Facultade de Informática
UNIVERSIDADE DA CORUÑA

FINAL PROJECT
DEGREE IN COMPUTER ENGINEERING
MENTION IN SOFTWARE ENGINEERING

Web module for widget based reports generation from report templates

Student: Juan Ramil Díaz

Direction: Paula María Castro Castro

Pablo Gómez Area

A Coruña, September de 2022.

To my mother

Acknowledgements

I would like to thank my family, for giving me the necessary boosts and support when it was needed. Also to Instituto Tecnológico de Galicia, for being a cradle. And, of course, to my project directors, from whom I learned priceless things.

Abstract

The objective of this project is the development of a report generation module in the context of an existing IoT platform. These reports will be composed of widgets and will be generated automatically from templates using the data stored in the platform, in order to extract useful information for the user in their decision making process.

Resumen

El objetivo de este proyecto es el desarrollo de un módulo de generación de informes en el contexto de una plataforma IoT ya existente. Estos informes estarán compuestos de widgets y se generarán automáticamente a partir de plantillas usando los datos almacenados en la plataforma, para permitir extraer información útil para el usuario en su proceso de toma de decisiones.

Keywords:

- Reports
- Graphics
- Templates
- Internet of Things
- Automatic generation
- Metrics

Palabras clave:

- Informes
- Gráficos
- Plantillas
- Internet de las cosas
- Generación automática
- Métricas

Contents

List of Figures

List of Tables

Chapter 1
Introduction

In this first chapter, it will be explained the context of this work, the motivation for the project, the main and secondary goals and the structure of this memory.

1.1 Context

The organization that wants to work with us is one that develops and maintains different integrations of an Internet of Things (IoT) platform. They follow the Sensor Observation Service (SOS) model, in concrete the implementation of 52North [2]. In this context, they are in charge of maintaining and extending an IoT platform, with different functionalities of monitoring, maintaining and managing different devices that can be uploaded to the platform. Ours is a module that will fit into the maintenance type, and will serve with reports with device data to the user, using graphical widgets.

1.2 Motivation

The organization that wants to work with us has already a more or less functional implementation of the system it wants us to develop: a report generation engine. Its technical problem with it is the following: it is an old tool, based on a library that has transitive dependencies with other possibly deprecated libraries, which makes it a very expensive module to maintain. Every time they want to release a new version of the platform that integrates the report engine, they have to spend time fixing dependencies issues. This means for the client that the development is more expensive and takes more time. The situation has extended for months, and they need a solution.

Each client of the organization can have different implementations or access many integrations of the system in charge of managing the data that comes from devices. This means that every customer may have different uses for the data collected, needing different types of

reports.

Our job consists in give to this organization a fully functional, custom application that they can effectively and easily maintain, fitting their wishes of control and maintainability over the tool, while also granting flexibility with respect to the kind of reports the application will process.

I chose this project since I think it can make me test and prove my abilities. It can be a great opportunity to learn web applications development, which is my aspiration.

1.3 Goals

In this section it will be described the goals of this project, the main goal and the secondary goals that have lead our work.

1.3.1 Primary goal

The main goal of this project is to create a web application module that serves the purpose of generating reports with graphs about devices that automatically upload data to the web platform of the organization.

1.3.2 Secondary goals

The secondary goals for this projects are the following:

1. A list of the available templates can be viewed from the application. A template can be clicked, and a device from the platform selected, so a report with the template's shape and the device's data is shown.
2. A concrete date range for the collection of the data can be selected.
3. Apart from hot generation of reports, these can be saved, so they don't have to be generated each time the user wants to inspect them.
4. The kind of graphic widgets that can be shown in a report must be open to expansion.

1.4 State of the art

Definitively, we are not the first ones that came up with the idea or the necessity of creating a report engine. For most companies of a certain size, automatic reporting is a priority, in order to maintain or improve the external or internal information flow. Even though we are not looking for this kind of requirement, our goal is similar: to automatically generate reports filled with data.

In this section we will explore some alternatives and a few libraries that let us generate automatic reports from templates, and try to explicit the reasons to develop our own application.

- **Enterprise Resource Planning**

  There are options such as Enterprise Resource Planning (ERP)s, which as part of their functionality, some allow for automatic report generation. However, it would not be worth it for our organization to contract this kind of product, because it is not such a large company that it needs all its services, nor is it worth doing so just to take advantage of a small part of its full functionality.

- **BIRT**

  Until the assignment of this project, the organization used a library for making reports, BIRT [3]. It is a widely used resource from Eclipse. Its main disadvantages are that it only permits to use its own tool for registering templates, that is has poor performance, and the major one: it generates dependencies with other old libraries that make its maintenance very costly. Those are the reasons why the company wanted to stop using this tool and make its own, as commented in Section 1.2.

- **JasperReports Library**

  As a tool from Jaspersoft [4], JasperReports Library is an open source tool that enables to generate reports from XML templates. It is widely used. Developers from the organization know this tool, yet they prefer a custom solution.

- **Conclusions**

  We need to reuse and adapt to the existing platform, that already uses JSON templates with other purposes. Also, we plan to execute all kind of custom graphs in our report with hot data, not just plain images and text. Given those facts, it is worth to implement our very own reports module, and fulfil our own specific needs.

1.5 Work structure

In this section, this final project memory's structure will be outlined.

1. **Chapter 1. Introduction:** in this chapter we introduce this final project, outlining its context, its motivation, the state of the art, its goals and this document's work structure.
2. **Chapter 2. Technology and tools:** here, it is described the technologies used to develop this project.

3. **Chapter 3. Requirements analysis:** as an essential part of software development, we carried out the extraction of the requirements. This process and its results are outlined in this chapter.

4. **Chapter 4. Methodology and planning:** in this chapter it is illustrated the methodology we are basing our work on, an adaptation that we will use, and the planning of the project together with an estimation of the cost and the follow-up.

5. **Chapter 5. Architecture and design:** this part is in charge of explaining the different architectural and design patterns used and their justification.

6. **Chapter 6. Implementation:** the development of the different functionalities is explained in this chapter.

7. **Chapter 7. Tests:** here, we explain the different tests carried away by the developer during the development, together with the testing philosophy we had to apply.

8. **Chapter 8. Report generation example:** in this chapter, an example of report generation is presented, together with an specification of the report templates used by the system.

9. **Chapter 9. Conclusions:** in this final chapter, we outline the main conclusions through a final discussion and list some possible future expansions of the module.

Chapter 2

Technology and tools

In this section, it is specified the technologies and tools that were used to develop this project. As mentioned above, the module that is the subject of this work is integrated into an existing application, so the tools that are used are mostly already defined. There are no compelling reasons to add complexity or use new technologies.

2.1 Software development

Next, the software development tools chosen to carry out this project are described.

2.1.1 PostgreSQL

PostgreSQL [5] is an open source and relational database management system. In this project, it is used to hold the data about widget templates, report templates, concrete widgets and concrete reports.

2.1.2 Java

Java [6] is a programming language. It is object oriented and its virtual machine allows its code to be executed in any type of hardware. Furthermore, it implements automatic memory management. It is used in this final project to implement the back end, and, so, the functionality of parsing the templates to generate readable reports and widgets and passing them to the front end.

2.1.3 Hibernate

Hibernate [7] is a tool that allows to establish automatic relationships between objects and relational database elements, through annotations. It will make easier to map our classes to the database, and we will use its implementation of the EntityManager [8] in Java.

2.1.4 MyBatis

MyBatis is a ”class persistence framework” [9] for mapping Java methods with concrete SQL queries. For that purpose, it uses, among other alternatives, XML ”mappers”. We will adapt an existing implementation so our system allows the upload of report templates.

2.1.5 Typescript

Typescript is a programming language based on JavaScript, but with typing. This feature helps with big projects and with object oriented programming. Our web behaviour will be programmed in this language.

2.1.6 Spring

Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can 'just run'. [10]

2.1.7 Angular

Angular [11] is a framework for TypeScript, converting front end development in an object oriented programming job. Its component-based and ready to implement any Single Page Application (SPA) [12]. It includes routing, component based design and other features that makes it very useful for this project. It will be used to implement the behavior related to the view of the reports.

2.1.8 Material

Material [13] is a compound of available components and a design style created by Google. We will use part of its components in the interface development, and follow its design principles to design the web.

2.1.9 HighCharts

Highcharts [14] is a web library for multiple frameworks (or plain Javascript), including Angular. It allows to depict many types of graphs. We will use it to implement our own graph widgets, making use of the graphs available.

2.1.10 HTML5

HyperText Markup Language (HTML) [15] is a well known markup hypertext language, very used in the creation of websites, and it is the language that web browsers understand. HTML5 will be used in order to implement the view of our web components.

2.1.11 CSS3

Cascading Style Sheets (CSS) [16] is a graphic design language used to make web pages attractive. In combination with HTML, it is used to implement the view of web applications. CSS will be used with these purposes to allocate and beautify the view of our interface.

2.1.12 Tailwind

Tailwind [17] is a web development framework for CSS. It allows to specify classes for HTML elements that are globally assigned to certain styles, saving time in the process so there is no longer need of working directly with CSS or specifying a .css file for each web component. Nevertheless, it is customizable and open to expansion, so new styles and classes can be defined. We will use it to save time developing, although we will complement it with pure CSS in some parts.

2.1.13 Maven

Maven [18] is a standard and a tool for building Java-written projects. It makes it easier to develop by pipe-lining the building of JAR files, for instance. It is open source, and allows to use plugins in its phase cycles, so the process of building applications is customizable. It also manages dependencies and establishes a workable structure. We will use it to build our Java application.

2.2 Development environment

This section describes the tools that we are using that have to do with the development environment.

2.2.1 Vagrant

Vagrant [19] is a tool to make and manage virtual machine instances. Among its advantages we could highlight its simplicity. In this project, it will be used with the goal of simulate a database of production, but during development. That is, a database to store the measurements to be displayed by the reports, the reports themselves with their widget instances and the templates for reports and widgets. This way, we will simulate the micro-service of our database with a virtual machine.

2.2.2 Docker

A container is an independent unit of software that holds the necessary libraries to execute an application and the application itself, so it can be executed among different operative systems.

Docker [20] is a containerization tool. It will serve the purpose of running a container with our database management system (PostgreSQL) in the virtual machine delivered by Vagrant, so we can access the database through it, simulating the isolation of the database from the other parts of the application.

2.2.3 VS Codium

VS Codium [21] is a "community-driven, freely-licensed binary distribution of Microsoft's editor VS Code". We will use it as an Integrated Development Environment (IDE) for the front-end development.

2.2.4 IntelliJ IDEA

IntelliJ IDEA [22] is an IDE by JetBrains. Its built-in tools make a good choice for programming and make the development an easier task. We will use it to develop the back-end of the application, and use its run configurations to start the application in a local environment in order to test it and give service to a local instance of the front-end.

2.3 Version control

These are the tools used to keep track of the different versions of the code:

2.3.1 Git

Git [23] is a tool for version control. It is distributed and widely used, becoming almost the standard for when software developers have to work together. It is used in the context of the organization to keep track of the history of the project, and to maintain a security copy of its code on its servers.

2.3.2 Gitlab

Gitlab [24] is a Git repositories platform builder. The organization we are working with has its own Gitlab instance, in order to maintain every project being developed and to perform its DevOps.

2.4 Text edition tool

The following tool has been selected to create and edit this work:

2.4.1 LaTeX

LaTeX [25] is a text editing specification, widely used in the publication of papers and scientific documents. We will use it to create this memory of the project.

Chapter 3

Requirements analysis

In this chapter it will be described the different requirements of the project, and in which iteration they were tackled.

3.1 Functional requirements

In this section, we will describe the different functional requirements. That is, the ones that are directly extracted from the communicated necessities of the client.

- **Upload device templates with report templates as an array property:** before the assignment of this project, there is already the possibility to upload device templates to the system, with different purposes. Our work here is to adapt the secondary system in charge of the templates upload so it accepts a list of report templates as one of the device template's properties. This requirement corresponds to iterations no. 1 and 3.
- **List report templates:** the user must be able to see which available report templates exist so hot reports can be generated from them. This requirement corresponds to iteration no. 1.
- **Hot-generate a report:** a template can be clicked, and a device selected, so a report can be hot generated from the processing of the template with the series of data from the device. Which series of data are shown in each graph is indicated in the template. This requirement will correspond to iteration no. 1.
- **View report:** a report must be able to be checked by the user. A report can be either retrieved from the database or hot-generated. This requirement is also part of the first iteration.
- **Saving a report:** if a report has been hot-generated from a template, it can be then saved in the database, so it is shown at the list of saved reports without generating it

again from zero. This is from iteration no. 2.

- **Selecting a range of dates for the report's data:** if the report is of the appropriate type (of type 'month'), a date picker will be available to select year and month for the data to be collected in that range. If the report is of the contrary type ('custom'), a date range picker will let the user select concrete dates for the data, with a maximum range of one month. This belongs to the third and last iteration.
- **Deleting a report saved in the database:** the user must be able to delete an existing report from the database. This requirement corresponds to the last iteration.

3.2 Non-functional requirements

The main non-functional requirement that we are going to face is the integration with the system that includes ours. For instance, the functionality of adding a device template (that already works in the system) has to be adapted to include report templates as an array attribute of the device template. We will have to adapt our whole system to the context in which it is being developed. This will define our architecture and design, and the code patterns that we will use.

3.3 Use cases

In this section it will be described the use cases related to the previously defined requirements.

Figure 3.1 represents the use cases of the module to be developed, as a description of the users and their interactions with the system, which we will describe next.

3.3.1 Actors

In this section we will outline the actors that interact with the system to be developed.

1. **User:** the final user of the application is the one that, mainly, will request reports with data for a concrete range of dates.
2. **Administrator:** an administrator responsible for the communication with the client of each integration of the platform (which is also usually a developer of the organization) can upload different templates to the system, in JSON format. The templates will include, of course, a list of report templates, that will be able for the User to select in order to hot-generate concrete reports.

3.3.2 Use cases concretion

In this section we will depict the use cases of our system, extracted from the analysis of the necessities of the client.

1. **UC-01. View all existing templates and saved reports:**

   The User must be able to see in the interface a list of the templates uploaded to the system. If any report template is from a device template associated to a type of device that exists in the platform of the user, then the report template is listed. This use case is detailed in Table 3.1.

2. **UC-02. Hot-generate a report:**

   The User can request the hot-generation of a report, directly from its template, after selecting a template and a device of its type to match report and data. Table 3.2 depicts the use case.

3. **UC-03. View report:**

   The User must be able to, once it has been saved into the database, check the detail of a report from the saved report list. More information is held in Table 3.3.

4. **UC-04. Save a report:**

   Once hot-generated, a button will appear in the screen so the report instance can be saved. This will be useful for when the User wants to view a report without generating it from scratch, perhaps because the template is to their liking. More details can be found in Table 3.4.

5. **UC-05. Select a month for the data to be displayed:**

   Whether it is a hot-generated or a retrieved from database report, and if in the template it is indicated that the 'defaultMode' attribute of the report is a string value equal to 'MONTH', the User will be able to select a concrete year and month to determine the range of dates (between the start of the month and its end) so the data is collected for it to be shown in the report. This use case is outlined in Table 3.5.

6. **UC-06. Select a custom date range of data to display:**

   Whether it is a hot-generated or a retrieved from database report, and if in the template it is indicated that the 'defaultMode' attribute of the report is a string value equal to 'CUSTOM', the User will be able to select a start date and an end date so the data is collected for it to be shown in the report. At most, a date range equal to one month may be selected. A depiction of this use case may be found in Table 3.6.

7. **UC-07. Delete a report:**

   The User can request the deletion of a saved report, by clicking a button and confirming their choice. The flow of the use case and other details are described in Table 3.7.

8. **UC-08. Upload a device template with report templates associated:**

   The Administrator must be able to upload a device template with a report template list associated, so those will appear in the User's application. This use case is explained in Table 3.8.

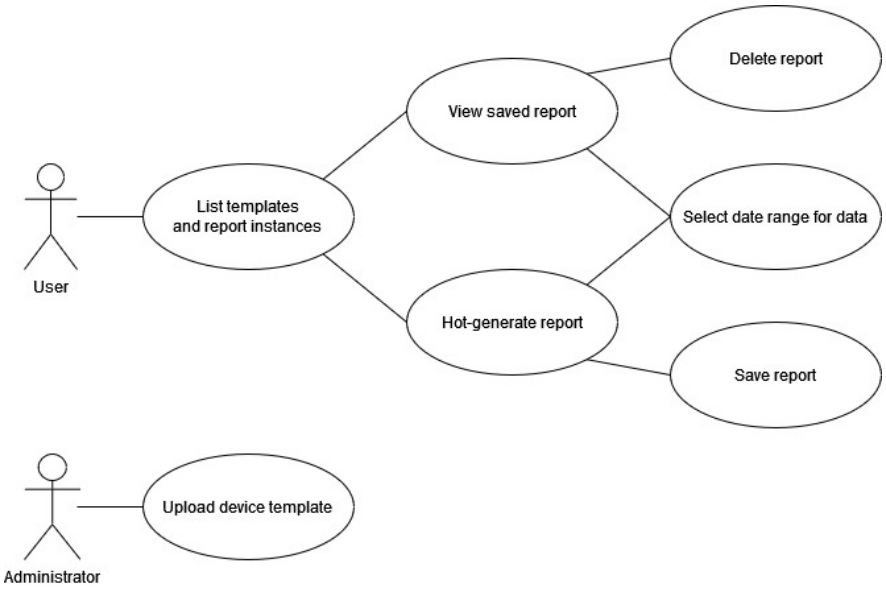

Figure 3.1: Use cases diagram

Use case no.	UC-01
Name	View all existing templates and saved reports
Description	Show all relevant templates and saved reports, with the name and small description. Only shows the templates that are associated to a type of device that exists in the platform of the user.
Trigger	User asks to list all reports and templates
Actors	User
Preconditions	The user must be logged into the system
Flow	1. The user requests the list of reports and the list of report templates. 2. The system returns both lists and displays them.
Alternative flow	-
Postconditions	Both lists are displayed
Exceptions	-

Table 3.1: View all existing templates and reports use case

Use case no.	UC-02
Name	Hot-generate a report
Description	Generate a report from a template, from zero and in the heat of the moment
Trigger	A user asks for the hot-generation of a report
Actors	User
Preconditions	1. The user must be logged into the system 2. The user has at least one device loaded into the system 3. There is at least one template loaded into the system of the type of the device
Flow	1. The user clicks on a template from the template list. 2. The system displays a modal screen with a list of the devices that are loaded into the system, and whose type is the indicated in the selected template. 3. The user selects a maximum of one device, then confirms their selection. 4. The system generates a report mixing the template's and device's data and shows it to the user.
Alternative flow	3.1 The user cancels the operation
Postconditions	A report is hot-generated and shown
Exceptions	-

Table 3.2: Hot-generate a report use case

Use case no.	UC-03
Name	View report
Description	A user checks a saved report
Trigger	A user asks to see a report of the list of saved reports
Actors	User
Preconditions	1. The user must be logged into the system. 2. There is at least one report saved in the database and listed in the report list.
Flow	1. The user clicks on a report from the report list. 2. The system displays a report from the database.
Alternative flow	-
Postconditions	A report is retrieved from the database and shown
Exceptions	-

Table 3.3: View a report use case

Use case no.	UC-04
Name	Save a report
Description	Once a report has been hot-generated, it can be saved in database, so there is no longer need of generating it from zero again
Trigger	The user requests the generation of a report
Actors	User
Preconditions	1. The user must be logged into the system. 2. A report has been hot-generated.
Flow	1. The user clicks on the ”save” button. 2. The system generates an instance of the report that is being shown in database, together with the widgets associated.
Alternative flow	-
Postconditions	A report is stored in database
Exceptions	-

Table 3.4: Save a report use case

Use case no.	UC-05
Name	Select a month for the data to be displayed
Description	If the type of the report is 'month', then a year and a month are selected to collect the data
Trigger	A user clicks the calendar icon in the view report screen
Actors	User
Preconditions	1. The user must be logged into the system. 2. A report is being viewed.
Flow	1. The user clicks on the calendar icon. 2. The system displays the corresponding date picker (year-month). 3. The user selects the desired date range, a concrete month. 4. The system loads the report with the data series associated to the new date range (from the 1st of month, to the last day of the month).
Alternative flow	-
Postconditions	A report is shown, with the new data
Exceptions	-

Table 3.5: Select a date use case

Use case no.	UC-06
Name	Select a custom date range of data to display
Description	If the type of the report is 'custom', two concrete days can be selected for the date range, with a maximum range of one month
Trigger	A user clicks the calendar icon in the view report screen
Actors	User
Preconditions	1. The user must be logged into the system 2. A report is being viewed
Flow	1. The user clicks on the calendar icon. 2. The system displays the corresponding date range picker. 3. The user selects the desired date range. 4. The system loads the report with the data series associated to the new date range.
Alternative flow	-
Postconditions	A report is shown, with the new data
Exceptions	-

Table 3.6: Select a date use case

Use case no.	UC-07
Name	Delete a report
Description	Delete a report stored in the database
Trigger	A user asks for the deletion of a report
Actors	User
Preconditions	A saved report is being viewed
Flow	1. The user clicks on the ”delete” button. 2. The system deletes the report from the database. 3. The screen is redirected to the list of templates and reports.
Alternative flow	-
Postconditions	A report is deleted from database
Exceptions	-

Table 3.7: Delete a report use case

Use case no.	UC-08
Name	Upload a device template with report templates associated
Description	The device template upload process must allow to include a report template list as a device template's property
Trigger	The user request the upload of a device template
Actors	Administrator
Preconditions	1. The user is logged into the manager web application. 2. The template has the correct structure.
Flow	1. The user clicks on the "add template" button. 2. The system displays a modal screen where the JSON object can be pasted. The object must have all the device template's specification. 3. The user pastes the object and confirms their action. 4. The system validates the input and inserts the template into the database.
Alternative flow	-
Postconditions	A device template and its children report templates are stored in database
Exceptions	-

Table 3.8: Upload device template with associated report templates use case

Chapter 4

Methodology and planning

Given the complexity of the elaboration of software process, it is standardized in Computer Engineering to use a clear and well known methodology to orchestrate the elements and team members necessary to develop the system to be achieved, in order to get the best quality and suitability to the necessities of the stakeholders of the product. In coherence with this, for this project a modification of Unified Software Development Process (USDP) methodology will be used, and in this chapter both the original methodology and the adaptation are described next. Also, the planning and estimation and costs are outlined.

4.1 Unified Software Development Process

Unified Software Development Process (USDP) is a methodology for the development of software. It is based in increments and prototypes, being iterative. This means that the work is divided into iterations that cross the different phases of the methodology, which result in prototypes, every time more elaborated and that include more functionality. It is also a risk and architecture focused methodology, which implies that in each iteration, the developers focus first in the most critical sections of the requirements, and in the way the software is arrayed to tackle these. It is strongly related to the Unified Modeling Language (UML), which is a specification for creating diagrams that help understand the software, documenting it. In USDP, UML is used in its different phases to graphically shape the system.

The phases that make up the unified process, and through which the developed product passes, are described below.

1. Inception

   In this phase, the developers focus on the task of defining the scope of the application, which architecture is going to be the most appropriate, and estimating the most risky parts of the software to be developed.

2. Elaboration

   In the elaboration phase, the team members of the development group mainly try to gather the requirements of the application. When this phase is done, it is considered that the development process reached a milestone: the capture of the use cases in a use case model, which is a typical artefact of the documentation in USDP and UML.

3. Construction

   This is the phase that envelops the implementation of the software. The objective is to end up with an executable system that fulfills the given requirements.

4. Transition

   During transition phase, the product is delivered and released to the stakeholders, perhaps refining it according to what is needed.

4.2 USDP applied to this project

Given the conditions and characteristics of the development of this final project, e.g. the fact that there will be only one developer, there is the need of adapting the methodology.

As it is usual in USDP, the different iterations will be centred in relevant features of the software, starting by those that suppose a greater risk for the realization of the objectives. In each iteration there will be analysis, design, implementation and tests, so each iteration includes more functionality about the previous one until the last one, that releases software with all the functionality. These iterations' workflow is described next:

• Analysis: description of necessary requirements in order to be able to build the specifications. As an incorporation to the methodology, we will execute the analysis of the new requirements if they exist after the previous iteration's tests.

• Design: conceptual creation of specifications according to the described previously requirements, ready to be concreted with the technology to be used.

• Implementation: translation of design into code, programming of the specified requirements.

• Tests: verification of the level of fulfilment of the specifications by the developed code.

• Documentation: elaboration of documents, models, user/technical manuals; or other artefacts that help understand the software and improve its usability and maintainability.

Also, we will focus more in adapting to changes in the requirements, and less in defining everything at the beginning, perhaps performing with this a more agile methodology. The main problem of USDP is that it doesn't contemplate any communication with the client. We will try to fix this by making user acceptance tests with them at the end of each iteration with

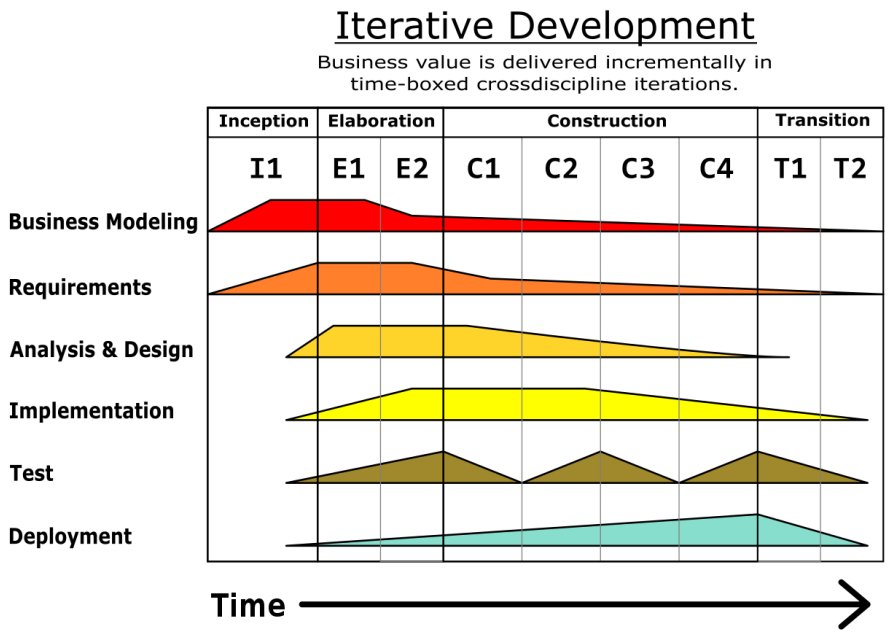

Figure 4.1: USDP workflows, extracted from [1]

the released prototype. At the beginning of each iteration, we will try to adapt to the results of these revisions.

The Unified Software Development Process focuses specially on generating artefacts for the documentation. It wants to determine everything so it is available the moment is needed by another developer during the implementation. This won't be necessary in this project, given the fact that there is only one developer. Instead, we will subscribe to the Agile Manifesto [26] on the point that states that documentation is not the goal of development, but that working software prevails over comprehensive documentation. Even though another developer in the future may have to expand the module, the main concern of the company is to understand the technical behaviour of this product, not how it has been done. Those are the main reasons that will make us dispense with deep or dense UML diagrams. We will keep for instance some class models, yet without as many diagrams as the tuple UML-USDP establishes.

However, we will describe now the different possible iterations the development will go through.

1. Iteration 1:

   In this iteration, we will focus on the most critical sections of the requirements: the hot-generation of reports, the retrieval of templates from the ones uploaded into the platform (integrating our module with the manager of templates, which is a web apart

from ours), and the viewing of a report detail, with the minimal functionality in the front end, so we can just check that the visualization works.

2. Iteration 2:

   This iteration will be focused on the functionality of saving reports with its widgets, modeling the necessary entities. Also, we will implement the retrieval of the saved reports list.

3. Iteration 3:

   In the last iteration, we will try to release a prototype with all the functionality: the previously described, plus the deletion of saved reports and the selection of date range for the data. For this last functionality we would have to adapt the component of each developed widget so it retrieves well-formed data according to the selected range of dates.

4.3 Planning

In this section, we will describe the planning of the development, an estimation of costs, and the follow-up.

4.3.1 Initial planning

In Figure 4.2, we can see the different iterations we planned to execute during this development. It is notorious that almost half of the work is done in the first iteration. This is in line with the philosophy of the methodology we apply: we take care of those functionalities that are critical to the system first, trying to develop a functional prototype since the first iteration, and at the end of every one.

The main time-consuming tasks are the ones that refer to the most risky requirements. That is, the ”Report generation from templates” of first iteration; the ”Visualization of hot-generated reports” (due to the CSS work that for sure will take) also from first iteration; the ”Saving reports, once hot-generated” task, from second iteration; and the ”Adaptation of widgets for data range selection”, from third and last iteration, since we would have to adapt every widget developed for the trial of the application.

As explained in Section 4.2, at the end of every iteration a revision and functional tests will be carried out, so we can adapt the development to the evolving necessities of the client.

There are two tasks named ”Integration with the platform”. For sure, and to carry out the non-functional requirement commented in Section 3.2, we will have to adapt our module to the system it belongs to. Not only in order to being able to upload report templates, but also

from the architecture point of view: the global platform where our module is encrusted has a well-defined architecture, that we must respect and extend.

4.3.2 Cost estimation

In this part, we will try to estimate how much the project will cost, taking into account the human resources, the software cost and the hardware cost.

1. Human cost

   In the planning, we have predicted a work of **40 days**. At **8 hours** per day, we have a total of **320 hours**. At a rate of **€17.5/hour**, we can predict that the human resources will suppose a cost of **€5.600**.

2. Software cost

   All the software and technologies used in this project are described in Chapter 2. Their licence is used for many projects, so we can assume that the cost of them is diluted and equal to **0€**.

3. Hardware cost

   The organization for which this software is being developed will give the student a laptop in order to carry the development out. The cost of the laptop is around **10000€**, but we will only use it for **320 hours**. If we assume that a laptop has a lifespan of **4 years**, using it **8 hours** per day, and a normal year has around **260 labour days**, we will spend a subtotal of

   $$€1000 \times (320h/(4 \times 260 \times 8h)) = €38.46 \tag{4.1}$$

   In Table 4.1 we can see the total cost estimation, derived from the previously mentioned concepts.

Resources	Cost
Human	€5600
Software	€0
Hardware	€38.46
Total	€5638.46

Table 4.1: Sum of costs

4.3.3 Follow-up

The organization for which this software is being developed has an internal tool for the follow-up of the hours spent by its workers. It allows the registration of hours worked on certain projects. We have access to this tool, and we will use it to register our work. Most of the days, from 5 to 7.5 hours were spent in the development. The result is the following:

In Figure 4.3, we have detailed the hours spent in this project during the month of February. As it shows, we took care first of the modelling of the template entities, and of the implementation of the upload of the templates, integrating ourselves with what already existed in the system. This means we had to update the 'manager' secondary system for the upload of templates and the auto-configuration of devices from the templates, including the report generation. Later, we put the focus on the visualization of those hot-generated reports.

In Figure 4.4, it is outlined the work carried out in March. Most of the time spent in this month was for the visualization of some widgets in the reports, the obtaining of data for them and the adaptation of the widgets to the data. We spent more time than estimated with this.

In Figure 4.5, it is outlined the work done in April. Already in the second iteration, this month was about saving hot-generated reports and some other widget visualization issues. Also, we spent some time generating technical documentation for the template creation. Along the way, there was a non-important hardware problem, and we had to replace the laptop.

In Figure 4.6, it is outlined the work during May and August. At the beginning of August, there was a big revision by the client, and we spent several days sorting things out. We took special care of the functionality of selecting a custom date for the data.

In Figure 4.7, we can see how the work during the month of September was almost exclusively dedicated to fix some bugs and details.

The total of hours worked in this project is **342.5 hours**. If we apply the salary of **17.5€** per hour, we obtain the total real human cost of this project, that is,

$$342.5h \times 17.5€/h = 5.993.75€ \tag{4.2}$$

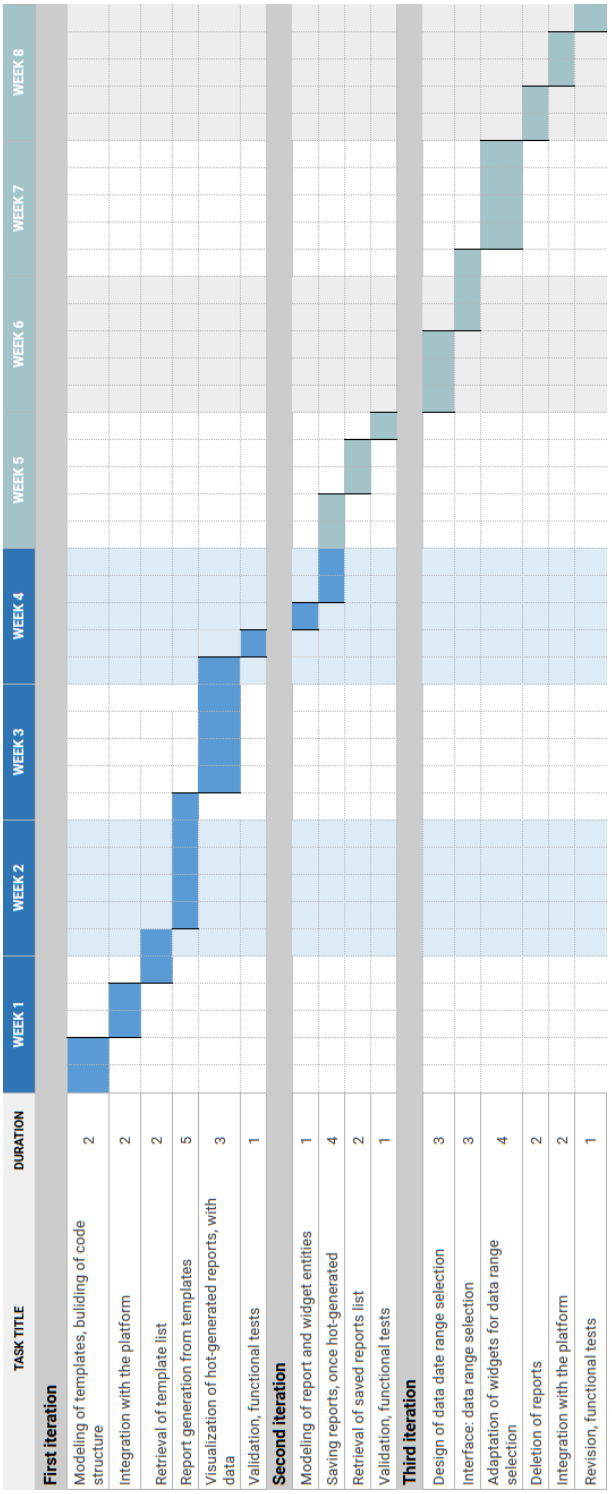

Figure 4.2: Gantt diagram for project's planning. All tasks are sequential

Fecha	Descripción	Horas
01/02/2022	Modelización de entidades	5,00
02/02/2022	Test de unidad device-template	5,00
03/02/2022	Test de unidad device-template	5,00
07/02/2022	Importación de plantillas	7,50
08/02/2022	Importación de plantillas	5,00
09/02/2022	Importación de plantillas	7,50
10/02/2022	Servicio autoconfiguración informes	5,00
14/02/2022	Servicio autoconfiguración informes	7,50
15/02/2022	Servicio autoconfiguración informes	5,00
16/02/2022	Adaptación manager	7,50
17/02/2022	Importación de plantillas	5,00
22/02/2022	Importación de plantillas	5,00
23/02/2022	Generación vista informes	7,50
24/02/2022	Generación vista informes	5,00

Figure 4.3: Follow-up of February

Fecha	Descripción	Horas
02/03/2022	Visualización de widgets de informes	7,50
03/03/2022	Visualización de widgets de informes	5,00
07/03/2022	Formato vista informe	7,50
08/03/2022	Formato vista informe	5,00
09/03/2022	Obtención de datos para informe por mes	7,50
10/03/2022	Obtención de datos para informe por mes	5,00
14/03/2022	Obtención de datos para informe por mes	7,50
15/03/2022	Widget de tabla de calor para informe	5,00
16/03/2022	Widget de tabla de calor para informe	7,50
17/03/2022	Widget de tabla de calor para informe	5,00
24/03/2022	Plantillas para informes	5,00
28/03/2022	Plantillas para informes	7,50
29/03/2022	Plantillas para informes	5,00
30/03/2022	Plantillas para informes	7,50
31/03/2022	Plantillas para informes	5,00

Figure 4.4: Follow-up of March

Fecha	Descripción	Horas
06/04/2022	Plantillas para informes	7,50
07/04/2022	Documentación para informes	5,00
18/04/2022	Diseño para funcionalidad de guardar informe	7,50
19/04/2022	Configurar nuevo SO	5,00
20/04/2022	Terminar de configurar SO y paginación de informes	7,50
21/04/2022	Plantillas para informes	5,00
25/04/2022	Plantillas para informes, gráfico para días laborales vs festivos y funcionalidad de guardar informe	7,50
26/04/2022	Plantillas para informes, gráfico para días laborales vs festivos	5,00
28/04/2022	Funcionalidad de guardar informe	5,00

Figure 4.5: Follow-up of April

Fecha	Descripción	Horas
02/05/2022	Funcionalidad de guardar informe	4,00
02/05/2022	Plantillas para informes, tamaños el informe	3,50
08/08/2022	Revisión informes	5,00
09/08/2022	Revisión informes	5,00
10/08/2022	Revisión informes:adecuar el datePicker a las fechas de las series del dispositivo elegido	2,50
10/08/2022	Revisión informes: arreglar lista de informes y plantillas cuando está vacía	2,50
11/08/2022	Revisión informes:adecuar el datePicker a las fechas de las series del dispositivo elegido	1,00
11/08/2022	visualización de varias plantillas de informe por cada plantilla de dispositivo	2,00
11/08/2022	Feedback de guardado de informe	2,00
12/08/2022	Corrección del orden de eliminación de plantillas de widgets	2,00
12/08/2022	Muestra de número de plantillas de informe en la web del manager	3,00
16/08/2022	Documentación de widgets	5,00
17/08/2022	Documentación de widgets	3,00
17/08/2022	Diseño informes semanales	2,00
18/08/2022	Limpieza de código	2,00
18/08/2022	Fix: generación de informes cuando la plantilla de dispositivo tiene mas de una plantilla de informe	3,00
19/08/2022	Endpoint para generación singular de informe	5,00
22/08/2022	Bug en generación de informe	5,00
23/08/2022	Diseño informes semanales	3,00
23/08/2022	Implementación informes semanales	2,00
24/08/2022	Implementación informes semanales	5,00
25/08/2022	Implementación informes semanales	5,00
26/08/2022	Implementación informes semanales	5,00
29/08/2022	Adaptación del front a informes semanales	5,00
30/08/2022	Resolucion de bugs en informes semanales	5,00

Figure 4.6: Follow-up of May and August

Fecha	Descripción	Horas
01/09/2022	Resolución de de bug en obtención de plantillas en la generación de informes	5,00
02/09/2022	Arreglo de bugs en heat map widget	5,00
05/09/2022	Refactor de la construcción de los ejes de heat map	5,00
06/09/2022	Debuggear muestra de widgets en informes	5,00
07/09/2022	Alineación horizontal widgets	5,00

Figure 4.7: Follow-up of September

Chapter 5

Architecture and design

In this project, a combination of different architectures and design patterns will be used at different levels, which are exposed and explained in the next sections of this chapter.

5.1 Back-end structure

In order to maintain an order for the construction of the code, and to enhance the legibility of its design, an architecture in layers will be used to program the functionality of the back end of the application, as represented in Figure 5.1, with three basic layers:

- Model: entity modelling and their persistence
- Service: where business logic is implemented
- Controller: a Representational state transfer (REST) Application Programming Interface (API) that gives access to these methods

5.1.1 Model

Given the fact that we are working with Java, an object oriented programming language in which basically everything is a class, we will build objects whose behaviour and methods respond to our necessities. Apart from the objects representing the stored report and the different widgets for those reports, a Data Access Object (DAO) pattern will be in charge of implementing the Java Database Connectivity (JDBC) (using the Hibernate EntityManager implementation for Create, Read, Update, and Delete (CRUD) operations), creating custom methods with custom queries for consumption of the database. The code corresponding to the pure and basic management of JDBC is already provided by the company for which the software of this final work is developed, despite the CRUD operations and the EntityManager implementation, which are provided by Hibernate. So, we would only need to implement the concrete methods and queries of the DAOs related to the concrete operations of our module.

Figure 5.1: Code structure in project

5.1.2 Service

The data collected from the database will be transformed into Java objects by the Entity Manager, and used by the Service layer in order to combine objects from different DAOs and manage the queries received at Controller layer.

Apart from the presented in Figure 5.1, the platform that includes the module ”reportV2” has other modules associated that serve the web application with different functionalities. In order to fabricate and persist the reports and widgets with certain attributes related to the devices, we have to store first their templates. The platform holds a module that is able to store templates, with general properties and configurations, which are used for multiple purposes. We would have to create in this module the report template class and the widget template class. Apart from this, a template for the header of the report, and another for the footer, will be implemented, so these can be also configured with, for example, a path or an Uniform Resource Locator to an image.

All of this is embedded in a previously existing module, called ”device-template”. Until now, the platform holds a functionality of auto-configure a device's information from a device template, generating automatically different instantaneous dashboards, tracing graphics, and so on. This is made from many other types of templates (always associated to a major device template that includes them, even our report templates), already integrated within the system, that a manager can upload from another platform. The different dashboards templates are read and dashboards instances are created, dynamically when the user presses a button. Our report templates must be read and reports must be created too when this functionality is triggered.

We need to depend on the ”device-template” module in order to generate a widget from a template and persist it in the database. But it also is needed to create reports from the templates in the ”device-template” module. Given this conflict (a cyclic dependency), an observer pattern is needed to trigger the methods of hot generation of reports from report templates when an auto-configuration of the whole device template is requested. In Figure 5.2, we can see a diagram explaining this pattern.

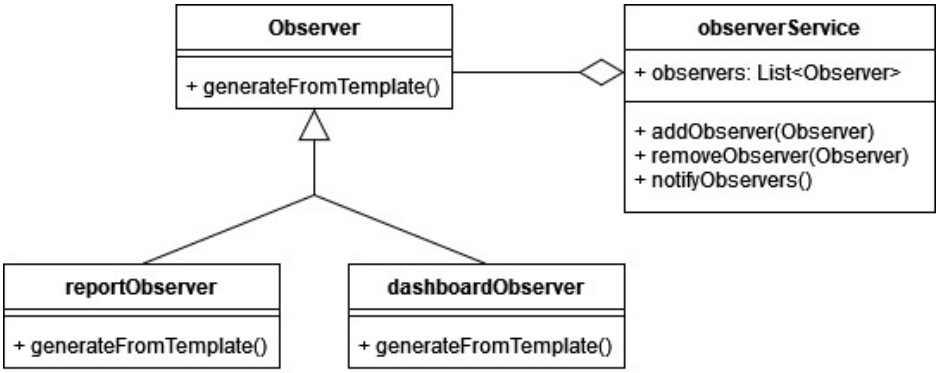

Figure 5.2: Observer pattern

To sum up, we need to implement an observer that generates the reports that the report templates have associated to a device when an auto-configuration is triggered. This pattern helps us avoid cyclic dependencies with the ”device-template” module.

Also, an strategy pattern will be implemented: we will choose which widget generation service to invoke depending on the type of widget indicated in the widget templates when processing each report template.

5.1.3 Controller

This controller layer is contacted by the user and integrates the web API of our module for them. If an entity needs to be consumed by the front end, we will implement the Data Transfer Object (DTO) pattern, modeling objects that contain the necessary attributes for the web part, and transforming our plain Java objects into them when it is required.

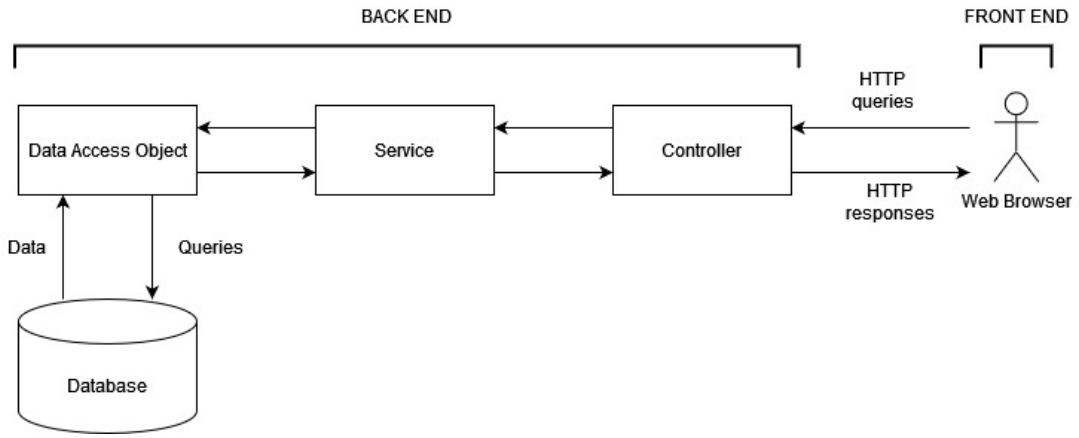

Figure 5.3: Architecture in layers for back end

5.1.4 Database design

In this subsection, the entity models of both the ”device-template” module and the ”reportV2” module shall be attached and explained.

As shown in Figure 5.4, previously existing DeviceTemplate now has a list of ReportTemplate associated; which has a HeaderReportTemplate (the template for the header of the report, actually only saving information of the path of the image to display and possibly, a list of ElementTupleConfigTemplate as configurations), a FooterReportTemplate (ditto for the footer) and a set of ReportWidgetTemplate. This ReportWidgetTemplate class would have a set of ElementTupleConfigTemplate and a set of ElementPropertyTemplate, which associates the template of widget with concrete property of a device to be read. The final Entity-Relationship (ER) model is pictured in Figure 5.6.

As it is outlined in Figure 5.5, a Report instance has a list of WidgetReport superclass instances associated. This superclass is concreted by any certain type of widget. For this project, three subclasses have been developed: TextWidgetReport (a simple widget that contains a usually short text message with a concrete color and size), GraphWidgetReport (a widget in charge of displaying a single or multiple graph, with associated GraphSimpleReport classes for the association with the series to be displayed) and TempTableWidgetReport (which is for the display of a heat map with a single TableWidgetReportSeries associated).

5.2 Front-end structure

In this section, it is explained the Model-View-Presenter pattern used to work with the back-end,

5.2.1 Model-View-Presenter

The combination of database, back-end and front-end that this application describes is encrusted into a well-known pattern for web applications: the model-view-presenter pattern (that Figure 5.7 outlines). Generally, when the view asks for information, the presenter takes it from the model, processes it and gives it back to the view, which shows it to the user. No direct interaction exists between view and model without passing through the presenter. Next, it will be explained the different parts of this pattern.

- Model: the model represents the information layer of the system. Conceptually, it is the part that is in charge of persistence and saving information between sessions (so we can for example read stored reports days apart). Everything that has to do with managing data is done by the model.
- View: the view is in charge of showing the data to the user. It is everything that has to do with an interface and with the interaction with the final user.
- Presenter: the presenter is the part of our application that is in charge of receiving user requests from the view, collecting the data from the model, processing it and pass it to the view in an appropriate format. In our case of work, the presenter is the back end of the application, since it manages the user's requests and accesses the model in retrieval of data.

5.2.2 Interface code structure

Since we are using Angular, the structure of our code will conform to this development framework. As pictured in Figure 5.8, the different modules of the web application are spread across the 'modules' folder, in a folder tree with implementations of different functionalities. The 'reports' module is the subject of this project. We will follow the convention for structuring Angular projects. More information about the workspace structure of Angular can be found in [27].

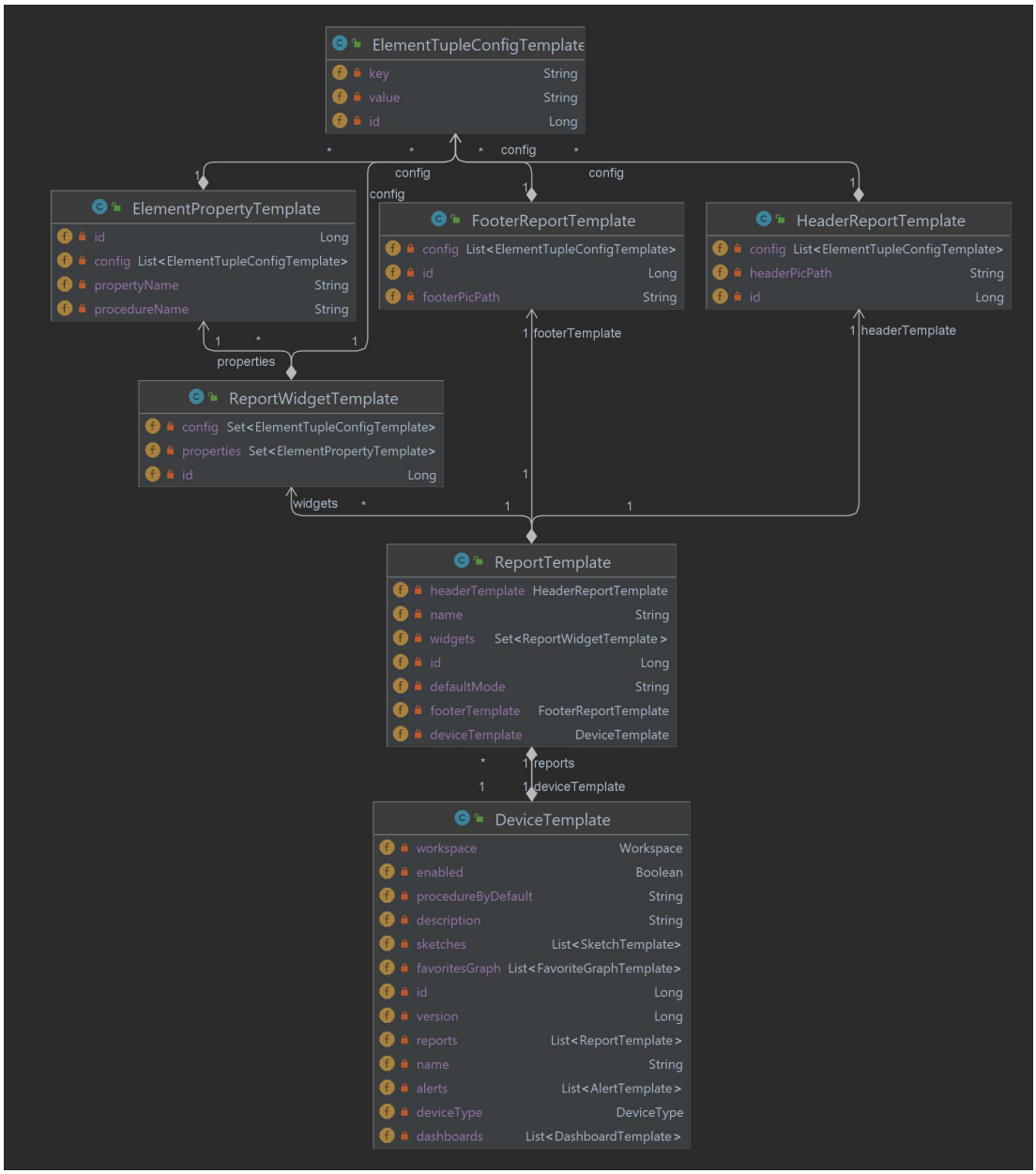

Figure 5.4: Report template entity model, class diagram

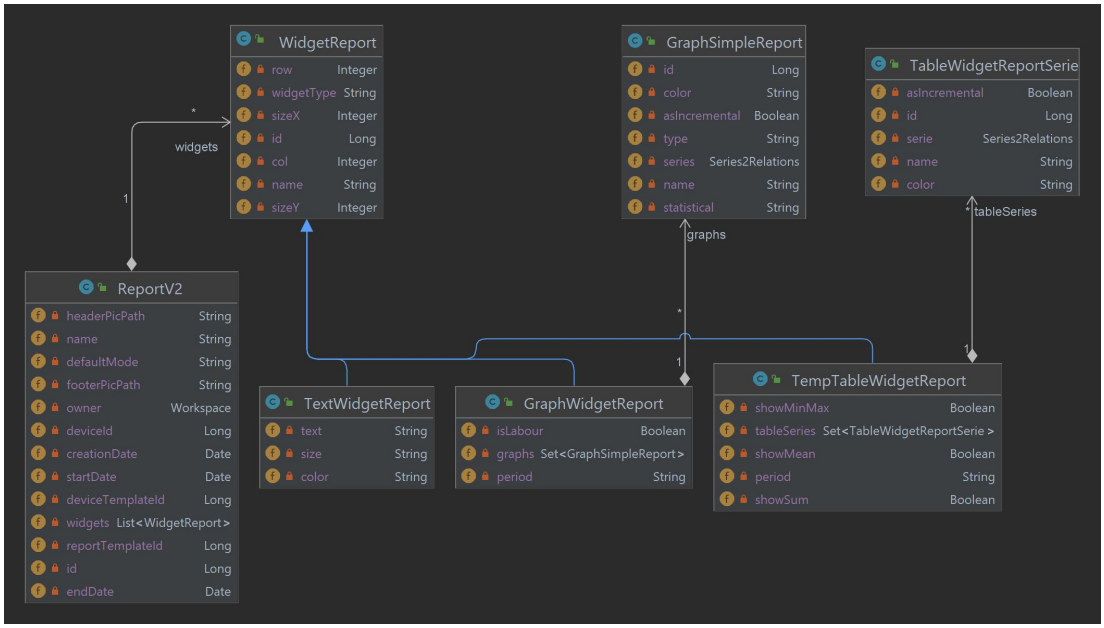

Figure 5.5: Report entity model, class diagram

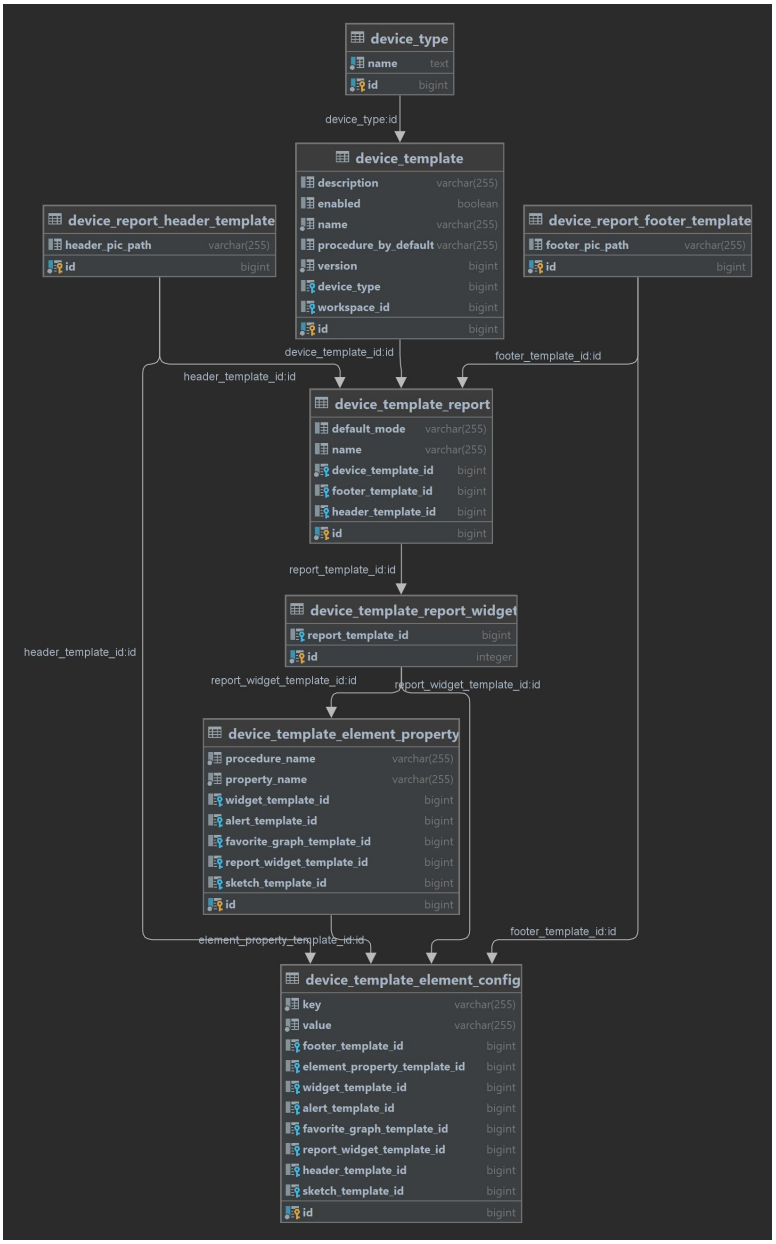

Figure 5.6: Report and device template ER diagram

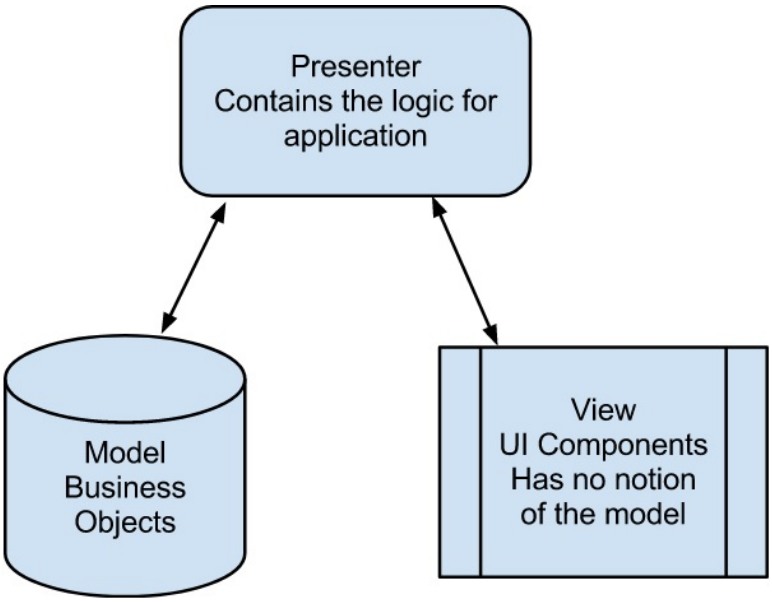

Figure 5.7: Model-view-presenter pattern

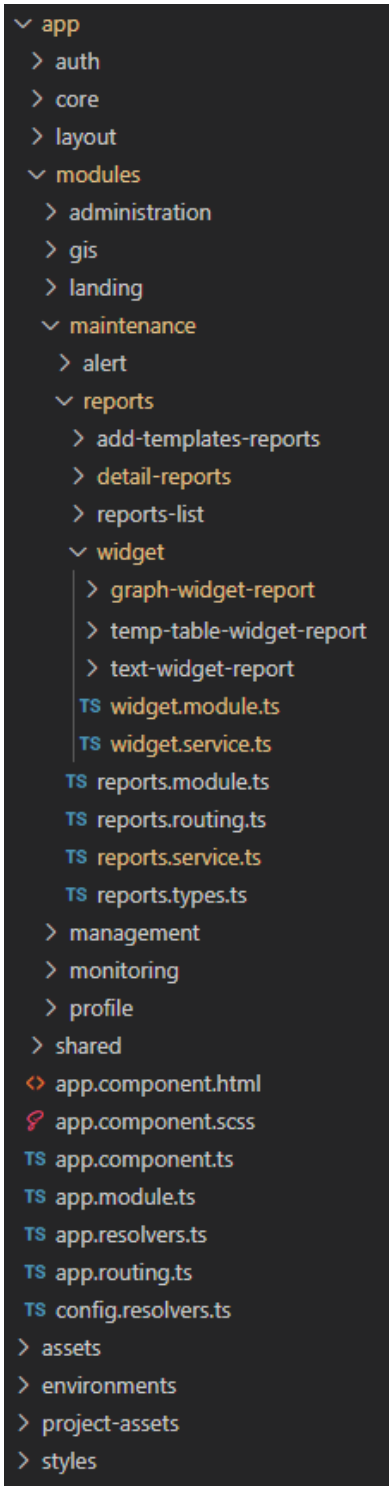

Figure 5.8: Front-end code structure

Chapter 6
Implementation

In this chapter, we will pass across different sections explaining how the different parts of the application have been implemented. We will explain the use of different technologies (Chapter 2) and patterns (Section 5.1 and Section 5.2) explained previously.

6.1 Back end implementation

In this section we will outline the implementation of the back end of the application. That is, the model, the service and the controller layers explained in Section 5.1

6.1.1 Database objects implementation

For the implementation of the classes shown in Figure 5.4 and in Figure 5.5 described in Section 5.1.4 of previous chapter, we will use Hibernate annotations for mapping and declaring relations between these objects. The annotations finally used are the following:

- **@Id [28]:** This annotation indicates that the concrete property is a primary key for the table associated to the object.
- **@GeneratedValue [29]:** This is used so a method for the primary key generation is specified.
- **@Column [30]:** This annotation is used to define concrete aspects about the column associated to the concrete attribute of the object. We use it to define the name of the column, the type of the field, if it is a nullable value, and if it is an unique value.
- **@ManyToOne [31]:** This one is used so a N:1 relation is declared, being the class on which the property is defined the N part, and the class of the property the 1 part. We use it also to define the fetch policy for when we have to call the related object (for most of our cases, eager fetch type is declared).

- **@Temporal [32]:** In case we have an attribute that is a temporal type, such a Date, with this annotation we can specify which SQL type to use in the column that represents the attribute.
- **@OneToMany [33]:** This one is used so a 1:N relation is declared, being the class on which the property is defined the 1 part, and the class of the property the N part. We use it also to define the fetch policy for when we have to call the related object (for most of our cases, eager fetch type is declared), to define the cascade and orphanRemoval policies (what to do with the foreign keys when an object is removed).
- **@JoinColumn [34]:** Same as '@Column', but with options to describe better the properties of the column that is going to be a foreign key. We can for instance specify the name of the foreign key column.
- **@Transient [35]:** This annotation is used to declare that a property is not persistent in database.

6.1.2 Service implementation

For the service implementation, we have to implement the business logic, that means, the true functionality of our system. We have to grant report generation from template, widget generation from template, saving of entities once generated, and retrieval of entities so saved reports can be viewed in the interface.

- **Report generation:** For this job to be done, a ReportTemplate is retrieved and its attributes read so a ReportV2 instance is generated to later be transformed into a DTO for the interface. One of the attributes read in the ReportTemplate is the list of WidgetTemplate. When this happens, and as we can see in Figure 6.1, we use a map of GenericWidgetReportServiceImpl (stored in WidgerReportServiceImpl) instances concreted by subclasses that extend it, implementing different manners of instantiate a widget from its template depending on the widget type. This is the implementation of the strategy pattern commented in Section 5.1.2. The method that implements it is shown in Figure 6.2. The different widget report service implementations are added to the map of WidgetReportServiceImpl when the application starts.
- **Saving a report:** When the user requests to save a report instance, an algorithm similar to the one used for report generation is invoked, with the exception that the report generated which was going to be transformed into a DTO is also persisted in database.
- **Delete a report:** The controller request for the deletion of a stored report is issued by the service with a simple call to the CRUD method of the model.

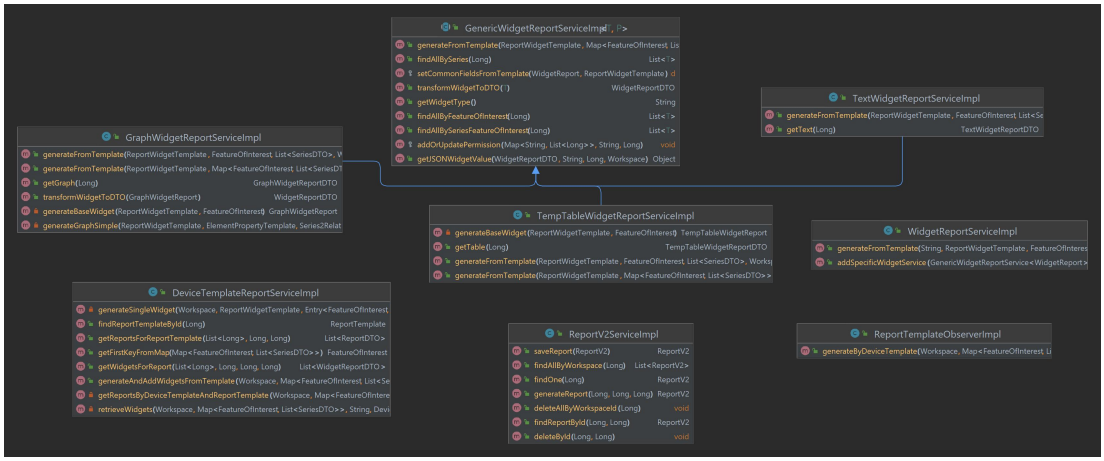

Figure 6.1: Services diagram for ReportV2 module

```
1 usage   Juan Ramil Díaz *
@Override
public WidgetReport generateFromTemplate(String widgetType, ReportWidgetTemplate widgetTemplate,
                                         FeatureOfInterest device, List<SeriesDTO> series,
                                         Workspace workspace) throws ServiceException {
    return this.widgetServiceMap.get(widgetType)
            .generateFromTemplate(widgetTemplate, device, series, workspace);
}
```

Figure 6.2: Code implementation of the strategy pattern

Following the expandability principle, concrete extensions of the GenericWidgetReportServiceImpl can be implemented. For the testing of the module, three widgets have been created:

- **TextWidgetReport:** the simplest widget made in this implementation, since it will only show some text with two properties: a size for the text font and a colour for the letters. The code for the generation from template of this widget is shown in Figure 6.3.
- **TempTableWidgetReport:** this widget is in charge of displaying a heat map. It reads series' names from the template and looks for them in the device's stored series. If there are coincidences, the service generates TableWidgetSeries objects, that relate the widget with concrete series of data and stores their properties read from the template. The code for the generation from template of this widget is shown in Figure 6.4. The generateBaseWidget method reads the configurations for the widget from the template.
- **GraphWidgetReport:** this widget is ready for the display of one or several series per line of a lines graph. The code for the generation from template of this widget is

```
1 usage  Juan Ramil Díaz +1
@Override
public WidgetReport generateFromTemplate(ReportWidgetTemplate widgetTemplate,
        FeatureOfInterest device,
        List<SeriesDTO> series, Workspace workspace) {
    TextWidgetReport textWidget = new TextWidgetReport();
    textWidget.setColor(DeviceTemplateUtils.getValueFromKey(COLOR_CONSTANT_WIDGET_FIELDS,
            new ArrayList<>(widgetTemplate.getConfig())));
    String text = DeviceTemplateUtils.getValueFromKey( key: "text", new ArrayList<>(widgetTemplate.getConfig()));
    textWidget.setText(DeviceTemplateUtils.parseNameWithDeviceName(text, device, text));
    textWidget.setSize(DeviceTemplateUtils.getValueFromKey( key: "size", new ArrayList<>(widgetTemplate.getConfig())));
    this.setCommonFieldsFromTemplate(textWidget, widgetTemplate);
    return textWidget;
}
```

Figure 6.3: Code of the generation of a TextWidget from a WidgetReportTemplate

very similar to the TempTableWidget generation code, generating GraphSimpleReport instances for the series associations, with the exception that there are different configurations that are read from the WidgetReportTemplate. For instance, a configuration called 'isLabour' can be set to true, and then the front end will process the data in a different manner, showing the mean of the data values per hour and differentiating labour days and holidays. We talk more profoundly about this in Section 6.2

6.1.3 Controllers

In order to build a REST API for the application that the web part can consume, we have to create different endpoints or access points in the application server, so a request is received and a response is returned. The following list is a collection of the two controllers with concrete sub paths and Hypertext Transfer Protocol (HTTP) methods that they can receive, and an explanation of the endpoint.

- **ReportController:** As shown in Figure 6.5, this controller has four endpoints: one for getting all the reports available in database (**/report/V2 [GET]**); another for retrieving a concrete report from database by its id (**/report/V2/detail/id [GET]**); another for generating a report from a concrete ReportTemplate id and a device id, and storing it in database (**/report/V2/generate/reportTemplateId/device/deviceId [GET]**); and a last one for the deletion of a ReportV2 from database, with its id (**/report/V2/id [DELETE]**).
- **DeviceTemplateReportController:** This controller builds the endpoints related to the hot-generation of reports. As shown in Figure 6.6, it has only one endpoint: the one that demands the hot-generation of a report from its template.

```
@Override
public WidgetReport generateFromTemplate(ReportWidgetTemplate widgetTemplate,
                                         FeatureOfInterest device,
                                         List<SeriesDTO> series, Workspace workspace) {
    TempTableWidgetReport tempTableWidget = generateBaseWidget(widgetTemplate, device);
    for (ElementPropertyTemplate property : widgetTemplate.getProperties()) {
        try {
            for (Series2Relations series2Relations : deviceTemplateService.getSeriesAssociated(
                    property, series)) {
                TableWidgetReportSerie tws = new TableWidgetReportSerie();
                tws.setAsIncremental(Boolean.parseBoolean(DeviceTemplateUtils.getValueFromKey(
                        ASINCREMENTAL_CONSTANT_WIDGET_FIELDS, property.getConfig())));
                tws.setColor(DeviceTemplateUtils.getValueFromKey(COLOR_CONSTANT_WIDGET_FIELDS,
                        property.getConfig()));
                tws.setName(DeviceTemplateUtils.getValueFromKey(NAME_CONSTANT_WIDGET_FIELDS,
                        property.getConfig()));
                tws.setSerie(series2Relations);
                tempTableWidget.getTableSeries().add(tws);
            }
        } catch (ServiceException ignore) {
            // Ignore if not series exists
        }
    }
    return tempTableWidget.getTableSeries().isEmpty() ? null : tempTableWidget;
```

Figure 6.4: Code of the generation of a TempTableWidget from a WidgetReportTemplate

6.2 Front end implementation

In this section we will explain the implementation of the component structure. Along the way we will look at other aspects of the implementation.

6.2.1 Component structure

As explained in Section 2.1.7, Angular is a component-based framework for SPAs. This means that the web interface development will be tackled by reusable parts, dividing as much as it is possible and adequate the view in components, and defining their behaviour by encapsulating it. More information can be found in [36].

However, we will tackle the necessities of our project's interface with several components, which are described in Figure 6.7 and next:

- **reports-list:** This component will be in charge of listing the different templates that the user will be able to see, and the reports stored in database. A report or a template can be clicked so the user is redirected to the "detail-reports" component. In Figure 6.8 we can see how these redirections work, with the concrete path and its arguments and the component that is loaded.

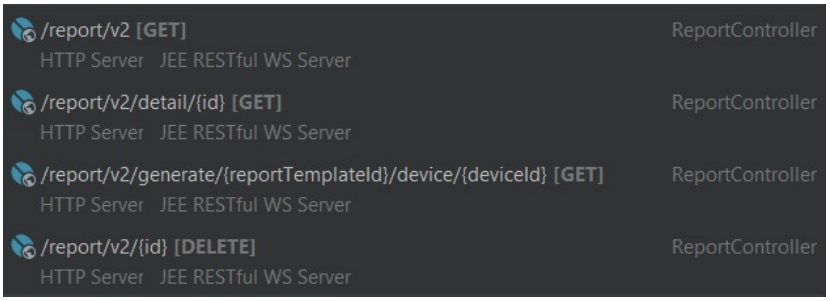

Figure 6.5: Report controller

Figure 6.6: Device Template Report controller

As usual for Angular, we have an HTML template and a component definition with a Typescript class. We go without the .css file or similar because we are using Tailwind, as described in 2.1.12. In fact, this will be like this for all the components we use except for the ”detail-reports” component, which has a .scss file.

- **graph-widget-report:** this is an example of a component for a specific type of widget. It processes the data so it is shown in one or several graph lines [37]. The properties associated to each line can be customized from the widget template in the aspects of colour and name, apart from the association with a concrete property and procedure from the device we are maintaining in the report.

- **temp-table-widget-report:** this is another example of a component for a specific type of widget. It processes the data so it is shown in a heat map [38]. The property associated to the widget will be read and its data shown in the color specified, so a temperature table is depicted with the color of each position visually indicating its value.

- **detail-reports:** This component is in charge of showing a report, whether it is hot-generated or retrieved from database. We have a HTML template, a Typescript definition of the component’s class and .scss file with the specific CSS for the representation of the page of the report (Figure 6.9). We use specific CSS for this component because Tailwind is not enough nor prepared for the depiction of DIN A4 pages, which is what we want for our reports.

  When the screen is redirected to this component from the ”reports-list” component, if a saved report is clicked, we will use the second redirection in Figure 6.8. If a template is clicked, and a device selected in the modal screen that it shows, then the third redirection is carried out. Once in the ”detail-reports” component, we will define one of

that component's properties called 'reportType' as "REPORT", or "TEMPLATE". This will define some details in the view, e.g. the depiction of a button to delete the saved report or to save the hot-generated one, as depicted in the use cases (Section 3.3.2).

For the implementation of the date range selection for the data, a date picker component is attached on this one, as implemented in Figure 6.10. For this kind of components, we will use the library of components called Material as described in Section 2.1.8. It is designed by Google and it can be implemented on mobile platforms and on the web. It is well documented and widely used. We will make use of this library to implement this date picker, but also for buttons and other reusable elements.

Once the date range is selected, the widget list is updated with the new start and end dates. Then, each widget component calls a reports service method that makes a request of concrete data about concrete series to the SOS platform. Finally, every widget is updated with the new data and shows it in its own way.

From this component view, we will also make available the deletion of a report, if it is retrieved from database, and the saving of a report, if it is hot-generated.

In order to list the different widgets that are related to the report, we will implement also a web library for Angular called "angular-gridster2". It allows to draw a grid with different elements. These elements must implement some attributes, as their position, width and height, so they are positioned correctly in the grid. As shown in Figure 6.11, we loop over the widget list and filter by type, then instantiate the concrete widget component passing the widget object as input.

6.3 Adding report templates to manager system

The organization has a secondary platform called 'manager', which is in charge of, among other things, displaying a web application that allows the upload of device templates. In order to fuse with the existing implementation, and to allow the upload of report templates, this process needs to change so this new kind of template is allowed in the system.

To the beginning of this project, MyBatis is used together with Spring in the manager application to elaborate the custom queries of the application that consumes the database. An XML template is created with the custom queries and arguments for the methods declared in a related interface located in the structure of the project. A chunk of this template can be inspected in Figure 6.12. Also, a part of the interface that declares the methods to be related with the queries is shown in Figure 6.13

The insertion of HeaderReportTemplate and FooterReportTemplate is made in the same way. For the deletion of the templates, a similar strategy is carried out, with the exception that

we have to modify the existing query for the device template deletion, making sure that all the template objects related to a report template (widget report, header and footer templates, together with their configurations and/or properties) are deleted in the right order. Else, we could leave orphan rows in the database, or violate some foreign key constraint.

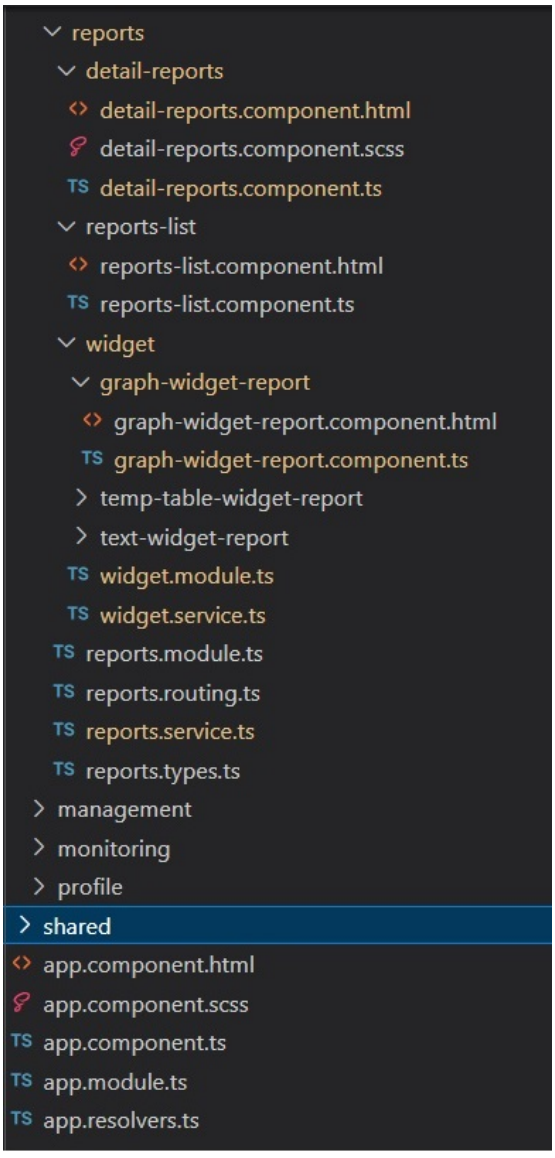

Figure 6.7: Components depiction for 'reports' module

```
export const reportsRoutes: Route[] = [
    {
        path: '',
        component: ReportsListComponent,
        canActivate: [AuthGuard]
    },
    {
        path: ':id',
        component: DetailReportsComponent,
        canActivate: [AuthGuard]
    },
    {
        path: 'template/:templateId/device/:deviceId',
        component: DetailReportsComponent,
        canActivate: [AuthGuard]
    },
];
```

Figure 6.8: Reports module routes

```css
body {
    margin: 0;
    padding: 0;
    background-color: #FAFAFA;
    font: 12pt "Tahoma";
}
.page {
    width: 210mm;
    min-height: 297mm;
    max-height: 297mm;
    height: 297mm;
    padding: 20mm;
    margin: auto;
    border: 1px #D3D3D3 solid;
    background: white;
    box-shadow: 0 0 5px rgba(0, 0, 0, 0.1);
}
.subpage {
    border: 1px rgb(255, 255, 255) solid;
    min-height: 257mm;
    max-height: 257mm;
    height: 257mm;
    outline: 20mm #d3d3d3;
}
```

Figure 6.9: Report detail .css file

```html
<div>
  <div class="flex items-center" *ngIf="reportDefaultMode==='MONTH'">
    <input matInput [matDatepicker]="dp" [min]="minDate" [max]="maxDate" />
    <mat-datepicker-toggle matSuffix [for]="dp"></mat-datepicker-toggle>
    <mat-datepicker #dp startView="multi-year" (monthSelected)="chosenMonthHandler($event, dp)">
    </mat-datepicker>
  </div>
  <div class="flex items-center" *ngIf="reportDefaultMode==='CUSTOM'">
    <mat-date-range-input #dpi [formGroup]="range" [rangePicker]="dp" [min]="minDate" [max]="maxDate">
      <input matStartDate class="hidden" placeholder="Start date" formControlName="start"
        (dateChange)="changeEvent($event, dp)">
      <input matEndDate class="hidden" placeholder="End date" formControlName="end"
        (dateChange)="chosenDateHandlerCustom()">
    </mat-date-range-input>
    <mat-datepicker-toggle matSuffix [for]="dp"></mat-datepicker-toggle>
    <mat-date-range-picker #dp [datepickerInput]="dpi" startView="multi-year">
    </mat-date-range-picker>
  </div>
</div>
```

Figure 6.10: Material date picker implementation

```
<div style="max-height: calc(100% - 115px)">
  <gridster class="gridster" [options]="gridsterOptions" *ngIf="gridsterOptions !== undefined">
    <gridster-item *ngFor="let widget of this.widgetList" [item]="widget">
      <div [ngSwitch]="widget.widgetType" class="h-full">
        <div *ngSwitchCase="'GRAPH'" class="h-full">
          <graph-widget-report class="h-full" [widget]="widget">
          </graph-widget-report>
        </div>
        <div *ngSwitchCase="'TEMP_TABLE'" class="h-full">
          <temp-table-widget-report class="h-full" [widget]="widget">
          </temp-table-widget-report>
        </div>
        <div *ngSwitchCase="'TABLE'" class="h-full">
          <table-widget-report class="h-full" [widget]="widget">
          </table-widget-report>
        </div>
        <div *ngSwitchCase="'TEXT'" class="h-full">
          <text-widget-report class="h-full" [widget]="widget">
          </text-widget-report>
        </div>

        <div *ngSwitchDefault class="h-full">
          <div class="flex flex-col w-full h-full justify-center">
            <div class="flex flex-row p-5 text-4xl justify-center font-semibold text-center">
              {{ 'MAINTENANCE.REPORTS.WIDGET_NOT_SUPPORTED' | transloco }}
            </div>
          </div>
        </div>
      </div>
    </gridster-item>

    <no-elements class="self-auto text-black" icon="heroicons_outline:chart-pie"
      message="{{ 'MAINTENANCE.REPORTS.NO_DATA' | transloco }}" *ngIf="widgetList?.length === 0"
      hideClickHere="true">
    </no-elements>
  </gridster>
</div>
```

Figure 6.11: Use of gridster component

```
<insert id="insertReportDeviceTemplate" flushCache="true" useGeneratedKeys="true"
        keyProperty="reportTemplate.id"
        keyColumn="id">
    INSERT INTO firefly.device_template_report (name, default_mode, device_template_id,
    VALUES ('${reportTemplate.name}', '${reportTemplate.defaultMode}', ${deviceTemplateI
            ${headerTemplateId})
</insert>
<insert id="insertReportWidgetReportTemplate" flushCache="true" useGeneratedKeys="true"
        keyProperty="reportWidgetTemplate.id"
        keyColumn="id">
    INSERT INTO firefly.device_template_report_widget (report_template_id)
    VALUES (${reportTemplateId})
</insert>
```

Figure 6.12: XML template with the custom queries

```
1 usage  Juan Ramil Díaz +1
void insertReportDeviceTemplate(@Param("reportTemplate") ReportTemplate reportTemplate,
                                @Param("deviceName") String name,
                                String defaultMode, @Param("footerTemplateId") Long footerTemplateId,
                                @Param("headerTemplateId") Long headerTemplateId,
                                @Param("deviceTemplateId") Long deviceTemplateId);

1 usage  Juan Ramil Díaz
void insertReportWidgetReportTemplate(
        @Param("reportWidgetTemplate") ReportWidgetTemplate reportWidgetTemplate,
        @Param("reportTemplateId") Long reportTemplateId);
```

Figure 6.13: Java interface with the methods to be mapped by MyBatis

Chapter 7

Tests

In this chapter it will be explained the testing philosophy of this project and the different functional user acceptance tests that have been performed.

7.1 Testing philosophy

Our business logic in the back-end can be summarized as follows: transforming report templates into report instances, and widget templates into widget instances, being these last ones associated to a report. Due to this relative simplicity (taking one object's associated properties and link them to another, by parsing the first ones), unit functional tests won't be that valuable, since there are no complex transformations or algorithms that could fail. Furthermore, we have to take into account the fact that the integration with the system is done by creating a new module, and that the templates are uploaded from a different application. These facts make it very difficult to test in an automatic manner the upload and transformation of the templates, so automatic integration tests at this level are also discarded.

Where we could implement integration testing is in the conjunction between back-end and front-end, and between the report generation application and the manager application (the one in charge of uploading the templates), at a manual level. We will detail the tests made like this in the next section.

7.2 Tests

In this section we will outline and describe some tests cases carried out by the developer. The following are functional integration tests.

7.2.1 Functional integration tests

In this section we will indicate the different main functional integration test cases that were carried out during the implementation.

1. **TC-01. A well generated device template with the ”reports” array property can be uploaded to the database:**

   After updating the manager application so it is able to add an array of report templates to the device templates that are uploaded, we have to test this use case. Table 7.1 summarizes the test case.

2. **TC-02. The reports application shows a list of uploaded templates:**

   Once the templates are uploaded, we must be able to see them in the application. Table 7.2 shows the test case in detail.

3. **TC-03. The application shows a list of uploaded report templates:**

   With the previous test case result, we concluded that we needed to show the report templates, not the device templates in the templates list. Table 7.3 shows more information about this test case.

4. **TC-04. The application successfully hot-generates a report with the correct widgets:**

   The system must successfully hot-generate a report, having parsed correctly the widget templates too. More information is available in Table 7.4.

5. **TC-05. The application successfully saves a hot-generated report:**

   We must check that the saved report (if there is any after the activation of the functionality) is correct and has the correct attributes, and the same for the widgets associated. Table 7.5 corresponds to this test case.

6. **TC-06. The widgets show the correct data:** If this test (Table 7.6) passes or not can be defined with other functionalities of the system that let the user check at a low level each piece of data that can be retrieved from a device.

7. **TC-07. The widgets show the correct data, according to the dates selected:** If this test (Table 7.7) passes or not can be again defined with other functionalities of the system for the interpretation of data at low level.

Test case no.	TC-01
Name	A well generated device template with the ”reports” array property can be uploaded to the database
Preconditions	1. The user is logged into the manager web application 2. The template has the correct structure
Test steps	1. The user clicks on the ”add template”. button 2. The system displays a modal screen where the JSON object can be pasted. 3. The user pastes the object and confirms their action.
Expected result	The system must validate the input and insert the template into the database
Actual results	The system validates the input and inserts the template into the database
Pass/fail	PASS

Table 7.1: Upload device template test case

7.2.2 Static code testing

Apart from these functional integration tests, we have performed static testing of the code, with tools like SonarLint [39] (a plug-in for IntelliJ IDEA) and ESLint [40] (a plugin for VS Code). These linters inspect statically the code to, for instance, prevent the creation of too long or difficult to understand functions. On one occasion, we detected that the method in charge of generating the widgets from their templates in the DeviceTemplateServiceImpl class from the ”device-template” module was too complex. We had to divide it so it didn't exceed the maximum cyclomatic complexity. Generally, these tools helped to create a clean and understandable code.

7.2.3 User acceptance tests

At the end of every iteration, user acceptance tests were performed, making sure that a communication with the final user is maintained along all the process, so they can make explicit the evolution of their requirements, at the time that the developer has a good feedback in order to know if the software developed is adequate. The result of these tests was always a note, a .txt file with annotations of what could be improved and what wasn't to the liking of the user.

Test case no.	TC-02
Name	The reports application shows a list of uploaded templates
Preconditions	1. The user is logged into the reports web application
Test steps	1. The user clicks on the ”Reports” navigation button
Expected result	The system must list the templates that have a device type associated equal to some uploaded device’s type
Actual results	The system shows correctly the list the templates, but depicting the device templates, not the report templates
Pass/fail	FAIL

Table 7.2: List device template test case

Test case no.	TC-03
Name	The application shows a list of uploaded report templates
Preconditions	1. The user is logged into the reports web application
Test steps	1. The user clicks on the ”Reports” navigation button
Expected result	The system must list the possibly multiple report templates that are included in the possibly many uploaded device templates, when the associated device type coincides with at least one of the types of any uploaded device
Actual results	The system shows correctly the list the report templates
Pass/fail	PASS

Table 7.3: Upload device template test case

Test case no.	TC-04
Name	The application successfully hot-generates a report with the correct widgets
Preconditions	1. The user is logged into the reports web application
Test steps	1. The user clicks on one report template
Expected result	A report is well hot-generated
Actual results	The report is well generated, yet the format of it is not satisfactory
Pass/fail	PASS

Table 7.4: Hot-generate a report test case

Test case no.	TC-05
Name	The application successfully saves a hot-generated report
Preconditions	1. The user is logged into the reports web application. 2. The user is checking a hot-generated report.
Test steps	1. The user clicks on the ”save” button
Expected result	A report must be saved correctly in the database, with its well-defined attributes
Actual results	A report is stored successfully in the database
Pass/fail	PASS

Table 7.5: Save a report test case

Test case no.	TC-06
Name	The widgets show the correct data
Preconditions	1. The user is logged into the reports web application
Test steps	1. The user clicks on a saved report or hot-generates one
Expected result	The data shown in the multiple widgets is correct and corresponds to the data stored in the platform
Actual results	The data is correct
Pass/fail	PASS

Table 7.6: Widgets show correct data test case

Test case no.	TC-07
Name	The widgets show the correct data, according to the dates selected
Preconditions	1. The user is logged into the reports web application
Test steps	1. The user clicks on a saved report or hot-generates one
Expected result	The data shown in the multiple widgets is correct and corresponds to the data stored in the platform for the date range selected
Actual results	The data is correct
Pass/fail	PASS

Table 7.7: Widgets show correct data for a date range test case

Chapter 8

Report generation example

In this chapter it is included a document explaining the specification of a report template inside of a device template (document that is part of the whole documentation of the module, and that the organization will keep); an example of device template with a report template ready to be uploaded to the manager and the result of hot-generate a report from that template (Figure 8.1); and its explanation.

ReportWidgetTemplate

Inside a DeviceTemplate, and as an attribute called "reports", there will be an array of ReportTemplate. Each one of them holding, among other things, another array "widgets" of ReportWidgetTemplate. These templates for report widget objects contain four attributes:

- **"id"**, a long generated automatically by the DBMS, identifying the template.
- **"report"**, the report template where the report widget template comes and has been read from.
- **"config"**, an array of objects with two properties:
    - **"key"**, name or key of the configuration element. For a general widget, we can have the following values: "name", "col", "row", "sizeX", "sizeY", "widgetType", "colour" or "statistical". Depending on the value for the widgetType, we can have more configurations, that will be documented in the section specific to every widget type.
    - **"value"**, value of the previous key; its possible values depend on the type of configuration of the key. For a key value equal to **"name"**, this will indicate the name of the widget; for **"col"**, a numeric value between 0 and 13 that indicates the position of the widget in the X axis; for **"row"**, a numeric value between 0 and 9 that indicates the position of the widget in the Y axis; for **"sizeX"**, a numeric value between 1 and 14 that indicates the vertical size wich the widget will occupy; **"sizeY"**, a value between 1 and 10 that indicates the size that the widget will occupy horizontally; for **"widgetType"**, one of the following strings: "TEXT", "GRAPH", "TEMP_TABLE"; for **"color"** a hexadecimal expression of the general colour for the widget (only applicable for "TEXT" and "TEMP TABLE" widget types); for **"statistical"**, one of the next values: "FIRST", "MIN", "MEAN", "SUM", "MAX", "LAST", which indicate which operation will be done with every group of data values obtained from the properties of a device (indicated by the next "properties" attribute); in case of **"text"**, for **"isLabour"**, a boolean value, indicating if a distinction between labour days and holidays is to be done, and that actually only the widget type "GRAPH" consumes.
- **"properties"**, array of properties, according to the IoT system, which the widget will refer to in its deployment. Normally, they will be a device's property to show graphically. Every object in the array will have the next properties:
    - **"procedureName"**, name of the logic grouping of properties to be read in a device
    - **"propertyName"**, name of the property that the data describes
    - **"config"**, an array of objects (similar to the widget's with the same name), each with the following properties:
    - **"key"**, name or key of the configuration element ("name", "colour", "asIncremental")
    - **"value"**, value of the configuration element. Its possible values depend on the type of configuration to which the previous "key" element refers to. For a value of "key" equal to **"name"**, the property "value" will be a string indicating the name of the property; for **"asIncremental"**, a boolean value that indicates wether to take the increments between samples instead of its values; for **"color"**, an hexadecimal expression indicating the specific colour for the property representation.

The following is an example of a device template with one report template with three widget templates, one of each type developed:

```
{
    "deviceType": "MONITOR",
    "name": "Doble Mensual",
    "description": "Tabla de calor y laborales vs festivos",
    "version": 1,
    "procedureByDefault": "system",
    "dashboards": [],
    "sketches": [],
    "favoritesGraph": [],
    "alerts": [],
    "reports": [
      {
        "headerTemplate": {
          "headerPicPath": "assets/images/header/mockheader.jpg",
          "config": []
        },
        "footerTemplate": {
          "footerPicPath": "assets/images/footer/mockfooter.jpg",
          "config": []
```

```
        },
        "name": "$DEVICE_NAME mensual",
        "defaultMode": "MONTH",
        "widgets": [
          {
            "config": [
              {
                "key": "col",
                "value": 0
              },
              {
                "key": "row",
                "value": 0
              },
              {
                "key": "sizeX",
                "value": 20
              },
              {
                "key": "sizeY",
                "value": 1
              },
              {
                "key": "widgetType",
                "value": "TEXT"
              },
              {
                "key": "text",
                "value": "Monitorización"
              },
              {
                "key": "color",
                "value": "#e84e40"
              },
              {
                "key": "size",
                "value": "S"
              }
            ]
          },
          {
            "config": [
              {
                "key": "col",
                "value": 0
              },
              {
                "key": "row",
                "value": 2
              },
              {
```

```
            "key": "sizeX",
            "value": 20
          },
          {
            "key": "sizeY",
            "value": 13
          },
          {
            "key": "widgetType",
            "value": "TEMP_TABLE"
          }
        ],
        "properties": [
          {
            "config": [
              {
                "key": "name",
                "value": "Temp"
              },
              {
                "key": "asIncremental",
                "value": false
              },
              {
                "key": "type",
                "value": "BAR_GRAPH"
              },
              {
                "key": "color",
                "value": "#32CD32"
              }
            ],
            "procedureName": "procedurePrueba",
            "propertyName": "rand25"
          }
        ]
      },
      {
        "config": [
          {
            "key": "col",
            "value": 0
          },
          {
            "key": "row",
            "value": 21
          },
          {
            "key": "sizeX",
            "value": 10
          },
```

```
            {
              "key": "sizeY",
              "value": 6
            },
            {
              "key": "widgetType",
              "value": "GRAPH"
            },
            {
              "key": "isLabour",
              "value": true
            }
          ],
          "properties": [
            {
              "config": [
                {
                  "key": "name",
                  "value": "Temp"
                },
                {
                  "key": "asIncremental",
                  "value": false
                },
                {
                  "key": "color",
                  "value": "#32CD32"
                }
              ],
              "procedureName": "procedurePrueba",
              "propertyName": "rand25"
            }
          ]
        }
      ]
    }
  ]
}
```

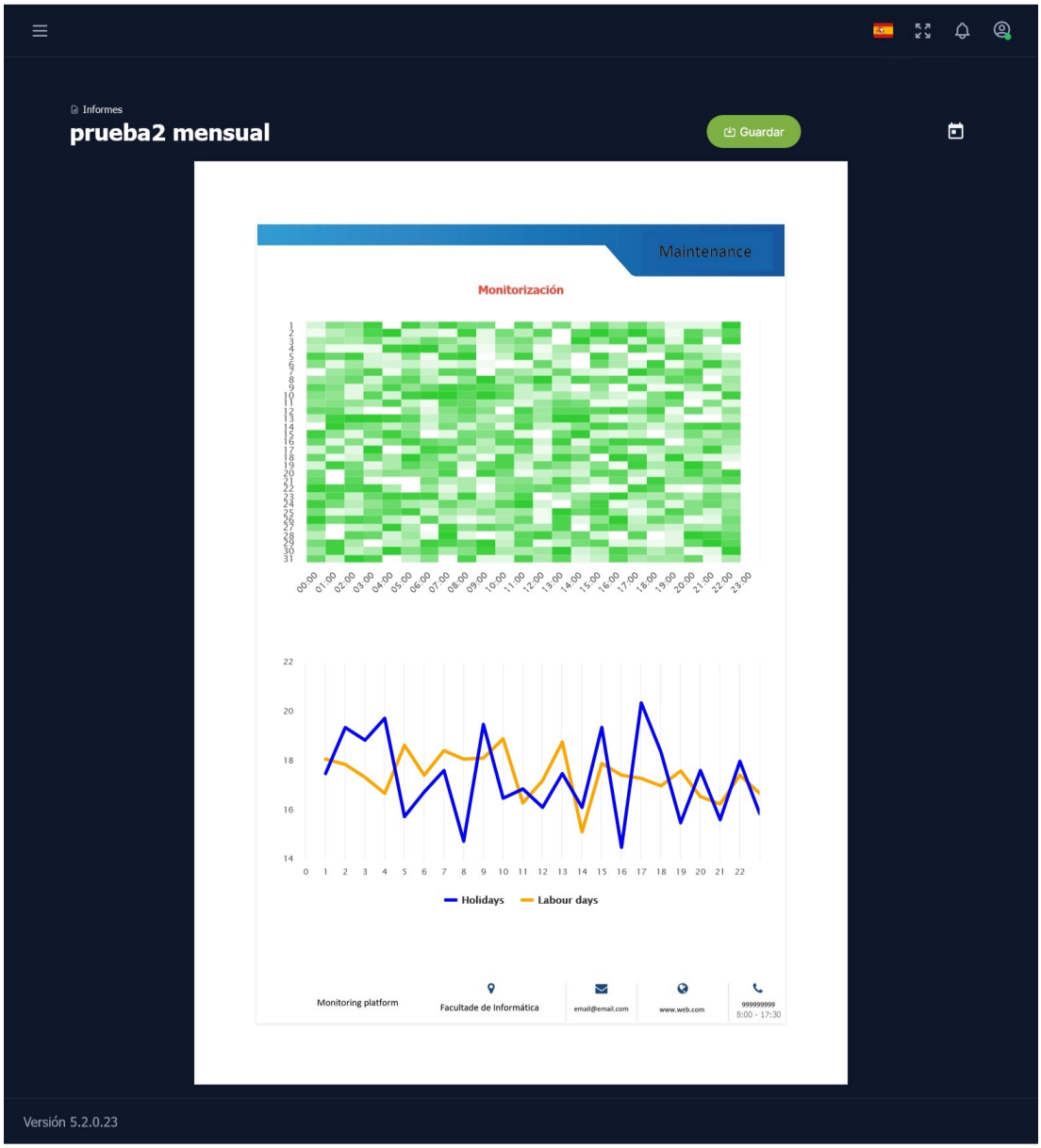

Figure 8.1: Report example, generated from the previously posted template

As it can be inferred from the template, this is built to generate a report with three widgets: a text widget with a simple world ”Monitorización” as content; a temperature table (or heat map, explained in Section 6.2.1) with data from a property called ”rand25” (which by the way stores random numerical values between 10 and 25); and a graph widget (also explained in Section 6.2.1) reading the same values with the configuration ’isLabour’, meaning that it shows the mean value of the data per hours, making distinction between labour and holidays.

The ’defaultMode’ attribute of the report template is set to ’MONTH’. This implies that

the data is collected for a whole month, from its first date to the last. The selected device is a test device that doesn't correspond to any real device, so the data of the series is not from a real application. If this were real data, the user could inspect for example the distribution of the values for the month, and check if there are variations between labour days and holidays. This is an example of how this automatic report generation could help in the decision-making process, but the user will be able to create their own reports, and the developers of the organization could add concrete widgets to the widget catalogue, enhancing the possibilities.

Chapter 9
Conclusions

In this chapter we will collect the conclusions of the work, and also some future work of implementations that could be added to the project.

9.1 Final discussion

After all the development, we have a functional module for the generation of reports, fulfilling the needs of the organization: hot-generating reports from templates, saving them into the system, and allowing the selection of concrete date ranges for the data shown. We have therefore completed like this the primary and secondary goals defined in the introduction of this memory (Section 1.3):

1. Primary goal: create a web application module that serves the purpose of generating reports with graphs about devices.

   This is the major goal, and it has been achieved. We have an application that reproduces a template, reading and parsing its properties, and then translates it into a human-readable report with graphical widgets ready to display the data collected from an associated device.

2. A list of the available templates can be viewed from the application.

   With the adaptation of the 'manager' system, report templates associated to a device template can be uploaded to the platform and shown in the main web application, so the user is able to select them for the hot-generation of reports.

3. A concrete date range for the collection of the data can be selected.

   This functionality is finally fully developed in the front end, which is the part that requests the data to the SOS platform and displays it through the widget components. The IoT system allows the determination of concrete date ranges for the retrieval of data, and the requests are customized to meet the user's demands.

4. Apart from hot-generation of reports, these can be saved, so they don't have to be generated each time the user wants to inspect them.

   We have successfully modeled the pertinent entities in our module in the back end, and mapped them to the database with Hibernate, so a strong model is in charge of persisting the concrete instances of reports. From the moment the user wants to store them, the reports are fully available to be checked anytime.

5. The kind of graphic widgets that can be shown in a report must be open to expansion.

   This goal is mainly achieved: we have applied an strategy pattern, so a concrete service can be invoked depending on the type of the widget to be generated. This way, only the implementation of a concrete widget generation strategy within a new class has to be created. Also, a superclass of WidgetReport is able to be specialized for new widgets. In the front end, a new widget component must be implemented and adapted for each new kind of widget, preparing a concrete Highcharts configuration and an algorithm for the consumption of the data.

To sum up, it should be noted that we have a final powerful tool for the final users to help them take good decisions based on data from widget-based reports. The organization is happy with the result, and is ready to keep maintaining and expanding the module, which was one of the motivations.

We have been able to adapt a well-known methodology to our necessities, making an effort to streamline a methodology like Unified Software Development Process (USDP), showing the capability to adapt the common knowledge of Software Engineering to our own context.

We have experienced the development and maintenance of a web application that works in the context of the Internet of Things (IoT), which is a growing industry, being able to learn how an instance of Sensor Observation Service (SOS) works in the real world. This is of great value for my experience and is a very good preparation for the labour market.

As a personal thought, I think I have a lot to thank both the organization that has collaborated in this project and my tutors, from whom I have learned a lot. I am now much more ready than ever before to tackle the web applications development, which as I said, is my aspiration with this project.

9.2 Future work

As part of a bigger platform, our module of reports generation is planned to be expanded and maintained by the organization for which it has been developed. Next, it will be described some possible future expansions of the module.

9.2.1 Multiple paging

Within the scope of this project, the tool is only able to generate one-page reports. This can be for sure expanded with the proper work with CSS and Angular. It can be done by changing the way the widgets are arrayed (checking if the rows of the widget exceed the maximum rows of one grid), or perhaps modifying the specification of the templates.

9.2.2 More widget types

The set of available widgets to show in the reports is open to expansion. This means that another developer could add support for new widget templates, implying the implementation in the back end of the proper class for the widget instance and its service (widget instance generation strategy from template). For the front-end, it would be necessary to implement a new component for the widget that processes properly the data and prepares it for the screen.

9.2.3 Automatic testing

Because the organisation uses a different web service to upload the templates, because the widget processing is mainly done on the front-end, and because our product activity is mainly visual, we have chosen to use functional and user acceptance testing. However, it is a possible future work to add automatic tests, specially at front-end level.

9.2.4 Better template management

To this day, the organization uses a different web application to manage the templates, which then are shared with the users. The templates have to be written in JSON and then pasted on the website. We could build a tool that would make this process easier for users, without them needing to know JSON.

9.2.5 Change the default mode of the report

With what is build until now, the 'defaultMode' attribute of the report is set with the report template. That means that when the hot-generated report is saved, this is set and immovable. We could for instance add a drop-down component in the reports detail component, so this attribute is changed in database when the user confirms their selection.

9.2.6 Larger date selection ranges

We could implement larger date selection ranges. For that, we would have to adapt the availability of dates in the date picker. Right now, if the report default mode is set to 'CUSTOM', when you click on a first date, available dates are limited to one month from the first date.

Also, the widget components configuration for Highcharts would have to be adapted (e.g. for the construction of the y axis, which to this day is limited to the calculation of one month length).

Appendices

List of Acronyms

API Application Programming Interface. 32, 45

CRUD Create, Read, Update, and Delete. 32, 43

CSS Cascading Style Sheets. 7, 47

DAO Data Access Object. 32

DTO Data Transfer Object. 34, 43

ER Entity-Relationship. iv, 35, 39

ERP Enterprise Resource Planning. 3

HTML HyperText Markup Language. 6, 7, 47

HTTP Hypertext Transfer Protocol. 45

IDE Integrated Development Environment. 8

IoT Internet of Things. 1, 68, 69

JDBC Java Database Connectivity. 32

JSON JavaScript Object Notation. 3, 11, 21, 57, 70

REST Representational state transfer. 32, 45

SOS Sensor Observation Service. 1, 48, 68, 69

SPA Single Page Application. 6, 46

UML Unified Modeling Language. 22

USDP Unified Software Development Process. iv, 22–24, 69

XML Extensible Markup Language. v, 6, 48, 53

Bibliography

[1] Unified software development process workflows. [Online]. Available: https://commons.wikimedia.org/wiki/File:Unified_Process_Model_for_Iterative_Development.svg

[2] Standardized, web-based upload and download of sensor data and sensor metadata. [Online]. Available: https://52north.org/software/software-projects/sos/

[3] Business intelligence reporting tool. [Online]. Available: https://eclipse.github.io/birt-website/

[4] Embed reports & dashboards in your app. perfectly. [Online]. Available: https://www.jaspersoft.com/

[5] Postgresql: The world's most advanced open source relational database. [Online]. Available: https://www.postgresql.org/

[6] Get java for desktop applications. [Online]. Available: https://www.java.com/en/

[7] Hibernate. everything data. [Online]. Available: https://hibernate.org/

[8] Interface entitymanager. [Online]. Available: https://docs.oracle.com/javaee/7/api/javax/persistence/EntityManager.html

[9] Introduction. [Online]. Available: https://mybatis.org/mybatis-3/index.html

[10] Spring boot. [Online]. Available: https://spring.io/projects/spring-boot

[11] The modern web developer's platform. [Online]. Available: https://angular.io/

[12] Spa (single-page application). [Online]. Available: https://developer.mozilla.org/en-US/docs/Glossary/SPA

[13] Material design. [Online]. Available: https://material.io/

[14] Simply visualize. [Online]. Available: https://www.highcharts.com/

[15] Html5. [Online]. Available: https://developer.mozilla.org/en/docs/Glossary/HTML5

[16] Css. [Online]. Available: https://developer.mozilla.org/en/docs/Web/CSS

[17] Rapidly build modern websites without ever leaving your html. [Online]. Available: https://tailwindcss.com/

[18] Apache maven project. [Online]. Available: https://maven.apache.org/what-is-maven.html

[19] Developer environments made easy. [Online]. Available: https://www.vagrantup.com/

[20] Develop faster. run anywhere. [Online]. Available: https://www.docker.com/

[21] Free/libre open source software binaries of vs code. [Online]. Available: https://vscodium.com/

[22] Intellij idea. capable and ergonomic ide. [Online]. Available: https://www.jetbrains.com/idea/

[23] git. [Online]. Available: https://git-scm.com/

[24] The one devops platform. [Online]. Available: https://about.gitlab.com/

[25] Overleaf for institutions. [Online]. Available: https://es.overleaf.com/for/universities

[26] Manifesto for agile software development. [Online]. Available: https://agilemanifesto.org/iso/en/manifesto.html

[27] Workspace and project file structure. [Online]. Available: https://angular.io/guide/file-structure

[28] Annotation type id. [Online]. Available: https://docs.jboss.org/hibernate/jpa/2.1/api/javax/persistence/Id.html

[29] Annotation type generatedvalue. [Online]. Available: https://docs.jboss.org/hibernate/jpa/2.1/api/javax/persistence/Id.html

[30] Annotation type column. [Online]. Available: https://docs.jboss.org/hibernate/jpa/2.1/api/javax/persistence/Column.html

[31] Annotation type manytoone. [Online]. Available: https://docs.jboss.org/hibernate/jpa/2.1/api/javax/persistence/ManyToOne.html

[32] Annotation type temporal. [Online]. Available: https://docs.jboss.org/hibernate/jpa/2.1/api/javax/persistence/Temporal.html

[33] Annotation type onetomany. [Online]. Available: https://docs.jboss.org/hibernate/jpa/2.1/api/javax/persistence/OneToMany.html

[34] Annotation type joincolumn. [Online]. Available: https://docs.jboss.org/hibernate/jpa/2.1/api/javax/persistence/JoinColumn.html

[35] Annotation type transient. [Online]. Available: https://docs.jboss.org/hibernate/jpa/2.1/api/javax/persistence/Transient.html

[36] Angular elements overview. [Online]. Available: https://angular.io/guide/elements

[37] Basic line. [Online]. Available: https://www.highcharts.com/demo/line-basic

[38] Heat map. [Online]. Available: https://www.highcharts.com/demo/heatmap

[39] Sonarlint. [Online]. Available: https://www.sonarsource.com/products/sonarlint/

[40] Find and fix problems in your javascript code. [Online]. Available: https://eslint.org/