UNIVERSITAT OBERTA DE CATALUNYA

END OF DEGREE PROJECT

Shift Scheduling System for a Full-Time Healthcare Center

Author:
Ángela LÓPEZ BENÍTEZ

Supervisor:
Oriol MARTÍ GIRONA

A thesis submitted in fulfillment of the requirements for the degree of Computer Engineering

January 9, 2019

UNIVERSITAT OBERTA DE CATALUNYA

Abstract

Computer Engineering - Software Engineering

Shift Scheduling System for a Full-Time Healthcare Center

by Ángela LÓPEZ BENÍTEZ

Staff-scheduling is a complex task nowadays accomplished mostly by hand. There are several tools available to help managers schedule shifts, but they have limitations, be it price, language or lack of features that make them difficult to use for organizations with a tight budget and human resource staff with limited technical experience.

This study is based on the needs of a concrete such organization and its needs in shift scheduling. It starts with an analysis of the problem resulting in a prioritization of features to include in a tool to automatize the scheduling of staff and manage calendar changes. A solution is proposed based on the most important features, detailing both the design of the interface and user workflow, and the data model and architecture of the application. Finally, a prototype of the application is included.

Keywords: human resources, shift scheduling

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Justification

Atendis is a non-profit organization that offers several services to people with intellectual disabilities and their families (Fundació Privada Atendis). Some of the main services it offers are full-time residences and day care centers for people with different degrees of autonomy (Fundació Privada Atendis).

One of the challenges in the management of such an organization is the necessity to have a number of staff present at all times. This means the matter of sorting the schedule of each staff member, as well as their pay sheets, is much more complicated than assigning a 9 to 5 workday and taking into account some holidays. For each center the organization managed, they had to take into account several factors when creating schedules and calendars, including:

- There must be a minimum number of care-taking staff present at all times
- Each staff member should have a yearly period of, at least, the minimal number of consecutive days agreed on their contract
- No staff member should work more than the number of consecutive days established on their contract
- No staff member should work for more than the established hours per year

Historically, this challenge was solved, as many others, manually. Each month, managers would draw schedules repeatedly until all conditions were met, hiring temporary extra staff if they were necessary for gaps created by holidays or free-days that were not taken into account in advance.

These needs are not unique to Atendis. Hospitals, nursing homes, day-care centers, many services such as public transportation, even factories with production flows that cannot be paused require staff to work on shifts and to always have a minimum number of operators present. Nowadays, with the rise of computer-aided management, there are several applications than try to cover different aspects of these needs. For example, according to Business Management Systems' website

[5], their product (Snap Schedule) is able to manage shifts, overtime, time off and many of the requirements of a health care center, and even does so in a MS Office style which might appeal to people transitioning from MS Excel.

However, for a small non-profit, these solutions have too steep a cost, even though they are significantly cheaper than dedicated systems. The managers at Atendis used the tools at their disposal – mainly Excel worksheets – to make the solving of these scheduling puzzles at least more organized, if still manual.

I first saw these calendar worksheets in November 2011. I was volunteering at Atendis, mostly offering computer help and advice to the technical staff (pedagogues, psychologists...), when the manager in charge of organizing schedules asked if I could take a look at the worksheets they used and help them automate some tasks. She showed me a calendar for an employee, where each day (a worksheet cell) was colored in a specific color that represented how many hours the employee should work that day. She simply asked if she could automatically calculate how many cells were painted in each color.

	AGOST								**SETEMBRE**								**OCTUBRE**							
	St.	Dll	Dts	Dcs	Dij	Div	Dis	Dg	**St.**	Dll	Dts	Dcs	Dij	Div	Dis	Dg	**St.**	Dll	Dts	Dcs	Dij	Div	Dis	Dg
	31			1	2	3	4	5	**35**						1	2	**40**	1	2	3	4	5	6	7
	32	6	7	8	9	10	11	12	**36**	3	4	5	6	7	8	9	**41**	8	9	10	11	12	13	14
	33	13	14	15	16	17	18	19	**37**	10	11	12	13	14	15	16	**42**	15	16	17	18	19	20	21
	34	20	21	22	23	24	25	26	**38**	17	18	19	20	21	22	23	**43**	22	23	24	25	26	27	28
	35	27	28	29	30	31			**39**	24	25	26	27	28	29	30	**44**	29	30	31				
	NOVEMBRE								**DESEMBRE**								**GENER 2019**							
	St.	Dll	Dts	Dcs	Dij	Div	Dis	Dg	**St.**	Dll	Dts	Dcs	Dij	Div	Dis	Dg	**St.**	Dll	Dts	Dcs	Dij	Div	Dis	Dg
	44				1	2	3	4	**48**						1	2	**1**		1	2	3	4	5	6
	45	5	6	7	8	9	10	11	**49**	3	4	5	6	7	8	9	**2**	7	8	9	10	11	12	13
	46	12	13	14	15	16	17	18	**50**	10	11	12	13	14	15	16	**3**	14	15	16	17	18	19	20
	47	19	20	21	22	23	24	25	**51**	17	18	19	20	21	22	23	**4**	21	22	23	24	25	26	27
	48	26	27	28	29	30			**52**	24	25	26	27	28	29	30	**5**	28	29	30	31			
									53	31														

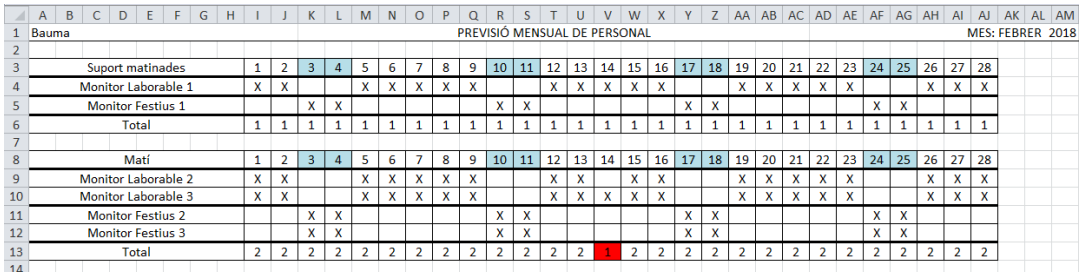
Recompte hores					
	Dies		Hores		
Matí	106	x	4	=	424
Tarda	126	x	4	=	504
Vacances	31	x	0	=	0
HORES TOTALS					928

Conveni	
Hores conveni	
(introduir manualment)	1000
Data inici contracte	
(introduir manualment)	01/02/2018
Data final contracte	
(introduir manualment)	31/01/2019
% Jornada	92,80%

FIGURE 1.1: Screenshot of the calendar of a staff member, with the count of days scheduled for each shift.

From a computer-savvy point of view, I was awed and horrified at the amount of manual work they did that could so easily be automated. Over the next few days, I was able to easily implement the counting of cells according to color (Figure 1.1).

Happy with the results, the next several years we added other features to their semi-automated worksheet each time she had to work out the schedules, such as an automated monthly summary of the staff on each shift (Figure 1.2).



1	Bauma																PREVISIÓ MENSUAL DE PERSONAL																				MES: FEBRER 2018	
2																																					
3	Suport matinades	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								
4	Monitor Laborable 1	X	X			X	X	X	X	X			X	X	X	X	X			X	X	X	X	X			X	X	X								
5	Monitor Festius 1			X	X						X	X						X	X						X	X										
6	Total	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1								
7																																					
8	Matí	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								
9	Monitor Laborable 2	X	X			X	X	X	X	X			X	X		X	X			X	X	X	X	X			X	X	X								
10	Monitor Laborable 3	X	X			X	X	X	X	X			X	X	X	X	X			X	X	X	X	X			X	X	X								
11	Monitor Festius 2			X	X						X	X						X	X						X	X										
12	Monitor Festius 3			X	X						X	X						X	X						X	X										
13	Total	2	2	2	2	2	2	2	2	2	2	2	2	2	1	2	2	2	2	2	2	2	2	2	2	2	2	2	2								
14																																					

FIGURE 1.2: Screenshot of a monthly summary for dawn and morning shift. You can see a warning for the February 14th morning shift, as there is only one staff member instead of the requisite two.

This semi-automated approach has numerous problems, though. Excel is a worksheet software, not optimized to be used as a scheduling system. The main problem we have encountered is that the processing of the calendars can be very slow, since the performance of reading and writing into the worksheets is very low. It also has a very awkward, stiff user interface, since it is severely limited by the options Excel offers.

As a software engineering student, I think this project is the perfect opportunity to take these semi-automated worksheets and create a full scheduling solution that will solve many of the UX and performance issues of forcing Excel to behave in ways it was not designed to. Furthermore, this will be an open-source project, enabling other organizations with limited means to leave clunky worksheets behind.

1.2 Alternatives

Looking at staff management software as a whole, one can distinguish two well separated types. The first one is software centered on projects, and the features dealing with staff - or "resources" - are focused on their assignation to certain projects. This is by far the most popular type, and there are dozens of applications that have features dealing with management of staff as resources, from Jira to Monday. These type of applications are the ones used to manage software projects, and as most tooling surrounding software development, there are several open source alternatives, such as Redmine [2, 10, 15].

The second type is focused on time or shifts instead of projects or tasks to be done. Not being used as often by technology teams, this type is a little behind and there are not as many open source options, although there are several commercial applications. In one recent article, PCMag compared 10 such applications. Most of

these, as well as several others not featured in this comparison, seem to be focused on smaller teams [14, 1, 11], with several showcasing the restaurant use case.

They have features to handle shifts of different types - waiter or cook, for example - and enable the manager to visually keep track of each day, making sure all shifts are covered. However, most of these applications do not offer any feature to semi-automatically schedule shifts for the whole staff, nor do they use any kind of signaling of scheduling conflicts, relying strongly on the visual representation for most management. LaborOfficeFree, which is in Spanish and free, although not open source, seems to fit into this group [13].

Many, especially the applications geared toward larger organizations, have a strong focus on keeping track of the time worked by each employee, with features such as time-clocking. This is the case of Identifica-t, which boasts integration with several biometric sensors to keep track of clock-ins and clock-outs [9].

Then there are a select few applications, such as SnapSchedule mentioned before, that have the features necessary to manage shifts in a larger scale [5, 12, 6]. They offer algorithms to generate schedules based on any number of customizable "rules" - shift hours, vacation days, rest periods... - and provide different kinds of signaling of scheduling conflicts. However, these features come at a cost. Snap Schedule, which has a public pricing page, costs $450 per one-computer, perpetual license or $450 per scheduler per year, if contracted as a service [4]. I have not been able to find any free-ware or open source application of this type.

If you take into account the language limitation, the options are fewer. There are some shift-management tools available in Spanish, but most focus on smaller teams or time-clocking. Most provide features to manage staff and contracts, such as Tamigo and EntiGest, but they do not offer the same schedule-management features as the larger applications [13, 9, 16, 7].

1.3 Scope

The main objective of this project is to design and implement a working prototype for a scheduling system that meets the requirements of a nursing home. As the time-requirements for this project are strict, the project will not include a fully working MVP, but an application that serves as a proof-of-concept, implementing the main differentiating features, and ignoring other features which would be necessary for a finished product but do not really provide nothing new to the project, such as authentication and authorization.

As such, the scope of the project includes:

- Stakeholder identification and requirement collection,

- Design via mock-ups of a solution that covers the main requirements discovered.
- Technical design of the application, including the data model to be used and the communication between different components.
- Implementation of a first version of the application, including the most important features collected.

On the other hand, it will not include:

- **Deployment**. Although deployment will be taken into account to make other decisions such as platform, the deployment itself is not in the scope of this project.
- **User authentication and authorization**. Given that the scheduling system might contain sensitive information, and that it should not be edited by anybody at an organization, the system should include some form of authentication. However, there already exist several solutions that manage users, and it will not influence the workings of the system itself.
- **Internationalization**. Despite being one of the main problems of the existing alternatives, internationalization of an application is a time-consuming process that does not affect the user experience for the immediate audience, so it will be excluded from this version of the application.
- **Automatized end-to-end tests**. While automatized tests are key to the maintainability of an application, they do not provide anything feature-wise, so they will be relegated to a second phase of the project.
- **Continuous integration and deployment**. Like the previous issue, this takes considerable effort to set-up while delivering no business value, so it will be excluded from this project.

1.4 Methodology

The analysis of the problem will start with the identification of the stakeholders of the system. This will be accomplished mostly by talking to my contact at Atendis and finding out each person who schedule the staff. The manager herself, and the staff who is being managed are also affected by the system, so they should be consulted too. These will be the starting points to finding out if there are any other stakeholders in the system.

For the discovery of requirements, design and implementation, I will use an iterative method using several tools of agile methodology. I have considered starting the project with a brainstorming session with all the stakeholders, which could reveal

hidden requirements that are difficult to see for anyone but a particular person. However, there are two strong points against it. First, given their schedules, it might be difficult to find a time all stakeholders are free. Second and most important, taking into account my previous interactions with most stakeholders, I expect them to be relatively inexperienced with computer software, so I believe it would be difficult to them to express their needs of the application without something to start of from.

Instead of brainstorming with all of them, I will jump-start the first iteration by talking to the manager of the center. She is more knowledgeable about computer software and has handled the scheduling of shifts several times. During this interview, we will map out the main user stories of the system. Afterwards, I will start the design process with several mock-ups of the different features desired. If there is any feature which requires it, such as some interaction that is difficult to understand in the more static wire-frames, I will also develop a proof-of-concept prototype to show case the user story and features involved.

The following iterations will follow a similar pattern. However, with user stories, mock-ups and possible prototypes in hand, I will talk to the other stakeholders. I expect that this will offer them a starting point that they might validate or oppose in more specific ways. During these interviews, I plan to use example mapping to dig into each user story and find out if there are any questions about each that needs to be answered. This will also help document each user story better than a vague objective.

In parallel to these iterations of interviews and examples, I will start to design the more technical part of the application, such as the data model and the component diagram. Although I will not show the stakeholders the progress in this area, I expect to further detail each diagram as the interviews progress and the user stories become more stable. Once the user stories start to solidify, I will choose the language, framework and tools I will use in the development of the application. During the last few iterations, I will start to develop the application itself, changing the meeting with the stakeholders to show the different user stories implemented instead of mock-ups.

I do not expect to finish the system as a whole in the duration of this project, however I plan to end the project with a prototype of the system that covers the main requirements collected and an idea of how to continue towards a working MVP of the system. As such, I will end the project with an analysis of further work necessary to make the application fully functional.

1.5 Schedule

I have used the online tool [17] to visually plan the project in a Gantt diagram captured in figures 1.3, 1.4 and 1.5. As I described in the methodology, I will use an iterative approach to the development of the project. I have roughly scheduled the

duration of each iteration, taking into account the days I will be able to speak with the stakeholders and the days I expect to be less busy (holidays from work, weekends, etc.). I have also planned for regular reviews of the report, especially before a partial check-in was due. When there were several tasks that may be accomplished in parallel, I have tried to plan them in the order I thought most logical. There are also several tasks that are scheduled in parallel, as I thought they might benefit from a back-and-forth between them before writing the final version of each part of the report.

To facilitate the scheduling of each task visually, I have colored purple the meetings with the stakeholders, blue design and implementation tasks and purple the documentation and report-oriented tasks.

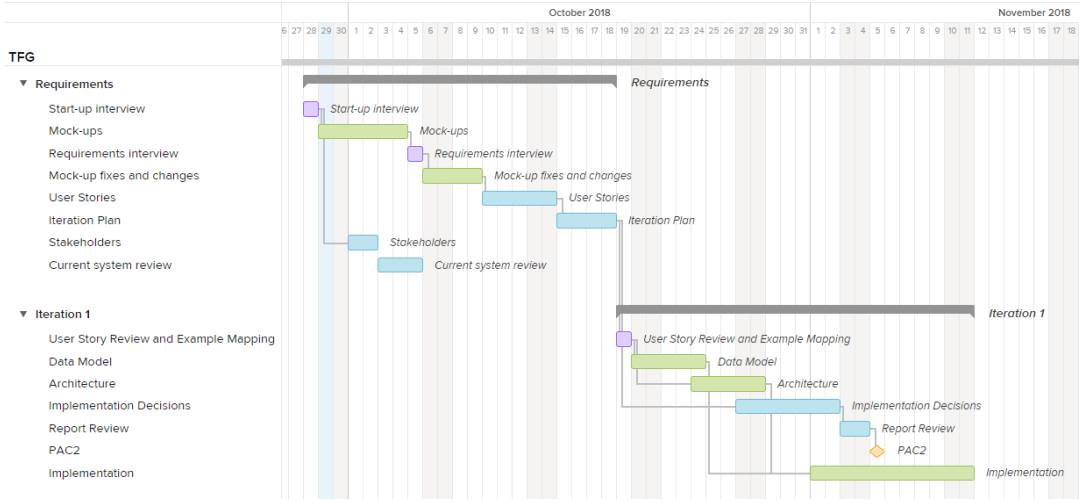


FIGURE 1.3: Screenshot of the planning of the tasks in the requirements and first iteration, including the second PAC.

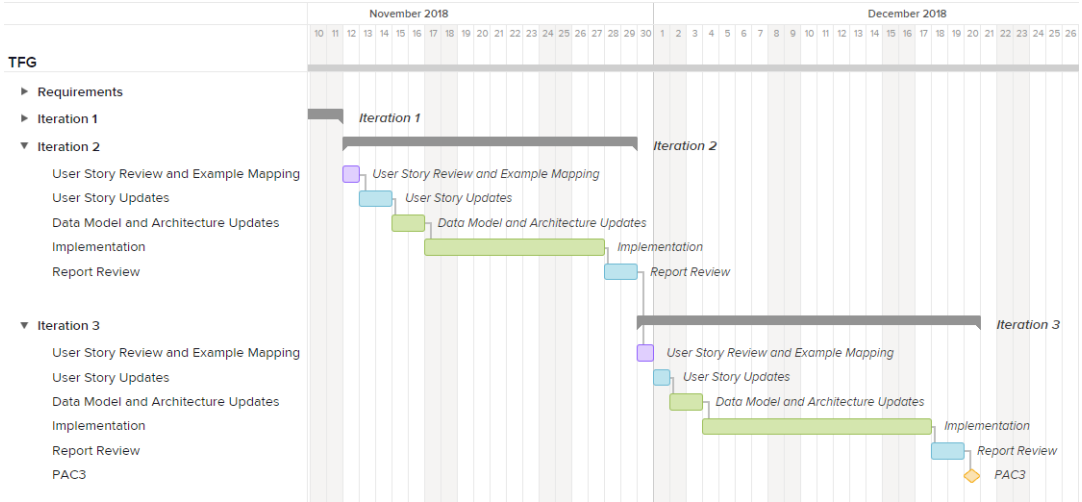


FIGURE 1.4: Screenshot of the planning of the tasks in the second and third iterations, including the third PAC.

Table 1.1 offers a summary of the planned and actual dates of each task in the project.

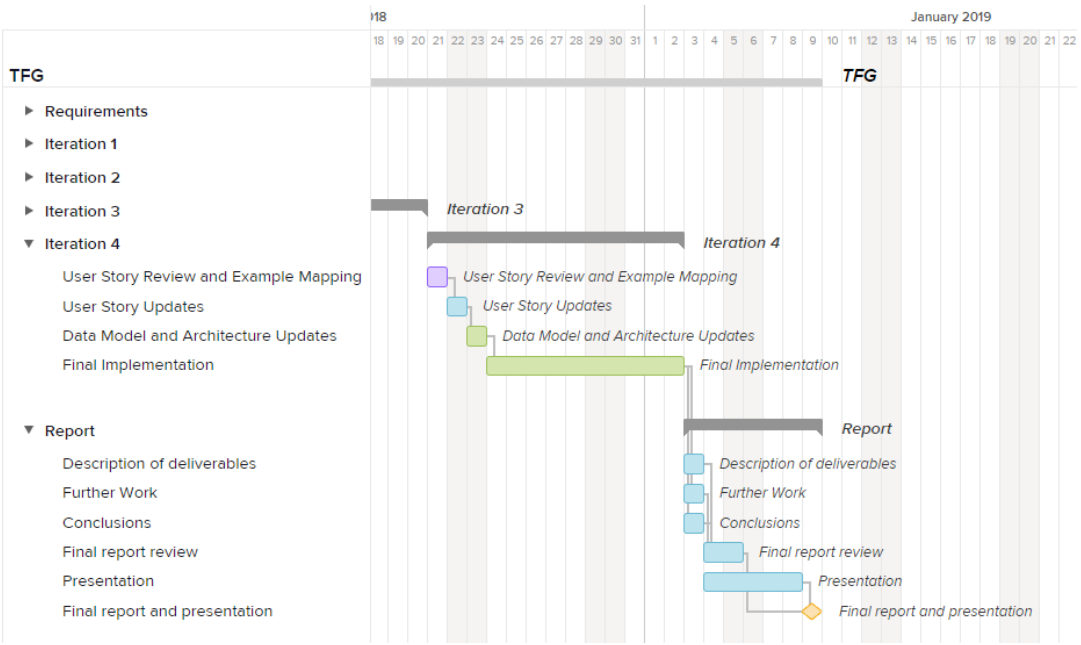


FIGURE 1.5: Screenshot of the planning of the tasks in the final iteration and the finalization of the project, including the final due date.

TABLE 1.1: Summary of planned and actual dates for each task of the project

Tasks	Planned Start	Planned End	Actual Start	Actual End
Project Plan				
Abstract	-	-	22/9/2018	22/9/2018
Justification	-	-	18/9/2018	22/9/2018
Alternatives	-	-	26/9/2018	29/9/2018
Scope	-	-	29/9/2018	30/9/2018
Methodology	-	-	22/9/2018	27/9/2018
Schedule	-	-	29/9/2018	30/9/2018
Requirements				
Start-up interview	-	-	28/9/2018	28/9/2018
Mock-ups	29/9/2018	4/10/2018	1/10/2018	9/10/2018
Requirements interview	5/10/2018	5/10/2018	10/10/2018	10/10/2018
Mock-up fixes and changes	6/10/2018	9/10/2018	11/10/2018	14/10/2018
User Stories	10/10/2018	14/10/2018	2/10/2018	14/10/2018
Iteration Plan	15/10/2018	18/10/2018	16/10/2018	16/10/2018

Tasks	Planned Start	Planned End	Actual Start	Actual End
Stakeholders	1/10/2018	2/10/2018	1/10/2018	1/10/2018
Current system review	3/10/2018	5/10/2018	1/10/2018	2/10/2018
Iteration 1				
User Story Review and Example Mapping	19/10/2018	19/10/2018	19/10/2018	24/10/2018
Data Model	20/10/2018	20/10/2018	19/10/2018	27/10/2018
Architecture	24/10/2018	24/10/2018	19/10/2018	19/10/2018
Implementation Decisions	27/10/2018	2/11/2018	19/10/2018	19/10/2018
Report Review	3/11/2018	4/11/2018	4/11/2018	5/11/2018
Implementation	1/11/2018	11/11/2018	27/10/2018	30/11/2018
Iteration 2				
User Story Review and Example Mapping	12/11/2018	12/11/2018	30/11/2018	30/11/2018
User Story Updates	13/11/2018	14/11/2018	-	-
Data Model and Architecture Updates	15/11/2018	16/11/2018	-	-
Implementation	17/11/2018	27/11/2018	1/12/2018	16/12/2018
Report Review	28/11/2018	29/11/2018	20/12/2018	20/12/2018
Iteration 3				
User Story Review	30/11/2018	30/11/2018	21/12/2018	21/12/2018
User Story Updates	1/12/2018	1/12/2018	22/12/2018	23/12/2018
Data Model and Architecture Updates	2/12/2018	3/12/2018	24/12/2018	27/12/2018
Implementation	4/12/2018	17/12/2018	27/12/2018	5/1/2019
Report Review	18/12/2018	19/12/2018	5/1/2019	5/1/2019
Iteration 4				
User Story Review and Example Mapping	21/12/2018	21/12/2018	-	-
User Story Updates	22/12/2018	22/12/2018		-

Tasks	Planned Start	Planned End	Actual Start	Actual End
- Data Model and Architecture Updates	23/12/2018	23/12/2018	-	-
Final Implementation	24/12/2018	2/1/2019	-	-
AWS Account Setup	-	-	5/1/2019	5/1/2019
AWS RDS	-	-	7/1/2019	7/1/2019
AWS ECR	-	-	7/1/2019	7/1/2019
AWS ECS	-	-	8/1/2019	8/1/2019
Report				
Description of deliverables	3/1/2019	3/1/2019	9/1/2019	9/1/2019
Further Work	3/1/2019	3/1/2019	9/1/2019	9/1/2019
Conclusions	3/1/2019	3/1/2019	9/1/2019	9/1/2019
Final report review	4/1/2019	5/1/2019	7/1/2019	9/1/2019
Presentation	4/1/2019	8/1/2019	8/1/2019	9/1/2019

1.6 Deliverables

Report: This report, with the different chapters detailing the requirements, design and implementation comments of the scheduling system.

Mockups: mockups.zip contains several BMPR files which may be viewed with the Balsamiq Mockup desktop application or Drive plugin (https://balsamiq.com/).

Iteration 3 diagrams: iteration3_diagrams contains several diagrams of the third iteration in full size.

Monthly Grid Source Code: Iteration2_MonthGrid_POC.zip contains the source code for the grid demo. Unzip and open index.html to execute.

ScheduleSystem Source Code: ScheduleSystem.zip contains the source code for the application, including the (uncomplete) implementation of the staff schedule editing.

Database Creation SQL: table_creation.sql contains the script to create the necessary tables of the database.

Presentation: Presentation.odp contains Impress presentation of the project.

Presentation Video: Presentation.mp4 contains narrated presentation of the project.

Presentation Transcript: Presentation_transcript.txt contains the transcription of presentation of the project.

Chapter 2

Requirements

2.1 Stakeholders

After the first meeting with the manager of the nursing home, we were able to identify the main stakeholders of the application:

- **Manager**. She is the person leading the technological advancement of the organization, and has expressed an interest on using a more modern approach to managing schedules several times. As the person ultimately responsible for all staff, she wants to alleviate the scheduling problems the organization has. Furthermore, she is interested on keeping the time spent on scheduling issues to a minimum, as this is time the scheduling employees could better spend doing other tasks.
- **Schedulers**. These are two or three employees who routinely handle the tasks of scheduling the shifts of the nursing staff. Most of the tasks related to this come in bulk at the beginning of each year, when they renew the calendars for most staff. Besides this, they sometimes have some schedule changes if there is new staff, somebody takes leave, or asks for a shift reduction.
- **Staff**. I will refer as staff to most of the employees whose shifts are to be planned with the application. Although they will not interact directly with the application, they are affected by it. For example, with an application that makes schedule changes easier, the schedulers will be most likely to grant a shift change.

We were also able to identify some special cases that merit special attention, so we should list them apart:

- **Assisted living manager**. He is the manager of a separate smaller center which houses higher-functioning people. Because of this, usually there is only one person scheduled to work, and puzzling together rest periods, leaves and reinforcements during peak hours - morning and evening routines - is a bit

trickier. His calendars end up looking much more like patchwork that those of the larger center.

- **Activities manager**. She is the manager of the activities each group makes. Besides taking into account the legal requirements of having a one care-taker for each number of residents, she wants to have an appropriate number of staff or volunteers for each activity.

2.2 Current System

Given the lack of experience using similar applications, all stakeholders are understandably biased towards the current system, and at least at first, all descriptions of features are expected to be in relation to the current Excel macros. As such, an explanation of the current application is necessary.

The first worksheet of the calendars workbook contains the configuration for the other sheets. It enables the schedulers to add or remove shifts and edit the properties of each shift: the minimum staff, the shift hours and the color used in both the working and holiday calendars. An extra worksheet has the calendar for the selected year, where the schedulers can mark the local holidays so the system will take them into account.

Another worksheet is used to manage the staff in the system. Each staff member belongs to a center and group. In practise, this is used to determine the number of staff in each center-group combination. Each staff member also has a main shift. One of the limitations of the current system is that each person can only be assigned a single shift from a very limited list, so the schedules are very restrictive from the onset.

To manage holidays, each "workday" staff member has an assigned partner that generally covers the same center and group in the days their partner has free. Of course, this cannot be a hundred percent true, as a workday staff member may have 30 days of holidays in a row, and their partner would not be allowed to have as many continued days of work. However, this is a technique that schedulers use to jumpstart the creation of calendars each year.

To start scheduling the shifts for a staff member, the manager can select a "workday" person to create their starting calendar. This creates a new worksheet for the staff member with a starting calendar. This calendar has all workdays colored with the person's shift. From there, the schedulers can color in the holidays, weekends worked, etc. At this time, they are also able to change the shift worked in any given day. Besides the calendar, each person's worksheet also is generated with formula-enabled cells that help the manager calculate the yearly hours a staff member is scheduled to work and their percentage of full-time.

Once finished the calendar for the workday worker, the managers can create the mirror calendar for the holiday partner. Besides the pre-filled calendar, this worksheet is equal to their workday partner's worksheet, so the managers will edit them to meet the scheduling rules. Once they are finished, or at any given time before, the schedulers can automatically check all calendars, and the system will warn them if any calendar breaks one of the programmed scheduling rules.

Finally, another worksheet is generated for each center, which contain monthly summary grids for every shift. These are the final validation the schedulers use to see who is working each day and if all days have the requisite staff scheduled. Besides the ability to automatically count the cells marked with each turn in the staff calendars, this worksheet is the main feature in the system. It is also the main reason behind the discrete nature of the shift concept in the current system. In an effort to visually represent the staff scheduled in during each day, the managers were forced to break the day into strict shifts that do not faithfully represent the reality of the staff's schedules.

2.3 Start-Up Interview

While I did not have any designs or ideas ready for this first interview, the manager came prepared with a list of limitations and missing features of the current system. The most significant issues are:

- There is no feature to easily change a contract or schedule. For example, if an employee asks for a shift reduction or a part-time employee changes full-time to cover a sick leave, there is no way to edit their calendars and yearly percentage of full-time. The system is working fine for puzzling together a yearly calendar, but in practice, these calendars rarely make it "as is" to the end of the year.
- The system has no continuity from a year to the next. The reality of the variable schedules and working hours of the employees means that, if they want to have a stable salary, each employee must keep a "pool of hours", a number of hours worked and not paid (or paid and not worked), to cover substitutions or small differences in contract hours from one year to the next.
- While the current system is based on "shifts" (morning, afternoon, evening...), many employees had varying schedules: one could have a 8 to 14 morning shift, while another could have a 8 to 12 because they had reduced hours. This makes features based on turns non-functional. For example, given the previous scenario, the monthly grid the schedulers use to check the number of employees at each turn would not tell that they were short one employee from 12 to 14.

The first two issues are rather straightforward, but the last merits more thought. To start, we decided to rethink the concept of "shift", so it is not a fixed range of hours. So, we'd call shift to any continuous time of work an employee does, acknowledging that the start and end times of these shifts may be different for each staff member.

The initial proposal, that I tried to explain with mock-ups, was to have an interactive grid which in it's main view showed which employee had any shift each day, but that could be expanded to show the hour-by-hour status of each staff member. For this to work, though, the main view should somehow show if there was any conflict during any day.

2.4 User Stories

Between the knowledge of the current system, the start-up interview, and the followup refinement of the expected features of the system, we were able to agree upon the following user stories:

- As a manager, I want to be assured that the requisite number of staff are scheduled at all times; and if possible, not more.
- As a manager, I want to each employee to be scheduled to work as close as possible to the hours agreed on their contract without breaking any of their contract restrictions.
- As a manager, I want to see a historic of the percentage of full-time an employee has had.
- As a manager, I want to assign a contract to a worker when I add them to my staff.
- As a manager, I want to remove people from my staff.
- As a manager, I want to automatically rotate contract details of my staff each year (holiday shift, annual worked hours, etc.)
- As a manager, I want to export the staff's schedules to a worksheet.
- As a manager, I want to edit the centers my organization manages.
- As a manager, I want to edit the minimum staff necessary at each time at each of the centers.
- As a scheduler, I want to be able to edit the local holidays for the next year.
- As a scheduler, I want to generate the schedules for all my staff.
- As a scheduler, I want to manually edit the schedules of my staff.

- As a scheduler, I want to be able to see who is scheduled to work any given day at any given time.
- As a scheduler, I want to be advised of any kind of conflict in the schedule of my employees.
- As a scheduler, I want to automatically recalculate salaries when I have to make contract changes halfway through a year.
- As a scheduler, I want to create a new schedule (for a new employee) halfway through a year.
- As a scheduler, I want to liquidate an account halfway through a year (balance hours paid with hours worked to adjust final payment).
- As a scheduler, I want to see which employees are free to cover a given shift.
- As a scheduler, I want to print the monthly list of shifts of my staff to be able to post it where my staff can see it.
- As an employee, I want to see a summary of my shifts during the following weeks.
- As an employee, I want to see a summary of my colleague's shifts to be able to ask for trades.
- As an employee, I want to trade shifts with a colleague.
- As an employee, I want to ask to exchange holiday days for free Mondays after I work on a weekend.
- As an employee, I want to ask for a shift reduction or to return to full-time.
- As an employee, I want to see a preview of the schedule and monthly salary I would have if I ask for a shift reduction.
- As an employee, I want to have a stable monthly and yearly salary, independently of small monthly and yearly hours differences.

However, as the effort required to implement the full set of user stories is greater than this project's scope, so we refined the backlog into a smaller set which would provide a working prototype:

- As a manager, I want to be assured that the requisite number of staff are scheduled at all times; and if possible, not more.
- As a manager, I want to each employee to be scheduled to work as close as possible to the hours agreed on their contract without breaking any of their contract restrictions.
- As a scheduler, I want to generate the schedules for all my staff.
- As a scheduler, I want to manually edit the schedules of my staff.

- As a scheduler, I want to be able to see who is scheduled to work any given day at any given time.
- As a scheduler, I want to be advised of any kind of conflict in the schedule of my employees.
- As a scheduler, I want to automatically recalculate salaries when I have to make contract changes halfway through a year.
- As a scheduler, I want to create a new schedule (for a new employee) halfway through a year.
- As a scheduler, I want to liquidate an account halfway through a year (balance hours paid with hours worked to adjust final payment).
- As a scheduler, I want to see which employees are free to cover a given shift.
- As an employee, I want to have a stable monthly and yearly salary, independently of small monthly and yearly hours differences.
- As an employee, I want to ask to exchange holiday days for free Mondays after I work on a weekend.
- As an employee, I want to ask for a shift reduction or to return to full-time.
- As an employee, I want to see a preview of the schedule and monthly salary I would have if I ask for a shift reduction.

There are some stories that seem to be prerequisite to others, such as being able to edit the minimum staff necessary at each center, but this can easily be "hard-coded" in the prototype, and treated as something configurable in a later phase of the development.

Besides the user stories themselves, we were able to determine which were the contract restrictions that should be taken into account when generating or checking schedules:

- Annual hours worked
- Maximum number of consecutive work days
- Minimum hours of rest between shifts in different days
- Maximum hours worked in a given interval of time
- Minimum consecutive natural free days in summer, spring and winter holidays
- Number of weekends worked every N weeks

2.5 Non-Functional Requirements

As this project is centered in a proof of concept development, most non-functional requirements will not be at the forefront of the project. Nevertheless, there are some issues that should be taken into account. I have used a couple of generic non-functional requirement checklists to help realize the requirements I should take into account in this early phase of the project [8, 3]. Non-functional requirements in the areas of auditing, maintainability and other "maintenance" areas are not included as I consider them out of the scope of this project.

- **Security**. There will only be one access level, with authorization to use all the system's features.
- **Performance**. Response times for navigation through the screens should be in the order of seconds, ideally less than 5 seconds. Process time (change a schedule, find a free staff member to cover a shift, etc. should be in the order of tens of seconds, ideally less than a minute.
- **Capacity**. The system should be able to handle at least 10 centers with 100 staff each.
- **Compatibility**. The schedules should be exportable to Excel worksheets.
- **Usability**. The system should be usable in screen sizes as small as 1768 pixels wide. The system should be available in Catalan and/or Spanish.

2.6 Iteration Plan

Given the described requirements and the time constraints we agreed to the following iteration plan:

- Iteration 1:
  - As a scheduler, I want to generate the schedules for all my staff.
  - As a scheduler, I want to manually edit the schedules of my staff.
- Iteration 2:
  - As a manager, I want to be assured that the requisite number of staff are scheduled at all times; and if possible, not more.
  - As a scheduler, I want to be advised of any kind of conflict in the schedule of my employees.
  - As a scheduler, I want to be able to see who is scheduled to work any given day at any given time.

- Iteration 3:
  - As a scheduler, I want to automatically recalculate salaries when I have to make contract changes halfway through a year.
  - As a scheduler, I want to create a new schedule (for a new employee) halfway through a year.
  - As a scheduler, I want to liquidate an account halfway through a year (balance hours paid with hours worked to adjust final payment).
- Iteration 4:
  - As a scheduler, I want to see which employees are free to cover a given shift.
  - As a manager, I want to each employee to be scheduled to work as close as possible to the hours agreed on their contract without breaking any of their contract restrictions.

Chapter 3

Iteration 1

3.1 User Stories

The first iteration is centered around the generation of schedules for the staff. There are several user stories from the shortened list that we could cover with this theme:

- As a scheduler, I want to manually edit the schedules of my staff.
- As a scheduler, I want to generate the schedules for all my staff.
- As a manager, I want to each employee to be scheduled to work as close as possible to the hours agreed on their contract without breaking any of their contract restrictions.
- As a scheduler, I want to create a new schedule (for a new employee) halfway through a year.
- As a scheduler, I want to liquidate an account halfway through a year (balance hours paid with hours worked to adjust final payment).
- As an employee, I want to ask to exchange holiday days for free Mondays after I work on a weekend.
- As an employee, I want to ask for a shift reduction or to return to full-time.
- As an employee, I want to see a preview of the schedule and monthly salary I would have if I ask for a shift reduction.

However, in this iteration I will focus on the first two. These two stories should be enough to start designing the entities of the system and the relations between each. Unfortunately, I was unable to do an example mapping [18] session with the stakeholders, but I has able to exchange several emails with them and detail the user stories planned for the iteration.

The first one should be self-explanatory. Given a staff member, the scheduler should be able to add, delete or modify shifts (change start and end times). This should be

done in a manner as visual as possible, for example, selecting a shift and "painting" the days corresponding to this shift in a calendar.

The second user story requires more explanation. There are basically two types of schedules: *weekday schedules* and *holiday schedules*, and the generation of schedules for the whole staff is usually based around the first. Weekday staff usually have the same shift from Monday to Friday and do not work on holidays. There are two versions of weekday staff: the first one do not work on Saturdays, but the second work one Saturday every four weeks. Of the latter type, they also have the option of exchanging holiday days for free Mondays after working on a Saturday. Furthermore, each worker is assigned a holiday turn. For example, for summer holidays, a staff member could have the first turn, in July, or the second turn, in August.

So, to meet the acceptance criteria for the second user story, the system should automatically generate weekday schedules from these conditions. In the case of the holiday schedules, the acceptance criteria for this story are somewhat simpler. The holiday schedule should fill all days its weekday schedule partner has free. Of course, this would result in an impossible schedule given the limitations listed in section 2.5. However, this is acceptable at this moment, and these schedule conflicts will be manually resolved.

3.2 Design

3.2.1 Wireframes

To start with, I have ignored the features around configuring the staff requirements for each center in the system. I have also supposed that the staff available is already in the system. Given these suppositions, the design I proposed is a simple list of available staff, where one could search through the staff and click on each one to see generate or see their calendar.

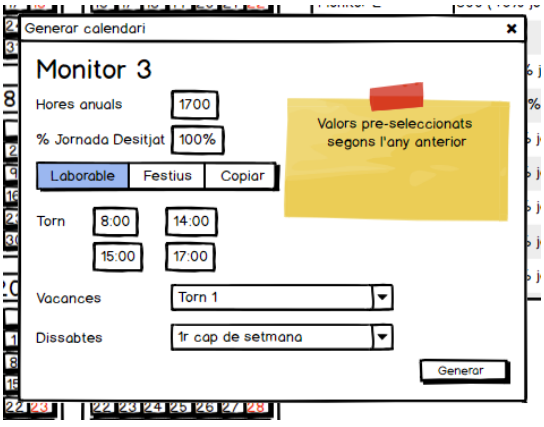


FIGURE 3.1: Wireframe of the calendar options selection window, showing the options for a weekday calendar.

Upon clicking a "generate schedule" button, the system will present a form to select the options for the calendar, including the type of calendar (weekday or holiday). Once the type of calendar is selected, further options are available, such as their shift and holidays and Saturday turns in the case of a weekday calendar seen in figure 3.1, or the partner-calendar in case of a holiday calendar seen in figure 3.2. In case the is a pre-existing schedule for the previous year, the values should be selected accordingly (all values should be the same, except for the holidays turns, which are rotatory).

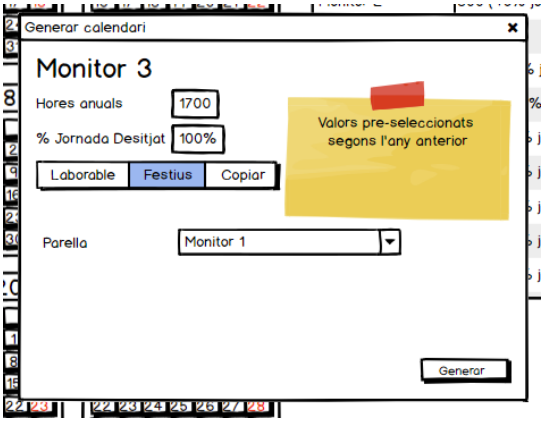


FIGURE 3.2: Wireframe of the calendar options selection window, showing the options for a Holiday calendar.

After clicking the "generate" button, the system would automatically fill in a calendar with the given specifications, navigating to the Staff detail view after it is finished as seen in figure 3.3. In this screen, the scheduler will be able to see the calendar of the selected staff member and manually edit it by selecting one of the shifts and "painting" over the days in the calendar.

3.2.2 Architecture

To start with, the system will use an architecture based on layers. It is specially important to have a domain layer, where all business logic will be implemented separate from the presentation and infrastructure layers. This is specially important since one of the secondary, long-term objectives of this project is to release the source code in a way that might be adapted to other scheduling systems that might need it. For this to work, it is important, for example, to decouple the business logic from the interaction with the database, as it might use a different service in other organizations.

In the presentation, I will use a model-view-controller pattern. This is a well-used pattern that allows the separation of responsibilities in a clear manner.

FIGURE 3.3: Wireframe of the detail view of a staff member.

3.2.3 Class Diagram

I will use an UML class diagram in figure 3.4 to describe the interaction between the domain entities. In this iteration, the domain focus is on the *Schedule*. An *StaffMember* can have an indefinite number of schedules, but only one active at any given time. Each schedule groups together the shifts between its dates. The schedules are generated from a *ScheduleModel*. This is an abstract class, and each of its concrete implementations has the responsibility to generate an schedule according to its specific rules.

However, besides creating the shifts themselves, it is also the responsibility of the domain to check the various schedules constraints each time a change is made. Because I could not fit this logic into any single entity, I added a *SchedulingService*. This is a service that will be called by the application layer when making scheduling changes, and will handle the logic that does not pertain to a single entity.

3.3 Implementation Decisions

The system will be implemented in C#, specifically using the .NET Core framework. The main reason for this decision is simply because it is a language and framework that I am familiar in. The .NET framework is also very popular and has a large community, so help is relatively easy to find on the internet, and there are many

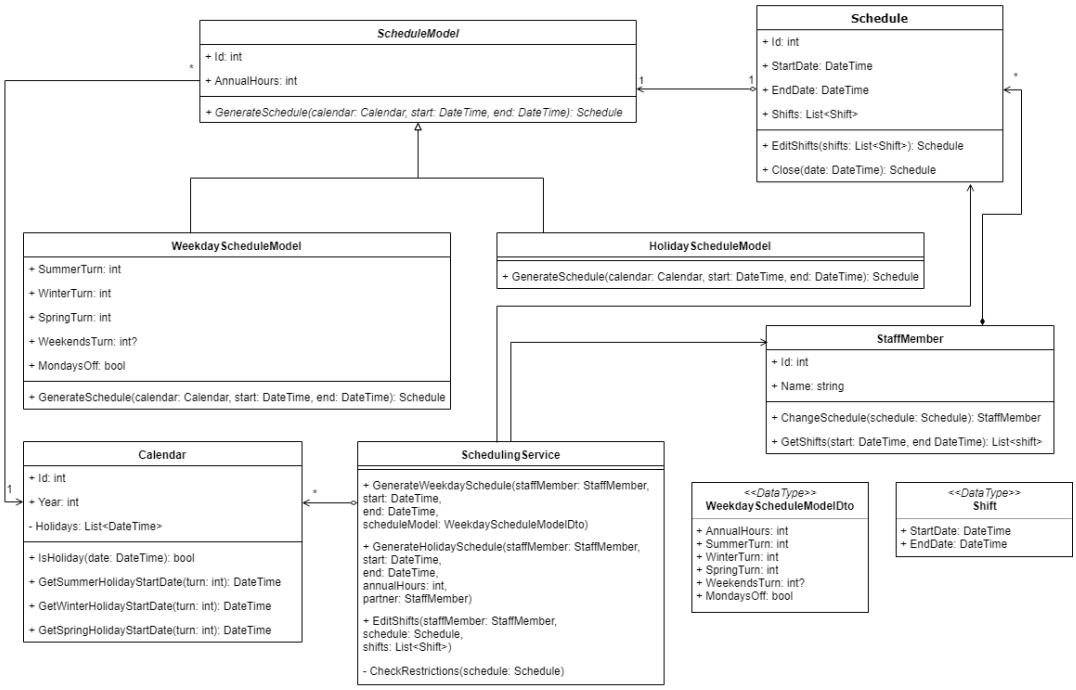


FIGURE 3.4: Class Diagram.

ready-to-use packages available. .NET Core is not as popular, but still has a considerable community behind it. Furthermore, it is able to run on any OS, so this is a great advantage if I want to make the system reusable by other organizations.

The system will be implemented as a website. While at first glance a desktop application might be easier to maintain, however, this is not within the scope of this project. An important part of this project, however, is the somewhat unconventional user interface for editing shifts. In my experience, this kind of interface is easier to implement with HTML, CSS and JS than traditional forms-based desktop interfaces, so the system will be developed as a ASP Core application. The ASP Core framework also has templates and features to ease the work with the MVC pattern, so it will be easier to follow the architectural decisions taken in the previous section.

Nevertheless, as I said, I have some concerns about deployment and long-term maintenance of a web application, so the separation of the presentation layer will allow me to change this in the future (out of the scope of this project) with minimal complications if necessary.

3.4 Retrospective

Halfway through the implementation, I was able to have a demo session with the stakeholders. In this I was able to realize there had been a misunderstanding in the way the managers scheduled their staff. While they use several "schedule models" to assign an individual schedule to each staff member, they were not interested in any

automatic generation of those models according to any set of rules. For this reason, I abandoned the features related to automatic generation. Instead, I focused on the manual edition of the calendars. However, I was also unable to meet the expected results in the planned deadline.

For these reasons, I have readjusted the objectives of the following iterations as well as their deadlines to work with the time I still have available. For this reason, I will try to squeeze two one-week iterations at the end of term, since I will have time away from work:

- Iteration 2: I will focus on a proof-of-concept of a grid that can be used to view gaps or over-scheduling in a whole month without using fixed-time shifts. This will advance towards the following user stories:
  - As a manager, I want to be assured that the requisite number of staff are scheduled at all times; and if possible, not more.
  - As a scheduler, I want to be advised of any kind of conflict in the schedule of my employees.
  - As a scheduler, I want to be able to see who is scheduled to work any given day at any given time.
- Iteration 3: I will resume my work on the manual editing of the calendars, as well as generating a "model calendar" which can be copied to any staff member
  - As a scheduler, I want to edit a "model calendar" that I can use as a starting calendar for any staff member
- Iteration 4:
  - As a scheduler, I want to automatically recalculate salaries when I have to make contract changes halfway through a year.
  - As a scheduler, I want to create a new schedule (for a new employee) halfway through a year.
  - As a scheduler, I want to liquidate an account halfway through a year (balance hours paid with hours worked to adjust final payment).

Chapter 4

Iteration 2

4.1 User Stories

This iteration will be centered around creating a proof-of-concept view that can be used find gaps or over-staffing in a whole month without using fixed-time shifts. This is important to show to the stakeholders at this time, since we found out I had several misconceptions of how they worked in my last meeting with them. They are also very used to the idea of working with fixed-time shifts, such as dividing the day in three shifts (morning, afternoon and night), plus two extra-staff shifts (morning and evening support).

I believe, however, that this rigid way of working with shifts is not adequate for their scheduling system, as it does not follow reality, where they have several people with slightly different shifts. However, when we talked about this, several stakeholders expressed their concern about not being able to see all the information they are used to seeing in a simple monthly grid.

Since my work with the whole system has gone much slower than expected, I decided to work on a simple demo application quickly show how a grid of this type could work with flexible shifts. This will not directly implement any features in the final system I am developing, however, it will help me advance towards the following user stories:

- As a manager, I want to be assured that the requisite number of staff are scheduled at all times; and if possible, not more.
- As a scheduler, I want to be advised of any kind of conflict in the schedule of my employees.
- As a scheduler, I want to be able to see who is scheduled to work any given day at any given
- As a scheduler, I want to see which employees are free to cover a given shift.

4.1.1 Example Mapping

During the starting meeting of this iteration, I tried to run an example mapping session [18]. Since in my previous interactions with the stakeholders they seemed to respond better to examples than to abstract requirements, I thought it would work help define the requirements of such a grid

Feature: Monthly grid.

Rules and Examples:

- Given a month and a time range, it will show in a grid all staff who work any shift in the given month at any time between the given times. Each column represents a day of the month, and each row a staff member
  - A staff member works the weekdays of March 2019, from 9AM to 2PM. The grid of March 2019, times 8AM to 2PM shows this staff member.
  - A staff member works the weekdays of March 2019, from 2PM to 10PM. The grid of March 2019, times 8AM to 2PM does not show this staff member.
  - A staff member works the 1st of March 2019 from 9AM to 2PM. The grid of March 2019, times 1PM to 10PM shows this staff member
- Each cell will mark if the corresponding staff member is assigned to work at any time during the selected time period on the corresponding day.
  - A staff member works the weekdays of March 2019, from 9AM to 2PM. On the grid of March 2019, times 8AM to 2PM, the cell corresponding to the 11th and this staff member is marked.
  - A staff member works the weekdays of March 2019, from 2PM to 10PM. On the grid of March 2019, times 8AM to 2PM, the cell corresponding to the 11th and this staff member is not marked.
  - A staff member works the 1st of March 2019 from 9AM to 2PM. On the grid of March 2019, times 1PM to 10PM, the cell corresponding to the 11th and this staff member is marked.
- There will be one extra row after all the staff member that will indicate the number of staff in the range of times selected for the grid.
  - Two staff members work the weekdays of March 2019, from 9AM to 2PM. On the grid of March 2019, times 8AM to 2PM, the last row shows "2" for all weekdays.
  - A staff member works the weekdays of March 2019, from 2PM to 10PM. Another staff member works the weekdays of March 2019, from 9AM to

2PM. On the grid of March 2019, times 8AM to 2PM, the last row shows "1" for all weekdays.

- If the total number of working staff varies depending on the time, the total staff row will show the minimum and maximum staff at any time.
  - A staff member works the weekdays of March 2019, from 8PM to 1PM. Another staff member works the weekdays of March 2019, from 9AM to 2PM. On the grid of March 2019, times 8AM to 2PM, the last row shows "1-2" for all weekdays.
  - A staff member works the weekdays of March 2019, from 8AM to 2PM. Another staff member works the weekdays of March 2019, from 2PM to 10PM. On the grid of March 2019, times 8AM to 10PM, the last row shows "1" for all weekdays.
- Clicking on a column header will navigate to the detail grid of the clicked day for the same times shown in the month grid. The rows will still be the staff members, but the columns will represent each hour in the selected time range.

Questions:

- How can you tell who is free or on holiday at any given day?
- If the total staff on any day varies between understaffed and overstaffed, should this day be marked (colored) as overstaffed?

4.2 Design

The solution will be based on the idea that each staff member can work any number of shifts on each day, and that each of those shifts can start and end at any hour. For visualization purposes, I will suppose that each start/end of shift can be at half-hour intervals, that is, a shift can start at 8:00 or 8:30, but not 8:15 or 8:20.

4.3 Implementation Decisions

I will implement this proof of concept with a quick static website, using javascript (with jQuery) for the interaction. I will hard-code some sample staff for the demo.

4.4 Retrospective

The stakeholders were happy with the implemented solution and understood how flexible shifts could help them better see when they had small ranges of time over or under-staffed

However, this approach had several problems compared to their previous grids using fixed shifts:

- If the grid shows only employees that work any given shift/month, it is difficult to find an employee that might be free to cover any leaves or personal days
- If the grid shows all available employees, the grid would be unmanageable
- Each day is marked equally for a person who works the full selected time range or only one hour

For these reasons, we decided a mixed approach would be more appropriate. We discussed a new feature where each staff member could be assigned one or several "tags". These tags could correspond, for example, to a fuzzy shift the person is usually able to cover. Then, when selecting the date range to view in the monthly grid, the schedulers could choose between labels instead of times. The grid then, would show all staff that has the selected label, not depending on whether they worked at the given times or not. Of course, custom time selecting should be still available to ease the discovery of over and under-staffed gaps.

Chapter 5

Iteration 3

5.1 User Stories

This iteration will return the focus to the edition of schedules. This time, I will implement the user stories of manually editing the schedule of a staff member. To enable this, I will also implement two features that are not specifically the focus of this iteration but that are necessary to manage staff: a list of all staff members and the option to create new staff members.

5.2 Design

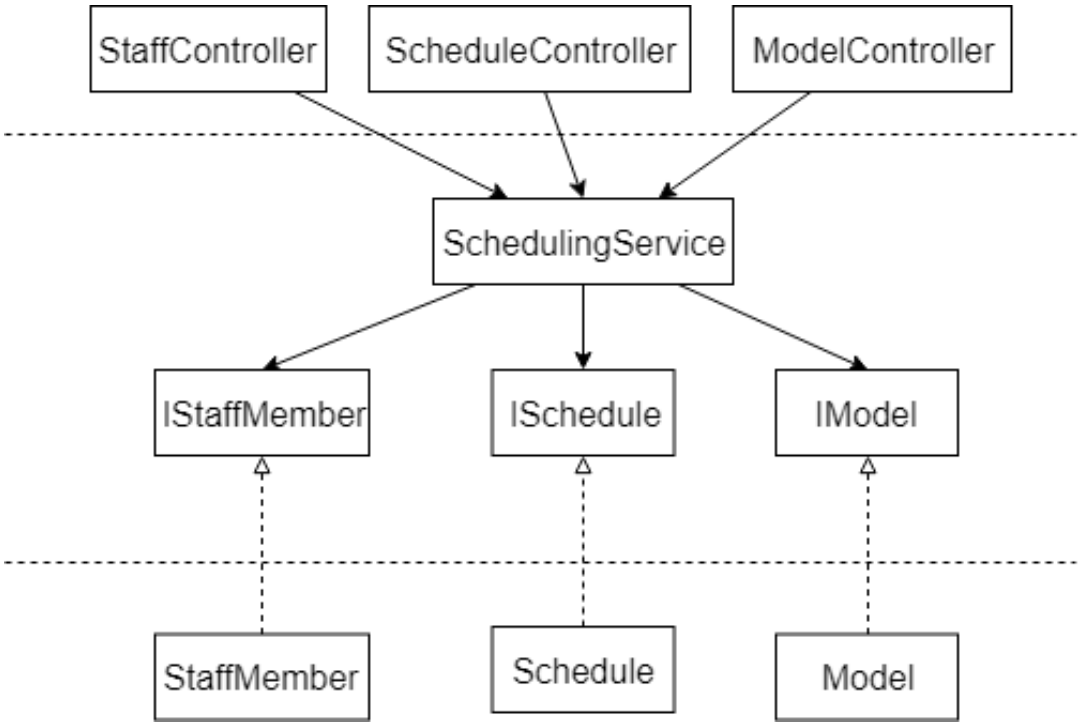


FIGURE 5.1: Schedule Editing Diagram.

In figure 5.1, we can see a global representation of the most important classes implemented in this iteration. In the presentation layer, there are three controllers. StaffController and ModelController are responsible for managing staff and schedule-models respectively, while the ScheduleController holds the calls necessary for managing the schedules themselves, independently if they belong to a model or to a real staff member. This way, I can implement the edition of staff schedules and model schedules without having to repeat duplicate calls.

The SchedulingService class serves as a façade for the domain logic. The domain layer also has all the interfaces for the necessary functionality of the persistence layer, which SchedulingService uses. The implementations of these interfaces, however, are in the persistence layer, so that if a different database was selected - even a non-relational database - I would only have to create new implementations of these interfaces in a new project.

5.2.1 Domain

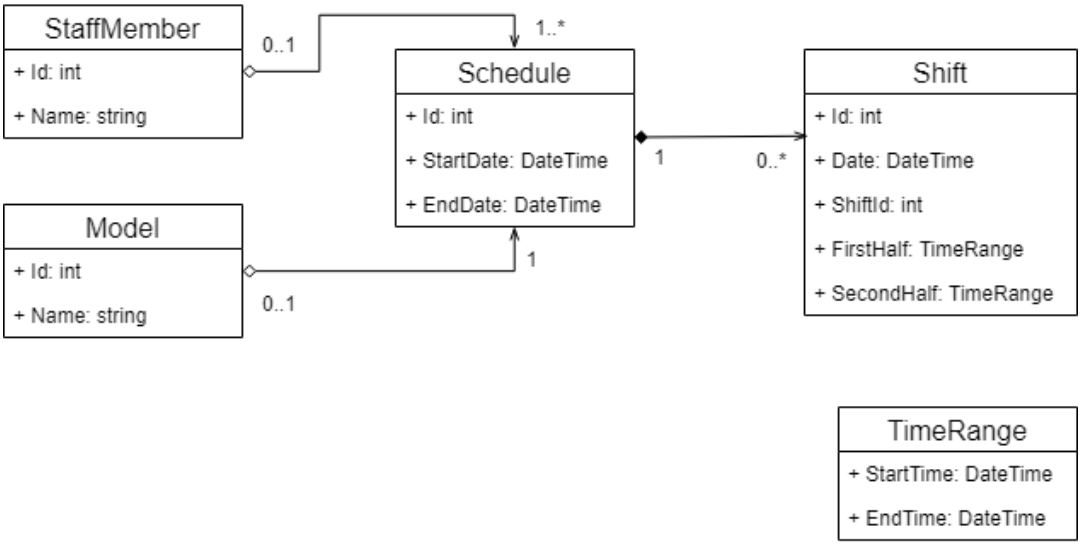


FIGURE 5.2: Domain Entities Diagram.

In the domain layer, as shown in figure 5.2, I have simplified the entities from the first iteration. I have also decided to work with thin, POCO-like entities, as using entities with methods was resulting in dependencies between entities that I was not sure how to resolve.

The center of the entities is the Schedule. Each Schedule can belong then to either one StaffMember **or** one Model. Besides having an start and end date, each Schedule is composed of several shifts. All these Shifts must have a date between the start and end dates of the Schedule, and there can only be one Shift for each date and schedule.

The Shifts themselves are composed by one or, optionally, two TimeRanges. In the case of an intensive shift, for example, from 8AM to 3PM, the SecondHalf of the Shift will be null. In the case of an split shift, such as from 9AM to 1PM and from 2PM to 5PM, FirstHalf will contain the first range, and SecondHalf the second.

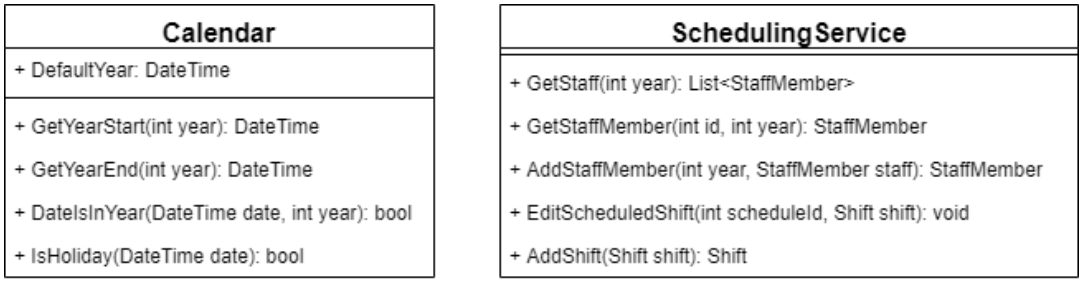


FIGURE 5.3: Domain Services Diagram.

Besides the entities and the interfaces for the persistence layer, the domain layer also contains the two services shown in figure 5.3. SchedulingService is the façade of the domain layer, and contains all the operations needed to edit schedules. However, the functionality related to the calendar themselves - which days are holidays, the default start of a schedule for a given year, etc. are encapsulated in the Calendar class.

5.2.2 Persistence

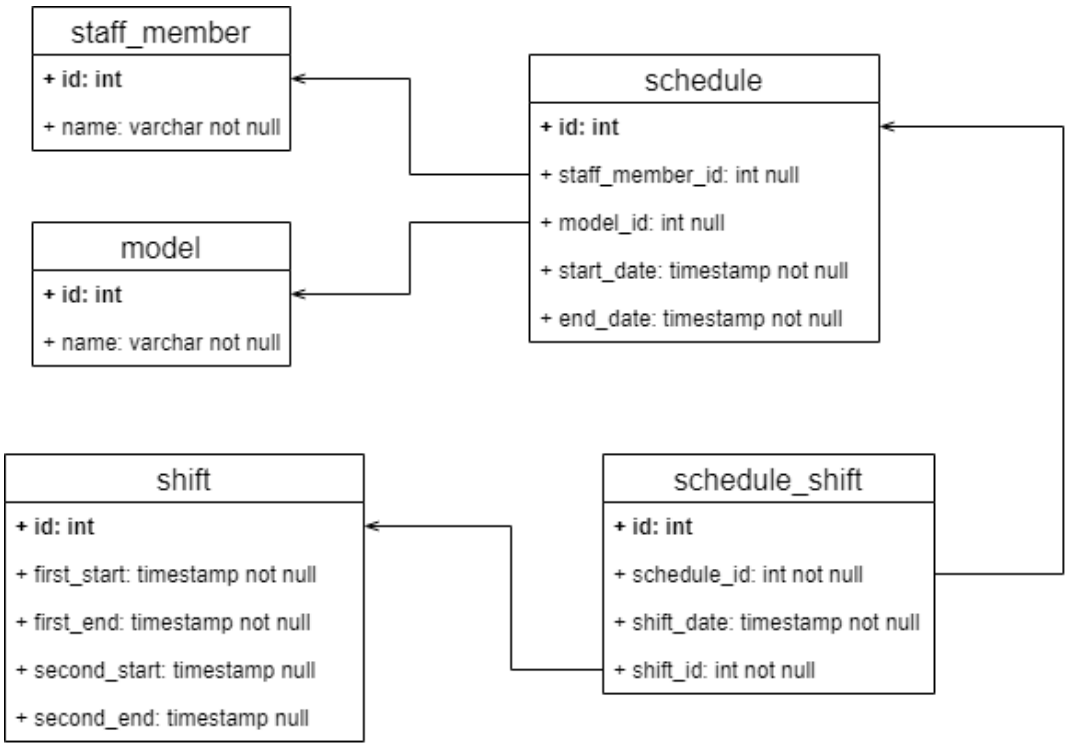


FIGURE 5.4: Database Diagram.

In figure 5.4 I have drawn the tables this iteration needs in the database. In the persistence layer, I have created DTOs that map one-to-one these tables. Note the difference between the shifts in the domain and persistence layer. Given that there are only a limited number of shifts possible, to make comparisons easier, I decided to persist shifts independently of their schedules.

Each table has its own data access object as shown in figure 5.5. The interface for each one is slightly different depending on how the data needs to be accessed and written. These are the classes the implementations of the infrastructure interfaces use to persist the data.

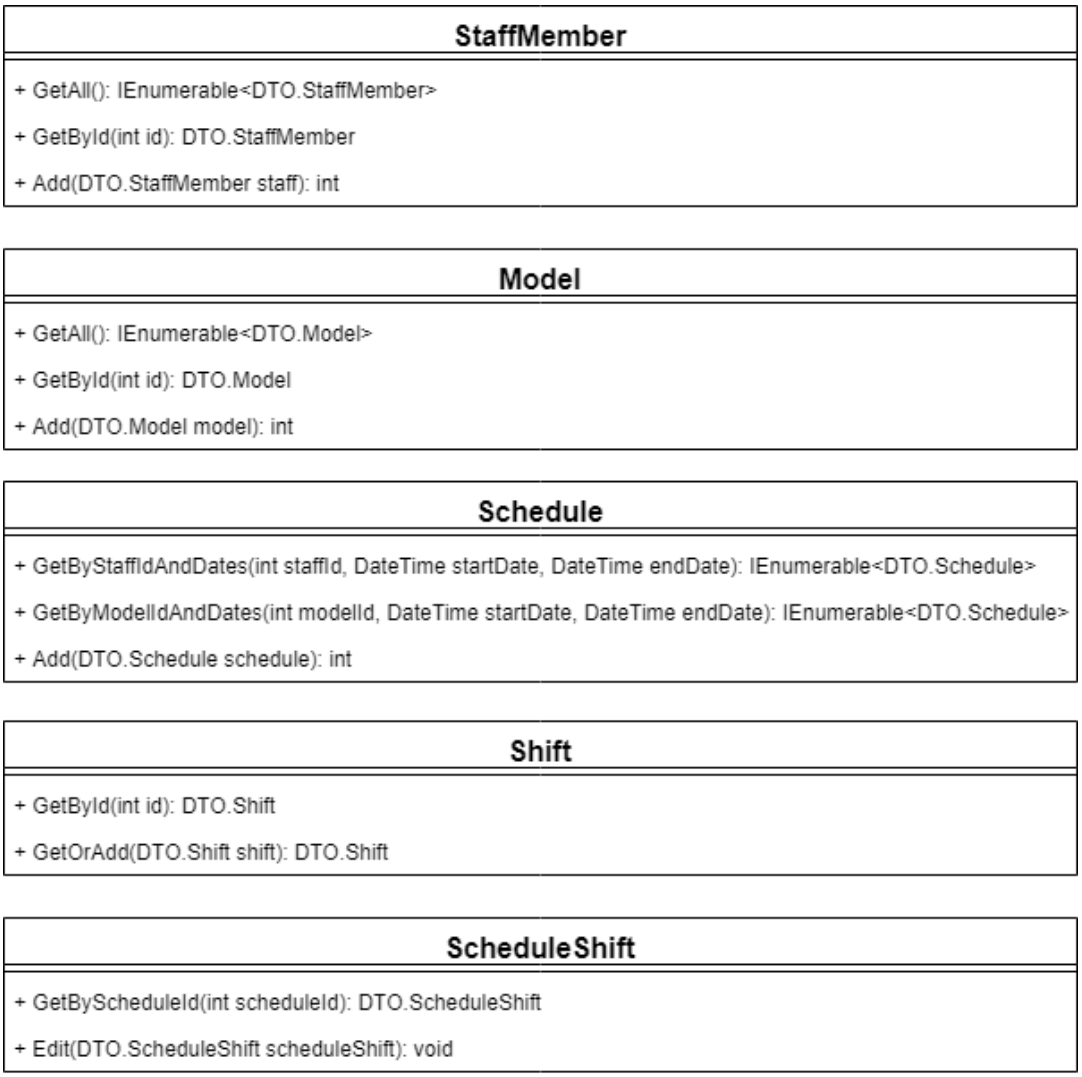


FIGURE 5.5: Data Access Objects Diagram.

5.3 Implementation Decisions

For the implementation of the data access objects, I have decided to use the library Dapper, as it is a well-supported, lightweight library that eases the communication

with the database without the overhead of Entity Framework.

For the implementation of the database, I have chosen postreSQL, as it is an open-source database system which is easy to sue and also supports json columns, which is often useful to store extra data for each object that does not need to be accessed in a relational manner.

5.4 Retrospective

In general, this iteration met the stakeholders' expectations. However, there are some points that need further work:

- The user interface is still somewhat awkward. At the very least, you should be able to "paint" several days with the same shift-color by dragging the mouse.
- The system should warn you when editing a shift breaks one of the calendar restrictions.

Chapter 6

Iteration 4

6.1 Introduction

I had not planned to do any kind of deployment, so I had not considered DevOps. However, after the last check-in, it was clear that some deployment would be necessary, so I changed the focus of the last iteration to be DevOps.

6.2 Containers

One of the advantages of using NET Core over traditional NET is that the applications can be packaged into containers. For this project, I used docker to containerize the application. To build the image, you need to execute the following command from the root of the application:

```
docker build -f ScheduleSystem/Dockerfile -t schedulesystem .
```

To run it, execute the following command:

```
docker run -d -p 8080:80 --name myapp schedulesystem
```

6.3 Deployment

I decided to use Amazon Web Services to deploy a development version of my application, since I am familiar with them and they have a free tier I could use.

First, I created a postgresql database in RDS as shown in figure 6.1.

I edited the connection string in the application settings of my application and launched the application, checking that everything worked correctly.

Afterwards, I created an image repository in ECR and pushed the image to it with the following script:

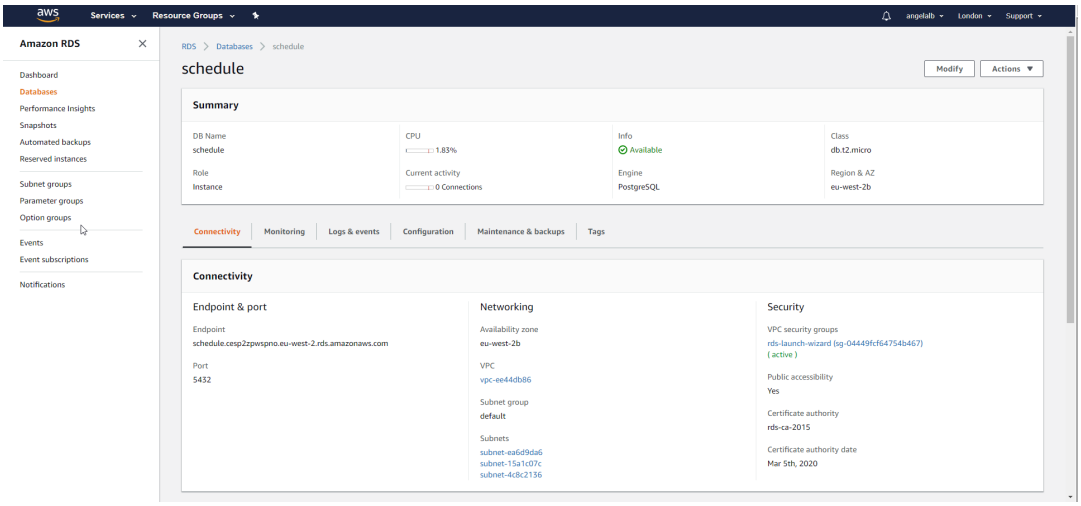


FIGURE 6.1: AWS RDS.

```
$(aws ecr get-login --no-include-email --region eu-west-2)
docker tag schedulesystem:latest ACCOUNT_ID.dkr.ecr.eu-west
    -2.amazonaws.com/schedulesystem:latest
docker push ACCOUNT_ID.dkr.ecr.eu-west-2.amazonaws.com/
    schedulesystem:latest
```

Finally, I configured a new container of this image using ECS, as shown in 6.2.

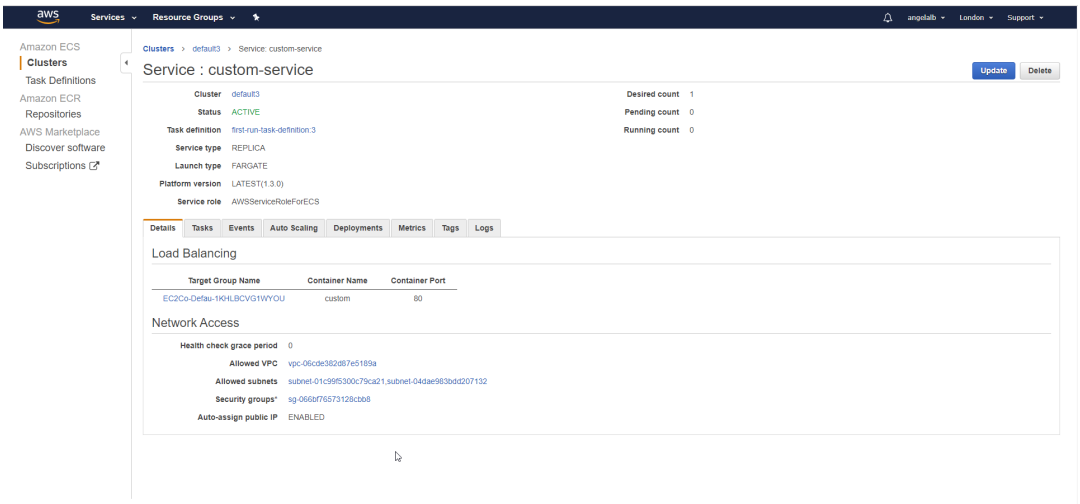


FIGURE 6.2: AWS ECS.

However, I was unable to connect to the database from this container.

6.4 Retrospective

Looking back, this last iteration was very rushed and I could have accomplished more if I had thought about deployment earlier.

Chapter 7

Conclusions

7.1 Conclusions

During this project I was able to reach two main conclusions:

First, the iterative approach used allowed for continuous communication with the stakeholders. Especially in the first iteration, stopping to check-in allowed to discover a miscommunication that would have meant dozens of lost hours otherwise.

Second, during the design phases of each iteration, I found it difficult to focus on the current iteration objectives. I often experienced "feature-creep", meaning I had double work, as I had to rethink the features not in the iteration anyway.

7.2 Further Work

This project needs further work to even get to a MVP version of the application. First and foremost, I need to continue working on the features to make a viable product, especially further work on the manual edition of schedules, the monthly grid, and the complete implementation of the warnings system.

Another task that is overdue is the implementation of unit tests, especially for the domain layer where most logic should be located. The models in the presentation layer also have considerable logic, so unit tests would be beneficial. In the persistence layer, the Schedule and StaffMember classes that are responsible for the mapping from domain objects to database DTOs and vice-versa would also benefit from having unit tests.

For the rest of classes and functionalities, integration and automatized end-to-end tests would be preferable. For example, integration tests for domain and persistence layers would mean an important increase in the confidence on each change, and they would still be useful even if the presentation layer changed. On the other hand, the presentation layer has a lot of interaction with the user, so end-to-end tests would cover this interaction.

As shown during the last iteration, I am not ready even for manual deployment. A pipeline for at least semi-automatic deployment and testing would ease the work further down when I start to have release candidates for the stakeholders to review.

Finally, as the presentation layer has a lot of interaction with the user, I should also consider using a front-end framework such as React that would help follow best practices in the front-end.

Bibliography

[1] Atlas Business Solutions. *Employee Scheduling Software*. 2018. URL: https://web.archive.org/web/20170630030817/https://www.scheduleanywhere.com/employee-scheduling-software/employee-scheduling-videos.aspx (visited on 09/28/2018).

[2] Atlassian. *Jira Product Page*. 2018. URL: https://web.archive.org/web/20180930221345/https://www.atlassian.com/software/jira (visited on 09/26/2018).

[3] Daljit Banger. *A Basic Non-Functional Requirements Checklist*. 2014. URL: https://dalbanger.wordpress.com/2014/01/08/a-basic-non-functional-requirements-checklist/ (visited on 10/12/2018).

[4] Business Management Systems. *Snap Schedule Pricing*. 2018. URL: https://web.archive.org/web/20180323192103/http://www.snapschedule.com:80/home/buy-now/snap-schedule/ (visited on 09/28/2018).

[5] Business Management Systems. *Snap Schedule Product Description*. 2018. URL: https://web.archive.org/web/20180627053038/http://www.bmscentral.com/products/schedule/overview.aspx (visited on 09/22/2018).

[6] Cary Snowden. *The Employee Scheduling Solution For Healthcare Managers*. Mar. 20, 2018. URL: https://www3.swipeclock.com/blog/employee-scheduling-solution-healthcare-managers (visited on 09/29/2018).

[7] Global Media Software. *EntiGest Product Page*. 2018. URL: https://web.archive.org/web/20180805210103/http://laborofficefree.com/ (visited on 09/29/2018).

[8] Mike Griffiths. *Non-Functional Requirements - Minimal Checklist*. 2009. URL: http://leadinganswers.typepad.com/leading_answers/2009/03/nonfunctional-requirements-minimal-checklist.html (visited on 10/12/2018).

[9] Identifica-t. *Identifica-t Product Page*. 2018. URL: https://web.archive.org/web/20180828045404/https://www.identifica-t.com/control-presencia-empleados/software-gestion-horarios-jornada-laboral-trabajadores (visited on 09/28/2018).

[10] Monday. *Monday Product Page*. 2018. URL: https://web.archive.org/web/20180930184002/https://monday.com/ (visited on 09/26/2018).

[11] Out Crowd. *FindMyShift Product Page*. 2018. URL: https://web.archive.org/web/20180925050726/https://www.findmyshift.com/ (visited on 09/29/2018).

[12] PDC. *Workforce Management and Staff Scheduling in Hospitals and Elder Care*. 2018. URL: https://web.archive.org/web/20180903013518/https://www.pdc.com/staffplan/health.html (visited on 09/29/2018).

[13] Prgtec Corporation. *LaborOfficeFree Product Page*. 2018. URL: https://web.archive.org/web/20180805210103/http://laborofficefree.com/ (visited on 09/29/2018).

[14] Michelle V. Rafter and Juan Martinez. "The Best Employee Scheduling and Shift Planning Software of 2018". In: *PC Mag* (Aug. 2018). URL: https://www.pcmag.com/roundup/345904/the-best-employee-scheduling-shift-planning-software.

[15] Redmine. *Redmine Product Page*. 2018. URL: https://web.archive.org/web/20180923100256/http://www.redmine.org/ (visited on 09/26/2018).

[16] Tamigo. *Tamigo Product Page*. 2018. URL: https://web.archive.org/web/20180906061537/https://www.tamigo.es/ (visited on 09/29/2018).

[17] Team Gannt. *Team Gannt Product Description*. 2018. URL: https://www.teamgantt.com/ (visited on 09/29/2018).

[18] Matt Wynne. *Introducing Example Mapping*. 2015. URL: https://cucumber.io/blog/2015/12/08/example-mapping-introduction (visited on 10/17/2018).