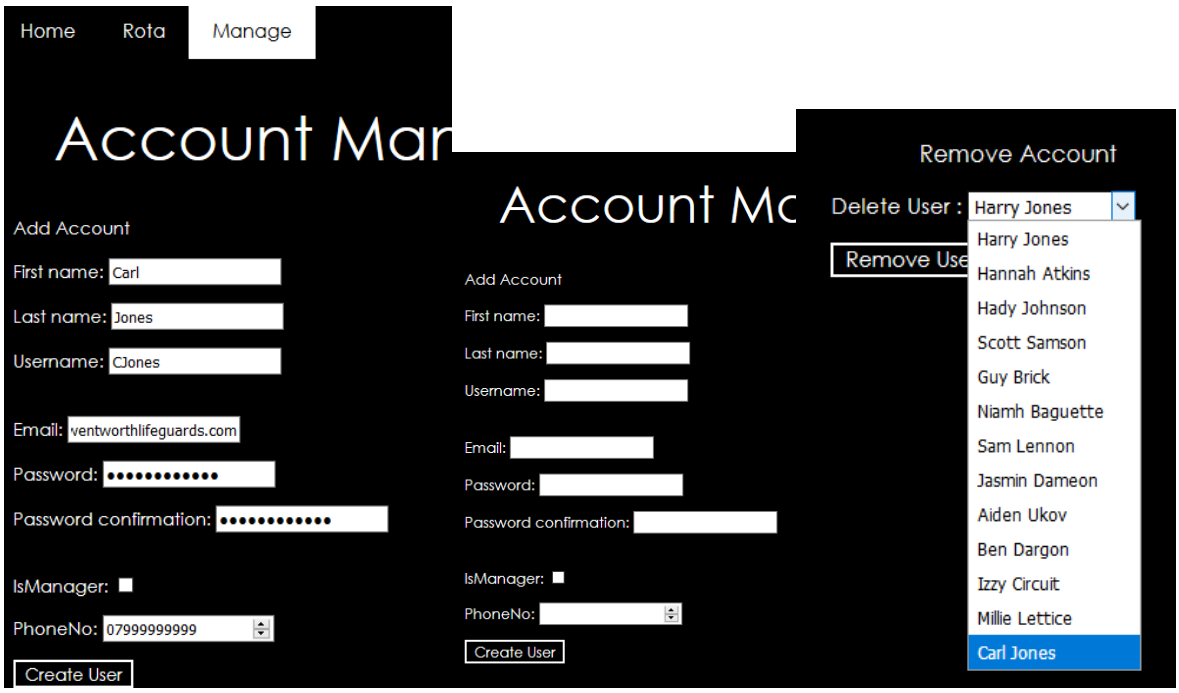


EVALUATIVE TESTING SCREENSHOTS (CONT.)

SCREENSHOT 1.2

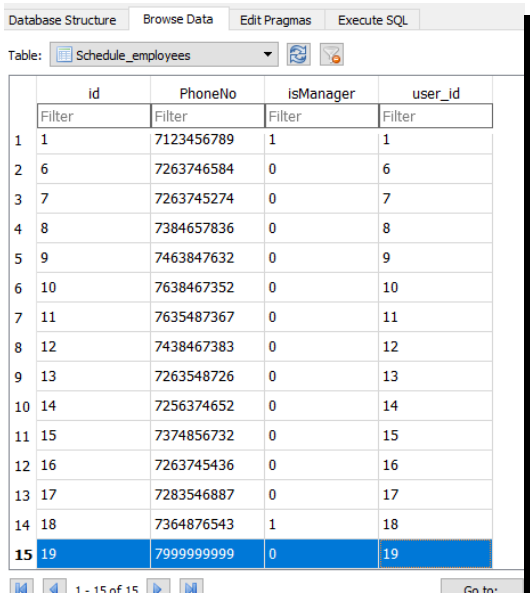


SCREENSHOT 1.3

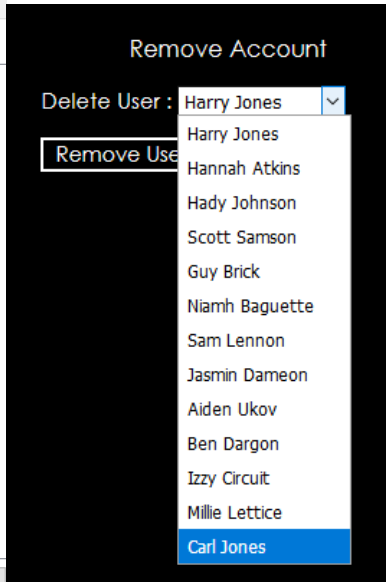


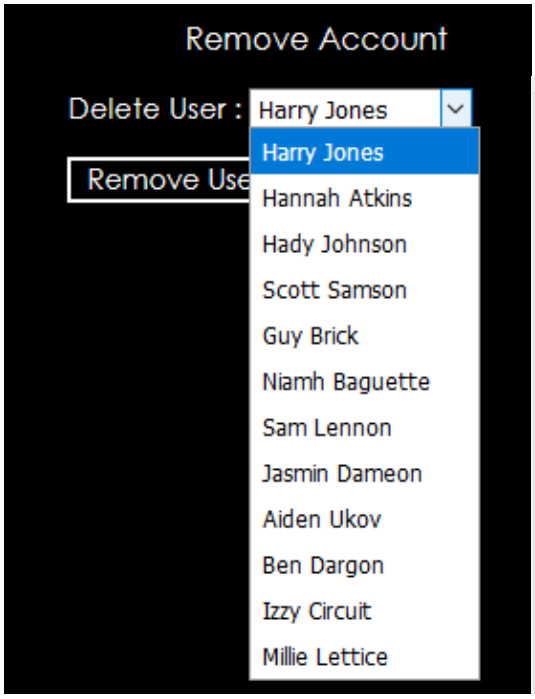
EVALUATIVE TESTING SCREENSHOTS (CONT.)

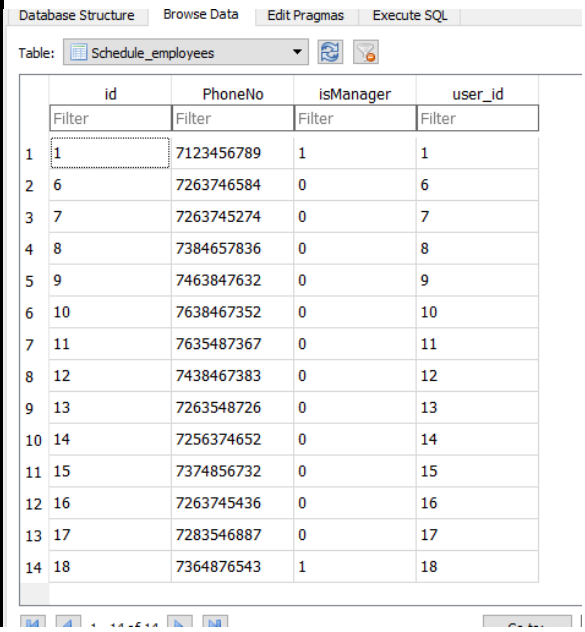
SCREENSHOT 2.0



	id	PhoneNo	isManager	user_id
	Filter	Filter	Filter	Filter
1	1	7123456789	1	1
2	6	7263746584	0	6
3	7	7263745274	0	7
4	8	7384657836	0	8
5	9	7463847632	0	9
6	10	7638467352	0	10
7	11	7635487367	0	11
8	12	7438467383	0	12
9	13	7263548726	0	13
10	14	7256374652	0	14
11	15	7374856732	0	15
12	16	7263745436	0	16
13	17	7283546887	0	17
14	18	7364876543	1	18
15	19	7999999999	0	19

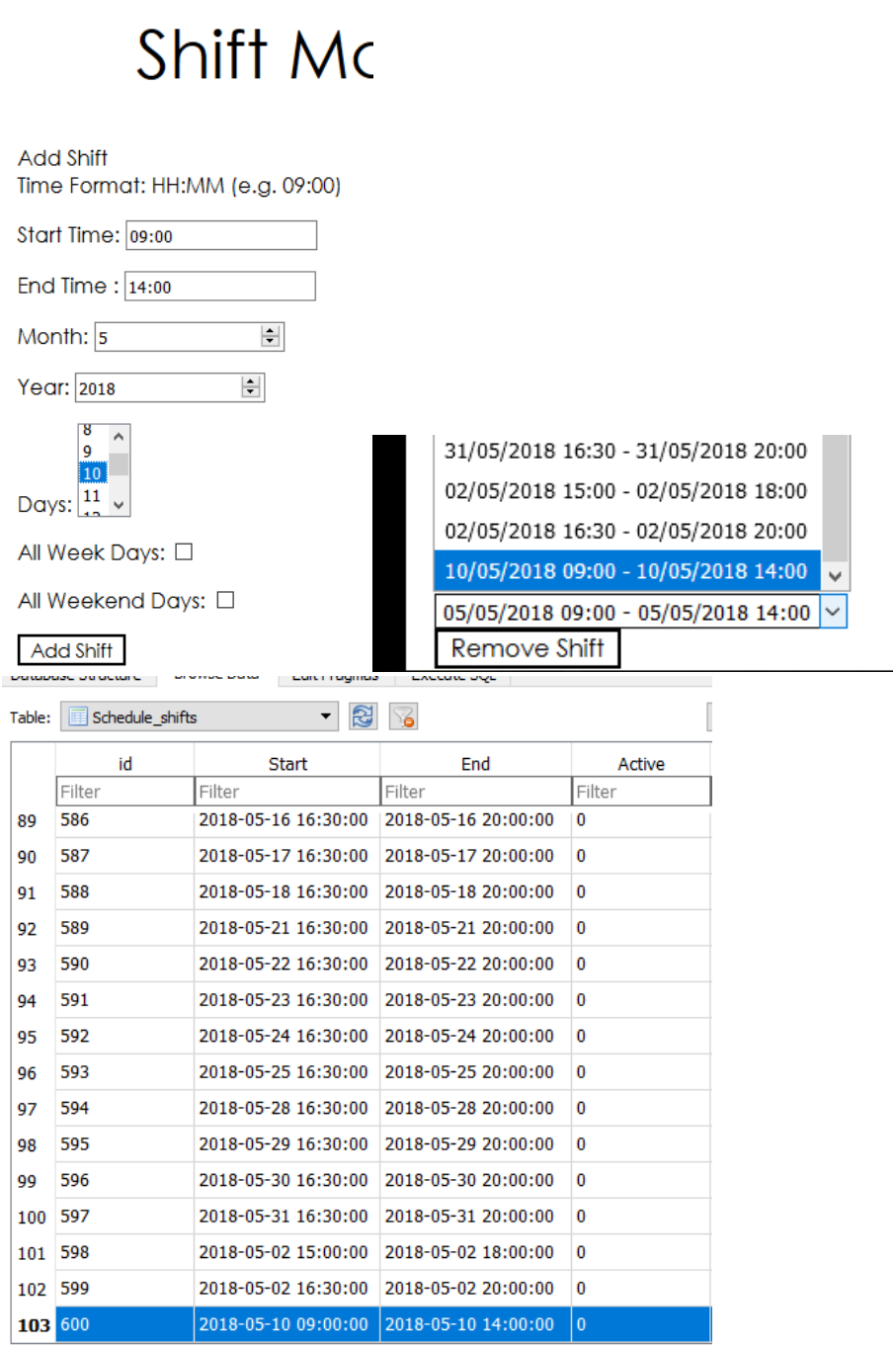






	id	PhoneNo	isManager	user_id
	Filter	Filter	Filter	Filter
1	1	7123456789	1	1
2	6	7263746584	0	6
3	7	7263745274	0	7
4	8	7384657836	0	8
5	9	7463847632	0	9
6	10	7638467352	0	10
7	11	7635487367	0	11
8	12	7438467383	0	12
9	13	7263548726	0	13
10	14	7256374652	0	14
11	15	7374856732	0	15
12	16	7263745436	0	16
13	17	7283546887	0	17
14	18	7364876543	1	18

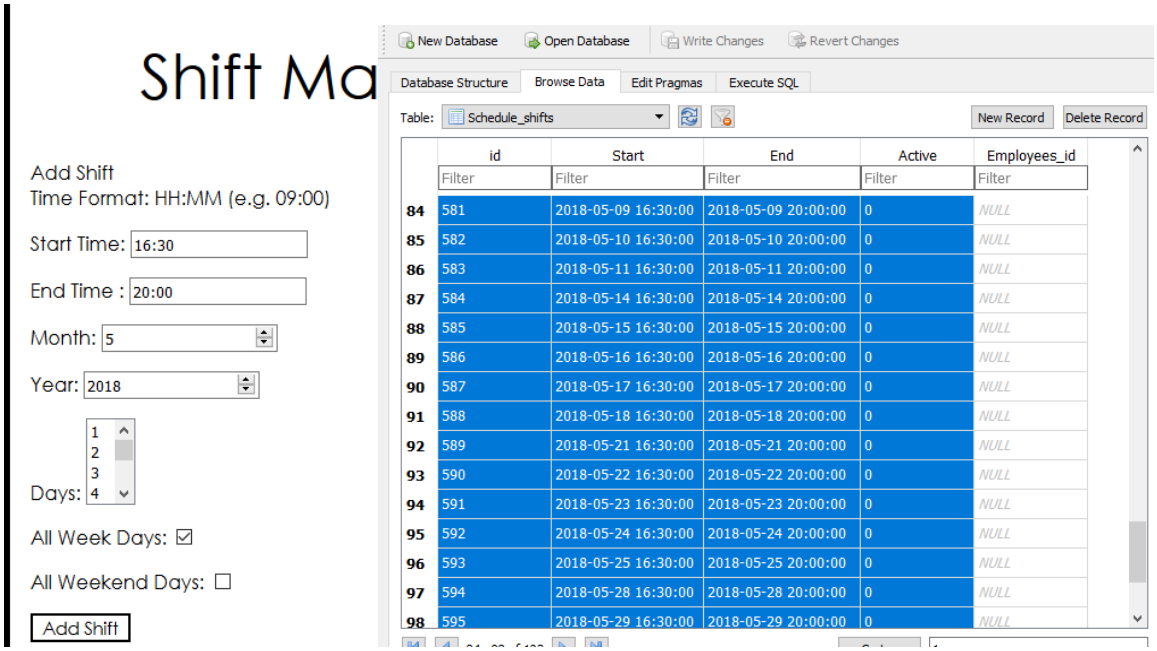
SCREENSHOT 3.0



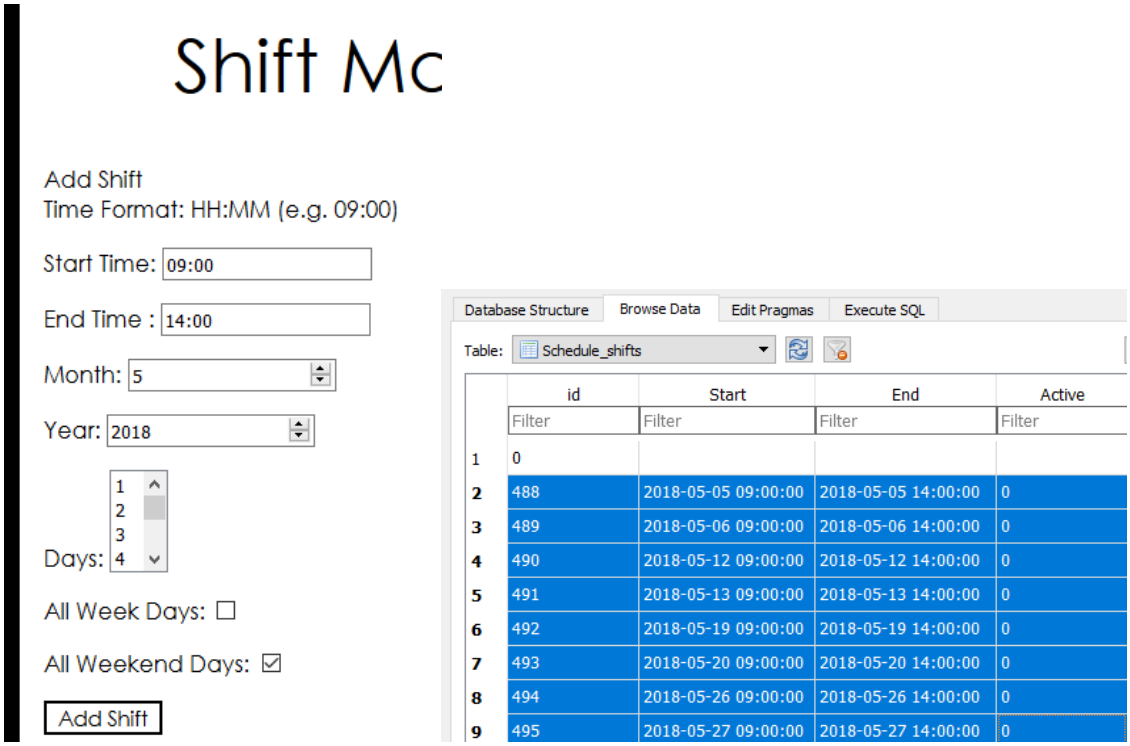
	id	Start	End	Active
	Filter	Filter	Filter	Filter
89	586	2018-05-16 16:30:00	2018-05-16 20:00:00	0
90	587	2018-05-17 16:30:00	2018-05-17 20:00:00	0
91	588	2018-05-18 16:30:00	2018-05-18 20:00:00	0
92	589	2018-05-21 16:30:00	2018-05-21 20:00:00	0
93	590	2018-05-22 16:30:00	2018-05-22 20:00:00	0
94	591	2018-05-23 16:30:00	2018-05-23 20:00:00	0
95	592	2018-05-24 16:30:00	2018-05-24 20:00:00	0
96	593	2018-05-25 16:30:00	2018-05-25 20:00:00	0
97	594	2018-05-28 16:30:00	2018-05-28 20:00:00	0
98	595	2018-05-29 16:30:00	2018-05-29 20:00:00	0
99	596	2018-05-30 16:30:00	2018-05-30 20:00:00	0
100	597	2018-05-31 16:30:00	2018-05-31 20:00:00	0
101	598	2018-05-02 15:00:00	2018-05-02 18:00:00	0
102	599	2018-05-02 16:30:00	2018-05-02 20:00:00	0
103	600	2018-05-10 09:00:00	2018-05-10 14:00:00	0

EVALUATIVE TESTING SCREENSHOTS (CONT.)

SCREENSHOT 3.1



SCREENSHOT 3.2



REFERENCES

https://www.seguetech.com/desktop-vs-web-applications/

https://www.sciencedirect.com/science/article/pii/S0166218X0100258X

https://simpleisbetterthancomplex.com/questions/2017/03/22/how-to-dynamically-filter-modelchoices-queryset-in-a-modelform.html

https://stackoverflow.com/questions/27715242/too-many-values-to-unpack-during-template-rendering

https://docs.djangoproject.com/en/2.0/topics/forms/

https://docs.djangoproject.com/en/2.0/ref/forms/fields/

https://www.tutorialspoint.com/python/time_strftime.htm

https://docs.python.org/2/library/datetime.html#strftime-strptime-behavior

https://djangodeployment.com/2016/12/23/which-database-should-i-use-on-production/

https://stackoverflow.com/questions/9081123/is-django-for-the-frontend-or-backend

https://docs.djangoproject.com/en/2.0/ref/validators/

https://simpleisbetterthancomplex.com/tutorial/2016/07/22/how-to-extend-django-user-model.html

https://www.w3schools.com/css/css3_animations.asp

https://stackoverflow.com/questions/18489393/django-submit-two-different-forms-with-one-submit-button

https://stackoverflow.com/questions/45184387/submit-two-forms-in-a-single-submit-in-django

https://www.w3schools.com/html/default.asp

https://www.w3schools.com/css/default.asp

https://gist.github.com/sirodoht/f598d14e9644e2d3909629a41e3522ad

https://docs.djangoproject.com/en/2.0/