Timetabling for Specific Scenarios

Staff Rota Management System for a Gastropub Manager

CS39440 Major Project Report

Author: Dongyen Keay (dok15@aber.ac.uk)

Supervisor: Dave Price (dap@aber.ac.uk)

Last Edited: 02/05/2024

Version: 1.1

This report is submitted as partial fulfilment of a BSc degree in Computer Science (G400).

Department of Computer Science

Aberystwyth University

Aberystwyth

Ceredigion

SY23 3DB

Wales, UK

Table of Contents

Declaration of originality

THIS REPORT’S TEMPLATE IS LARGELY JUST A COPY OF THE PROVIDED TEMPLATE ON GITLAB. I don’t know the GitLab link since I was informally sent the template in an email I deleted. That template’s heading number system broke so I made a simplified version.

I confirm that:

- This submission is my own work, except where clearly indicated.
- I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
- I have read the regulations on Unacceptable Academic Practice from the University’s Academic Registry (AR) and the relevant sections of the current Student Handbook of the Department of Computer Science.
- In submitting this work, I understand and agree to abide by the University’s regulations governing these issues.

Name: Dongyen Keay

Date: 13/03/2024

Consent to share this work

By including my name below, I hereby agree to this project's report and technical work being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name: Dongyen Keay

Date: 13/03/2024

Abstract

For my major project, I was assigned the vague topic of “Timetabling for Specific Scenarios” and the specific direction I took it in was inspired by my gastropub workplace. I decided to create software for a manager of a gastropub to create weekly timetables.

The purpose of the project is to create a digital tool which is helpful for a gastropub manager. In my work experience, I’ve seen problems created by manual timetabling, like assigning staff to shifts they are unavailable for, which then is resolved over private messaging between an employee and manager. This process is obtuse and slow, and could be improved with a timetabling tool which constrains a manager to making only valid edits to a staff rota / timetable.

There are some terms I use a lot throughout the project which should be clarified. When I say “roster”, I’m referring to a container of information about all employees, and when I say “rota”, I’m referring to a weekly timetable which says when employees are working and in which roles.

A variety of technologies make up the technical work. It is all programmed in Java, it writes to and reads from a PGSQL database, and it outputs staff rotas to PDF. Then supporting the project, I used GitHub for version control and OneDrive for manual backing up.

An agile Scrum methodology was used to tackle the work. This began with me designing a product backlog which specified my technical goal. It was here I worked out the specifics of what I wanted to create. Then throughout the project, I pulled related items from the product backlog into sprint backlogs, each of which had a sprint goal which when implemented, added significant functionality to my program.

My technical work delivered an offline desktop application which is mostly successful, because it provides a specific and restrictive timetabling tool which achieves most product backlog items.

1. Background, Analysis & Process

Background

My project's initial aim was vague and I first had to find a more defined direction, which I did by looking at existing timetable systems. I settled on a system based on my workplace's [A], because I am personally familiar with this, and I thought that the very specific staff timetabling problem would be less complex than more general timetabling like with Outlook Calendar [B]. I thought a weekly staff rota problem would be simpler because I wouldn't have to worry about clashes between events, timetabling to the minute and making longer term timetables based on a calendar.

I wanted to work on this problem because I have some understanding of it from my work experience. Choosing this enabled me to make quick planning and technical progress instead of being bogged down in trying to understand what the problem is.

Analysis

Considering a Web Application

Deciding on the type of application to make was an early task of mine.

It would have been very good for me to make my rota system a web application.

A web app would have been widely accessible via a wide range of devices [1], because web documents can easily be run by many devices, which would increase my program's accessibility and compatibility.

A rota web app would easily enable interaction from multiple users [1]. As it would be stored on a central server, manager and employee clients could access the system, which would enable employees to enter their unavailable times for managers to work around. The server system would also allow any staff to access the rota anywhere.

As a web app, my rota system could be very convenient, since a user would just need to use whatever browser they have, rather than install some software which may not work on their device [1]. An offline solution could add complexity for users. An online rota could be easier for a developer to maintain [2], since they would only have to manage one codebase on a server, rather than manage different versions for various platforms.

However, a web solution is not necessarily the best, because it depends on a client to server connection [1]. A web server going offline could render the timetabler inaccessible to managers and weekly rotas inaccessible to staff, which lends viability to an offline solution. Related to this issue, a major plus of a web app would have been allowing interaction from employees, but this benefit would not apply to employees with no internet access.

Another downside for a web solution that enables employees' interactions is that it adds complexity for them. Non-technical users would likely confuse a browser's back button for that of the web app [2]. This means either employees could be ineffective in using their tools or an employer would have to implement employee training.

Selecting a Programming Language

The main languages I considered were Java and C, because I'm most familiar with Java and C++. I was pretty settled into my Java choice, but I researched C to re-evaluate my options.

Using C, my rota system's performance could be improved. C is a lower level higher performance / more efficient language [1], which makes it seem like a good choice. However, I doubted that it would make much different for my relatively simple project, so this did not change my mind much.

Despite its benefits, C was not the clearly better choice for me.

C is procedural instead of object oriented [3]. I'm more familiar with the latter style and I find it very intuitive for representing real world scenarios, so if I chose C, my technical progress would be slowed by me trying to grasp procedural program design.

It's harder to make a C program robust since it lacks exception handling [3]. This means m code would have to be designed to a higher standard than if I was using Java, where I can put risky code in a try-catch block.

Software security in C is harder to achieve, due to it requiring manual memory management [4]. This requirement could hinder my technical progress because I would be bogged down in low level technical details of memory security instead of developing my system at a higher level.

I was very biased in favour of Java because it's what I'm most familiar with, and my research of it reinforced my bias.

It's quite easy to create Java software representations of reality because it's object oriented [3]. I could hit the ground running with program planning because any real object I wanted in software I could imagine as a Java class. This enables me to make good technical progress after a little planning.

Java programs are compatible with many devices because they run on systems via the Java Virtual Machine [3]. This reduces the need for Java developers to maintain multiple versions of software, which gives it a web-like advantage.

Java is a higher level language, and this is evident in it's automatic memory management [3]. Being able to worry less about code to hardware interaction, I could focus my efforts on developing software functionality.

Writing secure software is easier in Java than in C, because comparing the ISO C Standard and Java Language Specification documents, Java has slightly fewer total rules, and much fewer common rules for preventing "high severity attacks" [4]. This means C developers have to be more stringent to ensure software security.

Java is not necessarily the best choice either. Java's ease of use means its applications are slower than a C implementation. JVM enforces compatibility and the garbage collector handles unneeded memory, but these increase computing overheads and result in slower applications [3]. Since I valued this ease of use so much and my program is not very massive or complex, I assumed the performance factor would be minor.

Investigating Data Persistence Styles

At this project's start, I had only made data saving and loading systems using just plain text files, which worked well but I thought seemed to basic for a third year project.

Simple text files aren't necessarily bad.

They often contain unencrypted text which is widely human and computer readable [5]. Program data in a txt file would be compatible with many devices and their contents could be manually checked with no issue.

Text files have less pros than cons.

They are an inappropriate format for storing complex data structures because they are not designed to represent complex data structures [5]. Trying to store something complex like a staff rota in plain text could result in unwieldy lists and inefficient data storage [5].

The simplicity includes a "lack of data integrity measures" [5]. There is no correct format for text file contents, and whatever format gets used is entirely down to developers to decide, which means its harder to enforce correct data storage.

XML documents are like text files, but they have more inherent structure and constraints. Such documents can clearly represent complex data structures [6]. Using parent or child elements along with attributes in their opening tags, the complex and plentiful data content can be labeled and formatted in a human and computer readable way.

XML documents offer some integrity measures, because their content must be correctly placed within element tags [6]. Using this, software could quickly validate data to an extent, since XML correctness mainly considers structure and not content.

Being such a common format, XML document work on loads of hardware and software [6]. This is a great resource for an offline application since it can run on more devices.

Data storage systems using XML can be inefficient, since element tags can take more space than content [6]. If my staff roster got very large, the persisted data document could be very long due to XML's tag based nature.

Relational databases (RDBs) are the most complex data persistence means I've considered so far and they can model even more complex structures.

Using a RDB with multiple tables, a lot of data in with complex relationships can be effectively and tidily stored [7]. This is compatible with my project goal of storing a staff roster and weekly rota, since they depend on each other.

Integrity checking is done very well with RDBs [7]. Not only is data structure enforced, but the ability to constrain table column data to a certain type, or to be dependent on another table via foreign keys helps a lot. It helps because I wouldn't have to program as much to check for data correctness since the RDB management system does it for me.

Being so complex, a RDB is not always a great choice [7]. For example, a small program handling very simple data would be greatly over complicated by persisting data to a RDB, but I think my project is sufficiently complex that a RDB would be beneficial.

Since I chose Java, I decided to research serialization: its native and convenient data persistence method. Using serialization, the details of persisting complex data structures are abstracted away from me [8]. Using default classes and methods like readObject, I can quickly read in complex Java objects without having to first write code which processes a text file token by token.

Despite the massive convenience benefit, serialization could be a poor choice. The process is computationally demanding [8], which means saving and loading a lot would hurt system performance.

Being native to Java [8], files created from serialization are designed to be read by default Java methods. This means they are not standard or widely readable like XML, which hinders their compatibility with other systems. I still chose to use serialization because of its convenience benefits and because I had no plan for other systems to read the serialized files.

Data Distribution Formats

I identified that I wanted my program to export the staff rota into an accessible format, so I researched various methods.

Text files are a valid contender because they are widely accessible plain text documents [9]. A rota in a text file could be viewed by any staff member with a modern computer.

Text files' ease of access causes a problem which is that anyone can easily modify them [9]. A staff rota from my system should only be modified by the manager, so text files seem inappropriate.

Such files are perhaps too basic to effectively display a staff rota. Plain text lacks useful formatting abilities like making text bold and underlined [9], which means it can be harder to present a complex set of information.

A text file rota could result in bad user experience (UX) because software like Notepad doesn't preserve file formatting, so a narrow screen results in squished text [10]. I wanted to pick a format which offered consistent UX.

Exporting a rich text document seems both more suitable for a rota and more sufficiently complex for my project. Information types, like headers, can be quickly identified a glance by available formatting features like bold and colours [10].

Rich text is worse than plain text in some ways.

They use file types which are harder for programs to read data from [10], which would make PDF a bad choice for a saving and loading system. This is not an issue for my project, because I only want such documents to inform humans.

Their extra formatting comes at the cost of larger file size, which would hinder usability on very low capacity systems, but that's not a concern for my project.

Rich text formats include PDF and DOCX.

Like TXT, DOCX files can be edited by any holders [11], which is a feature I don't want.

DOCX files can be smaller than PDF due to compression [11], but I still favour PDF because of how it is static and widely compatible.

PDF was a strong contender because it can be viewed on very many devices and it is hard to edit [12]. This means that a staff rota with protected integrity can be viewed by many employees.

A consistent UX can be ensured with PDF, since a document’s appearance is not tied to the host device [12]. This means one clearly formatted document can look identical regardless of the computer used.

PDF files are not perfect.

The consistent UX can mean a poor mobile UX [13], because the static document type does not allow responsive design. A rota optimised for widescreen viewing could require a lot of scrolling on a phone screen.

Despite this glaring issue, I selected PDF for its accessibility and permanence.

Picking a GUI tool

The only UI I had programmed before was commandline based and I wanted to make my program more complex.

AWT could be an alright choice because it is widely compatible. Despite components like buttons being dependent on the OS, the library overall is not and will run on any JVM compatible machine [14]. This means an AWT GUI would run on many devices.

The library provides an intuitive way of programming a GUI, because window components are represented as objects [14]. This reduces the difficulty I would have in making a GUI.

Using Java AWT, my program’s UX would be inconsistent across devices. The library uses graphical methods from the OS [14], which means that a window with buttons on an old Windows machine would look a lot different than if it was on a new Mac.

Swing is an even more suitable choice than AWT because it builds on AWT with additions and improvements.

Swing offers faster performance than AWT because its code is not so dependent on waiting for the OS to do work [15]. This means my program could be laggier if it used AWT instead of Swing.

A related benefit is that a Swing UX is consistent across devices, because it is not as dependent on OS methods for drawing components like buttons [16]. Being more self sufficient, a Swing GUI would appear identical on different devices.

JavaFX is an even newer library with even more features [17].

Like Swing, JavaFX has a consistent appearance across devices [17], so I also favour it over AWT.

Non technical developers can use JavaFX effectively with Scene Builder [17]. This GUI tool allows developers to create a UI via an intuitive UI without any programming [17]. Using this, I would be able to create a complex GUI despite having no experience in doing such.

The library exploits modern hardware with GPU acceleration and it supports rich media like video [17]. This would benefit my project if it used a modern heavy UI, which I do not plan to develop.

Despite the strengths of JavaFX, I choose Swing because it is sufficiently advanced for my goals.

Choosing a Version Control System

I considered Git based systems including Git, GitLab and GitHub.

Picking either of the web solutions was not a significant choice since they were both just built on git [18]. I decided that an online solution would be better, since it would also work as a place to externally back up my project, rather than just having a Git repository on my laptop [18].

Picking a Methodology

The main options I considered were the agile Scrum model and plan driven Waterfall model.

A Waterfall approach would require me to have an intimate understanding of my project's details before starting technical work [20], while with Scrum, I would only need some higher level overview [19]. Since I have not made a timetabler before, I believe choosing the plan driven approach would have been bad and caused me to be bogged down in making a potentially faulty plan, that delays or prevents technical progress.

Project Requirements

Since I chose a Scrum methodology, I created my project requirements in the form of a product backlog: a list of high level descriptions about how a manager would use my system (stories). I built this by thinking about how a manager could interact with the timetable from appendix A.

Product Backlog Legend

To do **WIP** **Done**

- **As a manager, I want this product to be an offline desktop application so that its usage is only limited to managers.**
- **As a manager, I want to view the staff rota for the week, so that me and the staff can see what times which employees are scheduled to work.**
- **As a manger, I want to export the staff rota as a PDF so that it can be viewed on a wide range of devices.**
- **As a manager, I want to add a staff member to a free slot on the rota, so that I can create a staff rota for the week.**
- **As a manager, I want to be prevented from adding a staff member to a slot that is already occupied, so that my potential errors are prevented.**
- **As a manager, I want to remove a staff member from a slot on the rota, so that I can modify a filled staff rota.**
- **As a manager, I want to save a staff rota, so that the timetables I make are not lost.**
- **As a manager, I want to load a staff rota, so that a previous session's work can accessed or built upon.**
- **As a manager, I want a GUI in the staff rota creation interface so that my UX is intuitive and my inputs are more restricted to what is valid.**
- **As a manager, I want to add staff to a roster, so that I have staff to assign to rota slots.**

- **As a manager, I want to remove staff from a roster, so that I can have an up-to-date record of current employees.**
- **As a manager, I want to be unable to add duplicated staff to a shift, so that rota correctness is protected.**
- **As a manager, I want to copy the inputs I made for 1 day of the rota to every other day, so that UX is less tedious.**
- **As a manager, I want to save a staff roster to a PostgresSQL database, so that I don't need to recreate the staff roster every session.**
- **As a manager, I want to load a staff roster, so that a previous session's work isn't lost.**
- **As a manager, I want to store an email address for each employee so that they can be contacted and uniquely identified.**
- **As a manager, I want to be unable to enter invalid format emails so that staff roster correctness is better ensured.**
- **As a manager, I want each staff-roster item to contain an employee's name and speciality, so that valid rotas can be enforced and employees can be picked.**
- **As a manager, I want to be able to add multiple specialties for each employee, so that the system can represent an employee with multiple roles.**
- **As a manager, I want to be unable to add an employee to a slot they aren't specialised for, so that faulty rotas are prevented** *(e.g. no servers in chef slots)*.
- **As a manager, I want to add times in which an employee cannot work, so that faulty rota creation can be prevented** *(I haven't decided on a design for this. Could be an ArrayList ruleClassInstances variable that is stored in the Employee class or a more central location. The Rule class could contain Employee and UnavailableTime attributes)*.
- **As a manager, I want to be unable to add an employee to a slot they are unavailable for, so that faulty rotas are prevented.**

Process

Throughout technical development, I adhered to the Scrum methodology quite well. My initial planning and design was high level. It started with a product goal: a brief description of what I wanted to make, which was a gastropub staff rota system for a manager to use. Then I broke this down into a slightly more detailed product backlog with use case stories.

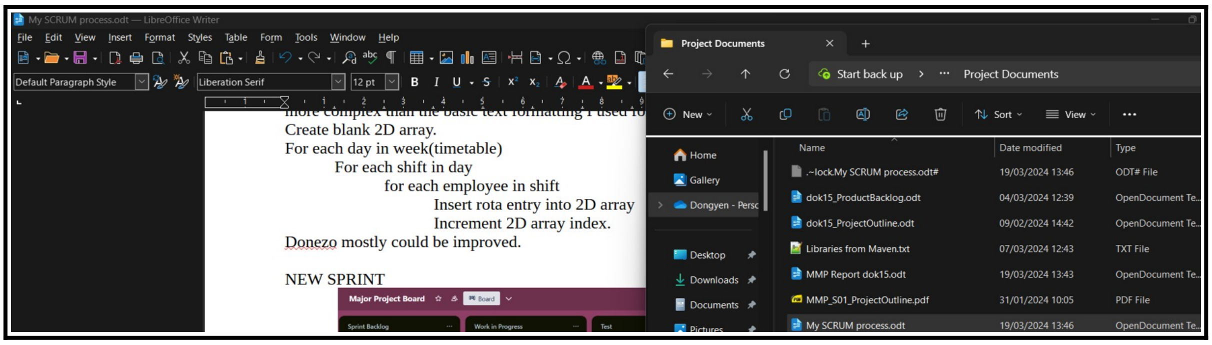
Trello was the Scrum management tool I used, and you can see my use of it in appendix D. When selecting the tool, I googled "Scrum tools" and tried a top results, which turned out to be just the product backlog manager I wanted.

Since Scrum is a group strategy, I had to adapt it for my solo project and act as the product owner, scrum master and development team. This did not negate the process' benefits, as I did all three roles and built the system in increments of functionality.

Areas of the process where I slacked were in sprints and testing. My sprints were not time-based, and instead were just based on groupings of related stories moved from the product backlog, that I would work to finish by no particular time. Despite the slack, I was able to build my program in frequent feature based increments.

Testing quality was mixed. Where it went professionally was when I wrote a method then wrote unit tests to check its correctness and resilience. Where it slacked was when I had no testing document during development, and instead just did ad-hoc usage based testing on code as I wrote it. Then testing quality shot up at the end with a big design document [Z] to test the program I considered mostly done.

Low level detailed planning happened too, but only in the scope of a sprint or single story. When thinking about how to write code, I would make think through logic and make pseudocode before coding. For example when trying to implement the GUI rota viewer I first made the notes below.



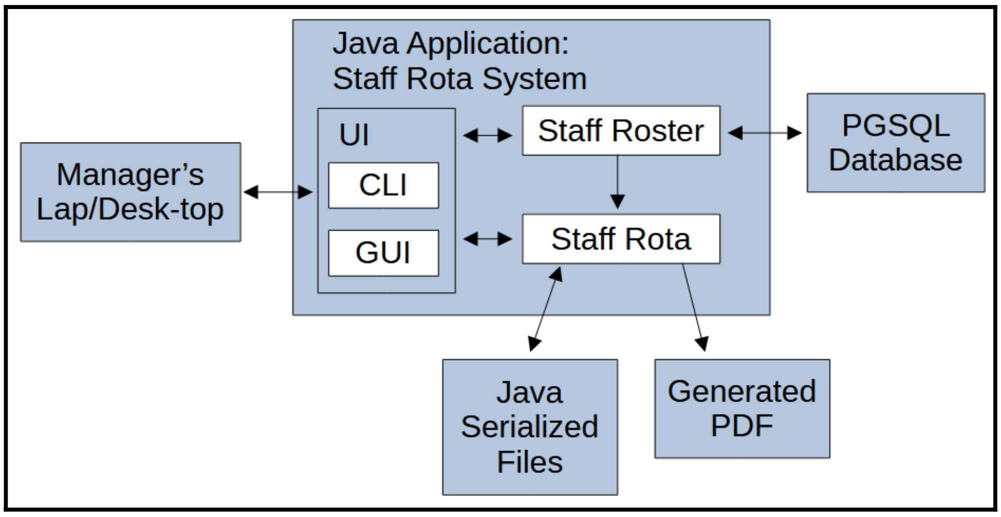
```
Create blank 2D array.
For each day in week(timetable)
    For each shift in day
        for each employee in shift
            Insert rota entry into 2D array
            Increment 2D array index.
Donezo mostly could be improved.

NEW SPRINT
```

Name	Date modified	Type
.~lock.My SCRUM process.odt#	19/03/2024 13:46	ODT# File
dok15_ProductBacklog.odt	04/03/2024 12:39	OpenDocument Te...
dok15_ProjectOutline.odt	09/02/2024 14:42	OpenDocument Te...
Libraries from Maven.txt	07/03/2024 12:43	TXT File
MMP Report dok15.odt	19/03/2024 13:43	OpenDocument Te...
MMP_S01_ProjectOutline.pdf	31/01/2024 10:05	PDF File
My SCRUM process.odt	19/03/2024 13:46	OpenDocument Te...

This meant implementation attempts were not careless hack jobs, so had a higher chance of success.

2. Design

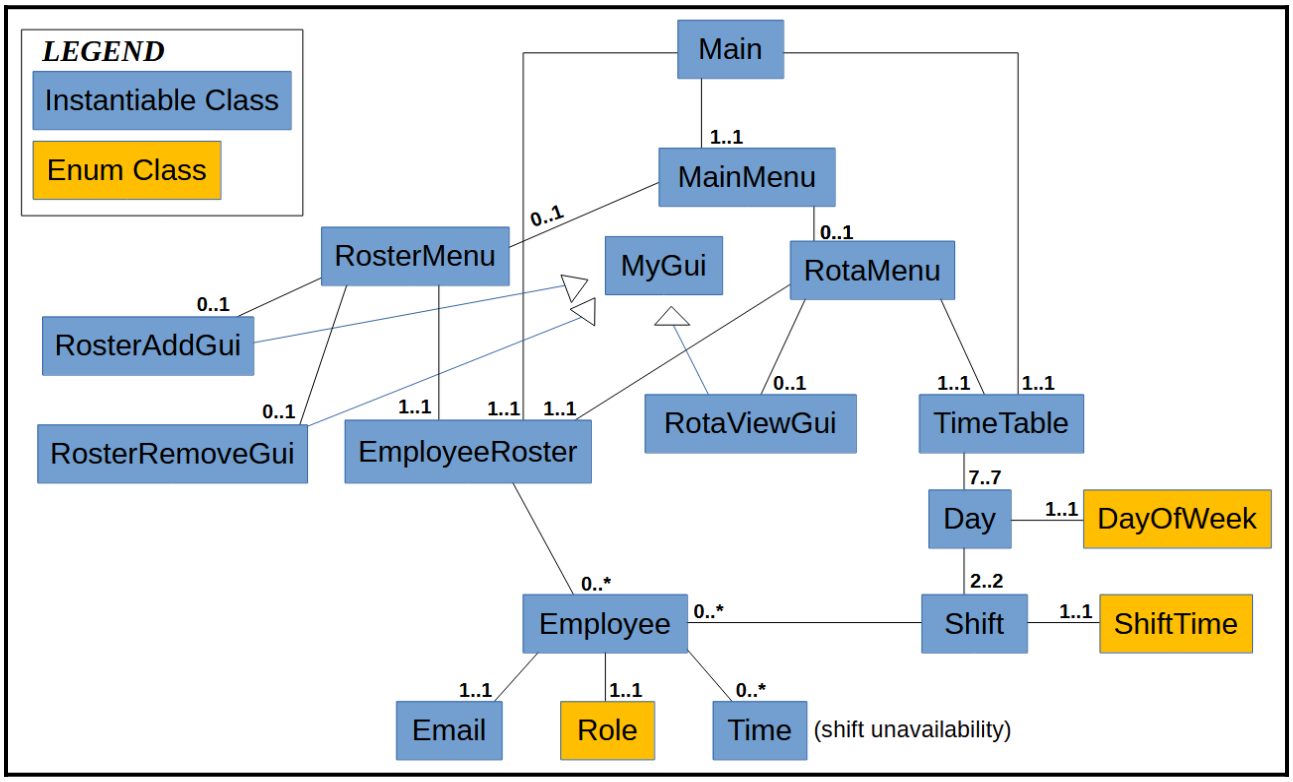
The overall architecture of my program is as follows.



The user device affects the UI, which affects the roster, which affects the rota etc. The exported PDF timetable does not affect the rota because my program does not parse the PDF.

Although no architecture diagram was written at the start of my programming, my product backlog describes this overall design with stories like "export the staff rota as a PDF".

Classes in my program are mapped below.



RosterAddGui inherits from MyGui, RosterMenu has exactly one EmployeeRoster, a Shift instance references zero or more Employee instances etc. Here you see the main split within the system: the largely separate staff roster and weekly rota aspects.

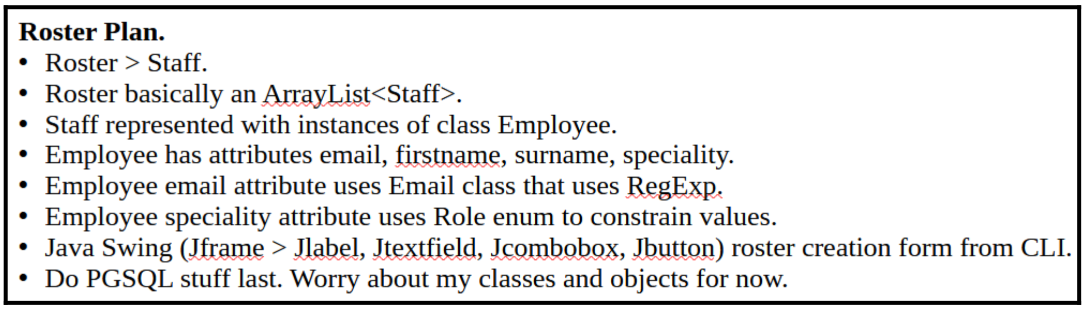
Roster Design

Information about individual employees gets entered into the UI, saved in the roster's RAM, then persisted to the CS department's PGSQL database. From this database, employee data can be read into the roster and displayed on the UI.

The rota is a timetable which gets filled with the roster's employee entities. A filled rota can be serialised by Java into a loadable file.

Looking at my product backlog and goal in whole, I saw my program had two main sections: the staff roster and the weekly rota. Since the rota is dependent on the roster, I decided to implement the simpler roster first. Implementing the rota first would have been suitable in a plan driven approach as it requires thorough design on what the roster would be like.

Analysing the requirements and implementing code to meet them could be very straightforward. When starting work on the roster system, a story says "add staff to a roster". From this, I planned low level details like there would be an Employee class which has instances stored in an instance of a Roster class.

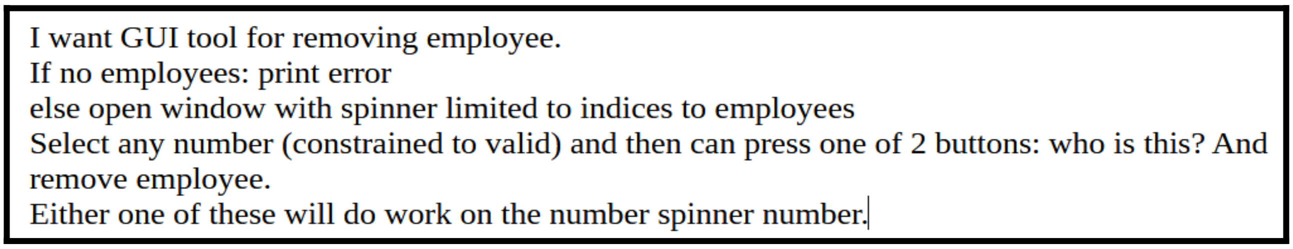


Roster Plan.
- Roster > Staff.
- Roster basically an ArrayList<Staff>.
- Staff represented with instances of class Employee.
- Employee has attributes email, firstname, surname, speciality.
- Employee email attribute uses Email class that uses RegExp.
- Employee speciality attribute uses Role enum to constrain values.
- Java Swing (Jframe > Jlabel, Jtextfield, Jcombobox, Jbutton) roster creation form from CLI.
- Do PGSQL stuff last. Worry about my classes and objects for now.

Another detail I realised is that the roster would have no cap on employee quantity, so an ArrayList instance would be a more suitable container than an array.

Requirement implementation was sometimes more complicated. From the story about emails used to uniquely identify employees, I realised there were two problems to solve. Email validity and uniqueness. I knew of regular expressions from year one and knew they could solve the validity problem, and since emails are just Strings, the provided equalsIgnoresCase() method could be used to ensure uniqueness.

A pseudocode design was used for planning a GUI employee deletion system.



```
I want GUI tool for removing employee.
If no employees: print error
else open window with spinner limited to indices to employees
Select any number (constrained to valid) and then can press one of 2 buttons: who is this? And
remove employee.
Either one of these will do work on the number spinner number.
```

A spinner seemed appropriate and robust because it can constrain user input to what is valid. This was an occasion where I thought ahead multiple steps and designed for the longer term. The remover's design requires the employee indices for employees in the roster, so I planned for a getEmployeesDetailsFromRoster type method which would print index numbers too. A major flaw in this design is that it reveals some lower level workings of my program: staff indices, which could be used by attackers to target the roster.

From the story about writing the employee roster to a database, I realised that Employee class attributes would have to match table columns, which would be easy to implement.

Rota Design

In planning the rota system, I realised it would not be as simple as the employee container roster. Appendix A was a major guide and using it, I imagined a Rota object storing all days of the week, Day objects storing the day and night shifts, and Shift objects storing employees pulled from the roster. I came up with this design by writing a rota description and highlighting nouns to be considered as Java classes.

Text description: highlight nouns as possible objects.
A **timetable** which contains information for each **day** of the week.
For each day, store information about each **shift** (day or night)
For each **shift**, store information about which **roles** are filled by which **employees**

With the number of days in a week and shifts in a day being constant, I realised that static arrays would be suitable code representations for these things.

Will use the Employees created by my completed roster system.
Time table could have a String variable "about" where you can input something like "Starting Monday 9th oct". Or research some sort of date storing class
Could use Java Enums like DayOfWeek and ShiftTime
TimeTable could contain a Days[7] array or 7 hard coded Day variables which would be OK since number of days are fixed.

With weekdays and shift times like "Monday" and "Night" being constant, I realised enums should be used to add information to Day and Shift class attributes.

The story about speciality attributes for each employee suggested that I would need code to ensure an employee is qualified for a shift role the manager is trying to assign them to. I decided that this would be achieved with the user having to specify which shift role they wanted filled, and the rota class having multiple employee containers, each for staff with different specialities like chef.

When designing the Shift class, I considered either hardcoding Employee variables in a way that matches the source rota [A] or using ArrayLists.

```
Private Employee chef1;
Private Employee chef2;
Private Employee server1;
…
OR private ArrayList<Employee> chefStaff; etc.
```

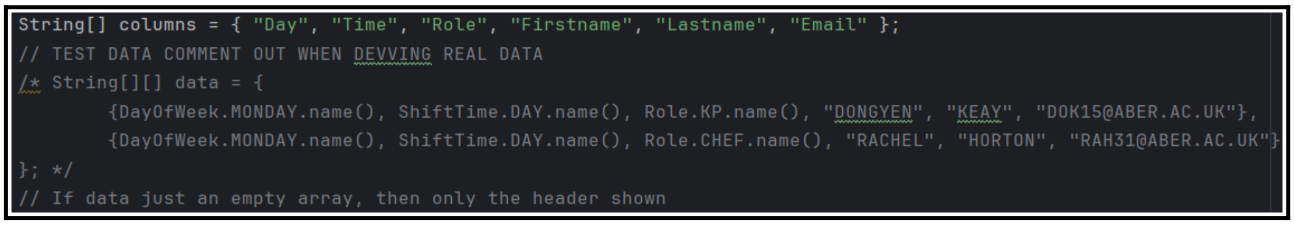
I settled on the latter solution because hardcoding the staff slots makes the software less adaptable to situations like the manager wanting more than two caterers.

For viewing the rota, I wanted my system to provide three means: CLI, PDF and GUI. An early document I made was a rota CLI prototype [C] based on my source rota [A]. It is basic but functional, as it clearly communicates who is working in which roles and on what shifts, and it is a design I largely stuck with for the CLI and PDF rota viewing tools. The main difference between design and implementation was that I made the prototype while thinking of hardcoding individual staff slots.

Designing for rota viewer programs was quite simple. Since my rota object was intended to be a container of containers, I decided there would be a loop over weekdays with a nested loop over shifts in a day with a nested loop over employees on a shift.

```
For each day in week(timetable)
        For each shift in day
                for each employee in shift
```

The above algorithm applied to all three viewers. Formatting on the CLI and PDF viewers was planned to be the same, with the day being printed with no indentation, the shift indented once, then each employee and their role indented twice. Designing for the GUI viewer was more complex and I settled on an unprofessional looking solution of inserting information about each employee into a Java Swing table.



```java
String[] columns = { "Day", "Time", "Role", "Firstname", "Lastname", "Email" };
// TEST DATA COMMENT OUT WHEN DEVVING REAL DATA
/* String[][] data = {
        {DayOfWeek.MONDAY.name(), ShiftTime.DAY.name(), Role.KP.name(), "DONGYEN", "KEAY", "DOK15@ABER.AC.UK"},
        {DayOfWeek.MONDAY.name(), ShiftTime.DAY.name(), Role.CHEF.name(), "RACHEL", "HORTON", "RAH31@ABER.AC.UK"}
}; */
// If data just an empty array, then only the header shown
```

I settled on this design despite it being less legible than my CLI and PDF solutions because I wanted to challenge myself to develop a range of functional Java GUI: something I had not attempted before.

Designing for a system that adds an employee to the rota was more complex than adding one to a roster. The latter is little more than an ArrayList for Employee instances, while the former is a container of containers representing all shifts in a week. Using pseudocode, I broke the algorithm into two main parts: the shift and employee getters.

```
PROBLEM: Adding staff from rota to time table.
Get data for which day, which shift on that day, which role, and which staff member then process all of that?
Obtain Shift object (via day and time)
Obtain employee object.
Try to add employee to shift :)
```

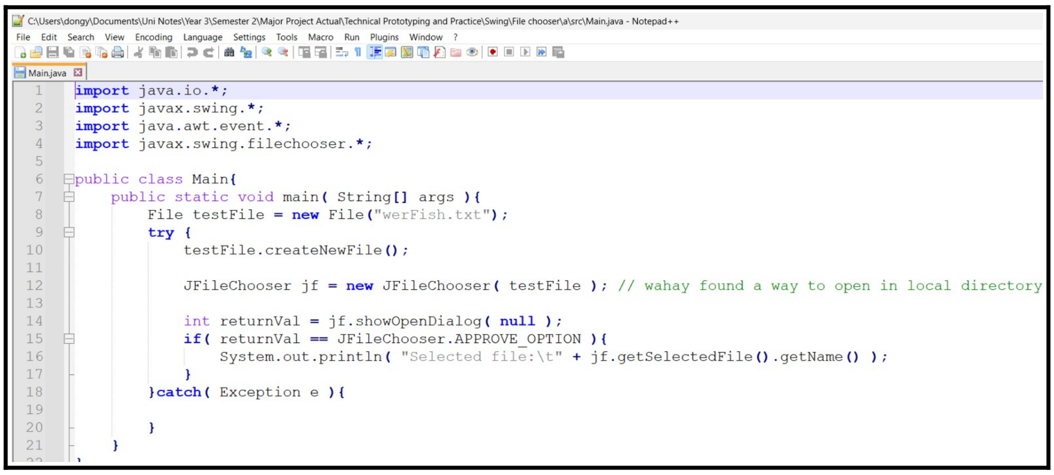
The first part would select a shift object from the rota, and the second would select an employee from the roster. I planned for this getter system to use a CLI that queried the user for indices. My code would also find out which role the manager wants to fill and slot the employee there if qualified.

Design for rota persistence was done in the form of some sentences.

For the sake of simplicity: I want to save data by serialisation.
Could hardcode a roster data filename or could have variable that stores filename provided by user.
Use regexp to ensure filenames valid by restricting to just letters and numbers etc.
Maybe have default hardcoded data file name like “roster1.txt” which you can override.
Roster loading function can have form to get text file name.

It was partly followed through as only the parts about serialisation and hardcoding a single datafile name were implemented, but the parts about design for using regular expressions to restrict file names was ported over to the design for the rota PDF exporting system.

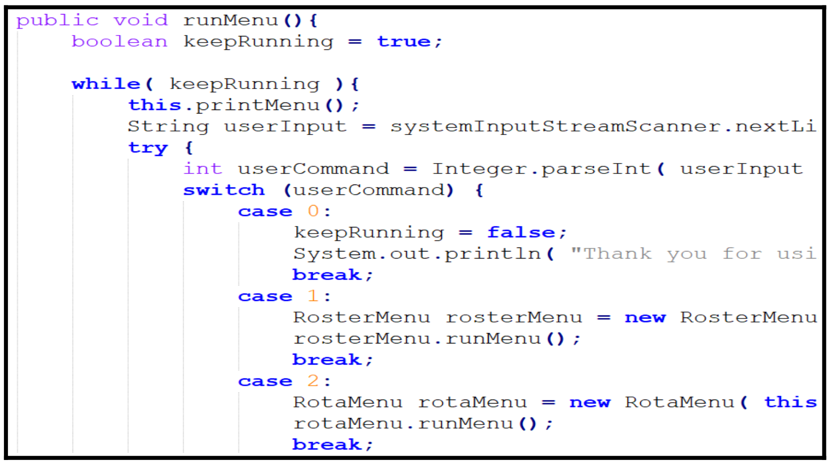
A more complex rota persistence system design was thrown out after prototyping.


C:\Users\dongy\Documents\Uni Notes\Year 3\Semester 2\Major Project Actual\Technical Prototyping and Practice\Swing\File chooser\a\src\Main.java - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Main.java

```java
import java.io.*;
import javax.swing.*;
import java.awt.event.*;
import javax.swing.filechooser.*;

public class Main{
    public static void main( String[] args ){
        File testFile = new File("werFish.txt");
        try {
            testFile.createNewFile();

            JFileChooser jf = new JFileChooser( testFile ); // wahay found a way to open in local directory

            int returnVal = jf.showOpenDialog( null );
            if( returnVal == JFileChooser.APPROVE_OPTION ){
                System.out.println( "Selected file:\t" + jf.getSelectedFile().getName() );
            }
        }catch( Exception e ){

        }
    }
```

The design used Java serialisation and the Swing and AWT libraries to allow users to select any rota datafile. It meant that users could serialise the rota to a file with a name of their choice. Then they could intuitively load the rota via a GUI file selector, rather than tediously type a filename on a CLI. Pictured above is a simple prototype program that opens a file chooser window for the program’s directory. I ditched this design because the code which worked in small tests would freeze my program mid run when I added it as a method which could be called from a CLI.

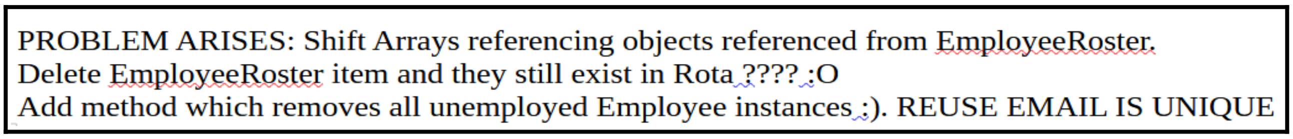
A later design choice was made to break up the system’s CLI main menu into smaller menus that you could traverse between: main, roster and rota.



```java
public void runMenu(){
    boolean keepRunning = true;

    while( keepRunning ){
        this.printMenu();
        String userInput = systemInputStreamScanner.nextLi
        try {
            int userCommand = Integer.parseInt( userInput
            switch (userCommand) {
                case 0:
                    keepRunning = false;
                    System.out.println( "Thank you for usi
                    break;
                case 1:
                    RosterMenu rosterMenu = new RosterMenu
                    rosterMenu.runMenu();
                    break;
                case 2:
                    RotaMenu rotaMenu = new RotaMenu( this
                    rotaMenu.runMenu();
                    break;
```

I initially only developed a single menu, but as features were added, it got long, which hurt UX in two ways: readability was reduced and more menu commands required two digits to be input instead of one.

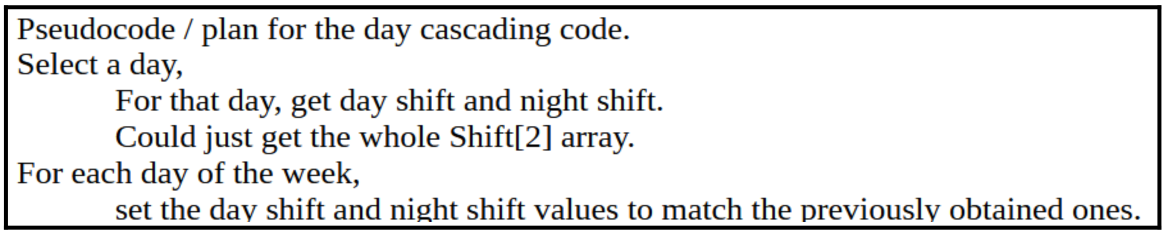
Another late discovered problem I designed for was discrepancy between the rota and roster: If an employee got sacked and removed from the roster, they could remain on the rota.



PROBLEM ARISES: Shift Arrays referencing objects referenced from EmployeeRoster.
Delete EmployeeRoster item and they still exist in Rota ???? :O
Add method which removes all unemployed Employee instances :). REUSE EMAIL IS UNIQUE

Above is a design for a simple method which uses the roster's email uniqueness method to see if an email is used by any contained employee. It would be used to see if each employee in the rota was also in the roster, and it would make deletions to fix discrepancies.

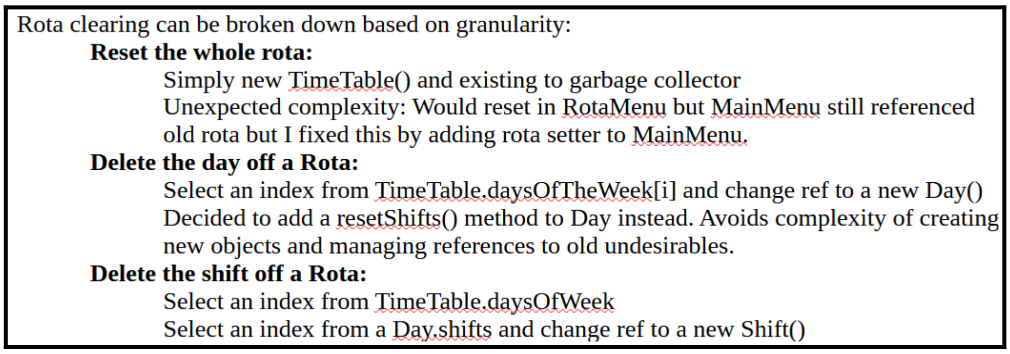
Pseudocode was used to design a system to copy the staff rota setup for one day across to every other day. This would enable the user to save time by copying entries for one day instead of tediously filling in the rota from scratch for each day.



```
Pseudocode / plan for the day cascading code.
Select a day,
        For that day, get day shift and night shift.
        Could just get the whole Shift[2] array.
For each day of the week,
        set the day shift and night shift values to match the previously obtained ones.
```

The design turned out mostly sound, but my initial implementation of it had a major fault to do with passing objects by reference, which I should discuss in the implementation section.

Rota clearing design was based on granularity. The design would enable the user to clear a whole rota, a day or a shift.



Rota clearing can be broken down based on granularity:
- **Reset the whole rota:**
  - Simply new TimeTable() and existing to garbage collector
  - Unexpected complexity: Would reset in RotaMenu but MainMenu still referenced old rota but I fixed this by adding rota setter to MainMenu.
- **Delete the day off a Rota:**
  - Select an index from TimeTable.daysOfTheWeek[i] and change ref to a new Day()
  - Decided to add a resetShifts() method to Day instead. Avoids complexity of creating new objects and managing references to old undesirables.
- **Delete the shift off a Rota:**
  - Select an index from TimeTable.daysOfWeek
  - Select an index from a Day.shifts and change ref to a new Shift()

The planned process for resetting some rota object like an instance of Day or Shift, was to create a new instance, reference that and dereference the old instance so that it gets handled by Java's garbage collector.

An important feature in a staff rota tool is the ability to store when staff are unavailable to work. There were two plans for ways this could be done.

```
Adding times in which an employee cannot work.
        ArrayList<CantWork> employeeUnavailabilityRules
                Very similar solutions to below, but would require extra processing. An extra
                ArrayList to check.
        Class CantWork
                private Employee e;
                private DayOfWeek d;
                private ShiftTime s
OR
        ArrayList<unavailableTime> unavailableTimes; attribute in Employee?
                Time attributes will use DayOfWeek and ShiftTime enums.
                Add unavailable times from RosterMenu.
```

The former would store the unavailable times within a central ArrayList. This collection would reference rule objects with attributes for an employee and a shift they are unavailable for. The latter would add an unavailable times ArrayList within the Employee class, for a record of the shifts an employee is unavailable for. I settled for the latter because an employee's availability is a vital aspect of that employee, which makes sense for it to be part of their representative class. The solution also requires less programming as I can access unavailability information from Employee objects directly instead of having to create a new class to an employee and a time, then accessing unavailability information through another ArrayList.

Unavailable times would be represented using a simple class with attributes describing which shift on which day someone cannot work.

```
When trying to add an unavailable Time to an Employee instance from Roster menu.
Select Employee.
Get inputs for DayOfWeek and ShiftTime enums.
Create new unavailable Time instance and add. Prevent duplicates!
```

Enum values like "MONDAY" and "NIGHT" would be use to describe such a shift, and instances of this simple class would be referenced from an Employee instance.

Design for how unavailable times would be enforced was quite tacked on to an existing system. My program could already assign an employee to a shift and prevent them from being assigned a role if not qualified.

```
Capture the Day instance's DayOfWeek enum value                    Header (Default Page Style)
        Capture the Shift instance's ShiftTime value
                Compare these with the unavailableTime instances of the Employee instance.
```

The design describes another check added right at the end of the assignment process to check if time was booked off for an employee. When adding an employee to a shift, the user must first select which day the shift was on, which shift on the day, then which role on the shift. After this, my program would be able to compare the selected shift information with the employee's unavailable time objects.

My design process was quite adaptable because of the Scrum approach. Some issues I did not consider until I encountered problems in using my program. For example, duplicate employees could be added to a shift, which lead to the creation of a new product backlog item about preventing this. Designing for duplicate employee detection was simple because I had the unique emails to compare. Similarly, some stories were added quite late into development because I wanted to increase my project’s complexity. Exporting a PDF rota was a late addition which was good because a PDF is an excellent shareable format [22]. The design for this was identical to my CLI rota and also inspired by the source rota [A].

3. Implementation

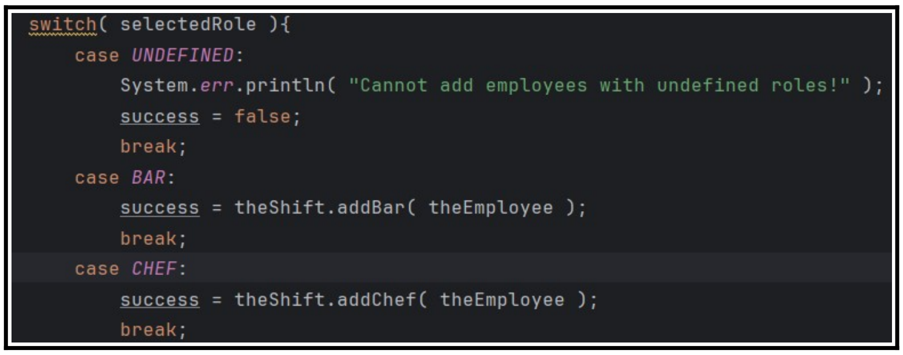
A major decision was to be made when starting technical work: implement the roster or the rota first? I chose roster first, because the rota requires the roster to draw employees from. Implementing the rota first would require employee test stubs and detailed longer-term low level planning of my program, which goes against the Scrum approach.

This ground-up implementation approach was done throughout development. For example, when implementing a rota, I first implemented the Employee, Shift, Day then TimeTable classes. Depended on components were done first.

Design and implementation were not done so separately. For example, my original menu design was to have one commandline interface with all the interaction options for the roster and rota, but after implementation, I realised it was unreadably long and that having to input two digits to access most options was tedious. Initial implementation lead to a design change which lead to reimplementation.

When implementing the system, I worked in a very incremental manner. For example when implementing the roster, I started with a method that prints the CLI menu with options like “add employee”, then I would work on implementing system features one at a time. The whole process was guided by Scrum, since product backlog items would move to the sprint backlog, from which I worked on one story at a time [D]. Each story would describe one feature of the system.

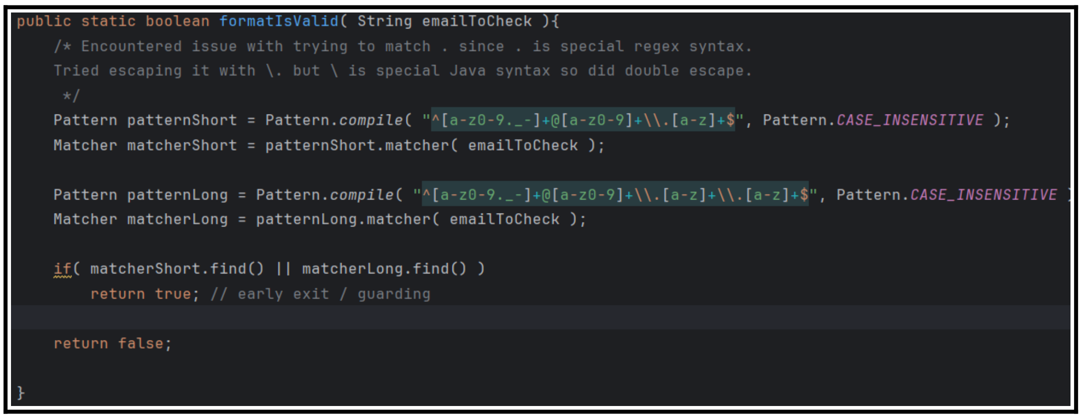
Enums were used to good effect by placing restrictions on code and reducing the chance of human error. Days of the week, shift times (night or day) and possible staff roles were predefined and constant, so representing them with enum classes allowed me to use enum constants instead of riskily typing out strings or integers to represent days or roles. Example enum use is pictured below.



```
switch( selectedRole ){
    case UNDEFINED:
        System.err.println( "Cannot add employees with undefined roles!" );
        success = false;
        break;
    case BAR:
        success = theShift.addBar( theEmployee );
        break;
    case CHEF:
        success = theShift.addChef( theEmployee );
        break;
```

When planning to implement Employee class, I thought about simply storing an email with a String attribute, but I decided to implement an Email class which places restrictions on what an email can contain.

Built in regular expressions (regex) libraries were very useful on this project. Regex code structure was copied from a tutorial [23], but most of the work is mine because I studied a general guide [24] and did experiments to find out how to setup regex to do what I wanted. The most complex feature achieved with regexp was enforcing valid emails and code for this is pictured below.

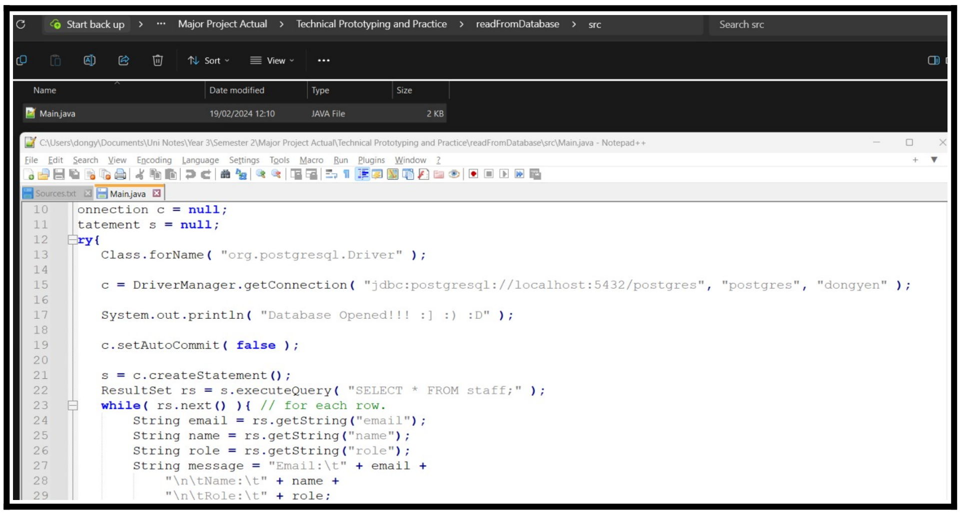


```java
public static boolean formatIsValid( String emailToCheck ){
    /* Encountered issue with trying to match . since . is special regex syntax.
    Tried escaping it with \. but \ is special Java syntax so did double escape.
     */
    Pattern patternShort = Pattern.compile( "^[a-z0-9._-]+@[a-z0-9]+\\.[a-z]+$", Pattern.CASE_INSENSITIVE );
    Matcher matcherShort = patternShort.matcher( emailToCheck );

    Pattern patternLong = Pattern.compile( "^[a-z0-9._-]+@[a-z0-9]+\\.[a-z]+\\.[a-z]+$", Pattern.CASE_INSENSITIVE
    Matcher matcherLong = patternLong.matcher( emailToCheck );

    if( matcherShort.find() || matcherLong.find() )
        return true; // early exit / guarding

    return false;

}
```

An issue I encountered was trying to do complex regex within Java when both languages have overlapping syntax. I had valid regex which would not work because it contained Java characters that had to be escaped.

Simple regex is key to my system's SQL injection prevention. When users input data to be stored in a PostgreSQL database, any inputs containing a semi colon gets rejected due to them being a likely injection attempt [25].

The system interfacing with a PostgreSQL database is a significant technical accomplishment. Code to interact with the database was largely copied from a tutorial [26], but I still did a lot of the work. Setting up a database and writing a program to connect with it was something I had not attempted before, so I downloaded PostgreSQL and created a local database along with a table and a simple prototype program on my laptop which is shown below.

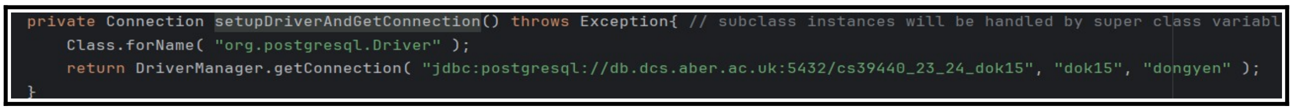


```
Name         Date modified       Type        Size
Main.java    19/02/2024 12:10    JAVA File   2 KB
```

C:\Users\dongy\Documents\Uni Notes\Year 3\Semester 2\Major Project Actual\Technical Prototyping and Practice\readFromDatabase\src\Main.java - Notepad++

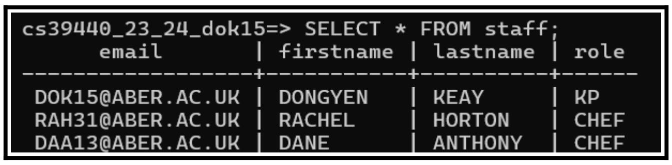
```java
onnection c = null;
tatement s = null;
ry{
   Class.forName( "org.postgresql.Driver" );

   c = DriverManager.getConnection( "jdbc:postgresql://localhost:5432/postgres", "postgres", "dongyen" );

   System.out.println( "Database Opened!!! :] :) :D" );

   c.setAutoCommit( false );

   s = c.createStatement();
   ResultSet rs = s.executeQuery( "SELECT * FROM staff;" );
   while( rs.next() ){ // for each row.
       String email = rs.getString("email");
       String name = rs.getString("name");
       String role = rs.getString("role");
       String message = "Email:\t" + email +
           "\n\tName:\t" + name +
           "\n\tRole:\t" + role;
```

The code was largely a copy of a tutorial [26], but I adapted the SQL to be suitable for a gastropub timetabler. Difficulties were encountered in trying to get connection code working on a campus computer. I tried to create a database like I had done on my laptop but found out this was not possible. After emailing module staff, a database was setup for me on a campus server. Then I adapted tutorial code [26] to work with my project database as shown below.



```
private Connection setupDriverAndGetConnection() throws Exception{ // subclass instances will be handled by super class variabl
    Class.forName( "org.postgresql.Driver" );
    return DriverManager.getConnection( "jdbc:postgresql://db.dcs.aber.ac.uk:5432/cs39440_23_24_dok15", "dok15", "dongyen" );
}
```

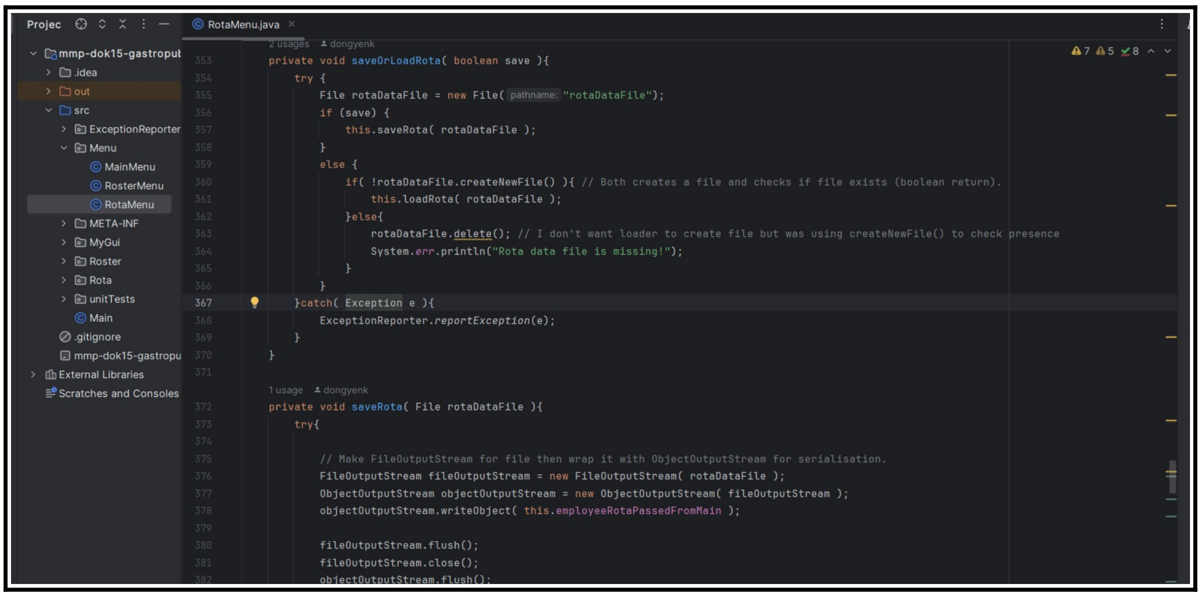
PostgresQL was used to a limited extent on my project since it is only part of the roster system where it stores staff information. This is because my staff table is simple with only a few columns, while a rota table would need many more columns and would possibly need foreign keys to link it to a roster table. My system's roster table is shown below.



```
cs39440_23_24_dok15=> SELECT * FROM staff;
       email        | firstname | lastname | role
--------------------+-----------+----------+------
 DOK15@ABER.AC.UK   | DONGYEN   | KEAY     | KP
 RAH31@ABER.AC.UK   | RACHEL    | HORTON   | CHEF
 DAA13@ABER.AC.UK   | DANE      | ANTHONY  | CHEF
```

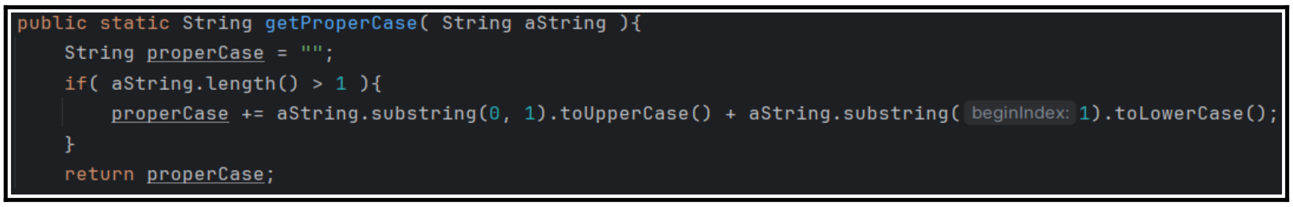
A library called JDBC was used which greatly simplified database connection by abstracting the connection process away into powerful methods. This let me focus on SQL and not get bogged down in connection errors.

Serialisation was used to store the complex rota object. Code which saves my timetable is based on a few tutorials [40][41][42]. Since serialisation is so basic, a lot of the code is copied from the tutorials, but I still adapted the code and made it work within a program more complex than the tutorials' examples. Code below is some amalgamation and adaptation of what I saw in the tutorials.



```
private void saveOrLoadRota( boolean save ){
    try {
        File rotaDataFile = new File( pathname: "rotaDataFile");
        if (save) {
            this.saveRota( rotaDataFile );
        }
        else {
            if( !rotaDataFile.createNewFile() ){ // Both creates a file and checks if file exists (boolean return).
                this.loadRota( rotaDataFile );
            }else{
                rotaDataFile.delete(); // I don't want loader to create file but was using createNewFile() to check presence
                System.err.println("Rota data file is missing!");
            }
        }
    }catch( Exception e ){
        ExceptionReporter.reportException(e);
    }
}

private void saveRota( File rotaDataFile ){
    try{

        // Make FileOutputStream for file then wrap it with ObjectOutputStream for serialisation.
        FileOutputStream fileOutputStream = new FileOutputStream( rotaDataFile );
        ObjectOutputStream objectOutputStream = new ObjectOutputStream( fileOutputStream );
        objectOutputStream.writeObject( this.employeeRotaPassedFromMain );

        fileOutputStream.flush();
        fileOutputStream.close();
        objectOutputStream.flush();
```

Displaying employee names in title case was a challenge I encountered. After googling the problem, I found a simple solution in a code example [27] which greatly influenced the solution I settled on pictured below.

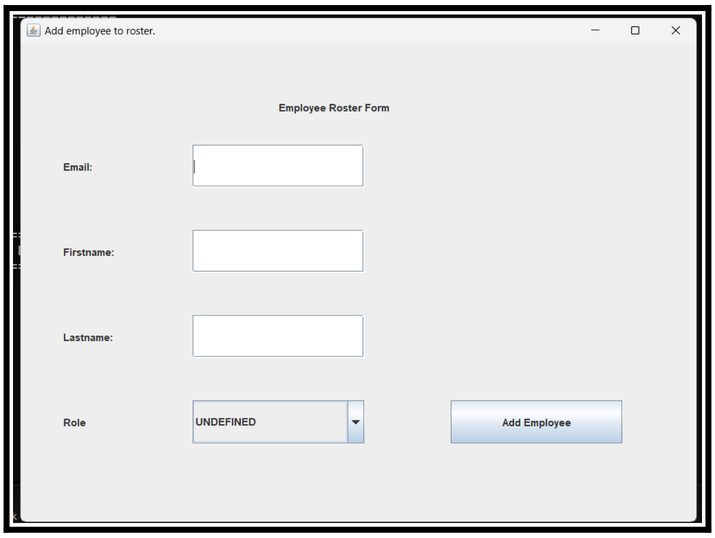


```
public static String getProperCase( String aString ){
    String properCase = "";
    if( aString.length() > 1 ){
        properCase += aString.substring(0, 1).toUpperCase() + aString.substring( beginIndex: 1).toLowerCase();
    }
    return properCase;
```

The solution is largely a copy of the example, but I adapted the code to more concisely solve my problem and I demonstrated a good understanding that the solution treats a String as an array and uses its segments to construct a new String.

The program's interface is mostly on the commandline, with some options opening a graphical tool to do things like create an employee or view a rota. CLI was implemented with a while loop containing a switch statement that handles user inputs, such as "0" which would exit the loop.

Implementing a variety of GUIs for some features on my system was a technical challenge. I used many tutorials [28-34] while implementing the code but the work is mostly mine. A powerful library: Java Swing was used for creating GUI elements. For example, when creating a tool for adding new employees to the roster, I studied some tutorials [28][29][30][32][33] to learn how to setup a Window and its elements like dropdown boxes. Then I did independent work to design the window, set it up and get its elements to send data to my program. A finished tool is pictured below.



Email:

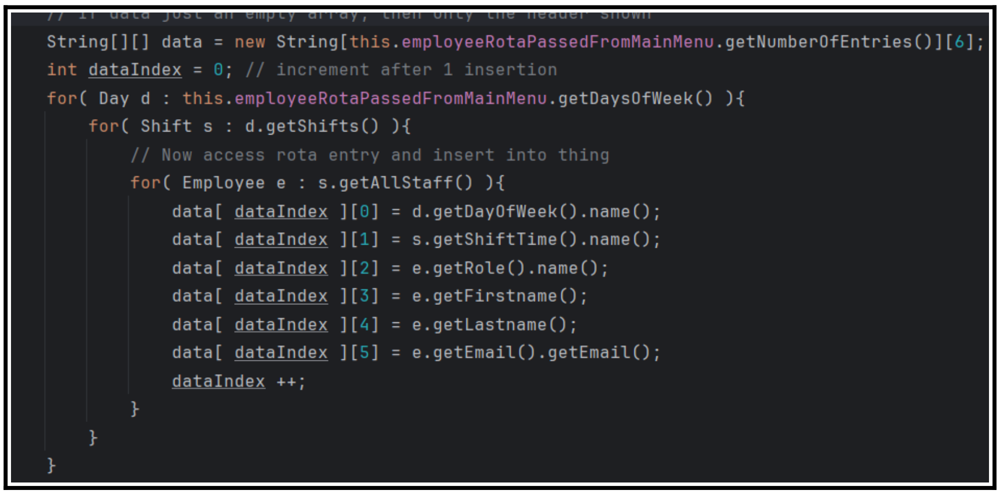
Firstname:

Lastname:

Role UNDEFINED

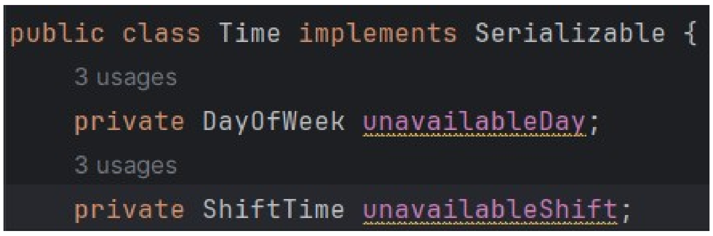
Restrictive input elements like spinners and dropdown boxes are used to good effect. Where input spaces only have a few valid values, I used elements more restrictive than text fields. In the above example, a dropdown box is used to select one of the few staff roles.

Making code for a GUI rota viewer was a technical challenge I overcame by implementing a simple table to represent the rota. The structure of code in RotaViewGui.java is based on a tutorial [34], but it is not a mindless copy. Work I did involved experimentation and engineering. I learned that Swing's JPanel class not needed for a readable table while JPane was needed for visible table headers. Tutorial code used hardcoded 2D array data so I had to adapt it to get data from my moddable staff rota, as you can see below.


```
// if data just an empty array, then only the header shown
String[][] data = new String[this.employeeRotaPassedFromMainMenu.getNumberOfEntries()][6];
int dataIndex = 0; // increment after 1 insertion
for( Day d : this.employeeRotaPassedFromMainMenu.getDaysOfWeek() ){
    for( Shift s : d.getShifts() ){
        // Now access rota entry and insert into thing
        for( Employee e : s.getAllStaff() ){
            data[ dataIndex ][0] = d.getDayOfWeek().name();
            data[ dataIndex ][1] = s.getShiftTime().name();
            data[ dataIndex ][2] = e.getRole().name();
            data[ dataIndex ][3] = e.getFirstname();
            data[ dataIndex ][4] = e.getLastname();
            data[ dataIndex ][5] = e.getEmail().getEmail();
            dataIndex ++;
        }
    }
}
```
The result is a table less readable than my commandline table viewer but the technical accomplishment made it worth keeping.

Allowing managers to record which shifts employees were unavailable for then preventing employees from being added to those shifts was a complex problem, which is why an attempt to solve it was left till after the system was mostly done. I felt a bit torn between two designs: should unavailable times be referenced from Employee instances or should they be in a more central location. I settled on referencing them from employees because unavailability is a key aspect of an employee so it can appropriately be represented with an attribute in Employee.

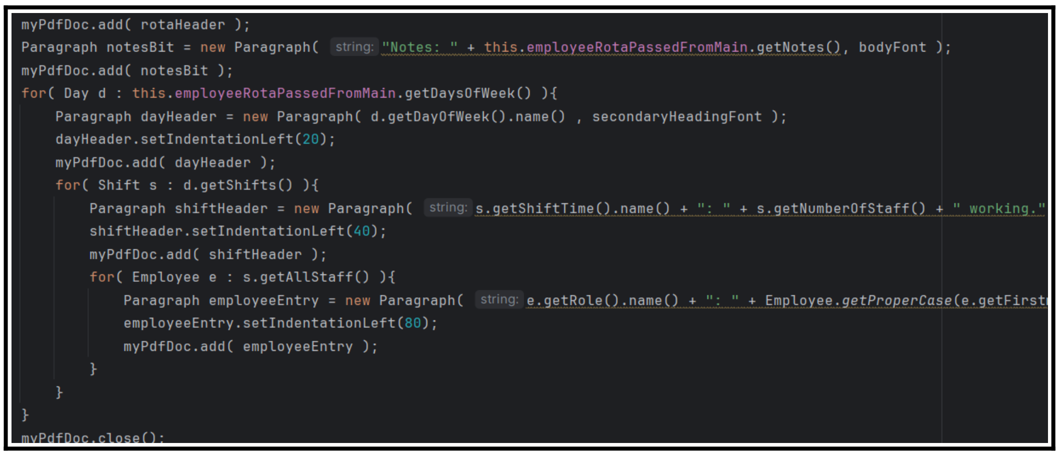
To represent shifts for which an employee was unavailable, I created a class to record a such shifts. This is pictured below.


```
public class Time implements Serializable {
    3 usages
    private DayOfWeek unavailableDay;
    3 usages
    private ShiftTime unavailableShift;
```
Its instances are referenced from an ArrayList that is referenced an Employee instance. Whenever a user attempts to assign someone to a shift, my code captures the weekday and time (day or night) of the shift and compares it to all the unavailable times associated with that employee to check for clashes.

Adding the ability to copy the rota for a day across to the rest of the week was a late design idea I thought of after using the system and realising how tedious it could be to fill in a week's rota, especially when most days have the same staff working.

Unintentionally passing objects by reference was a major technical problem I encountered and solved. Since filling in the rota one shift at a time could be so tedious, I started work on a convenience feature which would copy the staff assignments for one day across to every other. The first implementation would appear to work at first, then making a change to one day would also apply to every other day. Although my rota object was referencing seven Day instances, each one was referencing the same two night and day Shift instances. Through research of a tutorial [35], I found out I needed to make a deep copy of the day I wanted to copy across. This is because a Day instance references Shift instances which reference ArrayList and Employee instances. I needed the entirety of a Day object to be copied into a standalone entity. This would prevent changes to one object unintentionally affecting another. The Apache Commons Lang library and tutorial [35] were vital to the solution. The solution to my problem was just one line of deep copying code copied from the tutorial, but I did work in understanding the problem and how to solve it.

Exporting rotas into PDF files was an important technical and accessibility accomplishment on my project since they are so widely compatible [22] and making a program which outputs to PDF increased project complexity. My PDF code in RotaMenu.java is based on a tutorial [36] and uses the iTextPdf library, but I still did a lot of work. I learned how to use the library's stream based and top to bottom approach to PDF construction and adapted tutorial code to process rota data. Pictured below is code which generates a well designed document.

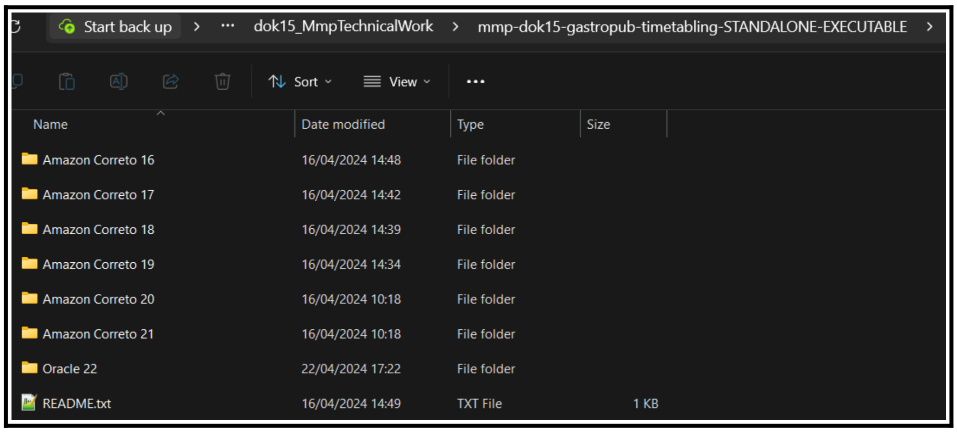


```
myPdfDoc.add( rotaHeader );
Paragraph notesBit = new Paragraph( string: "Notes: " + this.employeeRotaPassedFromMain.getNotes(), bodyFont );
myPdfDoc.add( notesBit );
for( Day d : this.employeeRotaPassedFromMain.getDaysOfWeek() ){
    Paragraph dayHeader = new Paragraph( d.getDayOfWeek().name() , secondaryHeadingFont );
    dayHeader.setIndentationLeft(20);
    myPdfDoc.add( dayHeader );
    for( Shift s : d.getShifts() ){
        Paragraph shiftHeader = new Paragraph( string: s.getShiftTime().name() + ": " + s.getNumberOfStaff() + " working."
        shiftHeader.setIndentationLeft(40);
        myPdfDoc.add( shiftHeader );
        for( Employee e : s.getAllStaff() ){
            Paragraph employeeEntry = new Paragraph( string: e.getRole().name() + ": " + Employee.getProperCase(e.getFirst
            employeeEntry.setIndentationLeft(80);
            myPdfDoc.add( employeeEntry );
        }
    }
}
myPdfDoc.close();
```

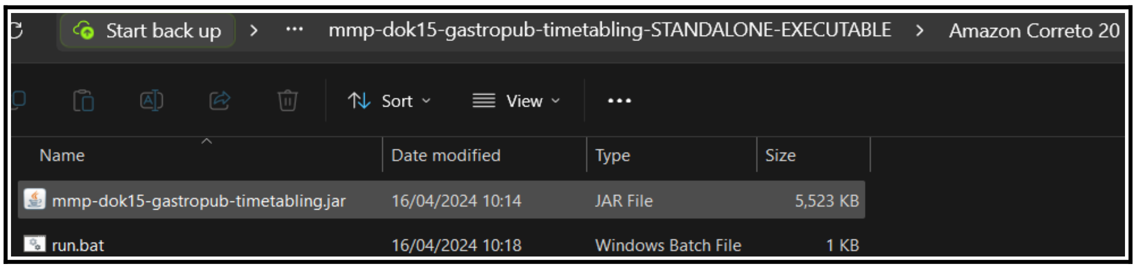
Work of designing PDF appearance, implementing the design, and using regular expressions to restrict file naming was mine too.

Making my program a standalone package which did not depend on Intellij was the final piece of technical work I did. Following a tutorial [37], I used the IDE to package my program into a JAR file but this was not enough. Clicking the file did nothing and my laptop's Command Prompt could not find Java. Following a tutorial [38], I resolved the locating Java issue by editing Windows' environment variables. Since my program largely uses CLI, I followed another tutorial [39] about how to open JAR files with Windows' Command Prompt. From past experience, I knew using Windows batch files would allow me to make a convenient executable for my program.

This initial packaging worked great on my laptop but on a computer in Llandinam C56, the executable would not open the JAR file and I would see no error message. Investigation was done by trying to manually open the file via Command Prompt, and from this I found out that the problem was the version of Java used to compile the JAR file cannot exceed the version of Java used by Windows' Command Prompt. I resolved this issue by making multiple standalone packages, each nearly identical with the only difference being the Java version they were compiled with. You can see the products below.



Name	Date modified	Type	Size
Amazon Correto 16	16/04/2024 14:48	File folder	
Amazon Correto 17	16/04/2024 14:42	File folder	
Amazon Correto 18	16/04/2024 14:39	File folder	
Amazon Correto 19	16/04/2024 14:34	File folder	
Amazon Correto 20	16/04/2024 10:18	File folder	
Amazon Correto 21	16/04/2024 10:18	File folder	
Oracle 22	22/04/2024 17:22	File folder	
README.txt	16/04/2024 14:49	TXT File	1 KB



Name	Date modified	Type	Size
mmp-dok15-gastropub-timetabling.jar	16/04/2024 10:14	JAR File	5,523 KB
run.bat	16/04/2024 10:18	Windows Batch File	1 KB

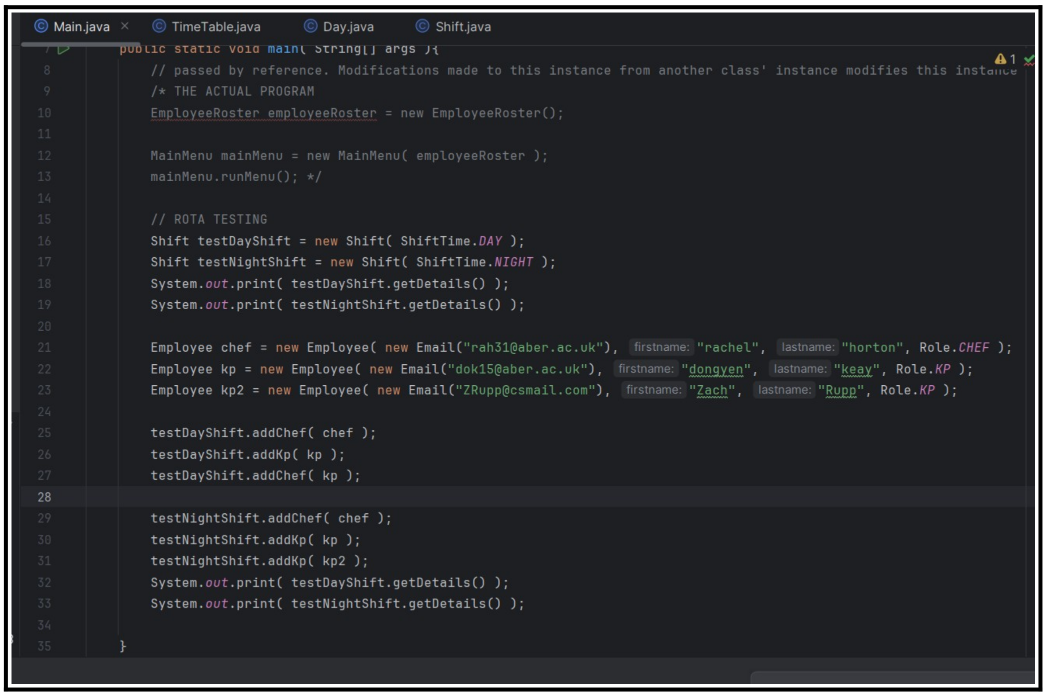
I got the package running on a campus computer by using “java -version” to findout which version of Java was used by its Command Prompt. Then I would select the most suitable version of the software and run it successfully.

4. Testing

A mixture of automated unit testing and manual test table work was used to check my program's correctness.

For automated testing, I wrote code based on a generic JUnit tutorial [21], and the tests can be found in a package of my program: mmp-dok15-gastropub-timetabling\src\unitTests. I approached testing in a test as you code manner. For example, when I wrote an email validator method, then wrote tests to see if it correctly handled different inputs. My testing process was to write then run a test and depending on if it passed or failed, I would write another test or fix the issue.

For manual testing, I made a detailed test table of tests to be conducted manually and from top to bottom, which can be seen in appendix Z. Throughout development I only did unit testing and ad-hoc testing, with the latter involving debugger print statements and replacing my Main.java code with code to exercise some classes or methods I just wrote.



```
    public static void main( String[] args ){
        // passed by reference. Modifications made to this instance from another class' instance modifies this instance
        /* THE ACTUAL PROGRAM
        EmployeeRoster employeeRoster = new EmployeeRoster();

        MainMenu mainMenu = new MainMenu( employeeRoster );
        mainMenu.runMenu(); */

        // ROTA TESTING
        Shift testDayShift = new Shift( ShiftTime.DAY );
        Shift testNightShift = new Shift( ShiftTime.NIGHT );
        System.out.print( testDayShift.getDetails() );
        System.out.print( testNightShift.getDetails() );

        Employee chef = new Employee( new Email("rah31@aber.ac.uk"), firstname: "rachel", lastname: "horton", Role.CHEF );
        Employee kp = new Employee( new Email("dok15@aber.ac.uk"), firstname: "dongyen", lastname: "keay", Role.KP );
        Employee kp2 = new Employee( new Email("ZRupp@csmail.com"), firstname: "Zach", lastname: "Rupp", Role.KP );

        testDayShift.addChef( chef );
        testDayShift.addKp( kp );
        testDayShift.addChef( kp );

        testNightShift.addChef( chef );
        testNightShift.addKp( kp );
        testNightShift.addKp( kp2 );
        System.out.print( testDayShift.getDetails() );
        System.out.print( testNightShift.getDetails() );

    }
```

It was with ad-hoc testing that I found issues to fix. I only made the test table once I was mostly finished with the project so that I could thoroughly check if its quality was acceptable.

My testing uncovered issues to fix and now shows my program is mostly functional.

5. Critical Evaluation

My project is flawed and basic in places but overall, it is quite good and deserving of about 50%.

Understanding the Problem

I am quite confident that the initial problem was understood well. From the start, I have sought a solution to timetabling for specific scenarios, and quickly settled on making a weekly rota creation system for a gastropub manager. There was some uncertainty about how well I understood the problem, since many timetables like ones on Outlook Calendar and Student Record look like appendix B and are for scheduling group events. The staff rota in appendix A is my project’s main inspiration and it is a timetable which details a “plan showing when events or activities will happen” [43]. A plan for me to show up for work on a Friday evening shift fits the definition comfortably, so despite my rota looking so unlike appendix B, it is still a timetable.

Implementation

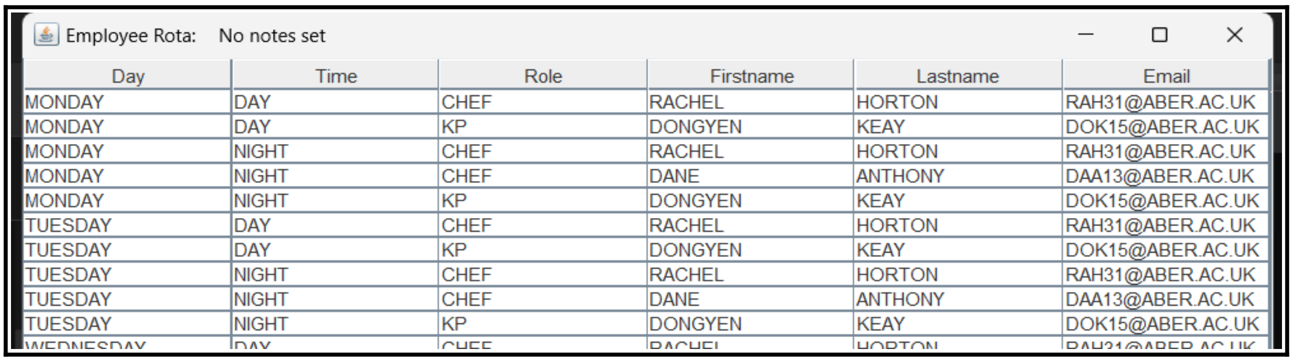
Weaknesses

The technical work has some important failings. Its UI is quite basic, since it is mostly used via a CLI. This is a weakness because GUI is the standard for commercial user software and this program mostly not using it makes it seem worse than competitors. Also GUI tends to be more intuitive and usable for non technical users, and it is fair to assume that a gastropub manager is not guaranteed to be excellent at using computers.

System fragility is a problem, since it is not entirely self dependent. The roster is saved to a database on campus and if that is down, an important aspect of the system: staff roster persistence, will not exist. This could have been resolved by having a backup persistence method like Java serialisation, which would have no external dependencies.

Managers can use the system to record which shifts employees are unavailable for, but the restrictions are ignored by my method which copies the rota for one day over to the rest of the week. This quality of life feature can defeat its own purpose by generating invalid rotas.

The GUI rota viewing tool is quite hard to read, as you can see below.


Employee Rota: No notes set

Day	Time	Role	Firstname	Lastname	Email
MONDAY	DAY	CHEF	RACHEL	HORTON	RAH31@ABER.AC.UK
MONDAY	DAY	KP	DONGYEN	KEAY	DOK15@ABER.AC.UK
MONDAY	NIGHT	CHEF	RACHEL	HORTON	RAH31@ABER.AC.UK
MONDAY	NIGHT	CHEF	DANE	ANTHONY	DAA13@ABER.AC.UK
MONDAY	NIGHT	KP	DONGYEN	KEAY	DOK15@ABER.AC.UK
TUESDAY	DAY	CHEF	RACHEL	HORTON	RAH31@ABER.AC.UK
TUESDAY	DAY	KP	DONGYEN	KEAY	DOK15@ABER.AC.UK
TUESDAY	NIGHT	CHEF	RACHEL	HORTON	RAH31@ABER.AC.UK
TUESDAY	NIGHT	CHEF	DANE	ANTHONY	DAA13@ABER.AC.UK
TUESDAY	NIGHT	KP	DONGYEN	KEAY	DOK15@ABER.AC.UK
WEDNESDAY	DAY	CHEF	RACHEL	HORTON	RAH31@ABER.AC.UK

It is functional, but the simplistic formatting increases the work needed to understand the rota. This design could be improved if instead of the duplicate day and shift times, they would only appear once as headers.

Integrity between the roster and rota is not checked or enforced. My system treats these entities as being very separate, so if an employee is deleted from the roster, they will remain on the rota. My system can create faulty rotas in which the unemployed are scheduled to work. This could be improved with some code which runs whenever an employee is deleted, which deletes any references to that employee from the rota.

Rota persistence done through serialisation only allows one persisted rota at a time. This makes it harder for managers to plan staffing for multiple weeks. The problem could be resolved by having code which allows users to serialise a rota with a custom filename. This sort of system is already in place with my PDF exporter.

The menus for my system can be too large and bloated. For example, the roster menu lets users add employees via a CLI or GUI. This is because I built my system in a very incremental manner and did not remove outmoded features because of new GUI alternatives.

Strengths

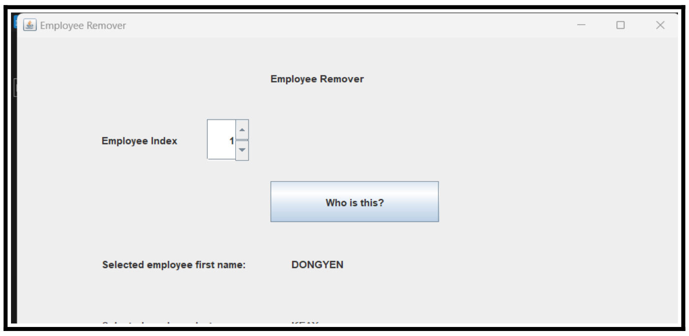
My project solves the problem of timetabling for specific scenarios. It allows users to create staff rosters, do weekly timetabling and then view or persist the rota in a variety of ways. It is not a question of “does my project offer a solution?”, but “how good is the solution?”.

Employee information storage is quite robust. Users can input details for an employee to be entered into the roster, which enforces integrity restrictions. Employee uniqueness is determined by their email. My code enforces email validity using regexp and employee uniqueness is enforced with rejecting entries with non unique emails. The result is an employee uniqueness system based on valid emails.

SQL injections are prevented with simple regular expressions in well picked places. The staff roster system is the only interface a user has with a database, so inputs of employee data will be rejected if a semi colon is detected. The system works, but it is very simplistic. The simplicity does not damage the system much because there is nowhere in the roster where a semi colon would be needed, except for maybe a very strange email.

My system is quite complex with some very different components. Staff roster data can be persisted to and read from a database on a campus server. This means my desktop application talks with a database component and both are aspects of my system. The interaction works well and represents a significant technical accomplishment since I have not programmed anything similar before.

Java Swing was used to good effect. It was used to create intuitive and restrictive graphical tools. Pictured below is an example.

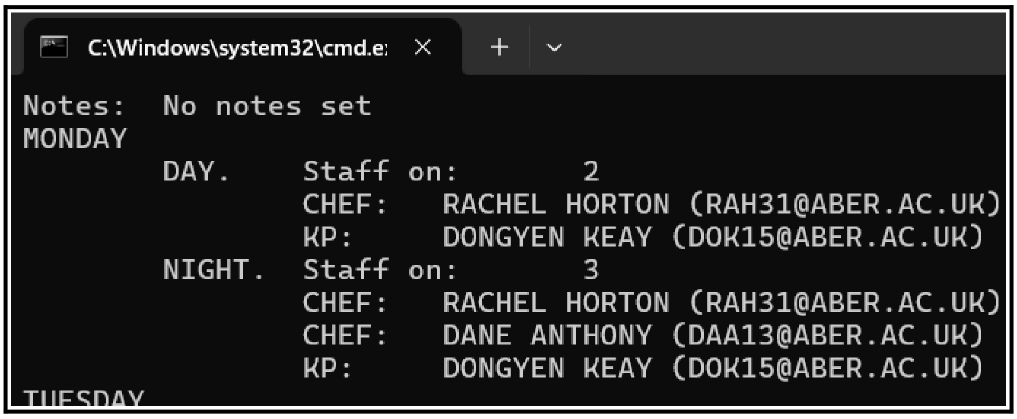


Employee Index 1

Selected employee first name: DONGYEN

The spinner restricts inputs to valid roster indices, which reduces how much thinking a user has to do and means my program does not need as much data validation code.

The CLI rota viewer, though basic, is very functional. Pictured below is the viewer.

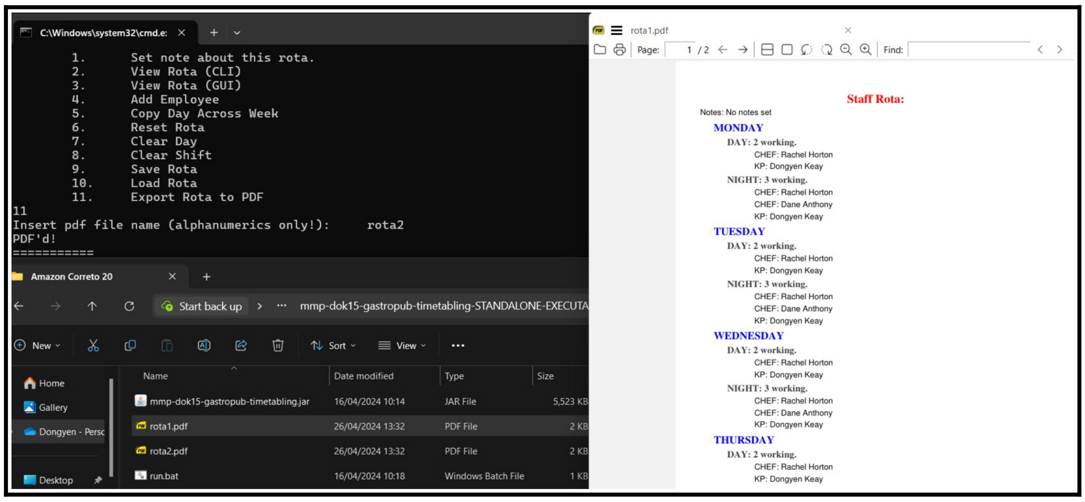


```
Notes:  No notes set
MONDAY
        DAY.    Staff on:       2
                CHEF:   RACHEL HORTON (RAH31@ABER.AC.UK)
                KP:     DONGYEN KEAY (DOK15@ABER.AC.UK)
        NIGHT.  Staff on:       3
                CHEF:   RACHEL HORTON (RAH31@ABER.AC.UK)
                CHEF:   DANE ANTHONY (DAA13@ABER.AC.UK)
                KP:     DONGYEN KEAY (DOK15@ABER.AC.UK)
TUESDAY
```

Users can quickly see what the timetabling is for any day or shift.

The GUI rota viewer, though harder to read, is a good technical accomplishment. Using Java Swing and a table, I made a functional viewer which significantly increased the range and complexity of GUI tools my system provides. The tool’s development also required a lot of work, since I had to adapt some tutorial code to work with a changeable rota instead of the tutorial’s hardcoded array [34].

Although rota serialisation only allows one rota to be persisted, the PDF exporting function allows any number of rotas to exist at once. Pictured below is this system in use.


C:\Windows\system32\cmd.e:

```
        1.      Set note about this rota.
        2.      View Rota (CLI)
        3.      View Rota (GUI)
        4.      Add Employee
        5.      Copy Day Across Week
        6.      Reset Rota
        7.      Clear Day
        8.      Clear Shift
        9.      Save Rota
        10.     Load Rota
        11.     Export Rota to PDF
11
Insert pdf file name (alphanumerics only!):     rota2
PDF'd!
===========
```

Amazon Correto 20
Start back up
mmp-dok15-gastropub-timetabling-STANDALONE-EXECUTA
New
Sort
View
Home
Gallery
Dongyen - Persc
Desktop

Name	Date modified	Type	Size
mmp-dok15-gastropub-timetabling.jar	16/04/2024 10:14	JAR File	5,523 KB
rota1.pdf	26/04/2024 13:32	PDF File	2 KB
rota2.pdf	26/04/2024 13:32	PDF File	2 KB
run.bat	16/04/2024 10:18	Windows Batch File	1 KB

Staff Rota:

Notes: No notes set

MONDAY
DAY: 2 working.
CHEF: Rachel Horton
KP: Dongyen Keay
NIGHT: 3 working.
CHEF: Rachel Horton
CHEF: Dane Anthony
KP: Dongyen Keay

TUESDAY
DAY: 2 working.
CHEF: Rachel Horton
KP: Dongyen Keay
NIGHT: 3 working.
CHEF: Rachel Horton
CHEF: Dane Anthony
KP: Dongyen Keay

WEDNESDAY
DAY: 2 working.
CHEF: Rachel Horton
KP: Dongyen Keay
NIGHT: 3 working.
CHEF: Rachel Horton
CHEF: Dane Anthony
KP: Dongyen Keay

THURSDAY
DAY: 2 working.
CHEF: Rachel Horton
KP: Dongyen Keay

The PDF and CLI rota viewers have very similar readable designs, which enable readers to quickly gleam timetable information. Implementing a PDF exporter was a significant technical accomplishment as I had to heavily adapt tutorial code [36] and take creative liberties so that a staff rota could be exported. It also increases the complexity of my system quite a lot by using a library to implement a powerful feature. The effort to implement this feature was worthwhile because PDFs are a very accessible file format that can seen identically on many devices [22]. This makes the format ideal for employee rota distribution.

Report

The report is quite good, because its creation was very driven by a guide [44], but there are flawed areas like the monolithic implementation section.

The report is thorough because just about everything meant to be in the report seems to be in [44]. This means it is a good complement to the technical work different stages like planning are discussed.

The report is quite balanced because it analyses the strengths and weaknesses project work. This means a reader can get a realistic view of accomplishment and make their own judgement on overall quality.

The report's weaknesses include inconsistent sentence professionalism since there was no consistent effort to avoid first person sentences. The overall quality suffers from the rushed nature of report work, which shows in the poorly segmented implementation section.

6. References

[1] Web Development Company in India. 2022. Top Advantages & Disadvantages of Web Application. LinkedIn. Accessed on 15/02/2024. https://www.linkedin.com/pulse/top-advantages-disadvantages-web-application-aalpha-muzammil.

Considering desktop application vs web application.

[2] Pop, Paul. 2002. Comparing Web Applications with Desktop Applications: An Empirical Study. Technical University of Denmark. Accessed on 15/02/2024. http://www.imm.dtu.dk/~paupo/publications/Pop2002aa-Comparing%20Web%20Applications%20wit-.pdf.

Since the article is decades old and has an abstract mentioning web browsers' "restricted interaction mechanisms", I suspect the article's findings may be out of date.

[3] Roberts, Sienna. 2023. C vs Java: A Comprehensive Guide. The Knowledge Academy. Accessed on 15/02/2024. https://www.theknowledgeacademy.com/blog/c-vs-java/.

Is Java the best choice?

[4] Svoboda, David. 2015. Is Java More Secure Than C? Software Engineering Institute of Carnegie Mellon University. Accessed on 15/02/2024. https://insights.sei.cmu.edu/blog/is-java-more-secure-than-c/.

[5] 7.5 Plain text files. Institute for Statistics and Mathematics at Vienna University of Economics and Business. Accessed on 15/02/2024. https://statmath.wu.ac.at/courses/data-analysis/itdtHTML/node56.html.

Program data could be stored in text files.

[6] 7.6 XML. Institute for Statistics and Mathematics at Vienna University of Economics and Business. Accessed on 15/02/2024. https://statmath.wu.ac.at/courses/data-analysis/itdtHTML/node57.html.

Program data could be stored in XML files.

[7] 7.9 Databases. Institute for Statistics and Mathematics at Vienna University of Economics and Business. Accessed on 15/02/2024. https://statmath.wu.ac.at/courses/data-analysis/itdtHTML/node60.html.

Program data could be stored in a database.

[8] 2024. Introduction to Java Serialization. Baeldung. Accessed on 09/02/2024. https://www.baeldung.com/java-serialization.

[9] What is a TXT file and how to open, view and edit one. Adobe. Accessed on 19/02/2024. https://www.adobe.com/uk/acrobat/resources/document-files/text-files/txt.html.

[10] 2024. What are the advantages and disadvantages of using plain text versus rich text formats? Linkedin. Accessed on 19/02/2024. https://www.linkedin.com/advice/1/what-advantages-disadvantages-using-plain.

[11] Kivi, Meelika. 2022. PDF vs DOCX: Which format is better? Accessed on 19/02/2024. https://pdfmailmerger.com/blog/pdf-vs-docx-which-format-is-better/.

[12] 2021. Pros and cons of PDFs. Appalachian State University. Accessed on 19/02/2024. https://accessibility.appstate.edu/news/pros-and-cons-pdfs.

[13] Krukowska, Dominika. Why PDF Advantages Are Not Worth the Risky Disadvantages. Storydoc. Accessed on 19/02/2024. https://www.storydoc.com/blog/what-is-the-advantage-and-disadvantage-of-pdf.

[14] Java AWT Tutorial. Geeks for Geeks. Accessed on 27/02/2024. https://www.geeksforgeeks.org/java-awt-tutorial/.

[15] Mansi. 2023. Difference Between AWT and Swing in Java. Accessed on 27/02/2024. https://www.scaler.com/topics/difference-between-awt-and-swing/.

[16] Introduction to Java Swing. Geeks for Geeks. Accessed on 27/02/2024. https://www.geeksforgeeks.org/introduction-to-java-swing/.

[17] JavaFX Tutorial. Geeks for Geeks. Accessed on 27/02/2024. https://www.geeksforgeeks.org/javafx-tutorial/.

[18] 2023. Git vs GitHub: Difference Between Git and GitHub. SimpliLearn. Accessed on 15/02/2024. https://www.simplilearn.com/tutorials/git-tutorial/git-vs-github.

[19] 2020. The 2020 Scrum Guide. Scrum Guides. Accessed on 15/02/2024. https://scrumguides.org/scrum-guide.html.

[20] Lutkevich, Ben. What is the waterfall model? TechTarget. Accessed on 15/02/2024. https://www.techtarget.com/searchsoftwarequality/definition/waterfall-model#:~:text=The%20waterfall%20model%20is%20a,the%20edge%20of%20a%20cliff.

[21] Vogel, Lars. 2021. Vogella. Accessed on 08/02/2024. https://www.vogella.com/tutorials/Junit/article.html.

[22] 2021. Pros and cons of PDFs. Appalachian State University. Accessed on 19/02/2024. https://accessibility.appstate.edu/news/pros-and-cons-pdfs.

[23] Java Regular Expressions. W3 Schools. Accessed on 16/02/2024. https://www.w3schools.com/java/java_regex.asp.

[24] Quick-Start: Regex Cheat Sheet. Rex Egg. Accessed on 16/02/2024. https://www.rexegg.com/regex-quickstart.html.

[25] Exploits Of A Mom. XKCD. Accessed on: 27/02/2024. https://xkcd.com/327/.

[26] PostgreSQL - Java Interface. Tutorials Point. Accessed on: 19/02/2024. https://www.tutorialspoint.com/postgresql/postgresql_java.htm.

[27] Rekin, 2010. StackOverflow. Accessed on 14/03/2024. https://stackoverflow.com/questions/3904579/how-to-capitalize-the-first-letter-of-a-string-in-java.

[28] JLabel | Java Swing. Geeks for Geeks. Accessed on 21/02/2022. https://www.geeksforgeeks.org/jlabel-java-swing/.

[29] Java JTextField. Javatpoint. Accessed on 27/02/2024. https://www.javatpoint.com/java-jtextfield.

[30] Java JComboBox. Javatpoint. Accessed on 21/02/2022. https://www.javatpoint.com/java-jcombobox.

[31] Java JSpinner. Javatpoint. Accessed on 27/02/2024. https://www.javatpoint.com/java-jspinner.

[32] Java JButton. Javatpoint. Accessed on 21/02/2022. https://www.javatpoint.com/java-jbutton.

[33] setDefaultCloseOperation(). Central Conneticut State University. Accessed on 21/02/2022. https://chortle.ccsu.edu/java5/notes/chap56/ch56_9.html.

[34] Creating a Scrollable JTable. Roseindia. Accessed on 01/03/2024. https://www.roseindia.net/java/example/java/swing/ScrollableJTable.shtml.

[35] Gurgul, Adam. 2024. How to Make a Deep Copy of an Object in Java. Baeldung. Accessed on 05/03/2024. https://www.baeldung.com/java-deep-copy.

[36] 2024. Creating PDF Files in Java. Baeldung. Accessed on: 19/02/2024. https://www.baeldung.com/java-pdf-creation.

[37] BoostMyTool. 2021. Intellij IDEA: Create an Executable JAR File with External Libraries. YouTube. Accessed on 15/04/2024. https://youtu.be/_XQjs1xGtaU.

[38] How to Fix Your Computer. 2020. javac is not recognised as an internal or external command Windows 10 \ 8 \ 7 Fixed. YouTube. Accessed on 15/04/2024. https://youtu.be/NmFzSyf7TTQ.

[39] Tamil Arasu10. 2023. PathFinder Community. Accessed on 15/04/2024. https://community.automationanywhere.com/developers-forum-36/how-to-open-the-jar-files-from-aa360-and-how-to-type-commands-in-command-prompt-using-aa360-87146.

[40] Java Create and Write to Files. W3 Schools. Accessed on 01/02/2024. https://www.w3schools.com/java/java_files_create.asp.

[41] Java Read Files. W3 Schools. Accessed on 01/02/2024. https://www.w3schools.com/java/java_files_read.asp.

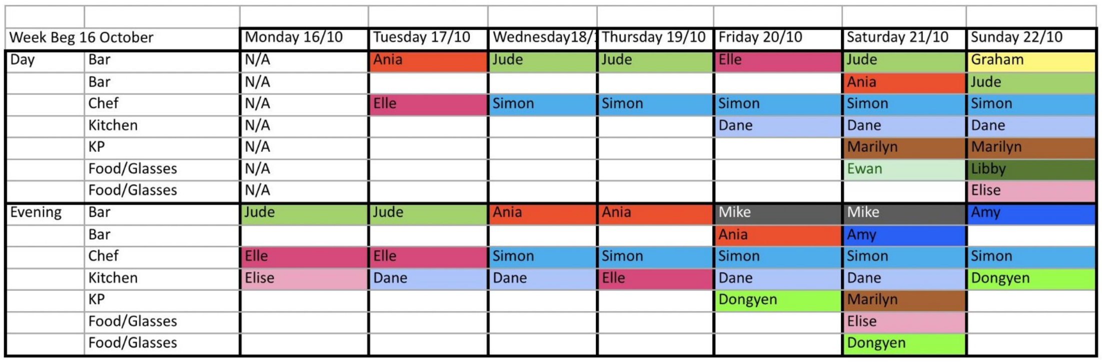
[42] 2024. Introduction to Java Serialization. Baeldung. Accessed on 09/02/2024. https://www.baeldung.com/java-serialization.

[43] timetable. Cambridge Dictionary. Accessed on 23/04/2024. https://dictionary.cambridge.org/dictionary/english/timetable.

[44] Taylor, Neil. “9. Report and Technical Work”. MMP Information. Accessed on 29/04/2024. https://teaching.dcs.aber.ac.uk/docs/2022/MMP/information/report-and-technical-work/index.html.

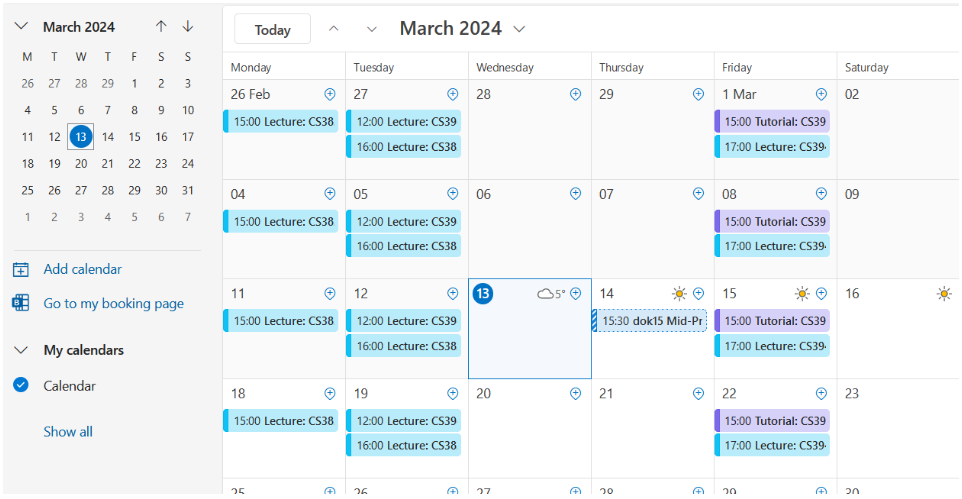
7. Appendices

A: Gastropub Rota



Week Beg 16 October		Monday 16/10	Tuesday 17/10	Wednesday18/1	Thursday 19/10	Friday 20/10	Saturday 21/10	Sunday 22/10
Day	Bar	N/A	Ania	Jude	Jude	Elle	Jude	Graham
	Bar	N/A					Ania	Jude
	Chef	N/A	Elle	Simon	Simon	Simon	Simon	Simon
	Kitchen	N/A				Dane	Dane	Dane
	KP	N/A					Marilyn	Marilyn
	Food/Glasses	N/A					Ewan	Libby
	Food/Glasses	N/A						Elise
Evening	Bar	Jude	Jude	Ania	Ania	Mike	Mike	Amy
	Bar					Ania	Amy	
	Chef	Elle	Elle	Simon	Simon	Simon	Simon	Simon
	Kitchen	Elise	Dane	Dane	Elle	Dane	Dane	Dongyen
	KP					Dongyen	Marilyn	
	Food/Glasses						Elise	
	Food/Glasses						Dongyen	

B: Outlook Calendar



March 2024

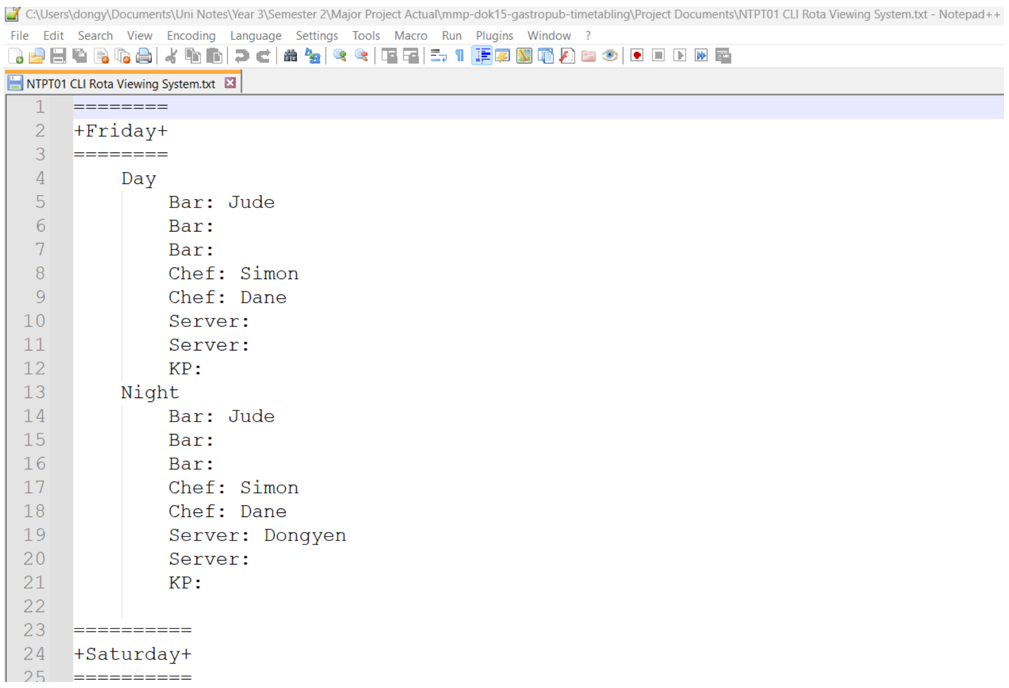
M	T	W	T	F	S	S
26	27	28	29	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31
1	2	3	4	5	6	7

Add calendar
Go to my booking page
My calendars
Calendar
Show all

Today March 2024

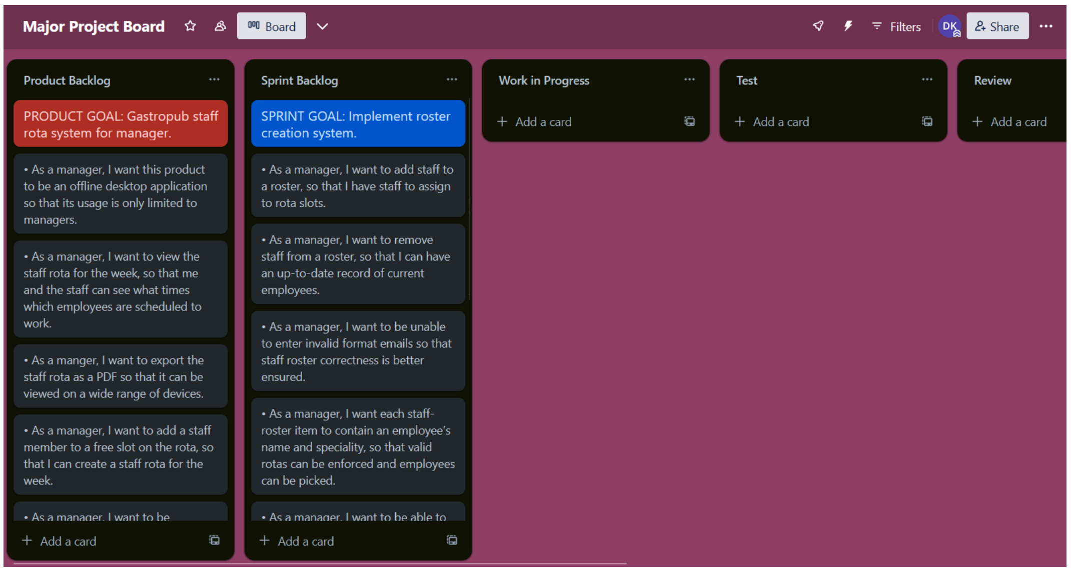
Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
26 Feb 15:00 Lecture: CS38	27 12:00 Lecture: CS39 16:00 Lecture: CS38	28	29	1 Mar 15:00 Tutorial: CS39 17:00 Lecture: CS39	02
04 15:00 Lecture: CS38	05 12:00 Lecture: CS39 16:00 Lecture: CS38	06	07	08 15:00 Tutorial: CS39 17:00 Lecture: CS39	09
11 15:00 Lecture: CS38	12 12:00 Lecture: CS39 16:00 Lecture: CS38	13 5°	14 15:30 dok15 Mid-Pr	15 15:00 Tutorial: CS39 17:00 Lecture: CS39	16
18 15:00 Lecture: CS38	19 12:00 Lecture: CS39 16:00 Lecture: CS38	20	21	22 15:00 Tutorial: CS39 17:00 Lecture: CS39	23

C: Prototype CLI Rota Inspired by Appendix A


C:\Users\dongy\Documents\Uni Notes\Year 3\Semester 2\Major Project Actual\mmp-dok15-gastropub-timetabling\Project Documents\NTPT01 CLI Rota Viewing System.txt - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
NTPT01 CLI Rota Viewing System.txt

```
========
+Friday+
========
    Day
        Bar: Jude
        Bar:
        Bar:
        Chef: Simon
        Chef: Dane
        Server:
        Server:
        KP:
    Night
        Bar: Jude
        Bar:
        Bar:
        Chef: Simon
        Chef: Dane
        Server: Dongyen
        Server:
        KP:

==========
+Saturday+
==========
```

D: My Scrum Implementation



Product Backlog

PRODUCT GOAL: Gastropub staff rota system for manager.

- As a manager, I want this product to be an offline desktop application so that its usage is only limited to managers.
- As a manager, I want to view the staff rota for the week, so that me and the staff can see what times which employees are scheduled to work.
- As a manger, I want to export the staff rota as a PDF so that it can be viewed on a wide range of devices.
- As a manager, I want to add a staff member to a free slot on the rota, so that I can create a staff rota for the week.
- As a manager, I want to be

Sprint Backlog

SPRINT GOAL: Implement roster creation system.

- As a manager, I want to add staff to a roster, so that I have staff to assign to rota slots.
- As a manager, I want to remove staff from a roster, so that I can have an up-to-date record of current employees.
- As a manager, I want to be unable to enter invalid format emails so that staff roster correctness is better ensured.
- As a manager, I want each staff-roster item to contain an employee's name and speciality, so that valid rotas can be enforced and employees can be picked.
- As a manager, I want to be able to

Z: Test Table

Test ID	Description	Test Data	Desired Result	Actual Result	Passed
MM01	Invalid main menu input.	Bread	Error message output followed by return to main menu.	Desired.	Yes.
MM02	Invalid main menu input. Boundary testing.	-1	Error message output followed by return to main menu.	Desired.	Yes.
MM03	Invalid main menu input. Boundary testing.	3	Error message output followed by return to main menu.	Desired.	Yes.
MM04	Exiting app from main menu.	0	A farewell message followed by app termination.	Desired. And process finished with exit code 0 (no error).	Yes.
MM05	Accessing roster menu.	1	Prints roster menu.	Desired.	Yes.
MM06	Accessing rota menu.	2	Prints rota menu.	Desired.	Yes.
RS01	Backing out from roster menu to main menu.	0	Prints main menu.	Desired.	Yes.
RS02	Trying to remove employee from empty roster.	3	Error message followed by return to roster menu (RM).	Desired.	Yes.
RS03	Trying to open GUI roster index querying / employee removal tool.	4	Error message followed by return to RM. No GUI opens.	Desired.	Yes.
RS04	Trying to remove all employees from an empty roster.	5	Error message followed by return to RM.	Desired.	Yes.
RS05	Trying to view an empty roster.	6	Prints roster header and number of employees (0).	Desired.	Yes.
RS06	Trying to add an unavailable time for an employee when roster empty.	9	Error message then return to RM.	Desired.	Yes.
RS07	Try view employee	10	Error message then	Desired.	Yes.

	unavailable times when roster empty.		print RM.		
RS08	Clear employee unavailable times when roster empty.	11	Error message then print RM.	Desired.	Yes.
RS09	Invalid roster menu input.	Bread	Error message then print RM.	Desired.	Yes.
RS10	Invalid roster menu input.	-1	Error message then print RM.	Desired.	Yes.
RS11	Invalid roster menu input.	12	Error message then print RM.	Desired.	Yes.
RS12	Add employee to roster CLI.	1, dok15@aber.ac.uk, dongyen, keay, 4.	Message saying employee added. Printing correct details of said employee.	Desired.	Yes.
RS13	Check that RS12 worked.	6.	Prints roster with 1 employee: the one I just created.	Desired.	Yes.
RS14	Check that RS12 worked.	4, select 1 on spinner, press “who is this?”.	GUI tool opens. GUI tools shows the RS12 employee.	Desired.	Yes.
RS15	Try add employee who is duplicate of RS12’s.	1, DOK15@ABER.AC.UK.	Error message about email duplicate. Return to RM.	Desired.	Yes.
RS16	Try add RS12’s duplicate employee.	2, then dok15@ABER.ac.uk, dongyen, keay, SERVER in appropriate fields. Press “add employee”.	GUI opens. GUI closes and CLI prints error message and returns to RM.	Desired.	Yes.
RS17	Check no duplicate employees added.	6	Prints roster with just the RS12 employee.	Desired.	Yes.
RS18	Show that employee uniqueness based on email.	2, then dok15@aber.ac.uk,	Error message about duplicate email. Return to	Desired.	Yes.

		rachel, rupp, CHEF, press “add employee”.	RM.		
RS19	Add employee to roster GUI.	2, then rar31@aber.ac.uk, rachel, rupp, CHEF, press “add employee”.	Prints message about successful operation along with details of new staff.	Desired.	Yes.
RS20	Check new employee added.	6	Prints roster with employees from RS12 then RS19.	Desired.	Yes.
RS21	Save to PGSQL database (DB).	7	Prints that operation successful.	Desired.	Yes.
RS22	Clear roster.	5	Prints that roster cleared.	Desired.	Yes.
RS23	Check roster cleared.	6	Prints empty roster.	Desired.	Yes.
RS24	Load roster from DB	8	Prints operation successful.	Desired.	Yes.
RS25	Check roster loaded	6	Prints same roster as in RS20	Desired.	Yes.
RS26	Invalid emails blocked.	1, then dok@aber.	Error message about invalid email.	Desired.	Yes.
RS27	Invalid emails blocked by GUI.	2, the dok@aber, don, k, KP, “add employee”.	GUI opens. GUI closes. CLI prints email error message. Return to RM.	Desired.	Yes.
RS28	Previous employee add attempt failed.	6	Roster shows the same 2 employees from before.	Desired.	Yes.
RS29	Adding a bar man.	1, PJ_paulJohn-borth.bread@csmail.com, paul, john, 1.	Employee added and their details shown.	Desired.	Yes.
RS30	Adding a server.	2, Meg, Thomas,	Employee added and their details	Desired.	Yes.

		borthHead@gmail.com, SERVER	shown.		
RS31	Check the employee roster has the 4 staff.	6.	All added staff are present.	Desired.	Yes.
RS32	Save them to DB.	7.	Success message.	Desired.	Yes.
RS33	Setting up for employee removal test.	1, remove1@gmail.com, remove, 1, 1.	Success message.	Desired.	Yes.
RS34	Setting up for employee removal test.	2, remove2@gmail.com, remove, 2, BAR.	Success message.	Desired.	Yes.
RS35	Remove employee CLI.	3, 5. Then check with rota viewer (6).	Employee from RS33 removed.	Desired.	Yes.
RS36	Remove employee GUI.	4, then select 5 on spinner and click delete. Check with rota viewer.	Employee from RS34 removed.	Desired.	Yes.
RS37	Make Paul John unavailable to work on Sundays.	9, then 3, 7, 2. 9 then 3, 7, 1, 9 then 3, 7, 2.	Messages printed saying Paul can’t work Sunday day or night. Error message printed about duplicate times.	Desired.	Yes.
RS38	Check Paul is recorded as being unavailable on Sunday. No sensible sorting of times. Not persisted.	10, 3.	Prints that Paul unavailable to work on Sunday night then day. No duplicate times.	Desired.	Yes.
RS39	Record that Meg can’t work Monday night and check this is added.	9, 4, 1, 2. 10, 4.	Monday night unavailability added.	Desired.	Yes.
RS40	Clear unavailable times for Meg then check.	11, 4. 10, 4.	Monday night unavailability removed.	Desired.	Yes.
RS41	Prevent SQL injection CLI.	1, zar37@aber.ac.	Error message given about SQL	Desired.	Yes.

		uk, zach, rob’); DROP TABLE staff; --. Check with roster viewer.	injection attempt. Employee not added.		
RS42	Prevent SQL injection GUI.	2, zar37@aber.ac.uk, zach, rob’); DROP TABLE staff; --, CHEF. Click add employee. Check with roster viewer.	Error message given about SQL injection attempt. Employee not added.	Desired.	Yes.
RT01	View empty CLI rota.	2.	Just the skeleton of a rota is there.	Desired.	Yes.
RT02	View empty GUI rota.	3.	Just the headers.	Desired.	Yes.
RT03	Set note about rota. And check it.	1, rota time, 2.	Rota skeleton now has the note.	Desired.	Yes.
RT04	Try to add employee when roster is empty.	Clear employee roster. Then in rota menu, input 4. Load roster from DB after this! And re-add unavailable times from RS37.	Error message about empty roster.	Desired.	Yes.
RT05	Export empty rota with note.	11, emptyRota. Then in project directory, open emptyRota.pdf.	A PDF rota near identical to that seen in RT03 is created.	Desired.	Yes.
RT06	Try export rota with invalid name.	11, newRota.png.	Error message about name. No PDF created.	Desired.	Yes.
RT07	Try export rota with invalid name.	11, rota_13	Error message about name. No PDF created.	Desired.	Yes
RT08	Export rota with valid name.	11, rota13.	PDF created identical to RT05’s.	Desired.	Yes.

RT09	Add employee for a time they're unavailable for.	4, 7, 1, 3.	Error message about Paul not able to work Sunday days.	Desired.	Yes.
RT10	Add employee for a time they're unavailable for.	4, 7, 2, 3. Check rota.	Error message about Paul not able to work Sunday nights. Rota still empty	Desired.	Yes.
RT11	Add chef for Monday.	4, 1, 1, 2, 2. 4, 1, 2, 2, 2. View rota.	Rachel working all Monday.	Desired.	Yes.
RT12	Prevent duplicate employees in a shift.	4, 1, 1, 2, 2. View Rota.	Error message about Rachel already working. No duplicate staff added to shift.	Desired.	Yes.
RT13	Prevent employees being added to positions that aren't their roles.	4, 1, 1, 3, 4. View rota.	Error message about incorrect roles. No staff added to shift.	Desired.	Yes.
RT14	Populate Monday rota.	4, 1, 1, 3, 1. 4, 1, 2, 3, 1. 4, 1, 2, 1, 4. 4, 1, 2, 4, 3. View rota.	2 people working day. 4 people working night.	Desired.	Yes.
RT15	Copy Monday rota across week.	5, 1. View rota.	Monday rota duplicate for whole week.	Desired.	Yes.
RT16	Unavailable times constraint is followed by copy-across code.		Paul is not assigned to Sunday.	Paul is assigned to Sunday. The copy across code doesn't check time constraints.	No.
RT17	Can clear Monday day.	8, 1, 1. View rota.	Monday day is cleared.	All shifts populated like in RT15, except Monday day.	Yes.
RT18	Can clear Sunday.	7, 7. View rota.	Sunday is cleared. Both Monday day and Sunday are unstaffed.	Desired.	Yes.
RT19	Can add have multiple people with the same	Add new chef to roster called	Saturday has both Barry and Rachel	Desired.	Yes.

	roles on a shift.	"Barry Cheddar" who has email bc@gmail.com . Add Barry to Saturday night. View Rota.	as chefs.		
RT20	Export populated rota as PDF.	11, 20240313rota.	20240313rota.pdf created and is near identical to what's shown in RT19.	Desired.	Yes.
RT21	Save rota.	9.	Message about rota saved. "rotaDataFile" created in project directory.	Desired.	Yes.
RT22	Clear rota.	6. View rota.	Rota cleared and can view skeleton rota with default note.	Desired.	Yes.
RT23	Saved rota can be loaded.	10. View rota.	Rota loaded and is identical to what's seen in RT19.	Desired.	Yes.
RT24	Can manipulate a loaded rota.	Clear Tuesday day. Add Dongyen to Sunday evening. View rota.	Rota now has a vacant Tuesday day shift and Dongyen is working on Sunday night.	Desired.	Yes.