UCN dmai0916

1st year final project

Group 2

Scheduling System for Restaurant Fusion

6th June 2017

Mirjana Erceg
Tamas Kalapacs
Zahro-Madalina Khaji
Sangey Lama
Andreas Richardsen

University College of Northern Denmark

Technology and Business

Computer Science AP Degree Programme

dmai0916

Scheduling System for Restaurant Fusion

Project participants:

Mirjana Erceg
Tamas Kalapacs
Zahro-Madalina Khaji
Sangey Lama
Andreas Richardsen

Abstract:
In this report, we address the scheduling problems for restaurant Fusion with managing its waiting and dishwashing staff.
Inside we analyse the business situation of Fusion and evaluate the potential IT solutions to their issue. We as a group designed a scheduling tool to assist the managers in creating schedules more easily and storing employee information / preferences.
While the system was not fully completed, the most critical use cases were incepted, elaborated and realised.
Schedules can now be easily created and schedule information now persists in a database. Employees can be managed easily and information updated as needed.

Supervisor:

Lars Landberg Toftegaard

Submission Date:

6th June 2017

Contents

Introduction

For our 2nd semester final project we decided to partner with an Asian seafood gourmet restaurant called Fusion. Fusion has enjoyed success as a fine dining restaurant for the past 7 years and delivers a high level of service to its customers. However, they do face some problems currently such as their scheduling and planning systems for future months. A lot of time is spent co-ordinating staff shifts for each month and calculating the number of hours worked for each employee to meet their required hours and preferences. The system they currently use is time consuming, inefficient and prone to human error (using Excel and hand-written notes) for the managers, which they feel could be better spent elsewhere. In this report, we will conduct an analysis of the business and its practices, evaluate and recommend a plan of action to deal with the stated problems. Then we shall begin to design and implement a solution which should solve the problems in regard to Fusion's scheduling for employees and tracking employee working hours.

Group contract

- We agree to be on time for group meetings, if in any case we are late or cannot attend the meeting, we shall notify the other group members in advance.
- We will all work together on this project, share work equally and with give our maximum effort, meaning that we will help each other to overcome difficulties that we may encounter during our learning process.
- Generally, the work will be done in pairs / trios, but it is also possible that each of the group members do some parts of work individually. The completed segments will be reviewed by all group members for approval.
- We chose Sangey as our group leader and he will lead us throughout majority of the project, however, if any of the group members express the will for leadership, so it will be proposed and discussed within the group.
- All disagreements and / or proposed changes will be discussed among the group members until a consensus is reached.
- We would like to sharpen skills and expand our knowledge in the following subject areas: Programming, System development, Business, Technology and how they relate to each other.
- If without a justified reason, a group individual stops actively participating in group work and / or meetings, the group will examine the problem and try to solve it internally.
  If nothing changes and the individual continues with the aforementioned behaviour, he will be excluded from the final project and it will be noted in the Evaluation of group work in the report.
- The group meetings will take place at UCN Sofiendalsvej, Library and / or Godsbanen's common rooms at least once per week.

UP Phase plan

In the project, we will be using the Unified Process (UP) methodology, this includes practice of use-case driven model to determine the initial requirements of the system up to the consideration for the code. It also involves working in phases with specific milestones and dividing work up into iterations. (For the purposes of our 1st year project we will follow UP until the elaboration phase documenting only the most critical/important use cases.)

Table 1: Unified Process phase plan up to elaboration for this project.

Milestone / phase	Inception	Elaboration	Construction	Transition
Goal	Business analysis Business case System Vision	Critical system functionality has been designed, coded and tested.	Final use cases are designed and system functionality is completed and tested	Present and test the application in production environment. Feedback from Customer
Number of Iterations	-	2-3	4-5	1
Finish Date	30.04.2017	05.06.2017	-	-
Artefacts	• Organisational Structure • Culture Analysis • Competition analysis • SWOT analysis • Mission / Vision • Strategic goals • Business Case • System Vision • Use Cases • Prioritisation of Use cases	• Architecture considerations • Domain Model • Relational Model • Fully Dressed use cases • Mock ups • System Sequence diagrams • Interaction Sequence diagrams • Design Class diagram • Code prototype system • Prototype tests	-	-

Project Planning

For planning our group work we used a variety of resources, including messenger apps, MS Project and Trello to plan and assign tasks necessary for the project.

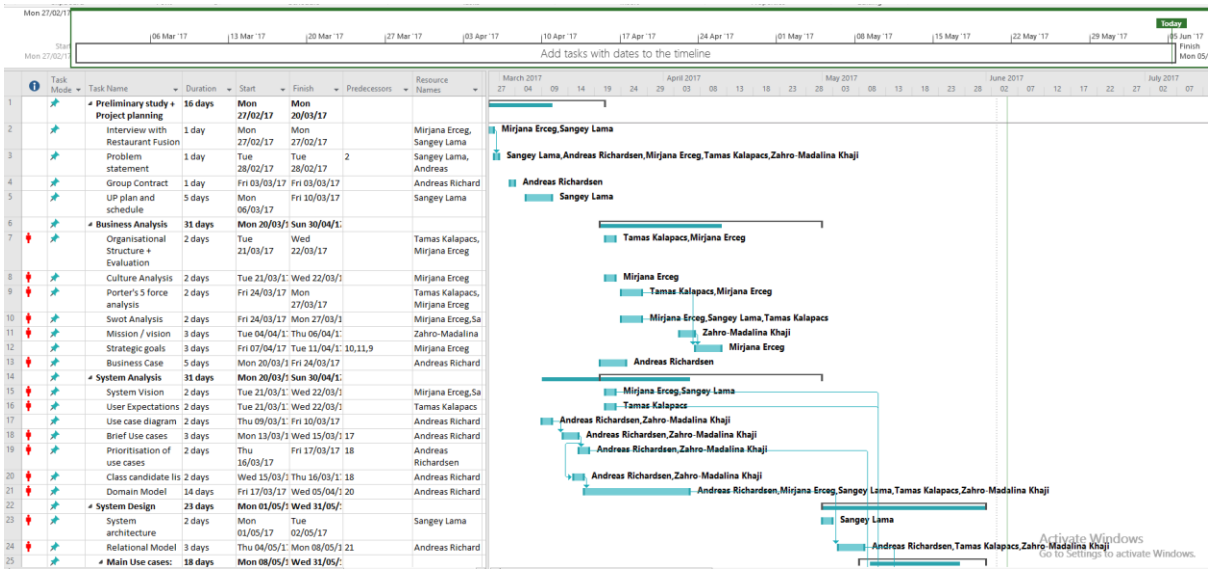

Figure 1: Screenshot of the group project plan with tasks and assigned responsibilities.

In the above figure is a screenshot from our MS Project plan where we break down the tasks into the necessary artefacts specified in the UP phase plan. We have rough estimated durations for each task with a start and end date as well as the assigned resources (project members). While there are some dependencies for certain tasks (e.g. Use case diagram -> Brief use cases -> prioritisation of uses case) a lot of tasks can be worked on independently and parallel to each other.

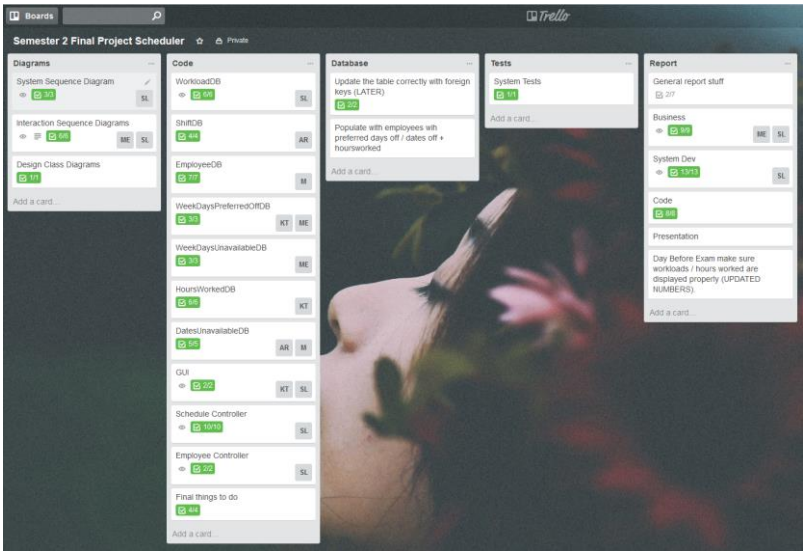

Figure 2: Screenshot of Trello used to delegate specific coding/diagram tasks in later phases of the project.

After the majority of group work concerning requirements was completed, we then switched to using Trello to subdivide the necessary coding / diagrams and testing that needed to be completed into cards with checklists. This allowed for greater flexibility in selecting work and then submitting it to the group without having to meet as regularly.

Business Analysis: Restaurant Fusion

Organisational Structure

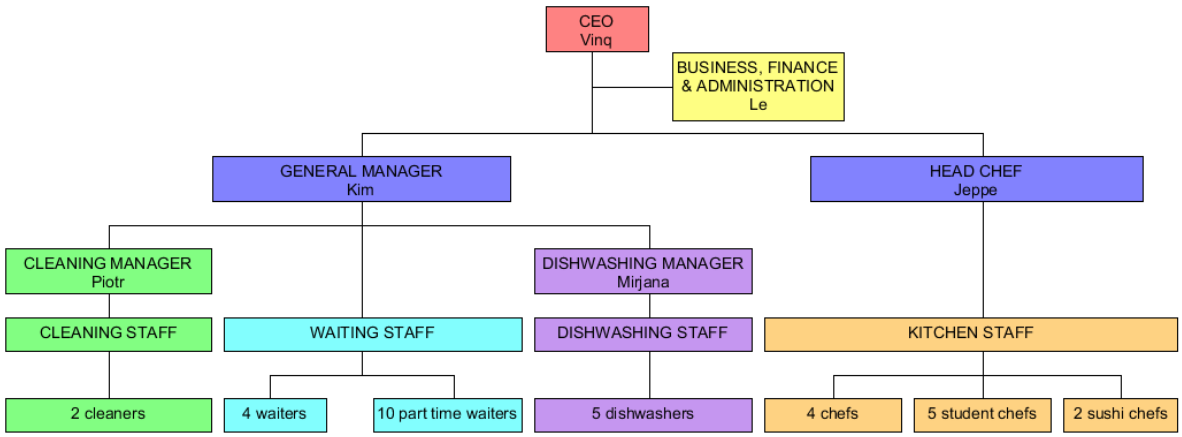

Figure 3: Organisational structure diagram for Restaurant Fusion.

Fusion's organisational structure is a functional structure, it is divided up into departments each with this own speciality / functional area. In this case; cleaning, waiting, dishwashing, kitchen and administration. Each functional area has its own functional head of operations who are responsible for communicating and co-ordinating with each other to ensure smooth operations.

For a restaurant of Fusion's size this structure is a good choice as it clearly defines the restaurants necessary functional areas and is flexible in how managers can control operations. It allows for departments to focus on their area of expertise and work together with other employees that have similar expertise and knowledge. Generally, this leads to more efficient business operations, however it could lead to potential tunnel-visioning. This means that employees from one functional area will see the business only from their functional point of view and this could lead to conflict / miscommunications between departments potentially leading to lower overall efficiency.

Culture/background

The restaurant has been open for 7 years and is owned and ran by a married couple. The husband's role of a CEO handles most of the restaurant's responsibilities and duties while the wife oversees all the business, financial and administrative tasks.
The organisational culture of the company can be described as a power culture. An organisation that relies on a central power (the owner and restaurant manager) which decides, controls or oversees all aspects of the company. Decisions are made quickly with little or no consultation and communicated informally to employees. This type of culture relies on trust both for the management to make the right decisions and for the employees to carry out tasks as delegated. In the end however, the success of a company with a power culture relies heavily on the owner / manager and the strength of their decisions.

The management style is very flexible, it is tightly controlled when the restaurant is busy and relaxed during slower hours. The CEO delegates the responsibilities between the Head Chef and General manager who then further divide the responsibilities, but the owners are involved in every sphere of the business. All the employees work as a team especially during busy hours when everyone is actively involved in helping each other.

The competitive situation analysis

1. **Threat of new entry**
   The threat of new entry in the Gourmet Asian restaurant industry is quite low. Knowledge to set up and run a restaurant like Fusion is very limited in the local area. Costs of setting up a similar restaurant are high.

2. **Competitive rivalry**
   The number of competitors in this specific area are very low or non-existent. There are a few more gourmet restaurants in this area but none of them offer Asian gourmet food. This makes the food they offer unique and gives them a lot of power in terms of competitive rivalry.

3. **Buyer power**
   The buyer power is quite low because the restaurant is not dependant on a few powerful buyers. A gourmet restaurant has a lower number of customers but the demand for Asian gourmet food is sufficient that the company does not worry too much about buyer power.

4. **Supplier power**
   Most of the seafood and drinks are ordered from local suppliers of which there are many other choices and switching is low cost, however some key suppliers are located abroad (Asia and Europe) and the number of importers are limited meaning Fusion is very dependent on these suppliers.

5. **Threat of substitution**
   The threat of substitution is low as making Asian gourmet food at home entails some obstacles. These include knowledge of specific cooking techniques, specialty ingredients and equipment, and finally, time and motivation to do so.

SWOT Analysis

Table 2: SWOT analysis table outlining Restaurant Fusion's current business position

Strengths	Weaknesses
The only Asian gourmet restaurant in the area	Many business tasks are done manually
They have established ties with local and overseas suppliers	Knowledge is centralised to the manager which makes him difficult to replace
Leaders are helping the employees to solve tasks as soon as possible during busy times, working together as a team	High turnover of waiting and dishwashing staff which requires constant need to replace and train new staff
They provide high quality food and service within the local area	High maintenance costs (e.g. rent, utilities) for the location
Some employees have been working there for years and have an extensive knowledge about that specific industry	Difficult working conditions could lead to conflict between the employees
The location of the restaurant is well selected	Limited growth
Opportunities	**Threats**
Possible expanding to other locations	Suppliers are late with the deliveries of products
Other sources to import their specialty goods	Ability to import specialty goods is affected eg. Trade embargo, overfishing, increased prices
More special events to attract customers coming to the restaurant (e.g. charity events)	Unforeseen costs: supplier cost increasing, tax increasing
Possibility for IT systems to help with everyday business management	

Mission

When the restaurant first opened, the owners dream was to own a gourmet restaurant. In 7 years the restaurant evolved from a family business to a high-end restaurant (more service and business oriented).

“To be the best restaurant in Denmark that serves Asian gourmet food. By constantly improving, Fusion wants to continue its tradition of offering fine-dining experiences and fantastic service in the food and restaurant industry in Northern Denmark.”

Vision

To provide the best possible customer service and dining experience in Northern Denmark. Fusion’s goal is to become the number one gourmet restaurant in Northern Denmark by 2020. Not only providing citizens of Aalborg with excellent dining experiences, the dedicated team at Fusion is determined to create a welcoming atmosphere and provide an intimate space for you and to those closest to you.

Strategic goals

Move tasks that are done manually to an IT solution to track and store data more easily. Decentralise knowledge and information of the manager and delegate simpler time-consuming tasks to an IT system, manager will have more time to focus on tasks such as events and promoting and / or staff training. Manager has more time to address working conditions and improve the work situation for employees.

Strengthen the brand of a company using IT solutions web based products such as websites promoting company's brand, promotional videos to influence market perception, improve communication with the customer base through monthly updates of the menu in different languages. Running seasonal campaigns and promoting them through various media channels (social media, newsletters, company website etc.).

Business case

Introduction:

The company is a restaurant called Fusion, which opened 7 years ago with the aim to become a gourmet restaurant serving quality food. While the company has enjoyed success the operation of a gourmet restaurant also comes with its own challenges. Restaurant managers need to manage schedules and ensure enough employees are working on each day depending on their expected number of guests.

Management summary:

The problem for the restaurant is that all the scheduling is done manually. They want a system that can calculate how many workers they need, based on how busy the restaurant is, track how many hours the students need, so that they have enough hours for SU but while not having too much overtime. They also want a system that logs the working hours digitally since it is currently done on paper.

Potential solutions:

1. Do nothing (not an option – managers waste time and currently higher risk of human error and mistakes).
2. Licensing an existing system (too expensive and offers unnecessary / unwanted functionality).
3. Creating a custom system to meet Fusion's needs (recommended).

We have recommended option number 3 (creating a custom system) since the company needs a solution to better utilise client's time to reach their desired goals. A tailored solution that satisfies the client's needs without paying for unnecessary features / services.

Cost benefit:

Costs: Price of implementing a new system, training staff, purchasing hardware and system maintenance costs.

Benefits: less time spent doing calculations, reduced risk of human error, easier to create schedules and find employee information, more precise records of hours worked, improved satisfaction for employees and management.

Impacts and risk:

Impact: employees need training for the new system, some part time workers might work fewer hours, therefore more money saved, reduced responsibility for employees to report their hours accurately or inform management of their required monthly hours.

Risk: employees may have difficulty adjusting to the new system, if the database fails or the information is corrupted then the records could be lost.

Conclusion:

In the end, our recommendation is to implement a new system that will offer them the functionality they require to run their business more effectively and efficiently. These functionalities are scheduling for the number of employees needed per day, calculating hours needed for SU, a way for the employees to check in and out for calculating the work hours.

System Analysis

System Vision

Purpose

The purpose of this document is to determine and analyse the requirements for the scheduling system. It will state the needs of the stakeholders and intended users, and why they are necessary. We will detail how the system will meet these needs through our use cases and supplementary documents.

Scope

The system we will design and create will replace their existing (manual) scheduling system. The system will display a daily workload of reservations and scheduled events for the upcoming weeks. It will automatically calculate the number of required shifts based on the number of reservations and events. It will hold all employee information; general contact details, minimum and maximum working hours, monthly hours worked, days unavailable for work, scheduled holidays etc. The system will support employees clocking in at the start of a shift and clocking out at the end of a shift.

Problem statement

The current problems with scheduling currently are as follows:

1. Too much time spent by management planning the schedule manually in Excel or with pen and paper.
2. Difficult to track and record the changing needs of employees e.g. hours worked, preferred working days etc.
3. Excessive time spent calculating necessary amount of work hours for each employee.
4. Occasional errors with shift scheduling either too many or too few workers schedule for work.
5. Unbalanced scheduling which leads to disproportionate overtime for certain employees.

Stakeholders

- Restaurant owners:
  Increased satisfaction of managers, employees and customers will lead to higher staff and customer retention, lower turnover of staff, resulting in customer loyalty.
- Finance and administration:
  Ease of tracking employee hours means more accurate wage calculations, less mistakes and happier staff overall.
- Restaurant managers:
  The new system would mean less time and stress spent planning the monthly schedules, freeing up more time to focus on daily service preparations and customer service.
- Employees:
  A fairer more balanced schedule for each employee, with accurate data tracking hours worked therefore freeing up responsibility to track working hours.
- Customers:
  Customer service will improve from the restaurant staff as work between staff is more evenly balanced.

System key features

- Retrieving and displaying daily workloads for the current month with accurate shift requirements for each workload.
- Simple switching between weeks in the current month.
- Easily creating new shifts and associating available employees to the correct shifts.
- Removing unnecessary shifts.
- Quick and easy updating of shifts previously created.
- Displaying important employee information when deciding who to schedule for each shift.
- Saving and updating shift information with the associated employees.
- Managing employee data, general contact information, days unavailable, holiday / vacation days.
- Adding new employees to the system with necessary employee details.
- Removing employee data from the system once no longer needed.

User Expectations

The company expects the new system to be able to do the following tasks:

- Making a weekly schedule for the dishwashers and waiters, considering their needs for hours, and their availability.
- To automatically take information from a 3rd party booking / reservation website and give notifications if extra shifts are needed.
- Calculate the estimated shifts length, the system should make use of a clock in and out system.
- Storing the actual hours worked for each employee.
- The system should store all employee information (contact information, unavailable days etc.).
- Calculate the number of required shifts based on the number of reservations in the restaurant, and sending notifications five days in advance.
- The system should check if the worked hours are balanced – if someone worked too much, too little, or if they don't meet the minimum requirements.
- The system itself must be user friendly and simple to understand.

Use case diagram

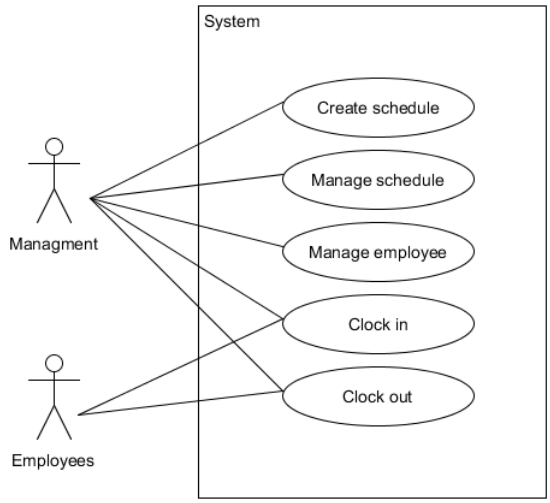

Figure 4: Use case diagram for the main tasks identified during the client interview.

Above is a diagram depicting the main use cases for our system. The key features for our system will cover creating and managing the schedule, managing the employee data, clocking-in and clocking-out for employees.

Ideally workloads would be managed automatically by the system (i.e. periodically updating the workload information from the reservation website used by the restaurant). However, given the time constraints for this project and possible legal issues in scraping the needed data from a 3rd party website, we have decided to input the workload details manually into the database for the purposes of this project.

Brief use cases

Use case: Create schedule

Actors: Management

Description: The manager logs in, checks the workload information and how many shifts are required then matches the most suitable employees to fill the shifts. This repeats this until all required shifts are filled. Manager checks the schedule, once the manager is happy he saves the schedule.

Use case: Update schedule

Actors: Management

Description: The manager logs in, retrieves the existing schedule and makes changes to the schedule as necessary. Upon implementing all the needed changes the manager saves the new schedule.

Use case: Manage Employee (Create employee)

Actors: Management

Description: The manager needs to add new employee data to the system. He / she inputs the employee's contact details, working hours, working preferences into the system and then saves the employee.

Use case: Manage Employee (Retrieve employee)

Actors: Management

Description: The manager needs to find information about an employee. He / she searches the employee by name and then views the necessary information including contact details and weekdays unavailable / preferred off for the employee.

Use case: Manage Employee (Update employee)

Actors: Management

Description: The manager needs to update an employee's information. He/she searches the employees by name, then change information (e.g. adding dates/weekdays unavailable/preferred off) and saves it into the system.

Use case: Manage Employee (Delete employee)

Actors: Management

Description: The manager needs to delete an employee from the system. He / she searches the employee by name and then deletes the employee from the system.

Use case: Clock in

Actors: Management, Employee

Description: At the start of the employees shifts he uses the credentials to log the start of his work.

Use case: Clock out

Actors: Management, Employee

Description: At the end of the employee's shift he uses the credentials to log the end of his work.

Class candidate list

Table 3: Class Candidate table, identifying potential classes based from nouns found in use cases.

Candidate class	Evaluation	Include in our system Y/N
Schedule	Consists of several individual workloads containing shifts not needed only represented in the GUI.	N
Manager	The manager is an actor that uses the system and he is just a type of employee.	N
Workload	Workload is important for the management of the schedule and shifts and it contains the no of reservations, the no of shifts, the date and possible events.	Y
Shift	Shifts is a part of workload and it contains start time, location, type of shift and the estimated shift length.	Y
Employee	An employee is an important part of the system, there are multiple types of employees and the system must store information about them.	Y
Hours-Worked	Consists of the date and number of hours the employee has worked.	Y
System	The system stores all the necessary information and is used by the employees to satisfy their needs. Not a class.	N
Credentials	Contains the user name and password of the employees.	Y
Work	Work refers to the number of hours the employees work and is already stored in the Hours-Worked class.	N
Worker	Worker is a synonym for Employee and therefore not necessary.	N
Date	Needed to represent which dates employees are unavailable but not necessary for its own class.	N
Weekday	Necessary concept to represent weekday's unavailable for work or preferred off but can just be an Enum.	N

Prioritisation of use cases

Table 4: Prioritisation of use cases based on their risk, coverage, and criticality as well as their priority.

Use case	Risk	Coverage	Criticality	Priority	Phase	Iteration
Create Schedule	High	High	High	1	Elaboration	1
Update Schedule	High	High	High	2	Elaboration	2
Manage Employee(Create employee)	Medium	Low	Medium	3	Elaboration	3
Manage Employee(Retrieve employee)	Low	Medium	Medium	3	Elaboration	3
Manage Employee(Update employee)	Medium	Medium	Medium	3	Elaboration	3
Manage Employee(Delete employee)	Low	Medium	Low	3	Elaboration	3
Clock in	Medium	Medium	Low	4	Construction	-
Clock out	Medium	Medium	Low	4	Construction	-

We have decided that our most important use case is Create Schedule. This is because it has the highest risk and criticality and has a high coverage of our system.

Update schedule is the second most important use case as its risk, coverage and criticality is very similar to create schedule however it is dependent on a schedule first being created.

Manage Employee (CRUD) generally has a lower priority, while managing employee information is important the main purpose of the system is to create schedules, therefore it is the 3rd highest priority.

Clocking in and clocking out are use cases we would like to implement given more time. Its criticality is low and not very important in our main use cases therefore we would leave it later in the construction phase.

Domain model

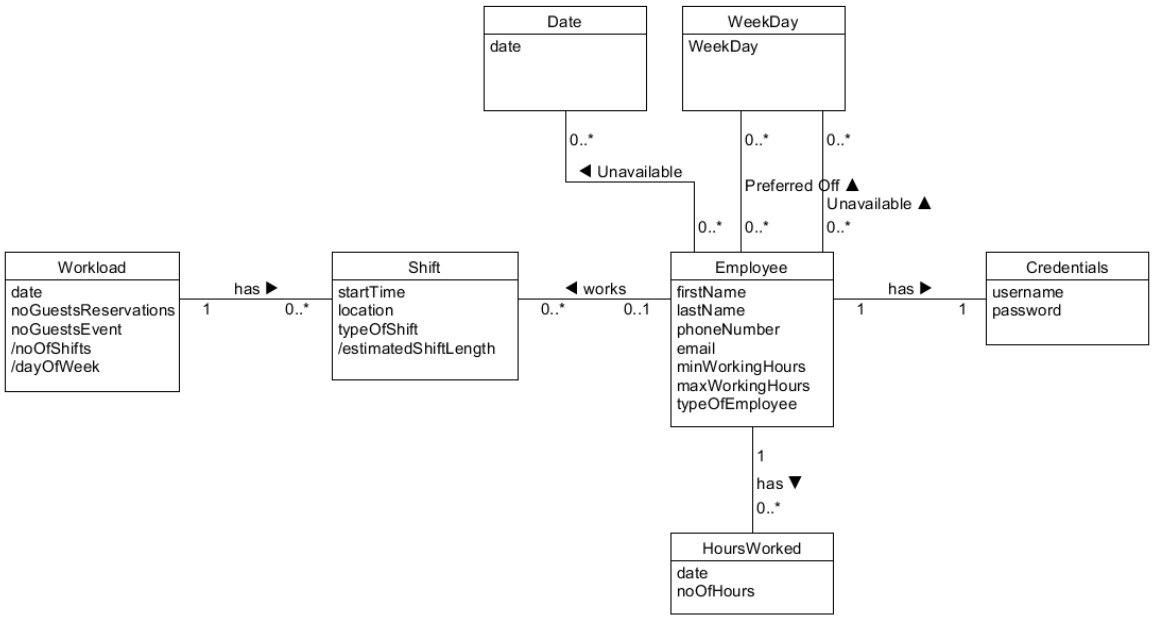

Figure 5: Domain model representing concepts important to Restaurant Fusion in terms of scheduling shifts.

Shown above is a picture of the domain model for the scheduling system. In it we have the classes Workload, Shift, Employee, HoursWorked and Credentials (Date and WeekDay are not classes but important concepts to represent within our system).

A Workload describes a set amount of work needed to be completed for a given date. The amount of work to be done is determined by the number of reservations and any scheduled events on the date. The work is divided up into shifts.

A Shift represents a division of work to be completed, it contains information regarding the shift start time, location of work, type of shift (Dishwashing, waiting, etc.) and a derived estimated shift length (calculated from no. of guests and the shift type / start time). Each shift is to be assigned to an employee (Note. A shift can temporarily have no employees assigned, while the manager decides which employee is suitable for the shift).

An Employee is a representation of an employee who works for the restaurant. They have general personal information such as name and contact details as well as their monthly minimum and maximum working hours. It also states their position in the company (typeOfEmployee: dishwasher, waiter, etc.). Each employee has associations with the amount of hours they have worked (HoursWorked), the week days they are unavailable or prefer off (WeekDay unavailable / preferred off), and finally dates where they are unavailable to work such as holidays / vacations (Dates unavailable).

HoursWorked holds information for a specific employee for how many hours they worked on a given date. A collection of HoursWorked can be used to calculate hours worked for a month or even a year.

Credentials are used to track hours worked for each employee, a set of credentials will match only one employee which would be used as part of the clock-in and clock-out use cases.

System Design

System Architecture

For our system, we will build an application in Java but we will store all of the scheduling and employee information in the database. To achieve this, we will use a 3-layered architecture including data access objects (DAO) which use embedded SQL in Java to build model object using ResultSet data retrieved from the database.

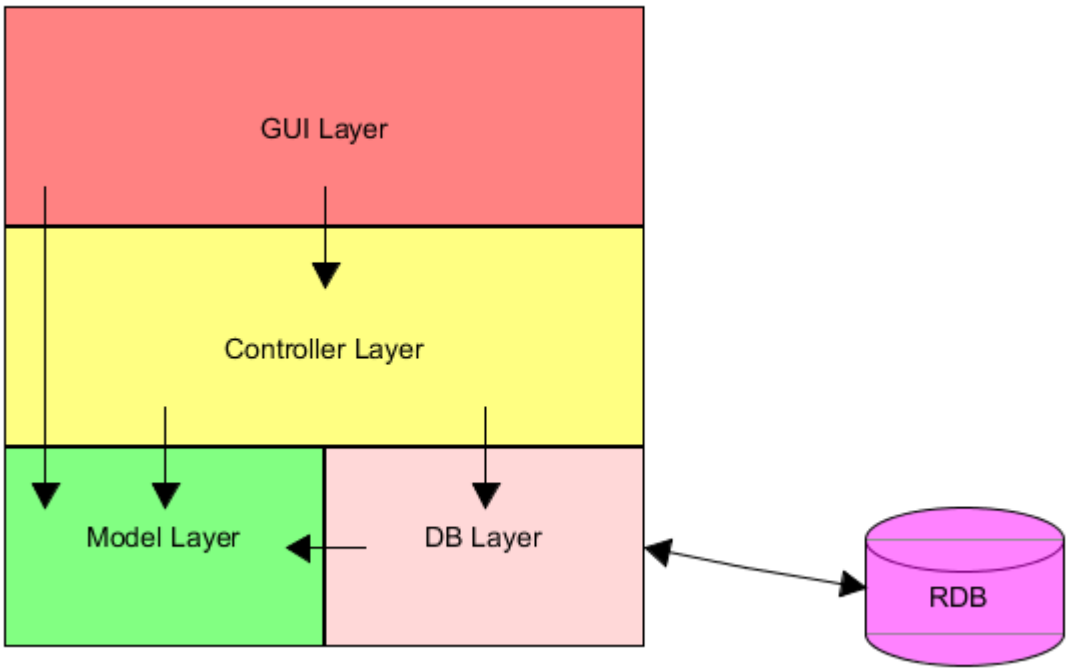

Figure 6: Three Layer architecture including DAO diagram.

For our 3-layered architecture, we will use a more flexible open style where the GUI Layer will be allowed to import from the model layer. We decided to use this style to take advantage of the object-oriented capabilities of Java rather than the more strictly controlled architecture which would require returning many data parts.

Relational model

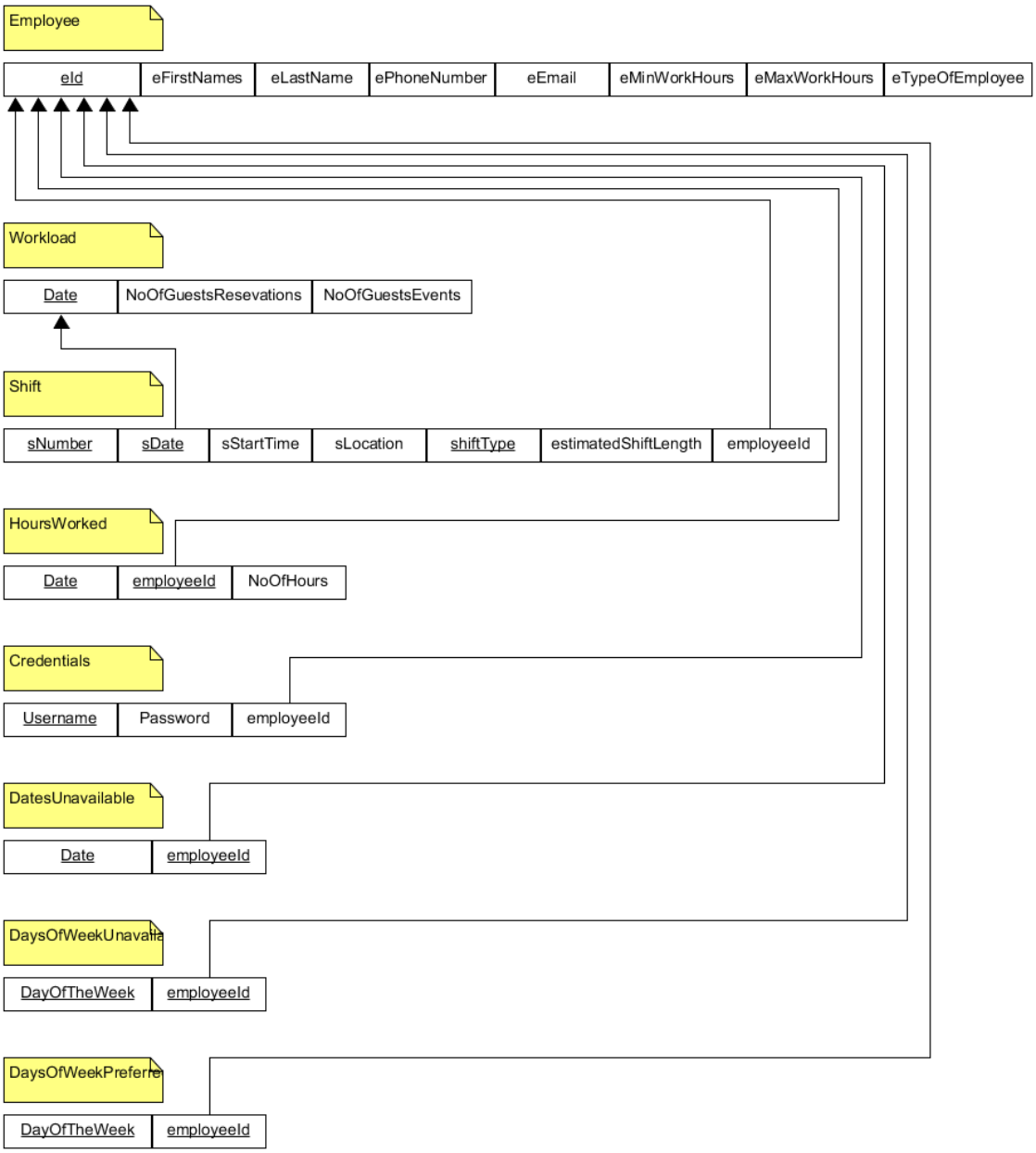

Figure 7: Relational model mapped from the domain model.

For our relational schema, we mapped our conceptual domain model into a relational schema for our database. Class names from the domain model were used as relation names, attributes became columns, primary keys were added to ensure uniqueness, and finally foreign keys referencing other tables were added.

Primary keys chosen were selected to guarantee uniqueness for each tuple (row) in a relation (table). For Workload, date is the primary key as there should only be one tuple to represent the amount of work for a given date. Shift has 3 primary keys, sNumber, sDate and shiftType this is because there could be multiple shifts on a given date and each could be of similar shiftTypes so an extra attribute sNumber is added to guarantee uniqueness. However, for shift we allow nulls for the non-primary key attributes – this is so partial shift information can be saved and completed at a later time.

Use case: Create schedule

Fully dressed use case

Use case: Create schedule
Scope: System for creating the schedule
Level: User goals
Primary actor: Manager
Preconditions: Manager is logged into the system, events and reservations are entered accordingly.
Success Guarantee: Schedule has been created and the workload information have been updated. Employee shifts are saved.

Main success scenario:

1. Manager chooses a week which must be updated
2. System displays all the dates in the week
3. Manager selects a date
4. System displays workload information with the number of events and reservations, estimated required shifts and available employees with their information
5. Manager chooses to create new shift
6. System creates a blank shift in the schedule
7. Manager enters the required shift information
8. Manager clicks on an available employee and adds them to the shift
9. System adds employee to the shift showing shift start time, estimated shift length, location and shift type
   Manager repeats steps 1-9 until the schedule is filled
10. Manager checks all the dates are correct and then saves the schedule
11. System records the updated dates

Extensions

a. System fails at any time (e.g. freezes)

1. Manager restarts the system and logs in
2. System opens previously saved dates
   Continue from main success scenario steps 1-11.

4a. System detects failure to communicate with server (no connection) and is not showing any reservations/events/available employees

1. System signals about a potential connection problem
2. Manager restarts the internet and the application and tries again

8a. System is not showing any information about the chosen employee

1. System signals an error and rejects adding the employee to the chosen date
2. Manager restarts the application and tries again
3. System records the update

Continue from main success scenario steps 8-11.

11a. System fails at saving dates

1. System signals error to the manager, displays the affected workloads that couldn't be saved
2. Manager reselects the erroneous workloads and re-enters the shift(s) information
   Continue from main success scenario steps 10-11.

Mock up

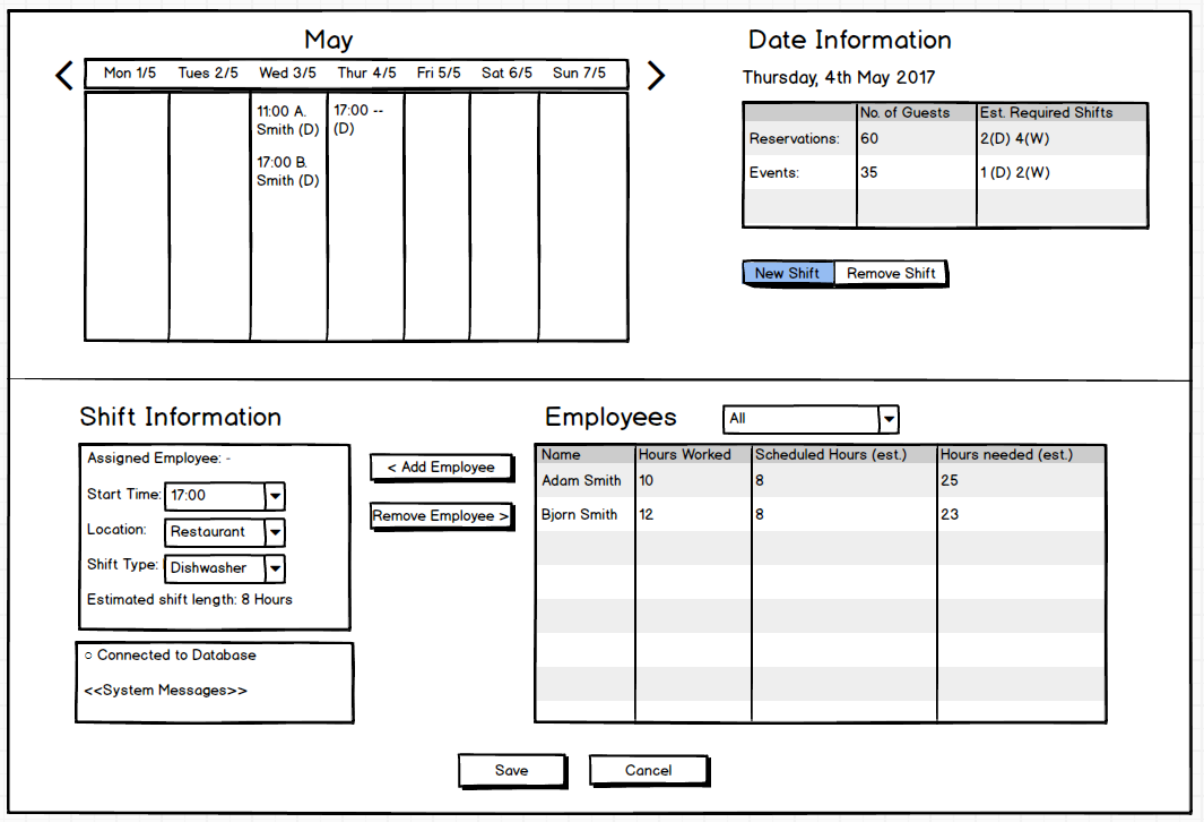

Figure 8: Mock up for our main use case Create Schedule, a visual representation of what we imagine the final product may look like.

Based on our fully dressed use case for create schedule, we decided to create a mock up to better visualise how our system may function once completed. Our aim was to simplify creating a schedule and making it as efficient as possible while still presenting all the necessary information to the user to be able to make a balanced schedule.

The idea is the user selects a date with workload information from the calendar in the top left.

Once selected, the information is displayed in the top right box showing the number of reservations, the number of people attending an event and the estimated required shifts.

The user can then press the new shift button to create a blank shift on the calendar. This will then enable the bottom left box "Shift information" and allow the user to customise the shift there.

When the user selects a type of shift, the employee table will automatically generate a list of employees available to work the shift, from which point the user can select an employee and simply click add employee to add the employee to the shift.

System Sequence Diagram

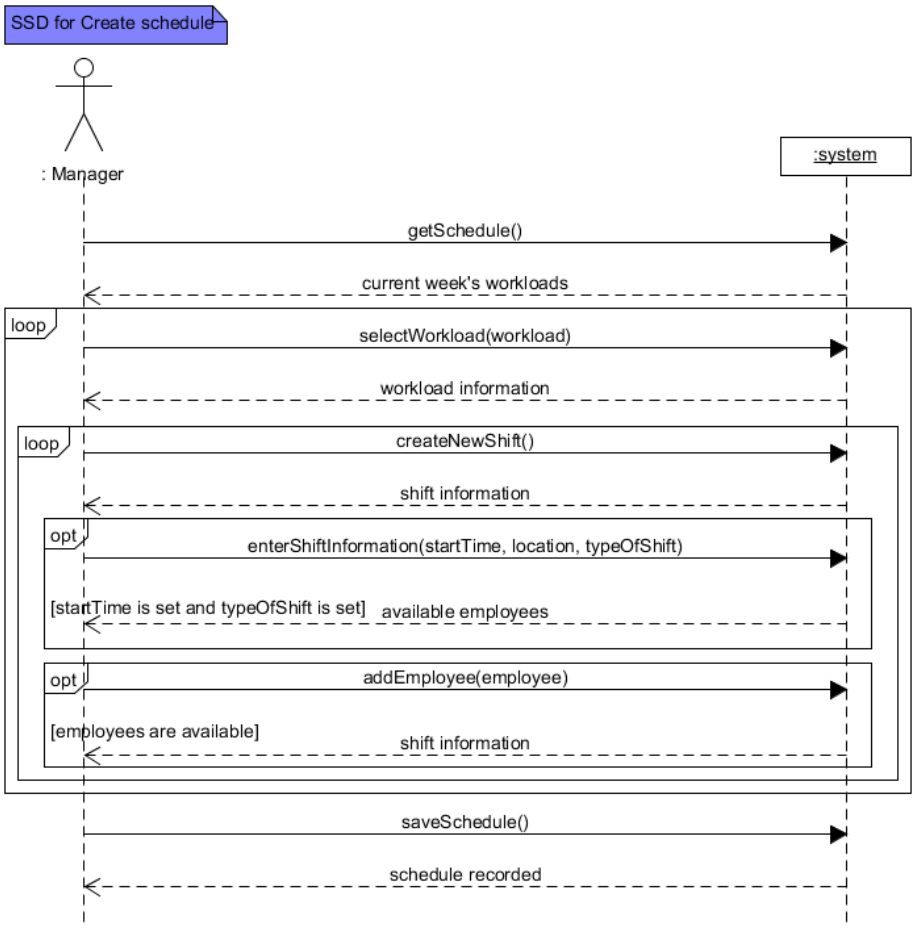

Figure 9: SSD for create schedule including method calls and system responses.

In our system sequence diagram, we show the interactions between the actor and the system responses. The actor enters the loop of selecting workloads, adding shifts, entering shift information and add employees to those shifts (Optionally he can leave them blank and complete them at a later date). Once finished the actor will save all workloads that have been modified to the database.

Interaction Sequence diagrams

getSchedule()

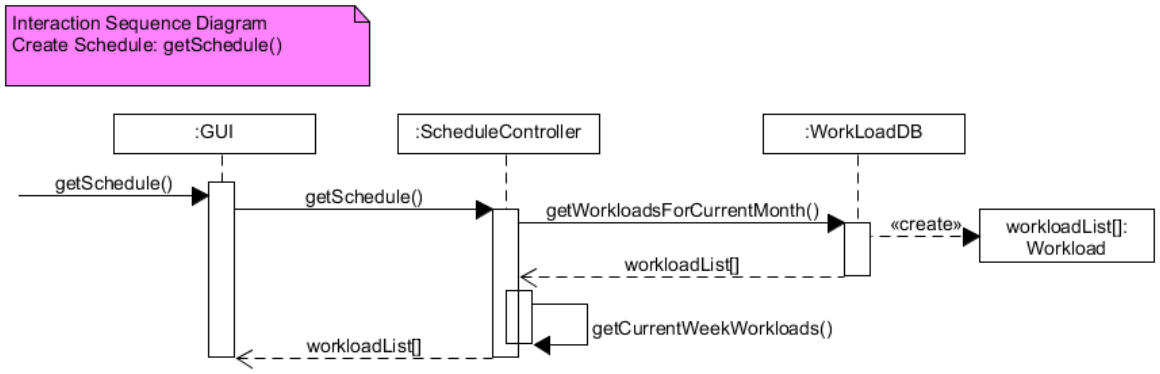

Figure 10: ISD for getSchedule() showing interactions between various layers and classes.

For getSchedule() the GUI calls the getSchedule() method on the ScheduleController, which in turn calls the getWorkloadsForCurrentMonth() from the WorkloadDB. A collection of Workload's are generated from the database and returned to the Schedule Controller, where the getCurrentWeekWorkloads() method selects the current week (based on the local machine's local date) and returns the current week to the GUI where it is displayed.

selectWorkload(workload)

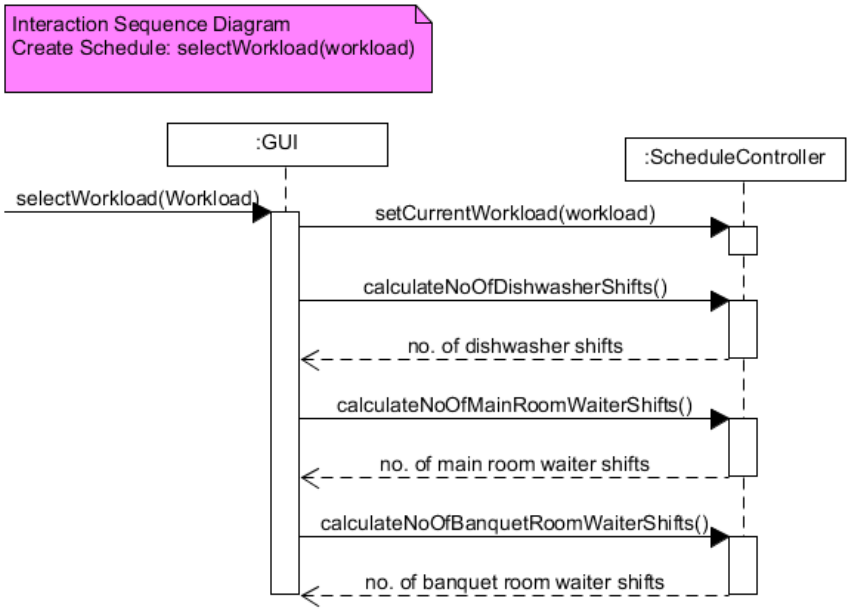

Figure 11: ISD for selectWorkload(workload) showing interactions between various layers and classes.

For selectWorkload(workload) when a workload is selected, the setCurrentWorkload() method is called on the controller, afterwards the number of dishwasher shifts, main room and banquet room waiter shifts are calculated and then displayed in the GUI through their respective method calls.

createNewShift()

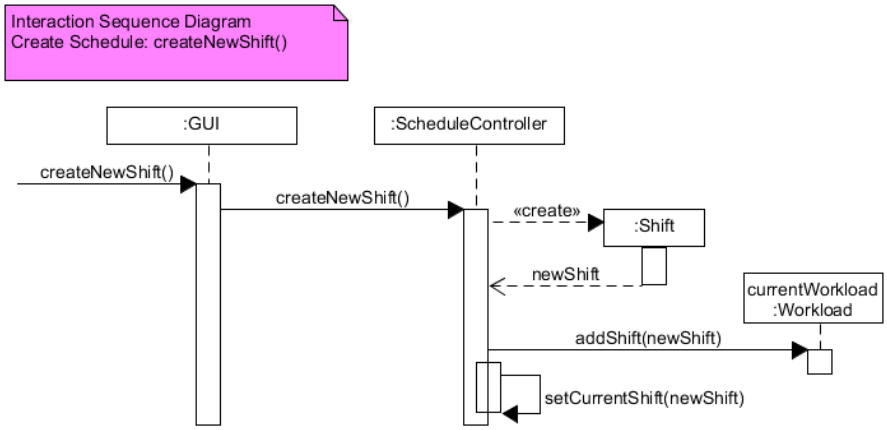

Figure 12: ISD for createNewShift() showing interactions between various layers and classes.

Once a workload has been selected, and the actor wants to add a new shift, the createNewShift() method is called on the ScheduleController. A newShift object is created and returned to the controller, this newShift is added to the currentWorkload, then finally the newShift is set to the currentShift on the ScheduleController.

enterShiftInformation(startTime, location, typeOfShift)

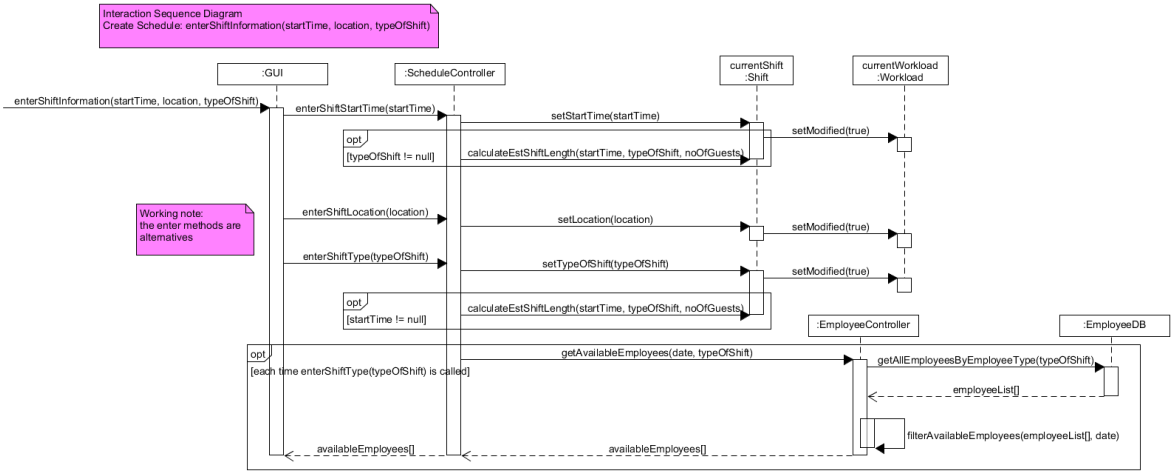

Figure 13: ISD for enterShiftInformation(startTime, location, typeOfShift) showing interactions between various layers and classes.

For enterShiftInformation(startTime, location, typeOfShift) the start time, location and type of shift can be set independently of each other, this means the actor can choose which attributes of the shift they wish to set in any order. However certain checks are called when certain attributes are set, namely setStartTime(startTime) and setTypeOfShift(typeOfShift). The reason for this is because the derived attribute estimated shift length requires both of these to be set in order to be calculated properly. Therefore the calculateEstShiftLength(startTime, typeOfShift, noOfGuests) method is only called when both are set. After each set method on the currentShift, an additional call

setModified(true) is called to the currentWorkload to signal that the workload has been modified and the shifts will need to be saved/updated on the currentWorkload.

Additionally every time the enterShiftType(typeOfShift) method is called and changed to a new shift the method getAvailableEmployees(date, typeOfShift) is called on the EmployeeController this returns the employees available to work the shift from the database based on the date and typeOfShift provided in the parameters. First it gets all the employees who can work the type of shifts from the database, then using the filterAvailableEmployee() (see next diagram for more details) it only returns the employees who are available to work on that specific date.

filterAvailableEmployees(employeeList[], date)

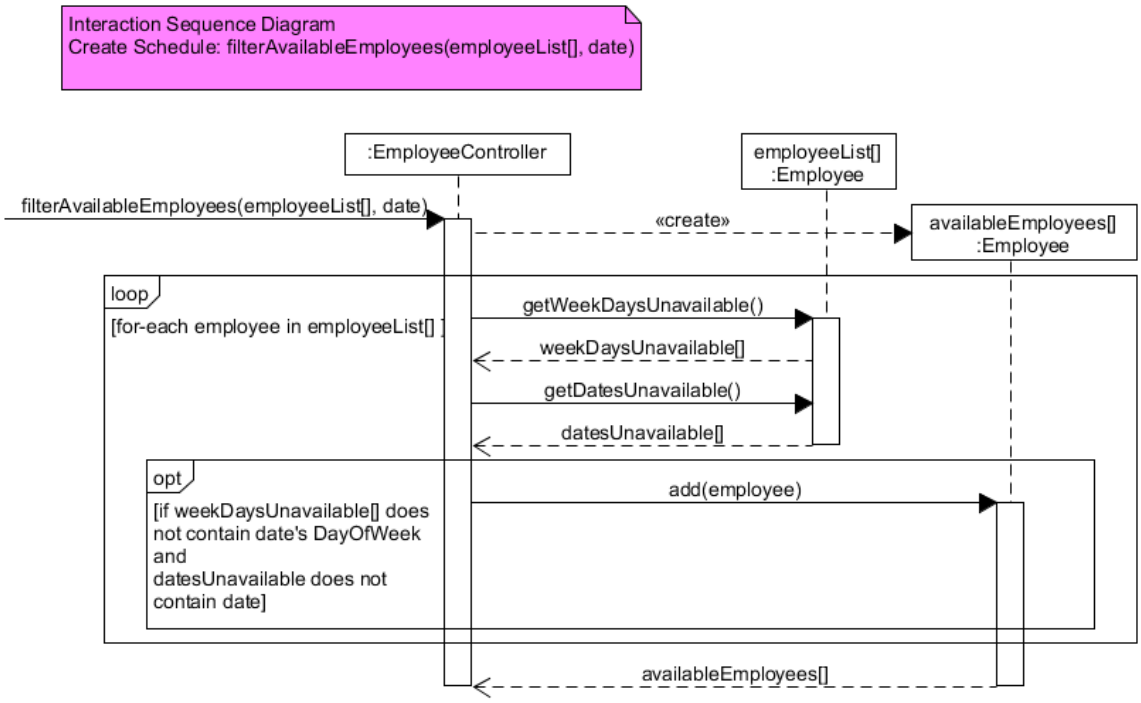

Figure 14: ISD for filterAvailableEmployees(employeeList[], date) showing interactions between various layers and classes.

This diagram shows how the filterAvailableEmployees(employeeList[], date) only selects the employees who are available for a given date. First a new list of Employee is created and named availableEmployees[]. Then entering a for-each loop for every employee in employeeList[] the weekDaysUnavailable and datesUnavailable are retrieved and stored in local variables. An if condition checks the contents for these local variables to see if they contain the dayOfWeek for the given date or the date itself. If they do not, the employee is then added to the availableEmployees[]. Once the loop has exited then the collection of availableEmployees[] is returned to the EmployeeController.

addEmployeeToShift(employee)

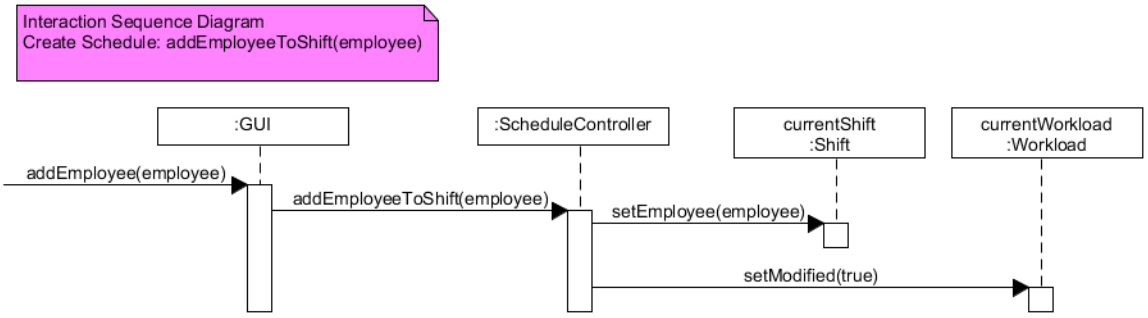

Figure 15: ISD for addEmployeeToShift(employee) showing interactions between various layers and classes.

After the available employees have been retrieved the actor selects one from the list and adds it to the current shift. This is done through the addEmployeeToShift(employee) method on the ScheduleController which in turn calls the methods setEmployee(employee) and setModified(true) on the currentShift and currentWorkload respectively.

saveSchedule()

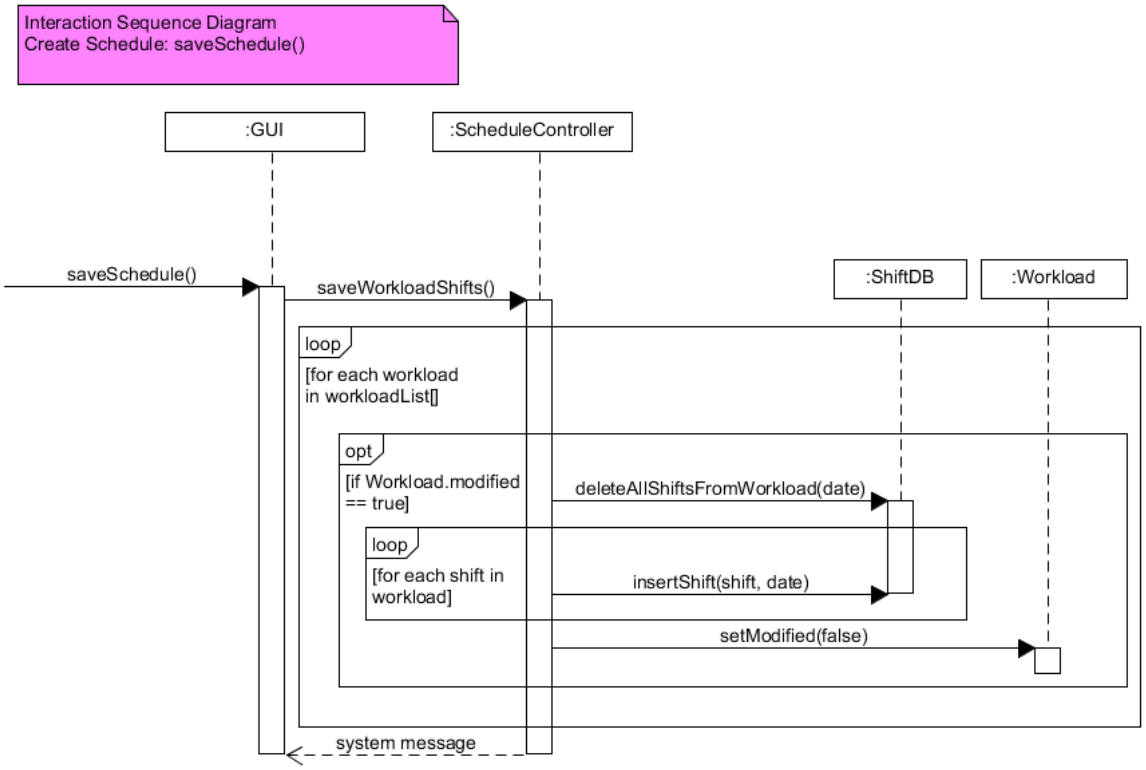

Figure 16: ISD for saveSchedule() showing interactions between various layers and classes.

Once the actor has finished creating/updating the shift information the saveSchedule() method is called. This calls the saveWorkloadShifts() method on the ScheduleController, this begins a for-each loop for every workload in the month's workloadList[], a check is done on each workload to see if it has been modified or not. If it has been modified the method deleteAllShiftsFromWorkload(date) is called on the shiftDB and then a new for-each loop for every shift in the workload is enter and a call to insert each shift into the database is made (through the insertShift(shift,date) method).

We decided on this approach as the alternatives would require many checks and flag for each shift on each workload (newly created shift, updated shifts, removed shifts). Because of this we felt it was simpler to delete all the shifts for the workload currently in the database and re-insert all the shifts from the workload object in the Java application.

After the shifts have been insert, the workload modified attribute is set back to false and a system message is generated stating the success of the save operation.

Design Class Diagram

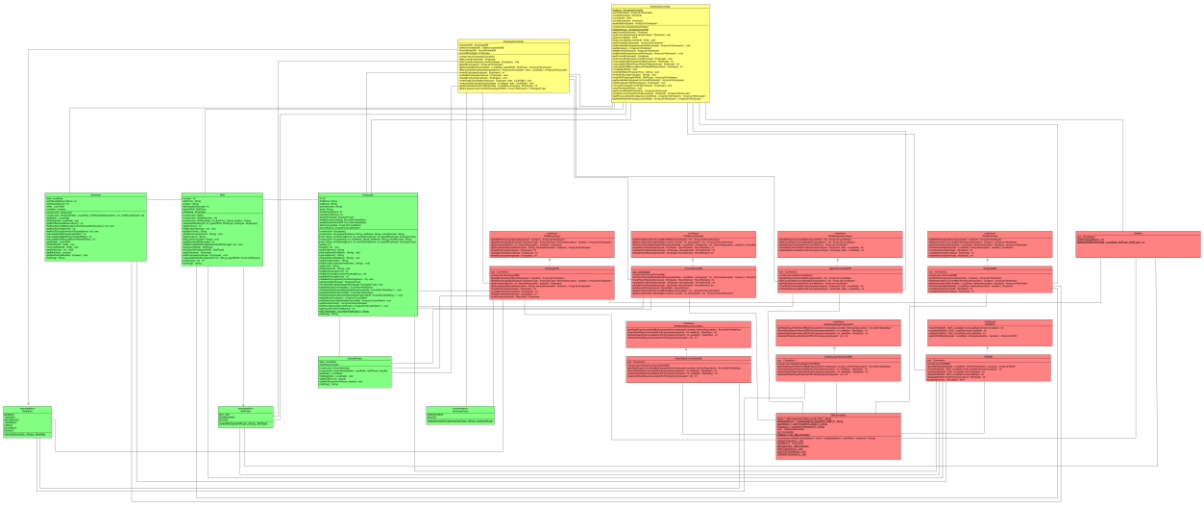

Figure 17: Design class diagram for our system including the model, DB and controller layers. (For a better view of the classes and associations please refer to the appendix)

Above is a diagram showing our classes for our system from the Controller, DB and Model layer.

The attributes and methods can be seen with their visibilities, return types and parameterised arguments. You can also see the associations between the classes and layers. The GUI layer has not been included in the diagram.

We have two controller classes the ScheduleController and EmployeeController to match our main use cases Create Schedule, Update Schedule and Manage Employee which control the main operational logic and functions for our system.

Code

MSSQL scripts for our database

For our project, we are using MS SQL to create and manage our database. We created tables following the relational schema set out in our relational model. Here we will show only a few of the most important scripts for our database:

Employee

```
use dmai0916_202617;

CREATE TABLE Employee(
eId int Not Null,
eFirstName varchar(99) Not Null,
eLastName varchar(99) Not Null,
ePhoneNumber varchar(99) Not Null,
eEmail varchar(99) Not Null,
eMinWorkHours int Not Null,
eMaxWorkHours int Not Null,
eTypeOfEmployee varchar(99) Not null,
PRIMARY KEY (eId));
```

	eId	eFirstName	eLastName	ePhoneNumber	eEmail
1	1	Lara	Croft	76321569	lara.croft99@hotmail.com
2	2	Clark	Kent	63599123	c.kent@aol.com
3	3	Bruce	Wayne	63186933	wayne.bruce@wayne-enterprise.com

Figure 18: Create Employee script and example table.

Create table script for Employee and examples tuples from the Employee table. The primary key for Employee is eId and it has no foreign keys.

Workload

```
use dmai0916_202617;

CREATE TABLE Workload(
Date [Date] Not Null,
NoOfGuestsReservations int Not Null,
NoOfGuestsEvents int Not null,
PRIMARY KEY ([Date]));
```

	Date	NoOfGuestsReservations	NoOfGuestsEvents
1	2017-06-01	49	0
2	2017-06-02	49	0
3	2017-06-03	71	50
4	2017-06-04	0	0
5	2017-06-05	13	0

Figure 19: Create Workload script and example table.

Create table script for Workload and example tuples from the Workload Table. We use the date as a primary key (there should be only one tuple for one date).

Shift

```
use dmai0916_202617;

CREATE TABLE [Shift](
sNumber int Not null,
sDate Date Not null,
sStartTime varchar(99) Null,
sLocation varchar(99) Null,
shiftType varchar(99) Not null,
estimatedShiftLength int Null,
employeeId int Null,
PRIMARY KEY (sNumber, sDate, shiftType));
```

	sNumber	sDate	sStartTime	sLocation	shiftType	estimatedShiftLength	employeeId
1	1	2017-06-01	17:00	Restaurant	DISHWASHER	10	9
2	1	2017-06-01	10:00	Restaurant	WAITER	8	13
3	1	2017-06-02	17:00	Restaurant	DISHWASHER	10	12
4	1	2017-06-02	10:00	Restaurant	WAITER	8	16
5	1	2017-06-03	17:00	Restaurant	DISHWASHER	12	5

Figure 20: Create Shift script and example table.

Create table script for Shift and example tuples from the Shift table. The primary key is a combination of the sNumber, sDate and shiftType. There are foreign keys for sDate and employeeId (but these constraints are added later in an Alter tables script.)

Alter Tables

```
ALTER TABLE dbo.[Shift]
ADD CONSTRAINT FK_Workload_Shift_Cascade
FOREIGN KEY (sDate) REFERENCES Workload(Date)
ON DELETE CASCADE

ALTER TABLE dbo.[Shift]
ADD CONSTRAINT FK_Employee_Shift
FOREIGN KEY (employeeId) REFERENCES Employee(eId)
ON DELETE SET NULL
```

Figure 21: Alter tables script adding referential keys and constraints to our database.

Here we add the constraints to Shift making sDate in the shift table reference the Date in the Workload table, and making employeeId in shift reference eId in Employee.

For the constraint FK_Workload_Shift_Cascade we added the line “ON DELETE CASCADE” so that if a referenced Workload is deleted any shifts with the same date will also be deleted from our database.

However, for the constraint FK_Employee_Shift we used “ON DELETE SET NULL” because if an employee is deleted from the database the shift won’t necessarily be deleted as well but rather set to null. This is because we may still want to find an employee to fill that shift.

Example from the DB Layer

```java
@Override
public int insertEmployee(Employee employee) throws Exception {

    int nextId = GetMax.getEmployeeMaxId();
    nextId = nextId + 1;
    int rc = -1;
    String baseQuery="INSERT INTO employee(eId, eFirstName, eLastName, ePhoneNumber, eEmail, eMinWorkHours, eMaxWorkHours, eTypeOfEmployee) "
            + "VALUES(?,?,?,?,?,?,?,?)";
    PreparedStatement preparedStmt = null;
    try{
        DBConnection.startTransaction();
        preparedStmt = con.prepareStatement(baseQuery);
        preparedStmt.setInt(1, nextId);
        preparedStmt.setString(2, employee.getFirstName());
        preparedStmt.setString(3, employee.getLastName());
        preparedStmt.setString(4, employee.getPhoneNumber());
        preparedStmt.setString(5, employee.getEmail());
        preparedStmt.setInt(6, employee.getMinWorkingHours());
        preparedStmt.setInt(7, employee.getMaxWorkingHours());
        preparedStmt.setString(8, employee.getTypeOfEmployee().toString());
        preparedStmt.setQueryTimeout(5);
        rc = preparedStmt.executeUpdate();

        WeekDaysPreferredOffDB wdpoDB = new WeekDaysPreferredOffDB();
        for(WeekDay weekDayPO: employee.getWeekDaysPreferredOff()){
            wdpoDB.insertWeekDaysPreferredOffForEmployee(nextId, weekDayPO);
        }
        WeekDaysUnavailableDB wduDB = new WeekDaysUnavailableDB();
        for(WeekDay weekDayU: employee.getWeekDaysUnavailable()){
            wduDB.insertWeekDaysUnavailableForEmployee(nextId, weekDayU);
        }
        if(rc == -1){
            Exception e = new Exception();
            throw e;
        }
        DBConnection.commitTransaction();
    }
    catch(SQLException e){
        DBConnection.rollbackTransaction();
        System.out.println("Employee insertion error.");
        throw e;
    }
    finally {
        try {
            preparedStmt.close();
        } catch(SQLException e) {
            e.printStackTrace();
        }
    }
    return nextId;
}
```

Figure 22: EmployeeDB example method insertEmployee(Employee employee).

This is an example method from one of our EmployeeDB class, insertEmployee() is used to insert a newly created employee into the database.

First, we retrieve the current max Id for employee from the database using our GetMax class, then we increment the max Id by 1 to get the next Id.

Next we prepare our embedded SQL base query that we will execute, we use the ?'s as placeholders for values we will insert later, this is because we are using prepared statements.

Inside the try block we begin by using a transaction and the method from our DBConnection class startTransaction(). The reason we are using a transaction in this method is because later we will try to insert the employee's WeekDaysUnavailable and WeekDaysPreferredOff, because we want to save everything at once and only if everything is inserted without exceptions.

Next, the preparedStatement is prepared using the base query and the ? placeholders are replaced with values from our employee object that the method takes as an argument. Using a prepared statement prevents against SQL injection attacks in case users have knowledge of SQL queries. Once the prepared statement is ready the update is executed to the database.

After that, the WeekDaysPreferredOff and WeekDaysUnavailable are inserted into the database using the insert methods found in their respective DB classes.

If everything is inserted without fail then the transaction is committed through DBConnection.commitTransaction() and then saved to the database.

However, if there are any problems with any of the insertions then the catch block with catch the SQL exception and then rolls back the transaction. It will then print an error message and also throw an exception.

Finally, we return the next Id so that the system knows what value to assign the new employee object's Id.

Logic for calculating the number of required dishwashing shifts

```
/* Minimum 1 shift for a working day.
 * More than 50 guests, need at least 2 dish washing shifts
 * If event = true && total no of guests >= 40, +1 shift (for banquet room)
 * More than 115 guest && no event, required 3 shifts
 * More than 210 guests total need 4 shifts
 */
public int calculateNoOfDishwasherShifts(Workload workload){
    int noOfRequiredShifts = 1;
    if(workload.getNoOfGuestsReservations() >= 50){
        noOfRequiredShifts = 2;
    }
    if(workload.getNoOfGuestsEvent() > 0 && (workload.getNoOfGuestsEvent()+workload.getNoOfGuestsReservations()) >= 40){
        noOfRequiredShifts++;
    }
    if(workload.getNoOfGuestsReservations() > 115 && workload.getNoOfGuestsEvent() == 0){
        noOfRequiredShifts = 3;
    }
    if((workload.getNoOfGuestsEvent()+workload.getNoOfGuestsReservations()) >= 210){
        noOfRequiredShifts = 4;
    }
    return noOfRequiredShifts;
}
```

Figure 23: ScheduleController method to calculate the number of estimated required dish washing shifts.

Here we have the pseudo code in comments with the actual code in the ScheduleController class. The calculation for the number of Dishwashing shifts is a bit complicated as it can depend on numerous factors.

Minimum required shifts for a working day is 1.

If there are more than 50 guests who have made a reservation ahead of time, 2 shifts are required.

If there is an event and the total number of guests (reservations + events) is greater than or equal to 40 an extra shift is required.

If there is no event and the number of guest reservations is greater than 115, 3 shifts are required.

Finally, if the total number of guests (reservations + events) is greater than or equal to 210, 4 shifts are required.

This was based on the logic provided to us by the restaurant manager.

Logic for calculating waiter shifts in the restaurant

```
/* Minimum no of waiters shifts = 1 (for < 10 guests)
 * 80+ guests = busy
 * <=80 guests = not busy
 * logic for main restaurant area
 *  if < 10, 1 shift
 *  if >10 <=20, 2 shifts
 *  if >20 <=60, 3 shifts
 *  if >60 <=80, 4 shifts
 *  extra shifts after 4 (when busy), 1 shift per 20 additional guests
 */
public int calculateNoOfMainRoomWaiterShifts(Workload workload){
    int noOfRequiredShifts = 1;
    if(workload.getNoOfGuestsReservations() > 10 && workload.getNoOfGuestsReservations() <=20){
        noOfRequiredShifts = 2;
    }
    if(workload.getNoOfGuestsReservations() > 20 && workload.getNoOfGuestsReservations() <=60){
        noOfRequiredShifts = 3;
    }
    if(workload.getNoOfGuestsReservations() > 60 && workload.getNoOfGuestsReservations() <=80){
        noOfRequiredShifts = 4;
    }
    if(workload.getNoOfGuestsReservations() > 80){
        noOfRequiredShifts = 4;
        for(int i = 0; i < (workload.getNoOfGuestsReservations()-80) / 20; i++){
            noOfRequiredShifts++;
        }
    }
    return noOfRequiredShifts;
}
```

Figure 24: ScheduleController method to calculate the number of estimated required waiting shifts.

Here we have the pseudo code in comments with the actual code from the ScheduleController class.

The required waiting shifts for the main room logic is simpler than the logic for dishwashing shifts as it only based on the number of reservations.

Minimum required waiting shift for a working day is 1.

Between 10 and 20 guests, 2 shifts are required.

Between 20 and 60 guests, 3 shifts are required.

Between 60 and 80 guests, 4 shifts are required.

After 80 guests the restaurant is considered ‘busy’ meaning more waiters are needed, 1 extra waiter per additional 20 guests (after 80).

(the calculations for the banquet room is almost identical to the above method).

Calculate estimated shift length

```
public int calculateEstShiftLength(String startTime, ShiftType typeOfShift, int noOfGuests){
    int defaultShiftLength = 8;

    if(typeOfShift == ShiftType.DISHWASHER){
        if(startTime.equalsIgnoreCase("17:00")){
            int estimatedShiftLength = 8;
            if(noOfGuests >= 40 && noOfGuests <80){
                estimatedShiftLength += 2;
            }
            if(noOfGuests >= 80){
                estimatedShiftLength +=4;
            }
            return estimatedShiftLength;
        }
        if(startTime.equalsIgnoreCase("20:00")){
            int estimatedShiftLength = 5;
            if(noOfGuests >= 40 && noOfGuests <80){
                estimatedShiftLength += 2;
            }
            if(noOfGuests >= 80){
                estimatedShiftLength +=4;
            }
            return estimatedShiftLength;
        }
    }

    if(typeOfShift == ShiftType.WAITER){
        if(startTime.equalsIgnoreCase("10:00")){
            return 8;
        }
        if(startTime.equalsIgnoreCase("16:00")){
            return 8;
        }
        if(startTime.equalsIgnoreCase("18:00")){
            return 6;
        }
        if(startTime.equalsIgnoreCase("20:00")){
            return 4;
        }

    }

    return defaultShiftLength;
}
```

Figure 25: Method used to estimate the length of a shift based on the startTime, typeOfShift and total number of guests.

In the image above we have our logic for calculating estimated shift lengths. The default shift length for both waiters and dishwashers is roughly 8 hours. The reason these are estimations is because in a restaurant setting the staff will work until the guests are finished dining, this means shifts could be shorter or longer depending on the day.

In the case of waiters, their shifts are regular and fixed. If they start earlier their shift lasts around 8 hours but if they start in the late afternoon / evening their shift ends roughly at midnight (sometimes longer).

However, for dishwashers it is a bit more complicated as the amount of work is tied directly to the start time and the number of guests for that day. Usually dishwashing shifts begin at 17:00 or 20:00 with shifts lasting 8 or 5 hours respectively. Depending on the number of guests the shifts can last longer than normal, anywhere between an additional 2-4 hours.

Regular Expressions

```
public class DateValidator {

    private Pattern pattern;
    private Matcher matcher;
    private static final String DATE_PATTERN =
            "^(19|20)\\d{2}\\-(0?[1-9]|1[012])\\-(0?[1-9]|[12][0-9]|3[01])$";

    public DateValidator() {
        pattern = Pattern.compile(DATE_PATTERN);
    }

    public boolean validate(final String date) {

        matcher = pattern.matcher(date);
        return matcher.matches();

    }
}
```

Figure 26: Example of regular expressions used in the GUI.

In our main use case, we do not make use of regular expressions, but we do use them in the manage employee use case for numerous sections. Above is one example of regular expressions we used, it is a date validator to check if a date inputted into a field is valid according to the regex pattern. A simple validate method is included in the class to allow easy checking of strings to see if they match the regex pattern.

Multi-Threading

```
private class DisplayAvailableEmployees implements Runnable{

    @Override
    public void run() {
        workloadTable.setEnabled(false);
        addEmployee.setEnabled(false);
        removeEmployee.setEnabled(false);
        newShiftButton.setEnabled(false);
        enableAvailableEmployeeInformation(false);
        schedCtr.setAvailableEmployees(schedCtr.getAvailableEmployeesForCurrentWorkload());
        availableEmpModel.rebuildAvailableEmpTableModel(schedCtr.getAvailableEmployees(), schedCtr.getCurrentWorkload().getDate());
        availableEmployees.getColumnModel().removeColumn(availableEmployees.getColumnModel().getColumn(4));
        enableAvailableEmployeeInformation(true);
        newShiftButton.setEnabled(true);
        addEmployee.setEnabled(true);
        removeEmployee.setEnabled(true);
        workloadTable.setEnabled(true);
    }
}
```

Figure 27: Example of multi-threading used in the GUI.

For our system, we use multi-threading for time complex operations to avoid GUI freezing / hanging. The example shown above is a private class from our ManageScheduleGUI class in the GUI Layer, the purpose of this class is to retrieve the available employees for a given workload and display it in a Jtable called availableEmployees. However, while the thread carries out the time complex operation of retrieving available employee there are certain parts of the GUI which need to be disabled and once the operation has completed they are re-enabled.

Error handling + Lambda

```
JButton save = new JButton("Save");
save.addActionListener(e->{
    String errorString ="Error saving shifts:";
    try {
        schedCtr.saveWorkloadShifts();
        JOptionPane.showMessageDialog(frame, "Workloads successful saved!", "Success", JOptionPane.PLAIN_MESSAGE);
    } catch (Exception e1) {
        e1.printStackTrace();
        errorString += " \n" + e1.getMessage();
        JOptionPane.showMessageDialog(frame, errorString, "Error", JOptionPane.ERROR_MESSAGE);
    }
});
```

Figure 28: Example of error handling and lambda expression use in the GUI.

This example shows the functionality of the save button from the ManageScheduleGUI class. Here we add an action listener and make use of the lambda function (e->) to allow us to write a method inside the parameter without creating a new actionListener class.

This method first enters a try block and attempts to save all the modifications made to the schedule. If it is successful without encountering exceptions a dialog box pops up informing the user that the workloads have been successfully saved.

In the event an exception is thrown by the saveWorkloadShifts() method, it is caught and an error dialog box pops up informing the user the save was unsuccessful and displays the message generated by the exception.

System test cases

For our main use case 'Create Schedule' making a system test case proved difficult as the way we had designed the GUI had very limited opportunity to allow for invalid inputs. Sections of GUI were disabled when they were not in use and only re-activated when necessary. And as we didn't allow for any text inputs (only allowing input through selection and combobox options) it meant there practically no way to attempt and invalid input.

Table 5: System test cases for create schedules.

Test case Id	Scenario	Date Is valid	Employee is available/valid	Press Save	Press Cancel	Expected result
1	User chooses date and employee and creates a shift	V	V	V	N/A	The new shift is displayed and its recorded in the database

Table 6: System test cases for create schedule (real values).

Test case Id	Scenario	Date	Employee	Press Save	Press cancel	Expected result
1	User chooses date and employee and creates a shift	2017-06-10	Tamás Kalapács	True	False	The new shift is displayed and its recorded in the database

For our use case 'Manage Employee' however we had many opportunities to test invalid inputs as new employee could be create or existing employees could be updated. Also, when entering dates employees are unavailable could also lead to problems.

Table 7: System test cases for manage employee.

Test case Id	Scenario	Employee Information is correct	Press save	Press cancel	Expected Result
1	Manage Employee Adding new employee valid input	V	V	N/A	Employee created/deleted/ Updated/displayed
2	Manage Employee Adding new employee invalid input	I	V	N/A	Error message is displayed detailing invalid inputs

Table 8: System test cases for manage employee (real values).

Test case Id	Scenario	Employee Information is correct	Press save	Press cancel	Expected Result
1	Manage Employee Adding new employee valid input	Tamás Kalapács, kapalacs@email.com, 12-34-5678	True	False	Employee created/deleted/ Updated/displayed
2	Manage Employee Adding new employee invalid input	Sangey Lama, Slama@, 123	True	False	Error message displayed, showing errors with email and phone number.

When entering dates unavailable for employees we allow both single date inputs and a range of date inputs, this could lead to a number problems when trying to add dates unavailable.

Table 9: System test cases for manage dates off.

Test case Id	Scenario	From date input	To date input	Press add button	Expected Result
1	Manage Dates off – valid single input	V	N/A	V	Date is added to the table
2	Manage Dates off – invalid single input	I	N/A	V	Error message – asking to enter a valid date
3	Manage Dates off – valid from input, valid to input	V	V	V	Range of dates added starting the From-date to the To-Date
4	Manage Dates off – valid from input, invalid to input	V	I	V	Error message – asking to enter a valid date
5	Manage Dates off – Valid from and to input, but excessive range	V	V	V	Adds the first 30 days starting at the From-date.
6	Manage Dates off – Valid from and to input, but to date is before from date	V	V	V	Error message – informing the user to check the to-date is after the from-date

Table 10: System test cases for manage dates off (real values).

Test case Id	Scenario	From date input	To date input	Press add button	Expected Result
1	Manage Dates off – valid single input	2017-06-06	N/A	True	2017-06-06 is added to the table
2	Manage Dates off – invalid single input	Hello	N/A	True	Error message – asking to enter a valid date
3	Manage Dates off – valid from input, valid to input	2017-06-06	2017-06-08	True	Dates from 2017-06-06 to 2017-06-08 are added to the table
4	Manage Dates off – valid from input, invalid to input	2017-06-06	Hello	True	Error message – asking to enter a valid date
5	Manage Dates off – Valid from and to input, but excessive range	2017-06-06	2018-06-06	True	Error message displays- date range is too large. Adds the first 30 days starting at the From-date.
6	Manage Dates off – Valid from and to input, but to date is before from date	2017-06-06	2017-06-05	True	Error message – informing the user to check the to-date is after the from-date

Conclusion

We as a group are pleased with the system we created during this process. We could make use of many techniques and practices taught to us during our time. The importance of planning and deciding on a work ethic when working together, how to analyse a business's situation and propose solutions based on that analysis, discovering the requirements through interactions with customers and making designs and diagrams to help plan our system's architecture. All of it has contributed and helped us to understand what goes into a project such as this.

Considerations for future

While we are happy with our system, no system is ever perfect and could always be improved upon. Given more time to expand our system there are some functions we would like to implement.

- Automatic scraping of reservation numbers / event information from 3rd party websites
- Functionality for clocking-in and out for employees for more accurate hours worked information
- Notifications to managers when number of required shifts change (depending on reservation numbers / event information changes)
- Ways to evaluate employee performance using statistics from the database

These are some examples of the things we would have liked to implement in the future.

Evaluation of group work

For the 1st year's Final project our class was given a task to find a company with a requirement for an IT system. Our group decided to create a scheduler desktop application which communicates with a database for Restaurant Fusion. The interview with the restaurant's managers was conducted at a date that was most suitable for them, and we gathered up all the essential information for our project.

Thereafter we met up and brain-stormed what kind of system would be the best meet Fusion's needs. Sangey as our group leader formed an UP Plan, which then guided us through all the iterations of the Inception and Elaboration phases. We created the Project schedule, group contract and divided up the work equally. We weren't always punctual when we decided to meet up, but each of us tried to inform the group in a timely manner.

In the beginning, everybody was trying the best they can, but with time the work focus slightly faded away, occasionally leading to demotivation of group members. The work was mainly split up between pairs, which often resulted in productiveness but sometimes it proved counter-productive, meaning that some work wasn't completed or it was incorrect. The biggest problem arose when it was time to write the system's code. Easier parts were somewhat equally divided, and made by each of us individually, while the more problematic parts were made by Sangey or with his help. GUI design was made by Tamas while Sangey implemented all GUI functions.

Generally, our group enjoyed the time spent together and any problems were resolved fairly. The main goal for all of us was to use the knowledge from the first two semesters to create our system, some of us learned quite a lot and we are still in the process of learning and trying to improve as much as possible.

References

1) Cadle, J. & Yeates, D.: *Project management for information systems*, 2007.
2) Bloisi, Wendy*: Management and Organisational Behaviour*, 2007.
3) Brooks, Ian: *Organisational behaviour*, 2006.
4) Larman, Craig: *Applying UML and patterns*, 2005.
5) Satzinger, Jonh W., Burd, Stephen D., Jackson, Robert*: Object-oriented Analysis and Design with the Unified Process*, 2004.
6) Elmasri, Ramez & Navathe, Shamkant B.: *Fundamentals of Database System*, 2016.
7) Stallings, William*: Operating Systems – Internals and Design Principles*, 2015.
8) Organisational culture: http://www.learnmanagement2.com/culture.htm
9) Porter's Five Forces: https://www.mindtools.com/pages/article/newTMC_08.htm
10) Porter, Michael E.: How competitive forces shape strategy (Harvard business review), 1979. https://hbr.org/1979/03/how-competitive-forces-shape-strategy?referral=03758&cm_vc=rr_item_page.top_right