Universidade de Évora - Escola de Ciências e Tecnologia

Mestrado em Engenharia Informática

Dissertação

Software development of SmartShiP: A smart shift planning tool

Ricardo Jorge Chambel Benedito

Orientador(es) | Teresa Gonçalves

Luís Rato

Évora 2021

Universidade de Évora - Escola de Ciências e Tecnologia

Mestrado em Engenharia Informática

Dissertação

Software development of SmartShiP: A smart shift planning tool

Ricardo Jorge Chambel Benedito

Orientador(es) | Teresa Gonçalves

Luís Rato

Évora 2021

A dissertação foi objeto de apreciação e discussão pública pelo seguinte júri nomeado pelo Diretor da Escola de Ciências e Tecnologia:

Presidente	|	Irene Pimenta Rodrigues (Universidade de Évora)
Vogais	|	Luís Rato (Universidade de Évora) (Orientador) Pedro Salgueiro (Universidade de Évora) (Arguente)

Évora 2021

Aos eternos insatisfeitos.

Acknowledgements

I would like to thank my family and friends for all the support provided during these years.

In loving memory of those who cannot be present.

Contents

List of Figures

List of Tables

Acronyms

DB Database

CI/CD Continuous Integration/Continuous Deployment

SmartShiP Smart Shift Planning Tool

USA United States of America

US United States

API Application Programming Interface

IDE Integrated development environment

TL Team Leader

MVP Minimum Viable Product

IT Information Technology

MS Microsoft

UK United Kingdom

CSP Constraint Satisfaction Problem

ELK Elasticsearch, Logstash and Kibana

CSW Critical Software

S1ES SMETS1 Enduring Support

SMETS1 Smart Metering Equipment Technical Specifications version 1

DAO Data Access Object

SQL Structured Query language

DEV Development

AC Acceptance Criteria

OOP Object-oriented programming

Abstract

Software development is a process used by engineers and software developers to create and produce software, as a part of the digital transformation of several industries in 2020.

Agile methodologies in Software development rose in popularity when the industry shifted from Product offering to Service offering, further stimulating the growth of Software companies.

This dissertation presents an example on how to use both Software development and Agile methodologies to develop a Service or Product that explores the thematic of work schedule planning. The tool developed, named SmartShiP aims at automating and improving work schedule planning processes.

Due to the incremental nature of SmartShiP, some of its core functionalities such as availability planning, intervention reporting and schedule generation are already implemented and their usage proves to be valuable for its users. The expansion to other teams is planned and further functionalities like multiple projects, different schedule strategies and code quality improvements are in the way.

Keywords: Software Architecture, planning, software development, agile methodologies

Sumário

Desenvolvimento de Software do SmartShiP

O desenvolvimento de software é um processo utilizado por engenheiros e desenvolvedores de software para criar e produzir software. Sendo uma parte da transformação digital de muitas indústrias em 2020.

A introdução de metodologias ágeis no desenvolvimento de software, acompanhou mudança da indústria de oferta de produtos para a oferta de serviços, estimulando ainda mais o crescimento das empresas de software.

Esta dissertação é um exemplo de como usar desenvolvimento de Software e metodologias ágeis para desenvolver um serviço ou produto que explora a temática do planeamento de horários de trabalho, automatizando e melhorando os processos com recurso a uma ferramenta de Software desenvolvida - o SmartShiP.

Devido a natureza incremental do SmartShiP, algumas das suas funcionalidades principais como planeamento de disponibilidade, justificação de intervenções e geração do calendário já estão actualmente implementadas e a sua utilização tem-se provado valiosa para os seus utilizadores. A expansão para outras equipas está planeada assim com mais funcionalidades como multiplos projectos, diferentes estratégias de planeamento e melhorias na qualidade do código, estão a caminho

Palavras chave: Arquitectura de software, planeamento, desenvolvimento de software, metodologias ágeis

Chapter 1

Introduction

It is part of the path of the Master studies in Informatics Engineering to develop a Master thesis that consolidates the learning from the first, second and third semesters' courses and all the related academic work.

This document presents and discusses the development of the project "Software Development of SmartShiP". Smart Shift Planning Tool (SmartShiP) is a system with a smart way of generating schedule shifts for teams. It consists of using the team availability to make sure there is a balance between workload and service availability. There was an opportunity to improve and automate the process of the schedule rotation with the use of software tools

This dissertation presents the thinking process and how the idea of SmartShiP became a reality: from a theoretical concept into a web application!

1.1 Motivation

SMETS1 Enduring Support (S1ES) provides support for the activities of Smart energy meters transition in the UK, with the objective of centralizing communications of energy meters. The transition to smart meters allows both energy supplier companies and their customers to have an accurate and instant energy consumption reading without the need for manual intervention, this also facilitates the access of more energy suppliers to more clients.

Enduring Support projects such as S1ES require the use of work shifts to maintain a 24/7 support line available to the customer. S1ES is part of a critical operation for Smart Metering Equipment Technical Specifications version 1 (SMETS1) in the United Kingdom (UK). To plan the work and

on-call shifts, a shift management tool is often used as the best approach for this problem of managing human resources schedule rotations.

While participating in a company project with a requirement of work planning and on-call shifts, it became clear that a simplistic approach to the problem would be a burden on management and eventually lead to poor balance of workload and end up with lower performance on the service.

A spreadsheet and strict scheduling for the team members were the only things that existed to plan on-call shifts. The idea was then to include all the requirements of the service and constraints into a model that would satisfy all the needs and eventually aid the management and finance departments to have a better visibility on the state of the service.

The solution encountered was to develop a tool that could better integrate the workflow of S1ES.

1.2 Objectives

After exploring several similar systems already used in the industry (Section 2.2 and Section 2.1), it became clear that for a finer tuning and more balanced attribution of shifts to the team, it would be necessary to adapt or create a tool that would manage these requirements properly.

Additionally, this tool would be a good opportunity to introduced the team members to software development and agile methodologies, since this subject is part of the company's training plan. This is specifically important as most of the members of the team don't have any software development background. It would be a complement to the training provided and would be important as the current S1ES project of the team is not related with software development.

For myself, this project would enable learning about software architecture, agile and software development and apply some of the concepts learned from the Master studies in Informatics Engineering.

The main objectives of this dissertation are:

- Introduce the agile methodologies for SmartShiP development
- Prove that SmartShiP is a viable solution for the proposed problem
- Explain the Engineering Design process used for SmartShiP
- Exemplify the development workflow for Software Development using Scrumban

As for SmartShiP its main objectives are:

- Develop a system that stores the availability and foreseen absences of the team members and records interventions during service;
- A web interface with a login;
- A calendar view to provide each user with its schedule;
- And finally an extraction of worked and on-call hours.

Each of these objectives were part of the core functionalities of our system.

1.3 Contributions

The usage of Agile and Software development practices on SmartShiP development contributed to the training of the team members in these areas.

The market was analysed and some software tools for schedule management were analysed and their functionalities explained.

SmartShiP made schedule management easier for the team, the usage of this tool contributed to a better workflow for Finance and Project management.

This dissertation linked software development, software architecture and agile methodologies together and is a real world example on the creation of a software product. All the techniques used from proof of concept until final product are explained.

1.4 Document Structure

The dissertation has the following structure:

On the second chapter the Schedule managements tools in the market are presented, referring current possible solutions to similar problems.

On the third chapter all the required theoretical fundamentals are presented briefly in order to introduce some basic topics to the reader such as technological requirements and agile methodologies.

Chapter four contains all the Engineering Design process since the beginning of SmartShiP, going through the concept and prototype phase and ending with the agreed functionalities for the first release of the system and future

developments. The System architecture and development framework are also presented in this chapter.

Chapter five provides all the actual development on the tasks breaking down each type of task to provide examples of the workflow. This chapter ends with an overview of the currently developed system.

Finally, the Conclusions present the reader with the achievements of the team and myself. All the progress made is acknowledged and summarized.

Chapter 2

Schedule managements tools in the market

There are several software tools focused on labor management, including shift management that are available in the market and are fully featured. Acknowledging the presence of Open Source, the tools tested during this phase, as guidance from the company, have proprietary licensing.

From the available tools, it was decided to focus on vendors that the company currently works with such as Microsoft and Atlassian.

2.1 Microsoft Teams Shifts

Shifts app is included in Microsoft Teams [Microsoft (2020)] and is free to use as part of the Office bundle. Its functionality is basic and simple and allows users to create schedules and manage them on a team level.

This application also allows users to be assigned to groups to more easily manage teams when working on a multi-team environment. On a shift you can add activities like training or specific tasks.

The export into a spreadsheet is a feature that can be imported in the future as well, it can be useful when determining the payroll of the team.

A user can make three types of requests: Time off, Swap and Offer.

- Time off is used to book a slot of time as holidays so you show your availability to the team;
- Swap can be used to switch shifts with a colleague;

- Offer is used to give another person your shift.

The interface contains several lanes that show the shifts for all members of the team. It can be grouped and filtered to only show the desired teams or shifts.

2.2 Atlassian Opsgenie - On-call schedule management

Opsgenie [Atlassian (2020e)] is a bundle software that includes among many other services an on-call management system that aids on building and maintaining on-call schedules, escalation and acknowledgement of incidents into a single system.

Unlike Shifts, Opsgenie is a much more powerful tool as it can automate the rotation of the schedules using custom rotations and allowing for different rules for different teams and also boasting a way of adapting as you go for sudden changes on availability of team members.

Opsgenie also integrates with JIRA allowing for a better visibility of newly created incidents and alerting via SMS/Email/Phone a person or team during the on-call period and working period as well.

Opsgenie is also more visually pleasing when an extract of the team effort is retrieved and contains some dashboards that will certainly aid not only management but also the finance teams.

To fully use Opsgenie, your service must take into account that Opsgenie is a full tool and may have more features than needed, meaning that the price of Opsgenie is something to take into account. Its features are available for every country but outside the United States (US), calls and notifications are slightly more expensive and may not be fit for projects that require communication to be done within a single country as data flows to Atlassian servers are hosted in the United States of America (USA).

2.3 Final overview

Comparing both Microsoft Teams Shifts and Atlassian Opsgenie - On-call schedule management, it is visible that Shifts is much simpler and its possibilities are limited, there are not many parts that are automated and it does work better with small teams with a limited range of schedules.

On the other hand, Atlassian Opsgenie - On-call schedule management is part of a fully fledged HR management solution and thus includes more integration and ways to personalize the experience of generating schedules with more rules and requirements. This means Opsgenie is better suited for bigger services that require a more structured approach to Support operations.

From both this applications, SmartShiP took ideas like the calendar views, the automated generation of the schedule and the availability system.

Chapter 3

Methodologies for Project development

In this chapter, the methodologies used in the development of SmartShiP, are presented and explained in a introductory manner. The adoption of Agile was enforced by the company therefore no other Software development methodologies were referenced here.

3.1 Agile Methodologies

Agile is an approach to software development and projected management that utilizes iterative deliveries to focus on incremental changes instead of a single delivery such as the one used in Waterfall. [Atlassian (2020h)]

With Agile, requirements, changes and defects are continuously evaluated and the outcomes are fitted in the product releases. This was part of "The Manifesto for Agile Software Development" [Beck et al. (2001)] that extended on the Iterative and incremental software development techniques.

3.1.1 Scrum

Scrum has its place in the core development of projects, it is widely adopted as the standard Agile way of breaking down work and distributing effort in a team. Its effectiveness depends on the accurate estimation of work per task. [Alliance (2020)]

Scrum has three roles: product owner, scrum master and the development team members. [Atlassian (2020f)]

The Product owner is responsible for taking all the inputs from the customer and share the vision of the customer to the team. His main responsibilities are:

- Managing the scrum backlog: adding and keeping current tasks up to date.
- Release management: Defines which sprints result in a release.
- Stakeholder management: Make sure the development team is delivering value to the stakeholders.

The Scrum master main role is to ensure the scrum events happen and that they are successful. They help the product owner manage the backlog and break down work.

The developer is any team member responsible to finish a task, they are required to deliver work through the sprint and ensure, during the daily meeting, that any difficulties or successes achieved during task development are shared to the team.

The five Scrum values (courage, focus, commitment, respect and openness) enable the transparency, inspect and adapt principles. [Latre (2019)]

As with most agile methods one must embrace the democratic nature of the estimation process, meaning that younger and less experienced teams will diverge widely during the estimation process whilst more mature and senior teams will often agree on the same estimation.

Scrum uses a calculated amount of working hours that can to estimate the delivery date of a software project version, setting Scrum as the most used framework during modern software development.

Scrum is a cyclical process, where each cycle is called a Sprint, which means that at the end of each cycle, the backlog is recalculated, tasks are estimated and the cycle begins until the cycle end is reached when all tasks should be finalised.

Scrum includes ceremonies (agile meetings) like Sprint Planning, Sprint Backlog refinement, Daily stand-up, Sprint Review and Sprint Retrospective all of which are the default minimum to practice Scrum, some teams may prefer to have one meeting for two topics if they so desire.

3.1.2 Kanban

The Kanban methodology of work dates back more than 50 years, after Toyota adapting their engineering processes to the model that supermarkets were using to stock their shelves. [Atlassian (2020i)]

While Scrum is often associated with the beginning and delivery of a software project, Kanban is related to the uncertainty of the workload, and often relies on the capabilities of the team to prioritize tasks instead of just estimating the effort.

One may look at Kanban and have the first impression that it is very flexible and therefore easier to practice than Scrum, but the creative nature of splitting tasks into smaller sub-tasks and having an agreed operation policy within the team, will soon make you realize that it isn't, in fact, easy to manage the expectation of the client or the higher management.

Kanban's origins relate back to stock management and service/product delivery and as stock fluctuates, the priority of certain tasks varies at all times.

In order to keep tasks from stalling in the pipeline Kanban was adapted to the modern day and now includes processes like the expedite lane which is used to make a blocker task move faster through the pipeline and soft limits on the maximum number of tasks on each state pipeline.

The success of Kanban derives from the capability of the team to deliver and prioritize with the client each task and maintaining a healthy backlog of tasks for the team.

Kanban is non-cyclical, there are no sprints, meaning that it does not get a new backlog every cycle, the backlog is in an alive state where it is fed on a daily/weekly basis.

3.1.3 Scrumban

With the need of adding flexibility to Scrum [Germanov (2019)] and to add cycles to Kanban the idea of Scrumban was adopted. Although it is not a pure form of Agile, it is an adaptation that makes sense in certain situations like in maintenance projects, where there are several streams of work arriving to a team. [Pahuja (2016)]

Scrumban is defined depending on the needs of the software project and it allows for fine tuning of the agile processes, therefore being the most versatile. Scrumban promotes easy management and utilizes some of the popular ceremonies from Scrum.

By including cycles, it is easy to change the effort for each period and adjust the needs for the next cycle while keeping a dynamic backlog of work similar to the Kanban one, ultimately being flexible enough to be widely used in teams with multiple projects.

Scrumban may adopt some of the ceremonies from Scrum like the Daily Stand-up, Retrospective and Backlog Refinement. Although they aren't referring to a sprint they may still be helpful in guiding the team for the next cycle and plan actions for the next days.

3.1.4 Final overview

All agile methodologies are well structured and will fit projects of the most diverse types, Scrum is the most structured and besides being very strict it is the easiest to follow.

Kanban is the simplest but reveals bottlenecks in the team more easily. It is better suited for irregular flow projects like support.

Scrumban being the hybrid version is a good fit for this team, the versatility is what SmartShiP project needs.

3.2 Software requirements

In this section, the tools, nomenclatures and some concepts are presented. All the presented material was enforced by the company as they are the same used in other projects in the company.

3.2.1 Software tools

Integrated development environment (IDE) - A software tool used by developers for software development, unlike text editors, IDEs are capable of compiling and execute code, debug and in some cases include a graphical interface with highlights of the code.

Web Browser - An application used to access websites.

Application Programming Interface (API) Test tool - A software tool used to test API endpoints, it allows the user to send specific HTTP requests, manipulate and monitor the requests.

Structured Query language (SQL) client - Database administration and de-

velopment tool that allows the user to connect to a database server and perform reads, writes and configure of the SQL server. Most of the tools also allow for data exports and imports.

Issue types

An epic [Atlassian (2020a)] is a large body of work that can be broken down into a number of smaller stories, or sometimes called Issues in Jira.

A user story [Atlassian (2020g)] is an informal, general explanation of a software feature written from the perspective of the end user. Its purpose is to articulate how a software feature will provide value to the customer.

A task is a set of software features that doesn't come directly from a requirement from the customer but is essential to the software project. For example it can be used to include a specific library into the software project.

A sub-task is used to split any User story or Task into more manageable work tasks and it is considered a dynamic unit of work on the agile approach.

Continuous Integration concepts

Jenkins [Jenkins (2020)] is an open source automation server which enables developers around the world to reliably build, test, and deploy their software.

Gitflow Workflow [Atlassian (2020c)] is a Git workflow that helps with continuous software development and implementing DevOps practices. The Gitflow Workflow defines a strict branching model designed around the software project release. This provides a robust framework for managing larger projects.

Continuous Integration/Continuous Deployment (CI/CD) Multibranch pipeline - In a Multibranch Pipeline project, Jenkins automatically discovers, manages and executes Pipelines for branches which contain a Jenkinsfile in source control. This eliminates the need for manual Pipeline creation and management.

3.2.2 Software paradigms

Architecture

Software architecture depends greatly on the on the service that will be provided and the underlying infrastructure where it will be deployed. The

options were monolithic or micro-services architecture.

Monolithic architecture pattern [Richardson (2020b)] is an easy to grasp concept and broadly used in software development. It consists on a single code base for the service with all the components in the same server. It results in large code bases and it is often a single technology stack.

Micro-services architecture [Richardson (2020a)] expands on the weaknesses of the monolithic architecture and with scaling in mind it is built with many small services that have a very specific purpose and are loosely coupled to other services meaning that each service is mostly independent from the others, making changes has little to no impact on the overall service.

As the purpose of the SmartShiP project is to introduce the team members to software development, for SmartShiP the use of monolithic architecture was agreed as it is easier to grasp and understand. This way all the logic of the system is under one component.

Programming Paradigms

As SmartShiP is written in Java, the SmartShiP project made use of the Object-oriented programming (OOP) paradigm that is inherent to the language. In the future it is planned to include a library that will allow the usage of Logic Programming to aid on the constraint satisfaction part of the system.

To access data, SmartShiP uses Data Access Object (DAO) which is a pattern that is based on creating abstract interfaces to access a database. The Data Access Object abstracts the underlying data access implementation for the Business Object to enable transparent access to the data source. The Business Object also delegates data load and store operations to the Data Access Object. [Oracle (2020)]

3.3 Engineering Design Process

The engineering design process is a set of steps followed by engineers that start on a problem and end on a solution. This process can be cyclical in some parts and it is very iterative and can be changed to fit specific ways of thinking and designing.

The engineering design process that was used on SmartShiP had the following steps:

- Problem definition: The problem is defined on this step and why it requires new solution;
- Requirements and Constraints: All requirements and constraints are gathered and brainstormed to create a model and define its boundaries;
- Identified Risks during early development: Following the identification of requirements and constraints, the risks must start being gathered, this is done in parallel;
- Concept: The concept is built using the defined requirements and constraints and tested in a abstract way to further improve the model;
- Prototypes: The prototype phase can be used to tryout different ideas and tools to build the final version;
- SmartShiP Project Startup: The last step is to define the structure of SmartShiP project, prepare the development and define the requirements for the first version.

Other projects may opt to cycle all the steps, in SmartShiP's case the cycle is between "concept" and "prototype" steps.

Chapter 4

SmartShiP: Engineering Design Process

In this chapter, the whole process from the idea until the start of development is defined and explained. The engineering design processed used was fit to SmartShiP because it didn't require much conceptualizing, as resolving using the Constraints managed to provide the expected results.

4.1 Requirements and Constraints

Without any tools available in the company at the time and taking the opportunity to improve some of the skills of the team members, it was decided to include SmartShiP as an introduction to software development in the team.

SmartShiP has the following requirements and constraints:

1. As in all support projects the load is client dependent and with that came the need to have the least amount of late day hours per person.
2. Engaging the team as most as possible
3. Each team member should stay on the late schedule the same amount of time as all the others. Everyone should cover weekends with the same weight and as frequently as possible.
4. A maximum of 3 days in a row on-call.
5. The schedule rotation needs to have at least 2 people at all times in order to maintain service stability and redundancy which, the number

of people in service in the after-hour period should be 2, in order to maximize the number of people in the normal hour period.

Software tool used

IntelliJ - A Java IDE (integrated development environment) that is configured with several plugins that allows debugging Maven applications running inside Docker containers, perform database read-outs and version control the code.

Google Chrome Browser - Any browser that has real-time debugging tools could be used, in this case Google Chrome. Mostly used to preview and debug the front-end component.

Postman - It was used to test the API endpoints of the service, without having to spin up the front-end.

DBeaver - A community based open source database software to interact with the databases.

4.2 Identified Risks during early development

Right after the SmartShiP project started, there were some details that were important to consider, risks were noted and proceeded to discuss their mitigation's.

1. High volume of incidents supported by only 2 people on weekends
    (a) Mitigation: Ad-hoc approach to bring more people to on-call
2. Not enough people available during holidays for support
    (a) Mitigation:
        i. Having at most 3 people on concurrent holidays
        ii. Create a scoreboard for covering absences, where the person covering an absence gets 1 point and the one being covered gets -1 point, the balance should be 0.
3. No backup plan for an absent person documented
    (a) Mitigation: Create documentation
4. Short period for handing over issues to next available support engineer on duty

   (a) Mitigation:
      i. Create handover plan
      ii. Contingency plan to handover when an unplanned absence happens
      iii. Handover completion criteria

5. People on-call may not have full days to rest
   (a) Mitigation: When intervention hours accumulate >7.5h, people can take full day off in coordination w/ Team Leader (TL) and aligned with schedule constraints

6. People who take Friday off aren't actually available the whole weekend
   (a) Mitigation: have a centralized calendar with team availability to capture these weekends/days away

4.3 Concept

The conceptualization phase uses an abstract approach to solve the problem and prepare the project for the next phase.

4.3.1 Initial concept

The initial concept was a playground to test out the models, constraints and roughly shape the logic of the system.

Management requirements made SmartShiP schedule have the following considerations:

- 2 schedules
  - Schedule AM 08:00 16:00
  - Schedule PM 14:00 22:00
- 1 on-call period
- Schedule PM has always 2 people.
- Remaining available people will be on Schedule AM, with a minimum of 2 people
- Extra hours done during on-call should be taken out from Schedule AM.

- On call starts at 22:00 until 08:00

The figure 4.1, is an aggregation of the presented requirements.

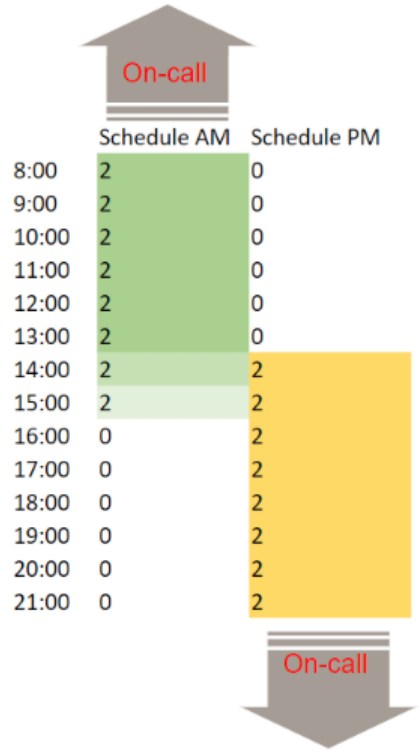

Figure 4.1: Rotation Sketch

After working out the constraints and to set a baseline, it was decided to assume 15 combinations for people=6 and groups=2 in the rotation with a rotation of 3 weeks. The table 4.1 contains an index that relates to the day in the rotation, a number designating each team member id (p1 and p2) and it is colorized to ease visibility of the data:

index	p1	p2
1	1	2
2	1	3
3	1	4
4	1	5
5	1	6
6	2	3
7	2	4
8	2	5
9	2	6
10	3	4
11	3	5
12	3	6
13	4	5
14	4	6
15	5	6

Table 4.1: First results

And after a few randomisation's to avoid having 1 number 3 lines in a row ended up looking like the shifts presented in following table 4.2, with index being the day in the rotation and both p1 and p2 are the team member id:

index	p1	p2
1	3	5
2	5	6
3	4	6
4	1	3
5	2	5
6	1	4
7	3	6
8	1	2
9	1	6
10	4	5
11	2	6
12	3	4
13	2	4
14	1	5
15	2	3

Table 4.2: End results

This was then translated into a horizontal calendar for the full rotation as

presented in table 4.3

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
PM Shift	3	5	4	1	2			1	3	1	1	4			2	3	2	1	2		
PM Shift	5	6	6	3	5			4	6	2	6	5			6	4	4	5	3		
On call	5	4	1	2	1	1	1	3	1	1	4	2	2	2	3	2	1	2	5	5	5
On call	6	6	3	5	4	4	4	6	2	6	5	6	6	6	4	4	5	3	3	3	3

Table 4.3: Horizontal results

And then it is a simple task to repeat this pattern to infinity to get the schedule for the project team of 6 members. For human readability numbers can be swapped for names and thus team members will know when they are on call or on the PM shift.

One important aspect of this first concept is that one missing constraint was addressed: who gets to be on call may be contacted during the night period and thus not getting enough time to rest, and the mitigation was to make who is on call stay on the afternoon shift so they can rest for a few hours.

4.3.2 Prototypes

The prototypes were mostly a test on the tools and software that would be use to develop SmartShiP, no further implementation of the rotation was made in this stage, only the technical setup.

Prototype 1

This Proof of concept is intended to clarify the right technical path for the Mini-Project 1 - SmartShiP.

Product Description:

Prototype 1 is a Java application (JAVA 8) where the technical challenge associated with the integration of Office 365 (Azure) in SmartShiP will be explored.

Features:

- Simple Java Interface (JAVAFX) with a form for the login (Critical Software (CSW) credentials)

- Simple Java interface (JAVAFX) with 2 buttons and a window view to visualize the features of the app.
- Ability to show on the application the scheduled shifts stored on Office365
- Ability to add a new shift to the scheduled shifts stored on Office365

In the following figure 4.2 there are two views on a Java Interface that communicate with an Office 365 cloud service.

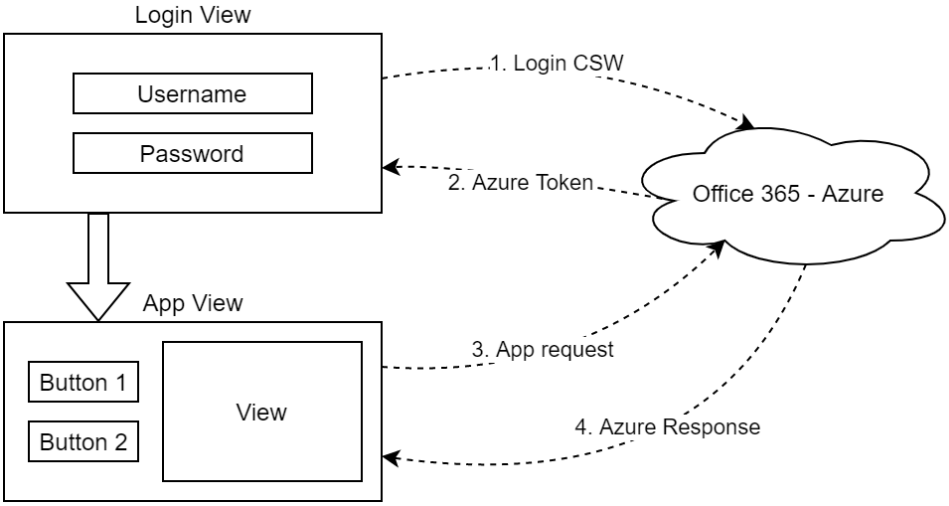

Figure 4.2: Prototype 1 - Diagram

Prototype 2

This Proof of concept is intended to clarify the right technical path for the Mini-Project 1 - SmartShiP.

Product Description:

Prototype 2 is a Java application (JAVA 8) where the technical challenge associated with the integration of a Web interface through a REST API in SmartShiP will be explored.

Features:

- Web interface with 2 buttons and a window view to visualize the features of the app.
- Database with a simple table containing some information of a shift.

- Ability to show on the interface the scheduled shifts stored in the Database (DB)
- Ability to add a new shift to the DB.

In the following figure 4.3 there is a database, a backend that interfaces with the database, and a simple frontend to send http requests to the backend.

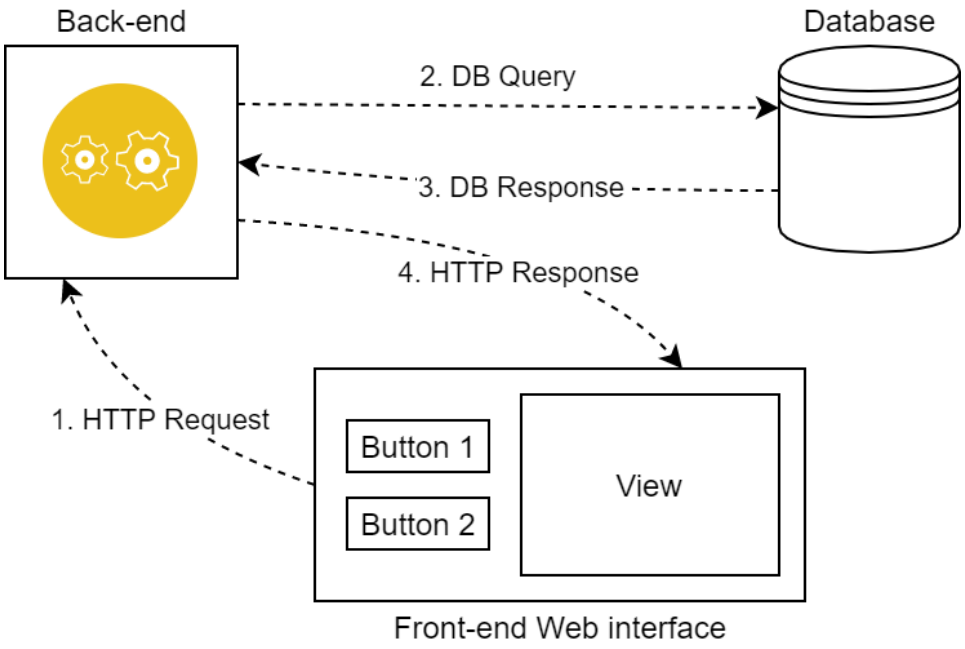

Figure 4.3: Prototype 2 - Diagram

4.3.3 Decision

After finishing both prototypes, the solutions were presented to management and after consideration, the technologies that were deemed as more important were selected as a way of to consolidate the knowledge on software development

To make SmartShiP complex enough and to provide learning opportunities on diverse web technologies, the team opted for the web service approach. Having a website that could be consulted for all the necessary information about the schedule and report the worked and on-call hours.

In figure 4.4 there is a database, a backend that interfaces with the database and utilizes Azure Active directory for the login system, a frontend to visualize data and send request to the backend.

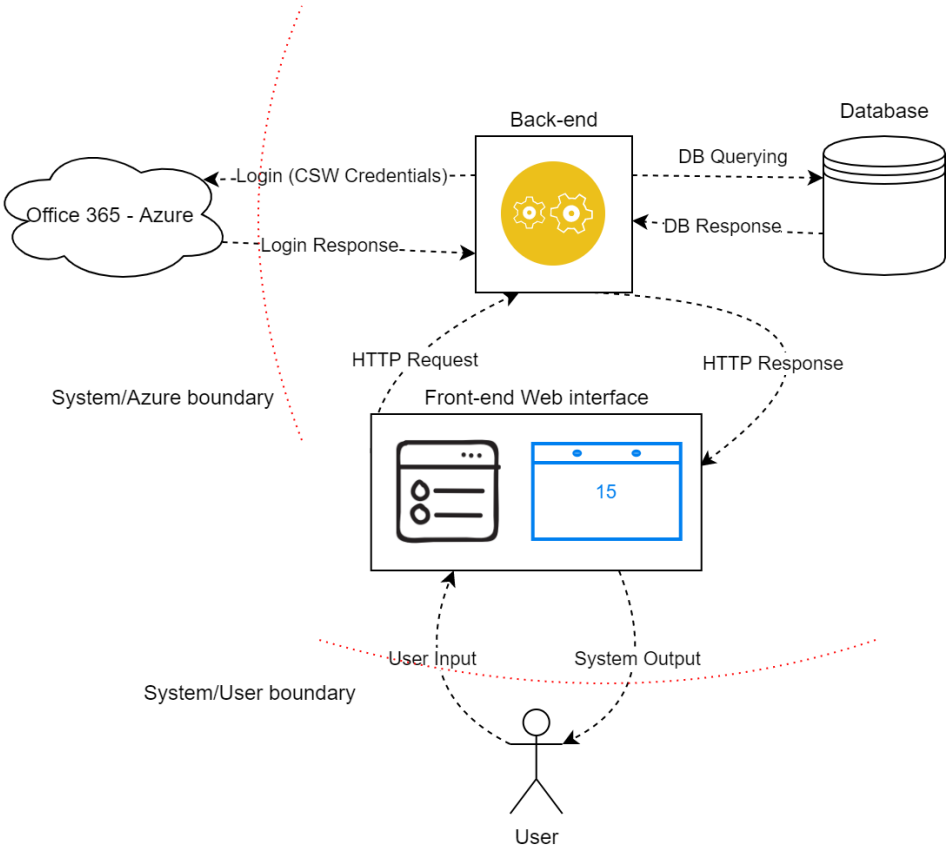

Figure 4.4: SmartShiP - Diagram

4.4 SmartShiP Project Startup

In this section the SmartShiP project gains further traction by defining the Minimum viable product and future developments.

4.4.1 Defining the Minimum Viable Product

To define the Minimum Viable Product [Ries (2019)], the team must strip down all the bells and whistles of the application and focus on what is really important for this application to have its core functionality and what makes it bring value to the stakeholders.

By thinking ahead, one can put himself in the role of software supplier and plan how the deployment of the solution will be performed and therefore, by looking at it from a higher level the solution was broken down as the

following components: Backend, Frontend, Database.

The process to define the Minimum Viable Product (MVP) was to set the requirements for the application to function and to provide the user with the necessary functionality. The Minimum Viable Product consists of the following core functionality:

- Store team members availability;
- Generate a calendar with everyone's availability;
- Generate calendar with on-call;
- Web interface to retrieve the calendar;

As for each component presented on the previous chapter (backend, frontend and database), the following functionalities were decided:

- For the Frontend there are some views needed to fulfill the MVP features:
  - Login
  - Report intervention;
  - Report availability/holidays;
  - Check schedule;
  - Check personal reports;
  - Check team report (manager).
- The Backend functionalities needed for the MVP are:
  - Perform DB queries;
  - Validate input data;
  - Generate individual and team reports;
  - Authentication and Authorization;
  - Generate a file report of changes;
  - Generate audit file;
  - Revert any action;
- Database MVP:
  - Normal working days (person, start_date, end_date, schedule, date)

  - Intervention (person, start_date, end_date, time spent , desc)
  - Availability (person, start_date, end_date, holiday, pulsar)
  - On-call (person, start_date, end_date, time_spent)
  - Date dimension (date , holidayUK, holidayPT, weekend)
  - Person Table (person, name, role)

- Non Functional Requirements, While developing the team should focus on having this in mind:
  - Maintainability - The ease with which the system can be changed, whether for bug fixes or to add new functionality. This is important because a large chunk of the Information Technology (IT) budget is spent on maintenance. The more maintainable a system is, the lower the total cost of ownership will be.
  - Portability - The ease with which software can be installed on all necessary platforms and the platforms on which it is expected to run.
  - Scalability - Software that is scalable has the ability to handle a wide variety of system configuration sizes. The nonfunctional requirements should specify the ways in which the system may be expected to scale up (by increasing hardware capacity, adding machines, etc.). Scalability ensures usability for five users or five thousand.
  - Security - Integrity requirements define the security attributes of the system, restricting access to features or data to certain users and protecting the privacy of data entered into the software.

4.4.2 Future Developments

During the design process, there were many ideas which would not be included in the MVP but that should be thought as additions on the application in the future. As such, SmartShiP got some possible future developments, should any be allowed:

- Integration with Microsoft (MS) Shifts;
- Java Choco lib to empower the application with Constraint Satisfaction Problem (CSP) capabilities;
- Integrate with calendar MS Exchange;
- Daily email with statistics;

- Configuration files;
- Metrics, statistics and indicators;
- DB Backup system;
- Holiday automatic absence on Office365 Calendar;

To further develop on the database, the following ideas were defined:

- Improve data quality and testing data;

Reporting and Dashboarding:

- Include frontend libraries for Reporting and Dashboarding like grafana;
- Possibility to have Elasticsearch, Logstash and Kibana (ELK);

Backend

- Refactoring of connectors with DB
- Refactoring of classes to comply with company code standards;

4.5 Architecture

In this section the architecture of SmartShiP is presented and each component is explained, at the end there is a brief overview of the stakeholders of the SmartShiP project.

4.5.1 Macroscopic System structure

All of the application components during development will be containerized using Docker containers. All docker containers will be orchestrated with Docker-compose to create a single network layer for SmartShiP.

Backend

The backend is a Java server application using Wildfly [RedHat (2020)]. It exposes the application via an API that the frontend can use to perform actions on the system.

The main components of the server are the Database connector class, requester handler for all the API endpoints and the DAO.

The Backend follows the DAO design pattern in order to streamline the creation of new endpoints. A DAO allows for several classes to interact with the Database using defined access points.

Frontend

The frontend is built using Angular and it will be the main way of interfacing with the application. At the beginning, the application will not be following any specific design pattern as this subject is still due for maturation and overall evaluation of the application's future needs.

Database

The Database system had a simple approach by just having an Azure hosted MySQL database to store the Data of the application. The data of the application will be aggregated in three groups: Availability, Intervention and Schedule.

4.5.2 Stakeholders

The stakeholders of SmartShiP are the Project managers, Finance and HR of the company, as the tool itself is for internal usage in the company.

The finance and human resources team utilizes the monthly report generated by SmartShiP to calculate the on-call and intervention payments to staff.

Project managers supervise the balance of the schedule and have an overview of the load of the service.

4.6 SmartShiP project roles

My roles during the development of SmartShiP were:

- System architect by conceptualizing, designing and document SmartShiP solution;
- Product owner by writing both the User Stories and Technical tasks of SmartShiP;
- Agile coordinator by promoting daily meetings, retrospective meetings and refinement meetings;
- Supervise and help with the progress of development tasks carried on by the team members;
- Developing and reviewing tasks.

SmartShiP was developed with a team composed by developers and myself the team leader. As a developer, my the participation on SmartShiP was:

- Develop and review tasks;
- Provide estimations during refinement meetings;
- Document the developed tasks.

4.6.1 Resource constraints

Due to SmartShiP intention to provide knowledge on the technologies used on other projects in the company, most of the resources will be inexperienced engineers. Also, SmartShiP is not running with any allocated time so it will only be used as a time filler during less busy working days.

The developers are free to choose any technology but Project Management is more inclined on using staple technology instead of cutting edge technology.

There are no restrictions on the amount of computing resources SmartShiP can use for this as well.

4.7 Development framework

In this section, the development framework known as the way of working, are explained and broken down into: roles, documentation, environments, testing and tasks. This framework is very similar to the ones used in the company.

4.7.1 Roles

Since SmartShiP assigns responsibilities to different people, a few roles were defined in order to maintain the communication simple and the different streams of work separated to reduce dependencies:

- Product owner
- Software Architect
- Developers

The role of Product Owner is assigned to the person who will be responsible for defining the functional requirements of the final product. There are also instants where the Product Owner will need to clarify or prioritize certain aspects of the development of the application so that the development backlog can stay in line with the business needs.

For SmartShiP to be designed, the role of Software Architect is taken by the team lead initially, who has the responsibility to design and prepare the SmartShiP project for the development team to work on. The whole concept and solution presentation is prepared by the Software Architect, having also the responsibility to ensure that the high level documentation contains meaningful information about the product.

At the base of the SmartShiP project there are the Developers, whose responsibility is to develop the codebase, maintain the technical documentation, create tests and review other developers' code.

4.7.2 Documentation

All the documentation is set to be produced and maintained either in the code to explain the code itself or in Confluence [Atlassian (2020b)] in a wiki format that is accessible to everyone easily.

Most of the documentation will follow the existing projects structure which is broken down into High Level and Low Level documentation.

High Level: The purpose of this documentation section is to provide a customer with all the information necessary to understand the purpose and use of the software.

Low Level documentation: This is mostly used to aid the developers, technical managers or system architects.

There is also specific information about deployment, development and testing/validation of the software. In some instances it is important to also include a Knowledge base for all the information that is relevant but doesn't fit the other topics.

4.7.3 Environments

When deploying the application it is important to consider the different environments that are part of the natural flow of an application life-cycle that starts in Development, goes through Testing and finally arrives into Production.

To perform some manual tests, the strategy adopted was to include a flag that is read by the CI/CD Pipeline to deploy the version of the application with the specific fix or feature onto a server.

Moving into Testing, SmartShiP can be deployed in several servers that run different test batches and are under different loads. There is also the option to have a pre-production environment that can be included after all testing passes and it is used to test data migration from previous production versions into the new one. The current CI/CD Pipeline doesn't include such a step.

When the batch of tests are finally complete and it returns a successful result the project is ready to prepare a Release, being a release the agglomeration of several improvements, fixes and changes into a single version that can be deployed into the Production environment when there is a planned deployment or maintenance window.

4.7.4 Testing

The test approach is very simple and no automated tests were implemented. Therefore the initial testing plan will include some dummy data to manually test the application and a small end-to-end test run.

After the first release, to ensure the application functionality is working as intended and as stated in the requirements, SmartShiP will include regression testing for any defects found after the first version release.

4.7.5 Tasks

In JIRA [Atlassian (2020d)], the SmartShiP project had a specific structure that is similar to a tree where on the root there is an Epic which branches

into Tasks or Stories and these into sub-tasks.

Developers work directly on subtasks except when the Tasks or User Stories are already small enough. This is done to avoid unnecessary overhead.

During task development, developers are encouraged to commit regularly to their task branch, at least once per day and leave a comment about the progress and barriers found until that point. After all is done a Pull request is opened for review and testing from other team members and once approved the branch is merged to Development (DEV).

4.8 Final overview

By utilizing the Engineering Design Process, SmartShiP managed to define all its requirements and prepare the development phase. It was important to conceptualize the model and given its requirements and constraints, prove that a solution for the schedule rotation problem is achievable.

All the components for development of SmartShiP were defined such as Database, Backend, Frontend and Infrastructure. Some non functional requirements were defined as well and they affect the way Software development is made on SmartShiP

Chapter 5

SmartShiP: Project development

In this chapter, all the details about the development of the SmartShiP project are explained. It contains several sections explaining each of the components and their development, the Agile practices used and an overview of the state of the project at the time of writing.

5.1 CI/CD

The Continuous Integration/Continuous Deployment solution will be based on Jenkins, where there will be a few pipelines dedicated to the deployments.

Currently the pipeline has the following steps Build, Static Analysis, Publish, Deploy.

The development workflow - Gitflow Workflow - is automated by the use of Jenkins Multibranch Pipelines projects. This project type enables us to implement different Jenkinsfiles for different branches of the same project (different repositories on SmartShiP's case).

The server has the responsibility of running the Jenkins pipelines. To be able to do this, everything set up in this machine is contained in dockers, and to manage the dockers there is a docker-compose file which restarts the dockers in the case that the machine is shut down or restarted. The docker-compose will need a secret for the Jenkins slave, this secret is stored in a .env file in the docker-compose file directory.

5.2 SmartShiP Project breakdown

As stated previously, there are three main components on SmartshiP: Backend, Database and Frontend; so for each component there is an Initial Setup task that consists on: Diagrams explaining the basic functionality of the backend, Server setups, DEV/TEST/PROD Environments setups on Dockerfiles, create dummy pages or endpoints.

For the Minimum Viable Product version, not only it had the Backend, Database and Frontend tasks but also CI/CD Infrastructure which contained tasks to configure the CI/CD Jenkins pipeline, associate a Jenkins job for pull requests, create stages to deploy and build the SmartShiP project for each branch and some documentation.

After this stage, its possible to deploy a dummy version of SmartShiP that had no functionality but it is the canvas for it. With this canvas ready there are two different types of tasks moving forward: User Stories and Tasks. For the MVP there are the following User Stories:

- MEMBER Reports Intervention
- MEMBER Gets the Individual Report
- USER logins to the application
- MEMBER reports availability
- MEMBER Gets the Team Report
- MEMBER verifies schedule
- MEMBER verifies availability
- LEAD generates Schedule

And the following Tasks:

- Create Permission schema in Database
- Create Back End component to handle requests from the FrontEnd
- Create Back End component to query database
- Create Back End component to generate CSVs
- Solve SonarQube Issues

With these Stories and Tasks prepared, their respective subtasks were prepared during two Agile Refinement meetings.

Given that each Sub-task is given an estimate, it is possible to determine the lead time for the first release of SmartShiP, which due to the nature of the team cannot be directly translated into a release date as the hours dedicated to SmartShiP development are unpredictable.

5.2.1 Tasks

Each task will have a similar workflow besides the technical differences, an example task will be presented for each component (backend, frontend, database, infrastructure and CI/CD)

The examples contain the main fields taken from the JIRA page of the task and the workflow done.

Backend example task

SubTask - Receive and handle the request for Intervention > Part of MEMBER Reports Intervention story

In JIRA this subtask was filled as such:

Component : SmartShiP - Backend
Fix Version: MVP
Labels: BackEnd, Induction, SmartShiP, TSTEAM
Original Estimate: 1 day, 5 hours
Dependencies: Create a form for the Intervention; Create Permission schema in Database

Description:

- Check permissions
- Validate the request fields
- Query the Database to insert the intervention

To start developing this task, begin by creating a branch from DEV in the git repository with the following name: feature/DCOCPESSETUP-218-receive-and-handle-the-request-for-intervention

With the branch created, the development can start by selecting the newly created branch on the IDE, opening or creating the classes needed to develop this task which would be the DAO for Interventions and all its CRUD

operations as well as an endpoint to test the raw access to the intervention operations.

The new classes are the following:

- Intervention.java - This includes the data types for the Intervention object and a simple toString() function
- InterventionDao.java - This is the interface that defines the DAO operations namely: getInterventionByPerson, createIntervention, updateIntervention and deleteIntervention.
- InterventionDaoImpl.java - Contains the implementation of the interface methods, plus some other private functions that may be helpful. This will contain SQL queries and the connection to the database to perform the queries.

And an endpoint is included in the API per defined operation on the InterventionDao, these are either GET, POST, DELETE or UPDATE HTTP commands, that are simply done by calling the InterventionDaoImpl methods with the provided variables in the request fields which can be validated at this point.

The current permission scheme allows for any command to be issued by anyone so the permissions won't be checked as previously desired.

Created a small guide on how to call the endpoints and test it manually.

After all this is done, the dev branch can be merged into the local branch so the local branch is updated, solve any conflicts and then create a pull request for this branch to be reviewed. The task is moved to the Review state until the pull request is accepted.

Frontend example task

Sub-task - Create a form for the Intervention > Part of MEMBER Reports Intervention story

In JIRA this subtask was filled as such:

Component : SmartShiP - Frontend
Fix Version: MVP
Labels: Frontend, Induction, SmartShiP, TSTEAM
Original Estimate: 1 day, 5 hours
Is Dependent By: Receive and handle the request for Intervention

Description:

- Create a service to feed the component
- The component is the view/form
- Handle responses from the backend
- Serve the user the form if he has permission
- Display a message in pulsar after if successful request

To start this frontend task, a branch from DEV is created in the git repository with the following name: feature/DCOCPESSETUP-219-create-a-form-for-the-intervention

With the branch created, development can start by selecting the newly created branch on the IDE.

The overall flow of the Frontend tasks is to create a service to interact with the backend and a component form to interact with the service. The service sends http requests and processes the response. The form is used to receive input from the user and show the results of the query back to the user.

When the component is ready, a plugin on IntelliJ that allows for hot changes on angular to be changed immediately on the browser will be used. This is useful to test how the application is running with the newly added frontend component.

There is no automated testing done at this point, only manual testing done at the review stage, which can be indicated either on the comment section or on the pull request itself.

After all this is done, dev can be merged into the local branch so the local branch is updated, solve any conflicts and then create a pull request for this branch to be reviewed. The task is moved to the Review state until the pull request is accepted

Database example task

Task - Create Permission schema in Database

In JIRA this task was filled as such:

Component: SmartShiP - Database
Fix Version: MVP
Labels: Database, Induction, SmartShiP, TSTEAM
Original Estimate: 1 day
Is Dependent By: Receive and handle the request for Intervention
Depends: Initial database setup

Description

- Create a table that contains (User_ID, Permission_item)
- User -> Item permission
- User = user_id +-
- Permission_Item = (Hierarchy | Permission) -> (Project A | Operation)
- Operation = AccessProject, GetCalendar
- Don't forget to update scripts that touch this code piece

To start this database task, a branch from DEV is created in the git repository with the following name: feature/DCOCPESSETUP-221-Create-Permission-schema-in-Database

With the branch created, development can start by selecting the newly created branch on the IDE in order to import the current Database schema.

To play around with the database, the docker container for the database is started and accessed it via the IDE or DBeaver. From DBeaver the tables that will be used for the permissions are created and tested, as for SQL tasks most of them are based on the description or technical requirements from the docs.

For manual testing, test data is generated for the current endpoints and its permissions.

SmartShiP does not use any technology like Flyway or Liquibase on SmartShiP so the database is always fully reconstructed when a new change is added.

After the database schema is as the task demands, the pull request is created and it remains in this state until reviewed and approved.

CI/CD example task

Sub-task - Configure Jenkins pipeline and build job for repo and dev branch
> Part of Initial CI/CD Infrastructure setup task

In JIRA this task was filled as such:

Component: SmartShiP - Infrastructure
Fix Version: MVP
Labels: Induction, SmartShiP, TSTEAM
Original Estimate: 1 week
Is dependent by: Create DEV and PROD environments

Description:

So that SmartShiP complies with the internal policies and guidelines of the company the task should have:

- a Jenkins pipeline configured for the SmartShiP repo and dev branch. The existing pipeline template should be validated to check if it fulfills the needs of the SmartShiP project and, if needed, it should be modified to meet those needs.
- a Jenkins job using the pipeline above in order to automatically trigger new builds. This job should poll the Bitbucket branch for new changes and trigger a build when new changes are found.

Acceptance Criteria (AC):

- AC1: Job created in Jenkins and triggered when new changes are made;

To start this CI/CD task, the team is informed to not merge anything into DEV until this task is finished in order to not interfere with the system.

During this task a branch named feature/DCOCPESSETUP-105-tests-in-jenkinsfile will be created to test the work in progress as it is required to have a jenkinsfile inside the repo for it to be used inside the development server during builds or deployments.

There are no special requirements for which IDE to use, so a text editor is used.

The stages on the Jenkins file are created as follows:

- Build: This stage is responsible for building the SmartShiP project.
- Static Analysis: This stage is responsible for analyzing the code against the Sonarqube profile.

- Publish: Is responsible to upload the artifacts to nexus (This will result in having the final .jar files in delivery-smartship-maven-snapshots nexus folder).

Most of the commands used for the pipeline use Maven which is prebuilt into Jenkins and are a matter of testing as you go to check if the results are what is expected.

When these steps are done, the pipeline is tested by pushing commits into the bitbucket and checking the CI web page of the company. The final changes can be pushed and a pull request is opened for this to be available in DEV.

Further checks should be done to make sure all Pull requests are building as expected.

Infrastructure example task

Sub-task - Setup the Server with Thorntail > Part of Initial Back End setup task

In JIRA this task was filled as such:

Component: SmartShiP - BackEnd
Fix Version: MVP
Labels: Backend, Infrastructure, Induction, SmartShiP, TSTEAM, Thorntail
Original Estimate: 5 hours
Is dependent by: Create DEV and PROD environments

Description:

- Similar to the prototype2
- AC1: The server is Dockerized
- AC2: There is a test endpoint when the backend is deployed in the Server
- AC3: The system is debuggable on IntelliJ

To start developing this task, a branch from DEV is created in the git repository with the following name: feature/DCOCPESSETUP-189 setup the server with thorntail

With the branch created, development can start by selecting the newly created branch on the IDE or text editor. The backend will be dockerized, therefore the task will start with the commands that are normally used to build and run the backend.

The dockerfile needs to be as independent as possible. All the system dependencies are included in the dockerfile such as Java 11 and Maven.

In the docker-compose files, the environment keywords that are sent to the built Java application are included, the main difference being the ports that the web service uses. There is a different docker-composes for each environment.

Documentation on how to debug and run the environments is produced on a Readme.md file.

After all this is done, dev can be merged into the local branch so it is updated, solve any conflicts and then create a pull request for this branch to be reviewed. The task is moved to the Review state until the pull request is accepted.

5.3 SmartShiP - Agile practice

In this section, the Agile adoption in the project is explained and the ceremonies that took place during development. All the decisions taken for our Agile practice were proposed by the company.

5.3.1 Daily stand up

The daily stand up is a round the table meeting where everyone in the team explains what was accomplished the day before, if anything is blocking progress and future tasks.

5.3.2 Refinement meetings

These meetings allowed the team to break down each Task and User Story into Sub-tasks, clarify any details regarding them, and estimate how long each task takes to complete.

To refine a task, first the task was broken down into sub-tasks for each component, then each sub-task is estimated using Planning Poker until at least 60 percent of the cards are the same value.

5.3.3 Retrospective

During the fortnightly retrospective, the following topics related with SmartShiP were discussed:

- List of tasks resolved or achievements since the last retro: This takes around 15 minutes to go through as you are supposed to assess the state of SmartShiP given the finished tasks and ask for any difficulties felt during the execution of selected tasks.
- Feedback of the company activities and team: The bulk of the meeting notes are taken here as this is the main driver for improvements and overall well being of the team members and thus is of major importance, this can take around 30 minutes to go through everyone's low and high points.
- Team member goals: This are general goals for the next cycle and are just general guidelines on how each person is going to progress next week.
- Actions for next cycle: During the retrospective actions from any point of the meeting are captured and assigned to the affected team member.

5.3.4 Risks and Lessons Learned

During the Risks and Lessons Learned meeting that happens between Project Manager and Tech leads, all the undergoing risks are noted in a tracker that can be always looked at and prioritized according to Probability and Impact.

5.4 Difficulties

As with all new projects there are adjustments that are required so that all difficulties of SmartShiP are eased. During the first weeks of SmartShiP development the pain points of the starting development project were noted down so that it could be tackled with some preparation and discussion.

Due to the sheer objective of SmartShiP being to train and welcome the newly built team on the technologies of S1ES project, it was expected that the proficiency with these technologies to be low. This led into a situation where some of the key tasks were too complicated to be done independently and resulted in delays on the SmartShiP delivery.

To accommodate these challenges further technical training by the title of Tailored Learning Units was set in place to freshen up some of the concepts of programming and engineering that were thought to be important not only for the immediate times but also on future endeavors of the team.

5.5 SmartShiP Overview

As it stands, currently SmartShiP is already used internally with the team rotation and schedule. It uses a static approach on the team rotation so it doesn't adapt for teams with fewer or more elements and it is only configured for this specific rotation plan.

It adapts to the holidays of each team member and is able to maintain a healthy balance on the hours done by each person. Most of the interactions done with the system are issued from the Frontend.

Unfortunately, and due to some topics being quite difficult such as the login and synchronization with Keycloak and Azure active directory, SmartShiP has not yet been released into production for other teams.

Currently the SmartShiP frontend is composed by a top bar with the SmartShiP logo on the left, notifications, messages and "My account" links on the right.

The user is then presented with the calendar and two buttons to insert availability or report intervention.

At the bottom of the page there are two boxes for schedule management: Schedule generation button and team reports.

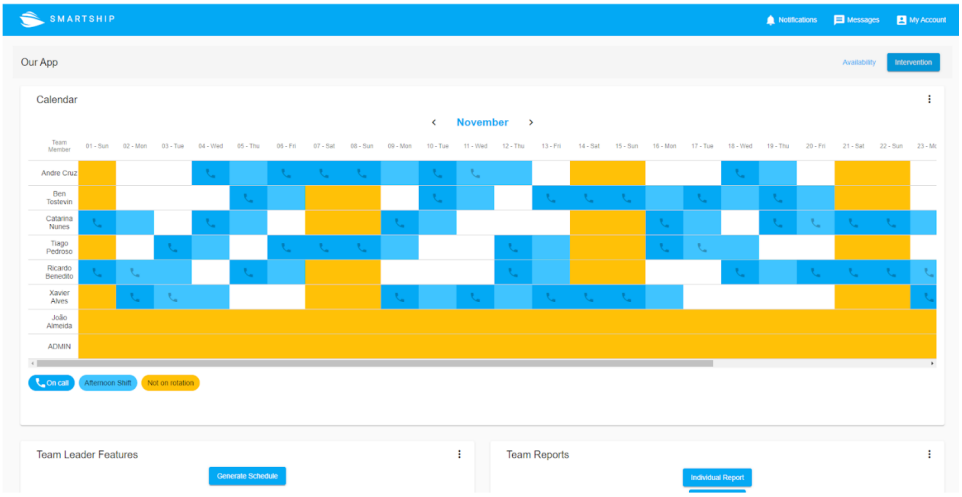

Figure 5.1: Front End

To report interventions the user clicks on the top right button that says "Intervention" and is presented with a modal to insert the start and end date and the time of the intervention.

It works by requesting the backend to insert the provided parameters into the database, making them available for extraction on the team reports.

Figure 5.2: Front End - Report intervention

A team report can be accessed by clicking the button "Team Report" at the bottom right of the page. It will then download a CSV file with the report for the selected team.

The team reports extract data from Interventions, Worked hours, availability and on-call hours. There are then some formulas applied to the fields that allow for interventions and on-call hours payments to be explained. This are the reports that Finance and HR can use. It is a temporary solution as a future dashboard is being prepared to address this topic.

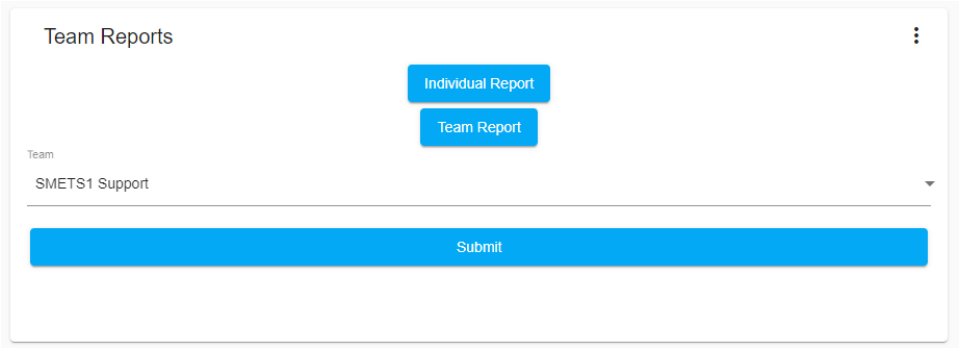

Figure 5.3: Front End - Team Report

To submit absences into the system, the user must press the "Availability" button on the top right corner. A modal will be presented to the user with an absence type and the start and end date.

By submitting absences, the database is updated with this data and it is then used to generate the schedule, which will make sure that the days of unavailability are not filled with a shift.

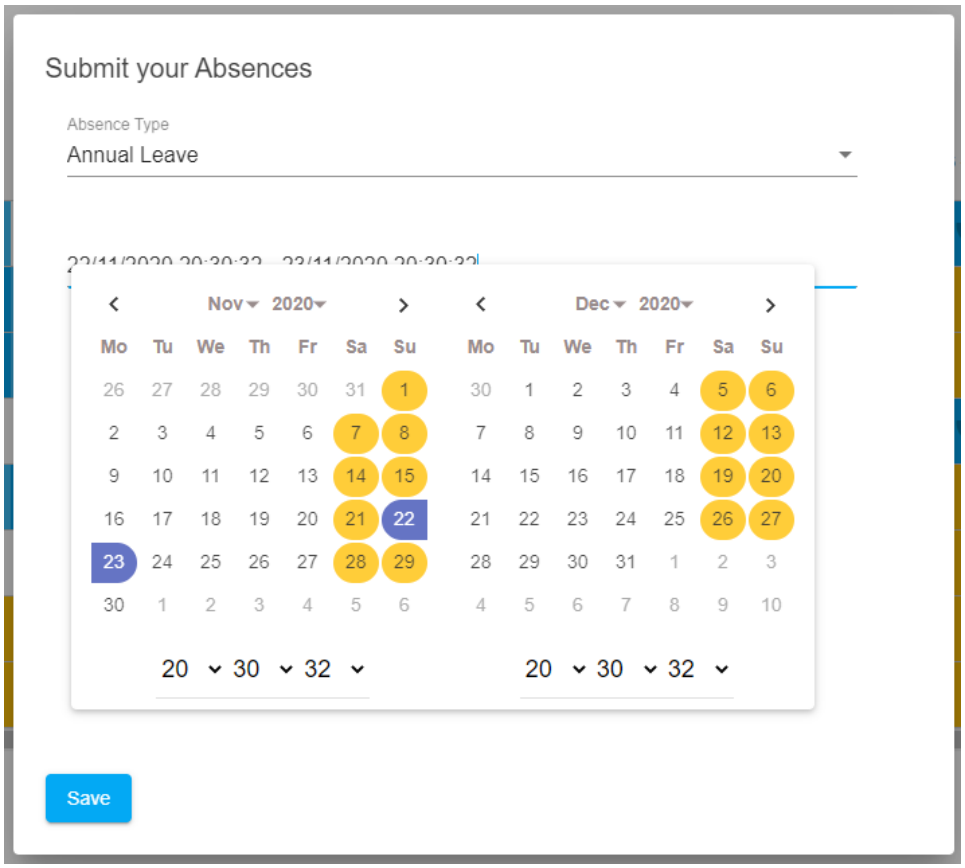

Figure 5.4: Front End - Absence Modal

The database holds all the data used in SmartShiP, the database data is accessed using DAOs created in the backend and currently the database has three main business groups and four support tables:

- Interventions
- Availability: personal absences
- Schedules: prevention and work schedule
- Person
- Schedule
- Absence types
- Dates

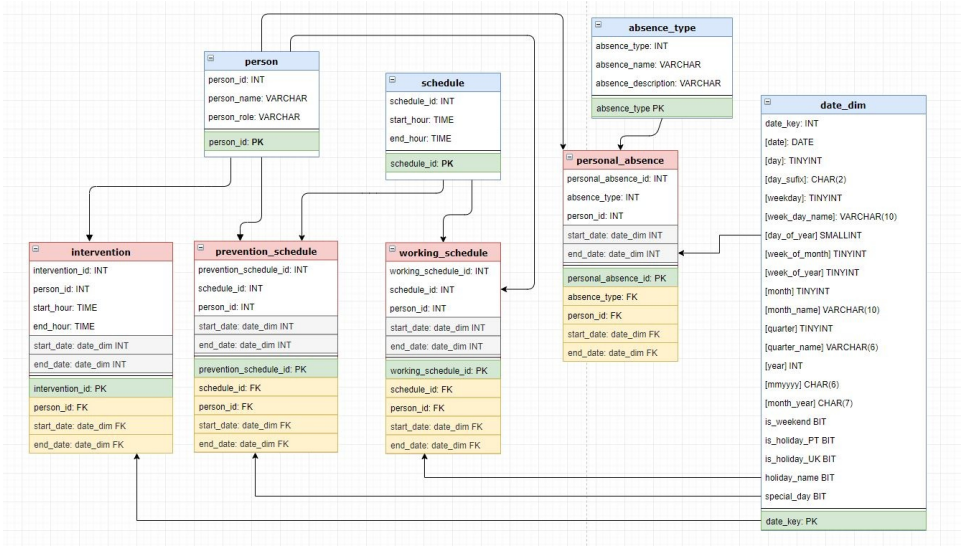

Figure 5.5: Database Schema

With this, any month of the year can be seen on the web application and it reflects part of what was already being used. Most of the functionalities planned for release are done besides the login.

Chapter 6

Conclusion

Software development using Agile methodologies is a practice that has increasingly popularity in software projects. By adopting a framework built with Continuous Integration in mind, there is a focus on the value provided by the software that considers more often the changes of a business and results in better performance of the developers overall.

This dissertation presents the implementation of the SmartShiP system, seeking to explore the concepts previously mentioned. This was done under the scope of the Master studies in Informatics Engineering of the University of Évora using the techniques and principles learned and applying them in real cases, as well as exploring subjects that were not part of the Masters course program.

By developing SmartShiP I can conclude that the usage of Agile methodologies plays an important factor during the software development phases in general and of SmartShiP in particular. It provides a middle ground for communication between the technical aspects and the project management and, with the increase of communication opportunities, soft skills are also improved.

This solution proved to be useful and can potentially save hours of work to all the involved people in the process of schedule rotations.

SmartShiP is still under development and is being internally tested with the team since May 2020 and is expected to be released for other teams in the same company project. Currently new features are being developed such as Login using Azure credentials, multi project functionalities and a testing infrastructure.

Although some of the technologies used were not bleeding edge, they are still

proven and very used ones within the company and, besides the technical aspects, all team members improved on problem solving skills.

In the future SmartShiP will be leveraged to include more technologies such as testing automation and new libraries on the backend to further improve the algorithms used.

As architect and team leader, my involvement with the surrounding parts of SmartShiP and the concept was really important as I developed sensitivity and creativity to solve the problems that were encountered.

This dissertation explained how software development, agile methodologies and software architecture are part of Software services and products. The techniques that were used during the development of SmartShiP were:

- Scrumban agile methodology;
- Monolithic architecture;
- Conceptualization and proof of concepts;
- Minimum viable product definition;
- Programming techniques: DAO

6.1 Future work

I, the product owner am already planning features for a newer version of SmartShiP that will have new features. The new versions of SmartShiP that are planned are v1.0.1 and v1.1.0.

The next versions of SmartShiP will follow a streamlined process to add the known as "good to haves" by including functionalities that broaden the user base and increase the use cases that the tool covers.

In parallel software quality will improve by adding code analysis and testing as part of the defined non-functional requirements.

The change log for v1.0.1 is the following:

- Generalize scheduling algorithm and include configuration file
- Add multi project capabilities to include multiple teams
- Fix interface bugs and remove unused buttons

The change log for v1.1.0 is the following:

- Create groups of permissions
- Intervention Report - Validation improvement
- Manager Adds People to project
- Manager Removes people from project
- Member edit/Delete intervention

Backlog of planned functionalities without a version:

- Integrate remote machine scripts with Bitbucket
- Setup SonarQube static analysis
- Create Stage for publishing dockers into nexus
- MANAGER manage MEMBER permissions
- MANAGER/LEAD receives notifications - Availability changes
- MEMBER Change Shifts
- SYS ADMIN Edits Project
- SYS ADMIN Deletes Project
- MEMBER verifies reported interventions
- LEAD schedule off-days for other members
- MEMBER edit/delete intervention
- Add Choco-solver library to the project
- Integration with MS Shifts
- Integration with MS Exchange Calendar
- Dashboard with Metrics, Statistics and indicators
- DB Backup system

The future SmartShiP backlog is still under discussion and its tasks and stories have not yet been written and reviews. With each version, more people will on-board SmartShiP and start utilizing it.

Bibliography

Alliance, S. (2020). Definition of scrum, scrum alliance (accessed 17 november 2020) <https://www.scrumalliance.org/about-scrum/definition>.

Atlassian (2020a). Agile epics: definition, examples, and templates, atlassian (accessed 17 november 2020) <https://www.atlassian.com/agile/project-management/epics>.

Atlassian (2020b). Confluence - features, atlassian (accessed 17 november 2020) <https://www.atlassian.com/software/confluence/features>.

Atlassian (2020c). Gitflow workflow: Atlassian git tutorial, atlassian (accessed 17 november 2020) <https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow>.

Atlassian (2020d). Jira software - features, atlassian (accessed 17 november 2020) <https://www.atlassian.com/software/jira/features>.

Atlassian (2020e). Opsgenie - on call management and escalations, atlassian (accessed 17 november 2020) <https://www.atlassian.com/software/opsgenie/on-call-management-and-escalations>.

Atlassian (2020f). Scrum roles and the truth about job titles in scrum, atlassian (accessed 17 november 2020) <https://www.atlassian.com/agile/scrum/roles>.

Atlassian (2020g). User stories with examples and template, atlassian (accessed 17 november 2020) <https://www.atlassian.com/agile/project-management/user-stories>.

Atlassian (2020h). What is agile?, atlassian (accessed 17 november 2020) <https://www.atlassian.com/agile>.

Atlassian (2020i). What is kanban?, atlassian (accessed 17 november 2020) <https://www.atlassian.com/agile/kanban>.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., and Thomas, D. (2001). Manifesto for agile software development, agile manifesto, (accessed 17 november 2020) <http://www.agilemanifesto.org>.

Germanov, I. (2019). Kanban vs scrum vs scrumban: What are the differences?, germanov, iliyan (accessed 17 november 2020) <https://ora.pm/blog/scrum-vs-kanban-vs-scrumban>.

Jenkins (2020). Jenkins - build great things at any scale, jenkins (accessed 17 november 2020) <https://www.jenkins.io>.

Latre, A. B. (2019). Contextless scrum: A principles or rules driven framework?, latre, alex ballarin (accessed 17 november 2020) <https://www.scrum.org/resources/blog/contextless-scrum-principles-or-rules-driven-framework>.

Microsoft (2020). What is shifts?, microsoft (accessed 17 november 2020) <https://support.microsoft.com/en-us/office/what-is-shifts-f8efe6e4-ddb3-4d23-b81b-bb812296b821>.

Oracle (2020). Core j2ee patterns, oracle (accessed 28 november 2020) <https://www.oracle.com/java/technologies/dataaccessobject.html>.

Pahuja, S. (2016). What is scrumban?, agile alliance (accessed 17 november 2020) <https://www.agilealliance.org/what-is-scrumban>.

RedHat (2020). About the wildfly project, redhat (accessed 17 november 2020) <https://www.wildfly.org/about>.

Richardson, C. (2020a). Pattern: Microservice architecture, richardson, chris (accessed 17 november 2020) <https://microservices.io/patterns/microservices.html>.

Richardson, C. (2020b). Pattern: Monolithic architecture, richardson, chris (accessed 17 november 2020) <https://microservices.io/patterns/monolithic.html>.

Ries, E. (2019). What is a minimum viable product (mvp)?, ries, eric, (accessed 17 november 2020) <https://www.agilealliance.org/glossary/mvp>.