Software Engineering Project Report

Alessio Pepe - 0622701463
Paolo Mansi - 0622701542
Teresa Tortorella - 0622701507

17 Dec 2020

1 Introduction

The project concerns the development of a web application to support an industrial maintenance process. There are various types of maintenance that can be performed based on certain conditions or characteristics, classified into two categories: planned and unplanned.

The planned approach to maintenance requires that maintenance strategies can be classified into corrective maintenance and preventive maintenance strategies, which are based on the execution of all those activities that tend to avoid the occurrence of damage to the machines and factory equipment. Unplanned maintenance refers to what can intervene to repair a sudden failure and is entirely reactive to the errors of the Machine.

To facilitate the maintenance of machinery and to manage more efficiently the faults presented in these business resources, it is essential to use IT tools that allow the management of planned and unplanned maintenance tasks.

The application will allow the management of the following maintenance activities:

- Scheduled activity (scheduled maintenance)
- Close Emergency Work Order (EWO)

The application must use a database to log maintenance - related tasks, exposing an intuitive and user – friendly interface that supports the different roles involved in the process: Planner and Maintainer.

The project aims to develop an information system capable of managing information from planned and unplanned maintenance activities, as well as those related to EWO records. Specifically, it will be a web application to provide to the roles of Planner and Maintainer during the management cycle of a machinery maintenance process.

The software development process focuses on the idea of application as a maintenance activities management tool that contributes to the company's continuous quality improvement program.

The team initially consisted of 5 people. Before the start of the first sprint, one member left the group, and before the beginning of the second, another member left the team. The current team, at the delivery date, is composed as follows: Alessio Pepe, Paolo Mansi and Teresa Tortorella.

The following technologies were used to develop the application: Java EE, Java SE, JavaScript (and particularly the jQuery library), HTML5 & CSS (and the Bootstrap framework), Tomcat, and Jetty that are both Java Web Servers and Java Servlets container.

To facilitate collaboration and versioning, the team used GitHub. GitHub is built precisely for partnership. Set up an organization to improve the way where the team works together and get access to more features.

2 Scrum

The work was carried out following the Scrum methodology.

Scrum is an Agile development methodology that has demonstrated promising results. Takeuchi and Nonaka first introduced this methodology based on its successful usage in the manufacturing industries. Many of the terminologies used in Scrum are adopted from the game of rugby. The Scrum framework consists of team roles, events, artefacts, and rules. You must use all these components but might add other techniques or features as well.

Scrum is an iterative method and is based on short interactions called sprints. A sprint is the basic units of development for units of functionalities of the total software product. They are short (one month or less) and time-boxed (their duration is held constant, but scope may be adjusted if needed). Ideally, a potentially releasable product is produced at the end of each sprint.

Scrum defines the following artefacts to be used for controlling the project:

- The *product backlog*, an ordered list of all the remaining requirements or user stories for a product. The product owner prioritizes and ranks the requirements.
- The *sprint backlog* is the ordered list of tasks that need to be done for the current sprint. Tasks here are broken down to be small. Rather than tasks being assigned, developers choose their next task based on the sprint backlog and their skills.
- The *burndown chart,* which is a frequently updated (daily or more often), publicly displayed chart showing the remaining log in the current sprint backlog.

In the preliminary phase, user stories were defined, and priorities were assigned to them.

Trello is used to organize the work and follow the Scrum methodology. An initial view of our Trello workboard is shown in Figure 2.1. Story points have been assigned to each story, which classifies the cost in terms of effort, through the technique of the planning poker (where each component initially gives a score, and we come to meet it until we get the best estimate for everyone).

The set of written stories constitutes the product backlog. In the first sprint (two weeks), a collection of stories were identified which are all related to DbAdmin and System Administrator. It was decided to develop this part, whose stories are prioritized as stories of high importance because, without these stories, the application cannot work. The application without these stories would be just a demo because the data would have to be managed manually through the database.

The development of this first set of stories allows us to manage the fundamental data needed to carry out all the other operations.

The Scrum for Trello plugin, the graphic display of the story points, the points consumed, and the "Doing" and "Done" card lists allowed to create the burndown chart automatically and accurately for both sprints.

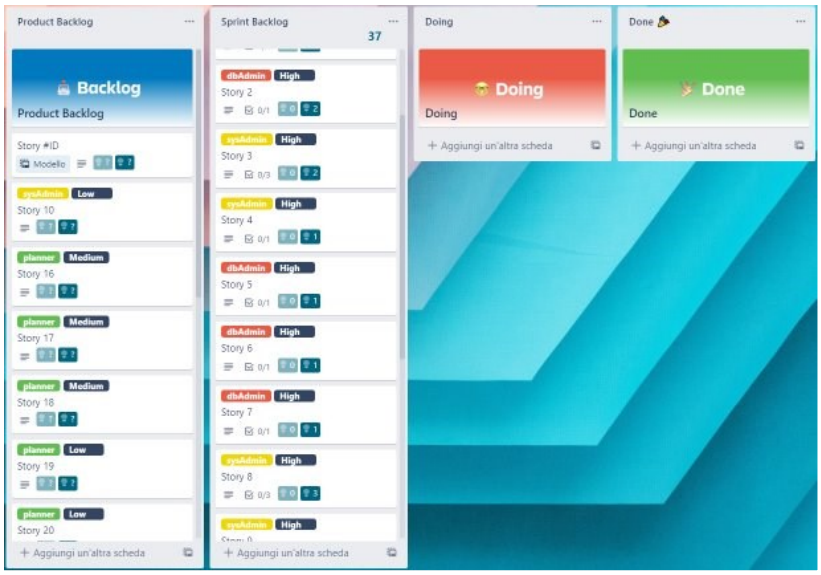


Figure 2.1 - The initial state of the scrum board.

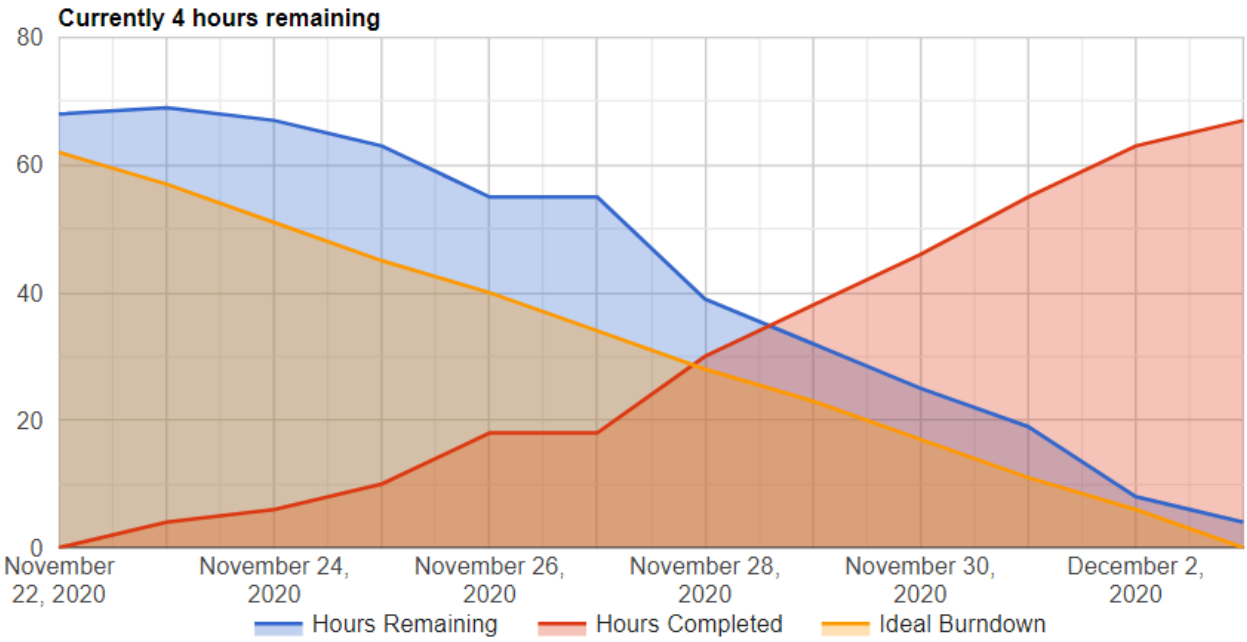


Figure 2.2 - The burndown chart of the first sprint

From Figure 2.2 you can see how initially the blue curve (real burn down chart) is prolonged compared to the orange curve (ideal burndown chart) because the team was inexperienced in the java and front–end web technologies used.

The points of a story of the first sprint have not been completed. This story for the methodology used cannot be considered complete.

At the end of the first sprint, the team found itself with one less member.

In the second sprint for the reasons mentioned above, fewer points were planned based on the operating speed of the previous sprint.

In this sprint, all the stories of medium and low importance have been planned. In this release, it is not yet possible to manage all the parts of the application intended for the Maintainer. Also, it is not possible to view the status of the assigned activities. However, it is possible to assign the activities and notify the Maintainer by e-mail.

The client can use this application by verifying the correct functioning so that he can provide feedback for any changes.

Figure 2.3 shows how the estimates made for the second sprint were correct: the blue curve this time follows the ideal trend.

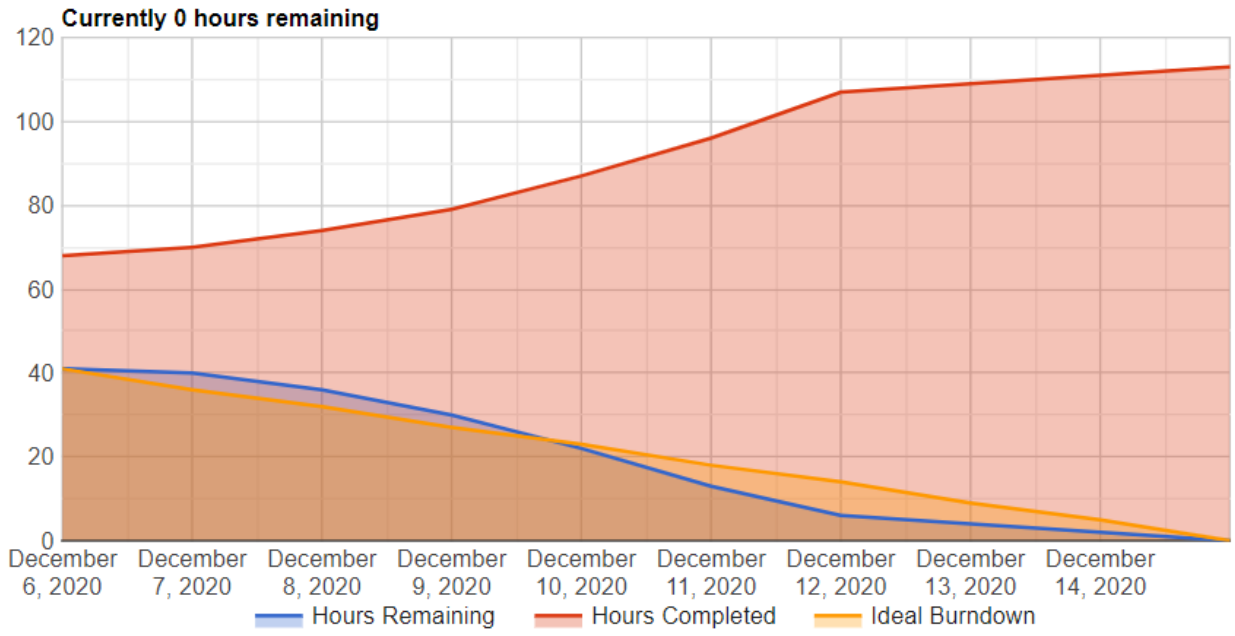


Figure 2.3 - The burndown chart of the second sprint.

3 High Level Design

At first, in sprint planning, the initial high-level architecture very similar to the current one developed (Figure 3.1).

The architecture shows the browser refers to an HTTP server to obtain the static pages that will then be dynamically compiled through the asynchronous request functions. These will require data that will be taken from JSON interfaces. The interfaces have always been designed to return the same type of data to services mounted in the same server.

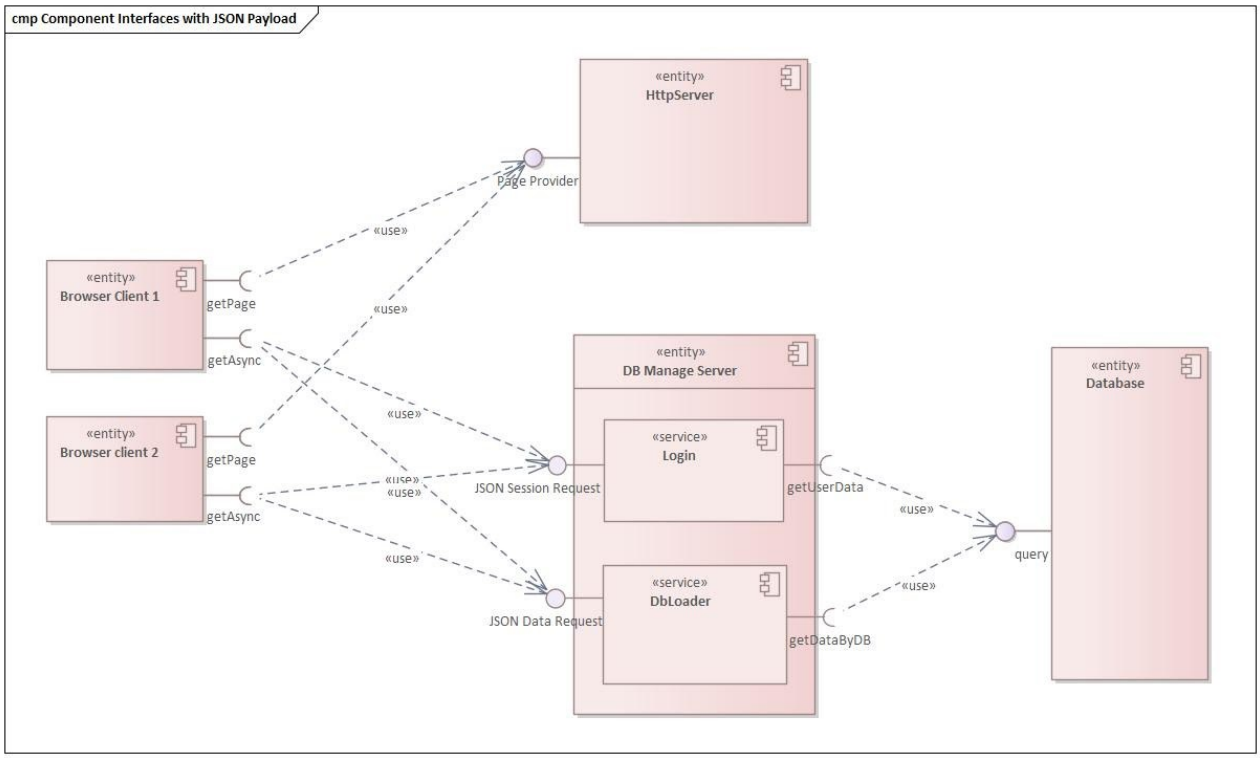


Figure 3.1 – The High-Level Design

4 Database

4.1 Introduction

In creating the application, it was necessary to use a database to store persistent data already in the design phase as the use of files or libraries that simulated a database would not have been as efficient as Isolates of the same given the large quantity of data. Furthermore, they would not have guaranteed security, the possibility of multiple accesses and the necessary support, as well as performance in the same way.

It was decided to create the database using RDBMS technology, as, although part of the data lent itself well to a NoSQL representation (especially for documents), part of the data had solid relationships between them and therefore the possibilities to graphically visualize the connections between the data have drastically simplified the subsequent development of the application.

In particular, it was decided to use the RDBMS PostgreSQL v12 and not to particularize the syntax of the queries or the creation of the structures with the support languages offered by the system, but only the vanilla SQL (as far as possible). In this way, it is possible to make the database portable from one technology to another without the need for major changes and, for many parts, without any modification.

4.2 Design Choices

The e-r diagram of the database was designed gradually with the realization in the stories, adding each time (when required) the part of data necessary for the operation of the application. Obviously, in the following report, not all the versions of the diagram are reported but only the final version at the state of the second Sprint. The graph can be viewed in Figure 4.1 or, for a better resolution, at the link in the description of the figure.

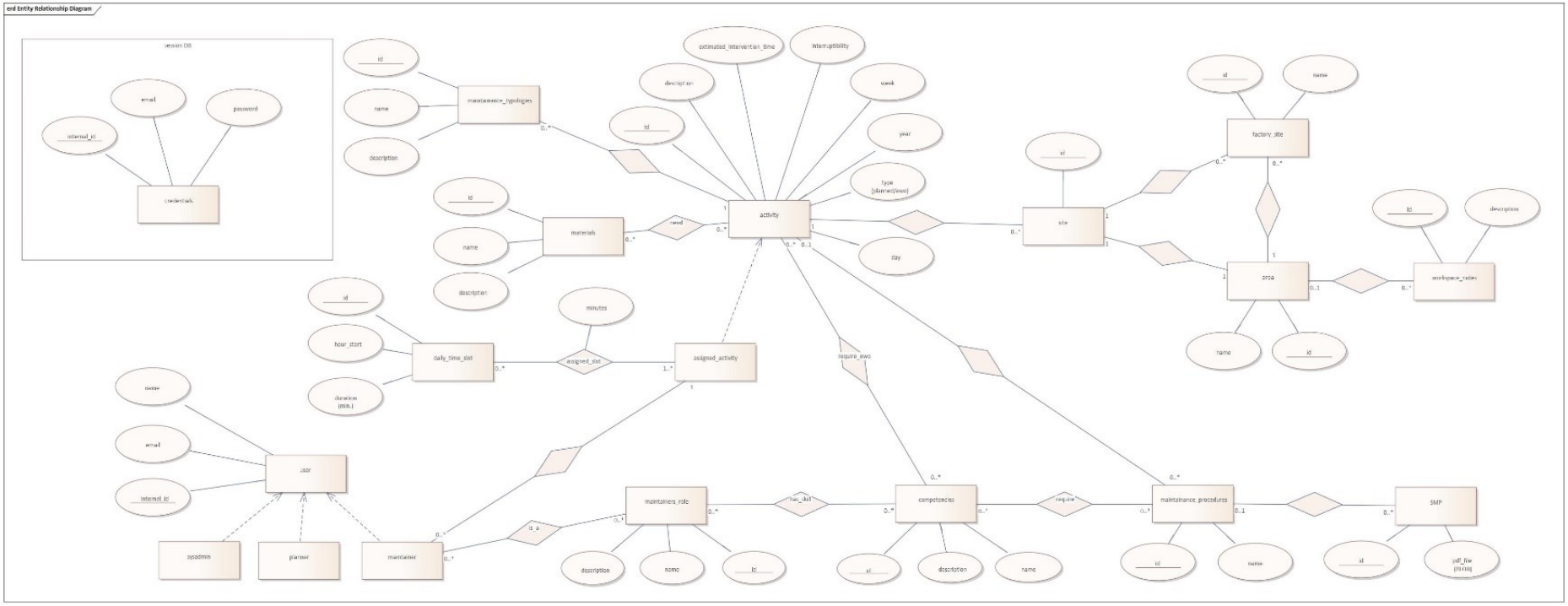


Figure 4.1 – The e-r diagram at the state of the second sprint (https://github.com/apepe108/seProgectTeam3/blob/main/Design/se_v0.3.eapx)

We want to explain some of the less immediate choices:

- The activities, both planned and unplanned, were managed with the same data model without a flag because the form of the data is the same; the uses made of it are different.
- It was decided to keep the assignment data in a rawer form and to make it more complex to obtain the information on availability from this last section in favour of non-redundancy in the data.
- We have chosen to have a separate table, not necessary, to store the SMP blob files as they are managed differently by the different databases and therefore, to make the application more portable.
- Given the presence of a maintenance procedure in the planned activities and the non-presence in the EWO activities, it was decided to assign the competences both to the procedures and directly to the activities, to avoid data redundancy in the planned activities (even if this is intrinsically managed in the database structure, so it would allow a simple upgrade).

The physical implementation, through SQL scripts, is available in the GitHub repository at the following link: https://github.com/apepe108/seProgectTeam3/tree/main/db/src.

Two representations of the logic diagram resulting from the implementation are shown below, obtained by reverse engineering with the DbVisualizer tool:

- A hierarchical view, shown in Figure 4.2, where it is easy to compare compliance with hierarchies and consistency with what is the design.
- An organic view, shown in Figure 4.3, where it is easy to notice the relationships between the tables and therefore, the links between them. It is also easy to understand what the information flow is.

Both images are available at a higher quality in the GitHub repository at the links indicated in the image caption.

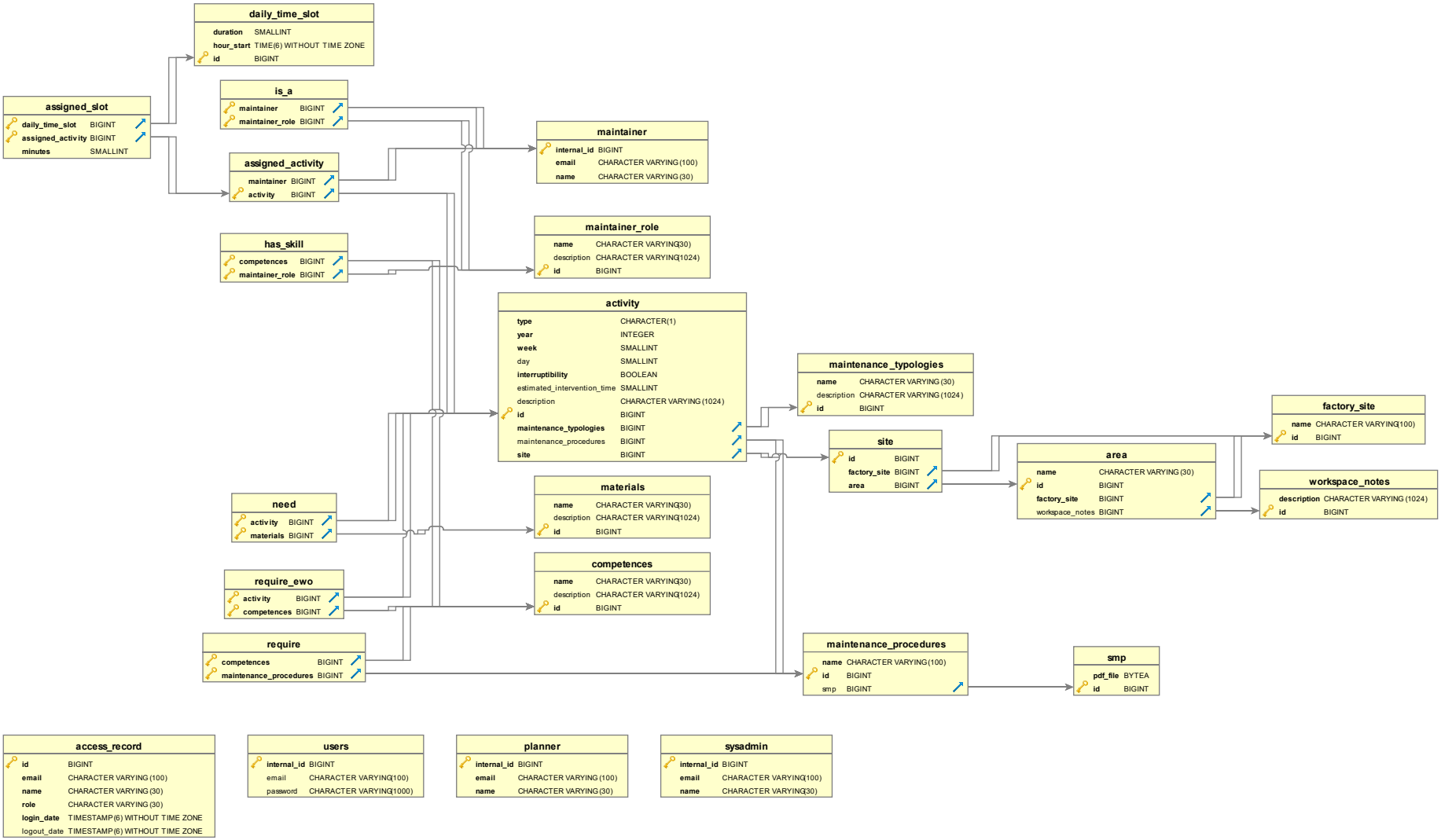


Figure 4.2 - A hierchical view of Db logic scheme. (https://github.com/apepe108/seProgectTeam3/blob/main/db/hierarchical_db_v1.3.svg)

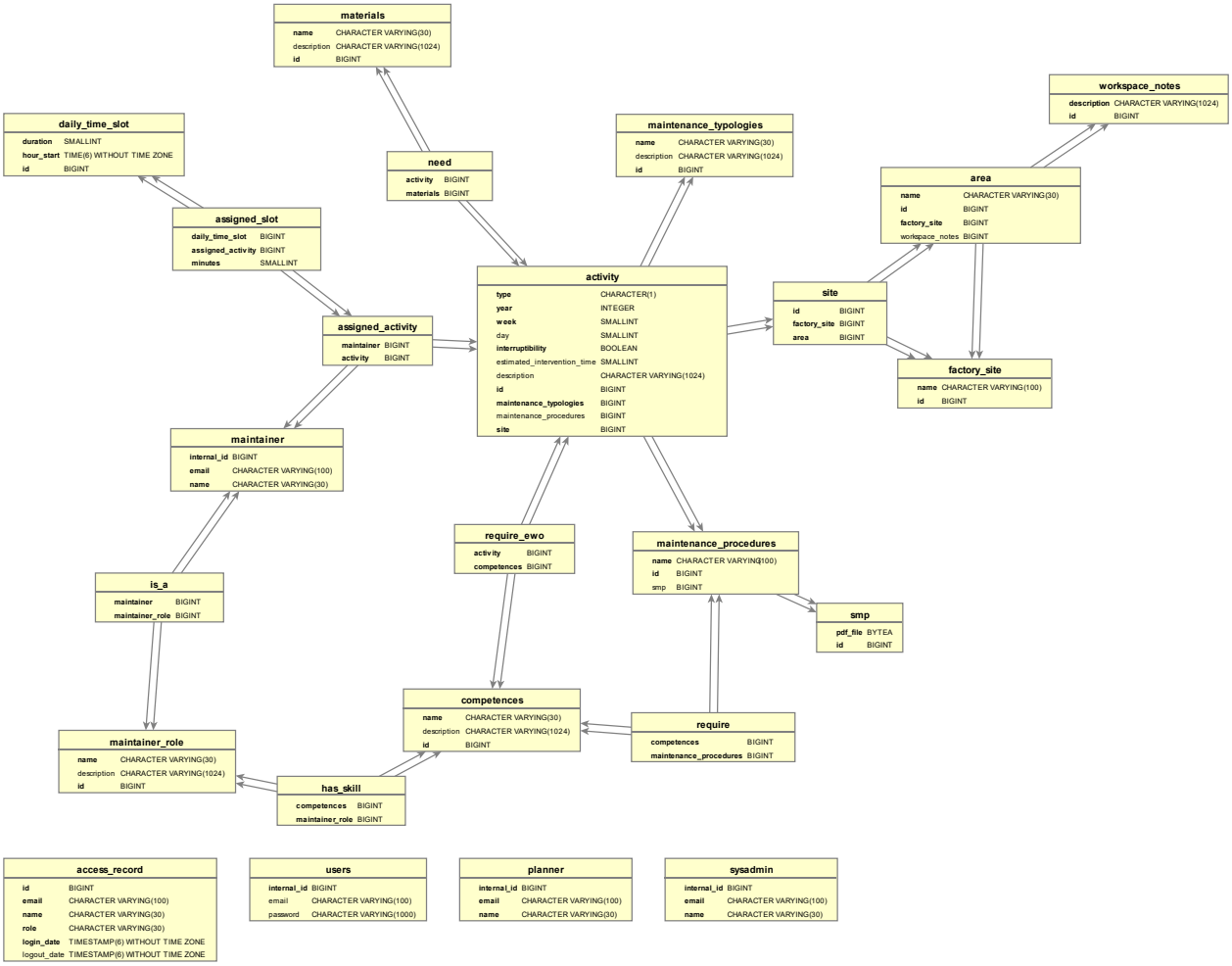


Figure 4.3 - A organic view of Db logic scheme.
(https://github.com/apepe108/seProgectTeam3/blob/main/db/organic_db_v1.3.svg)

5 Backend Application

As for the server-side application part, we have chosen to use Java technology for web applications, by inserting the product code into a Tomcat server, as shown in the architecture in Figure 3.1.

IntelliJ Idea 2020.2 is the ide used for development. As for the Java version, the JDK 15 was used for the build but always setting the lowest possible version of Java to be used to generate the .class files, to improve compatibility even on less performing servers.

Still, regarding technologies, Java Maven was used to managing dependencies. The .pom file, therefore, contains all the libraries necessary to build the application, making sure that you do not have to add external libraries either in the IDE being developed or in Tomcat libraries. We want to report some of the leading libraries used:

- `javax.servlet` for the part of the application that makes the APIs available to request data or makes changes;
- `org.junit.jupiter` for automated testing with Junit5;
- `org.postgresql` for the JDBC drive;
- `org.eclipse.jetty.aggregate` to instantiate a servlet and test it with JUnit5;
- `com.sun.mail` for mail sending.

5.1 Design Choices

The only identified service, which in its entirety makes up the server-side application, is structured in three packages, shown in Figure 5.1:

- The classes used to interact with the database used for data storage are provided in the `dbinteract` package. These are implemented using the JDBC driver mechanism.
- The models package provides classes that represent and contextualize the data extracted from the database, obtaining the information associated with them. They also offer the simplifications of providing a `toJson` interface that allows you to convert this data into Json format in a straightforward way.
- The servlet package provides the implementations of the HttpServlets, used to create the APIs used by the client-side application. It can be appreciated in the pom.xml file, but also in Figure 5.2, that the 4.0 servlets have been used to ensure more excellent performance and to simplify the deployment of these (since XML files will not be compiled, but the Java compiler will manage everything through annotations).

In the rest of this section, the contents of these packages will be explained in more detail.

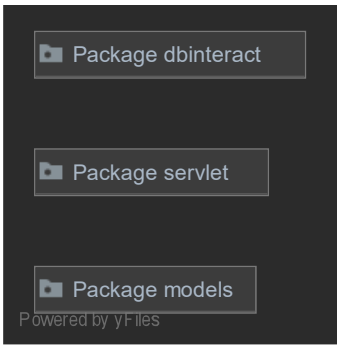


Figure 5.1 - Reverse engineering on all source code.

```xml
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>4.0.1</version>
    <scope>provided</scope>
</dependency>
```

Figure 5.2 - Servlet dependency (extracted by pom.xml)

5.1.1 Models

A representation of the package, utilizing the UML class diagram, can be displayed in Figure 5.3 or, for higher quality, at the link in the caption of the figure.

In this diagram of the classes, in addition to the classic inheritance and merging relationships, we have also chosen to display the dependencies between the various classes to visualize the flow of information more effectively.

The most crucial choice to note is that all the classes of the package implement the Model interface, which requires the implementation of a `toJson` method and the equals method (necessary within the test classes to then use the `assertEquals` method provision from JUnit). In this regard, it can be noted that some of the classes have an association relationship with other classes of type Models, with cardinality not always equal to 1 because, in some cases, it is necessary to store within an object of type Model, a list of objects of type Model (e.g., the materials necessary for an activity, the skills required for a certain maintenance procedure). To simplify the conversion in these cases (and not only) we see that the `JsonUtil` class provides a static method `toJson` that converts a list of objects of type Model into a single JSON string. This util is also necessary during the regular operation of the application, as even requesting the display of a list of activities will require that all activities be placed inside a JSON file.

Furthermore, for each class, there is a test class which tests all the methods of the class, following a white-box approach. For this reason, the test coverage on this package is 100%.

5.1.2 DbInteract

A representation of the package, utilizing the UML class diagram, can be displayed in Figure 5.4 or, for higher quality, at the link in the caption of the figure.

It was chosen, again to pursue the principle (which should be clear by now) of database portability. Furthermore, it was decided that, since this section could often vary according to the type of requests, the features added/modified or even for changes to the graphic interface, to make it easy to adjust by spending more in its design.

The decorator pattern was used to respect the two choices described above:

- Both the concrete implementations and the decoration classes are managed through an interface which, at present, has only a concrete PostgresDb performance. It is easy to add other concrete implementations and then be able to use the queries present in the decoration classes. Furthermore, this also makes it possible to use two types of databases (e.g., if these are replicated and therefore, the transition takes time, because the data is separated).
- It is possible to modify the queries already present by having to retest even the single decoration method (even if the integration test is recommended and has always been performed) thanks to the principle of single responsibility deriving from inheritance in the object-oriented programming paradigm.
- To better separate the various decoration classes, we have chosen to use a decorator to manage access to a specific area of the database where the data are closely linked to each other (example, send a decorator for materials, one for activity, one for the sites).

A test class is provided for each concrete implementation for the interface and each decoration class. Since the methods of these classes, as it will be pointed out further below, use Modal objects or Modal lists especially when displaying data, once the necessary modals have been created and tested, the decorators have been tested, excluding the defects from the modals for the principle of single responsibility.

In this case, a white-box approach was also used to try to make the application as stable as possible. The coverage is however 90% because some points (which also increase the cyclomatic complexity of the application) introduce cases that can never occur according to the logic defined for the application (e.g., a decorator could throw an exception if they are not passed all the minimum required parameters; however, this never happens as a servlet makes the call after checking the parameters).

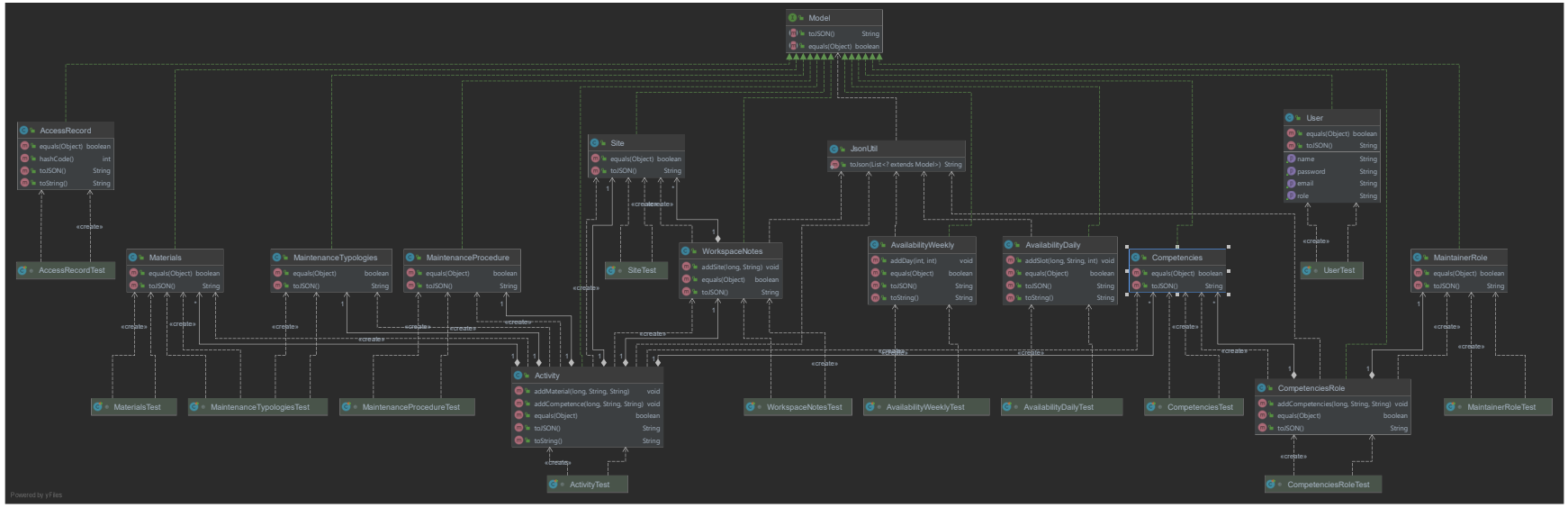


Figure 5.3 - Reverse engineering representing the package models
(https://github.com/apepe108/seProgectTeam3/blob/main/backEnd/Documentation_v2.0/Package%20models.svg)

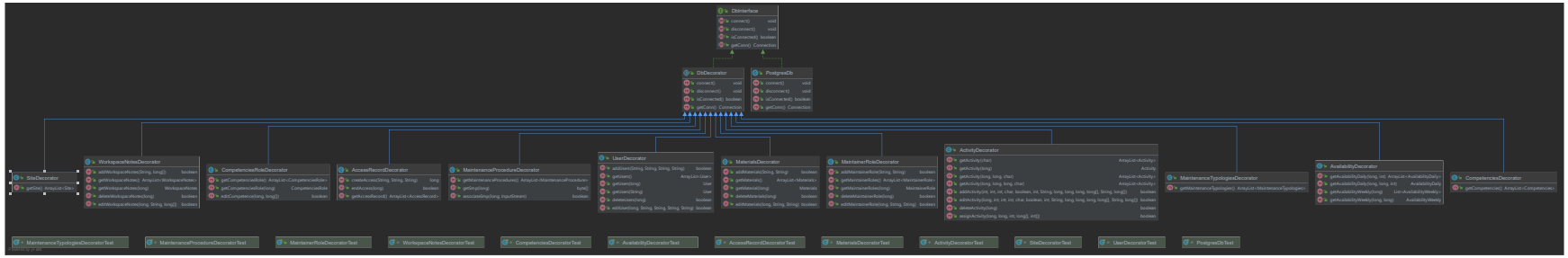


Figure 5.4 - Reverse engineering representing the package dbinteract.
(https://github.com/apepe108/seProgectTeam3/blob/main/backEnd/Documentation_v2.0/Package%20dbinteract%201.svg)

5.1.3 Servlet

A representation of the package, utilizing the UML class diagram, can be displayed in Figure 5.5 or, for higher quality, at the link in the caption.

As already said before, all the classes created go to give an implementation to the abstract class HttpServlet, which simplifies the creation of a web servlet as it increases the abstraction from a generic servlet (and it is for this reason that even the latter, although not present in the package, are shown in the class diagram).

As mentioned earlier, servlets 4.0 were used in the development of servlets, making possible not to compile XML files and to be able to manage the options than necessary for deployment in the application on the server through annotations:

- `@WebServlet`, which allows you to specify various settings, including the URLs to which the servlet responds and the name of the servlet. This is used in all servlets.
- `@MultipartConfig` to make it possible for the servlet to receive multipart objects from web pages, used only in the servlet linked to maintenance procedures, to receive the pdf file to be associated with the type.

The following conventions were used for all servlets:

- In the case of data visualization, if requested a list is shown in JSON format (also empty), while if an object is asked for and it does not exist, an error 417 (expectation failed) is returned.
- If all the necessary parameters are not passed to an API, the server responds with an error 400 (bad request).
- The case in which the request generates an error (not possible because, due to the logic defined for the application, this can only occur if not all the parameters are managed), error 417 is returned once again.
- In all other cases the server responds with 200 (ok) and the data in case they need to be returned.

It should be noted that also for this package, a test class is provided for each class to make the code reliable. The coverage is about 90% as some errors, which must be managed in the code, can never occur according to the defined logic.

5.2 Information flow example

Not all the possible cases that can occur during execution are reported, but only a simple case of logging and viewing access, by means of the UML class diagram, shown in Figure 5.6.

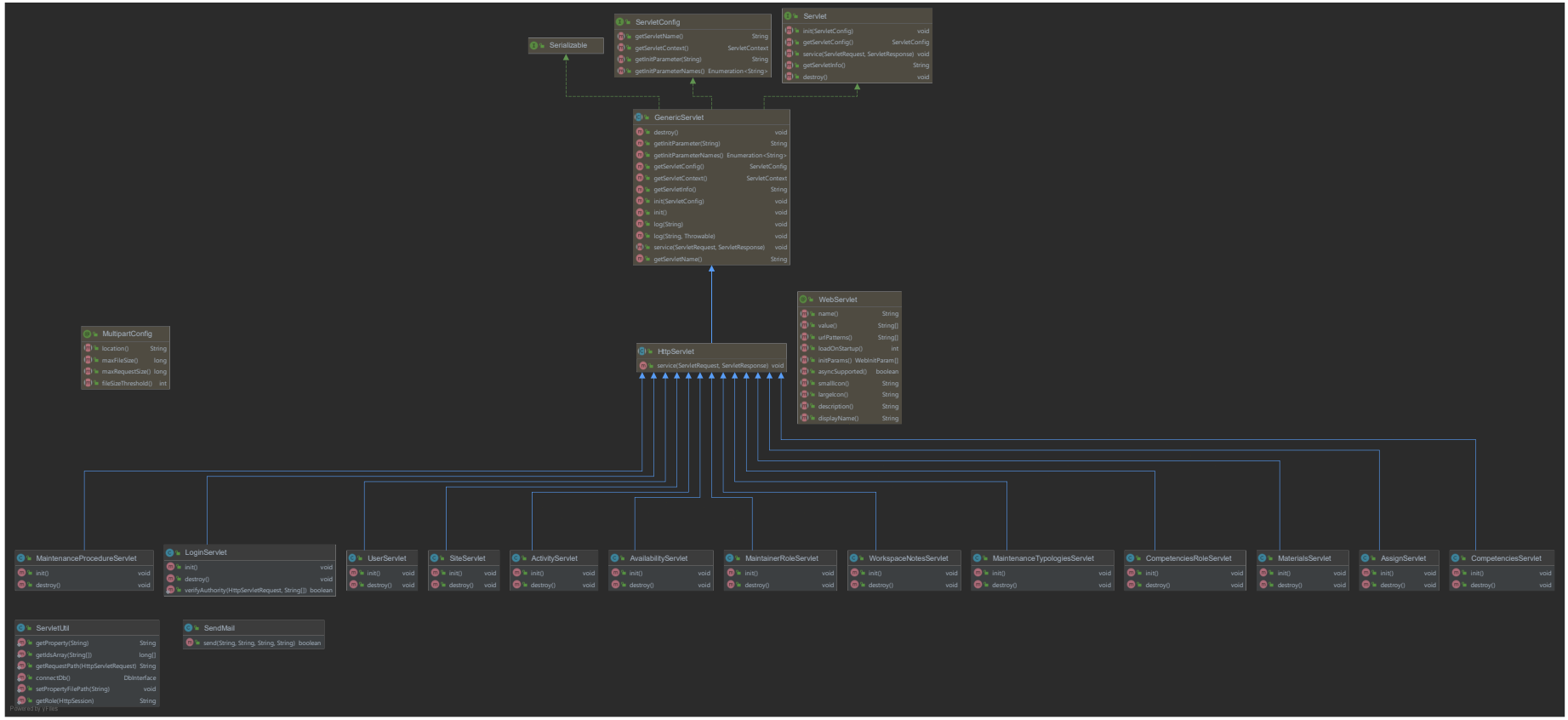


Figure 5.5 - Reverse engineering representing the package servlet.
(https://github.com/apepe108/seProgectTeam3/blob/main/backEnd/Documentation_v2.0/Package%20Servlet.svg)

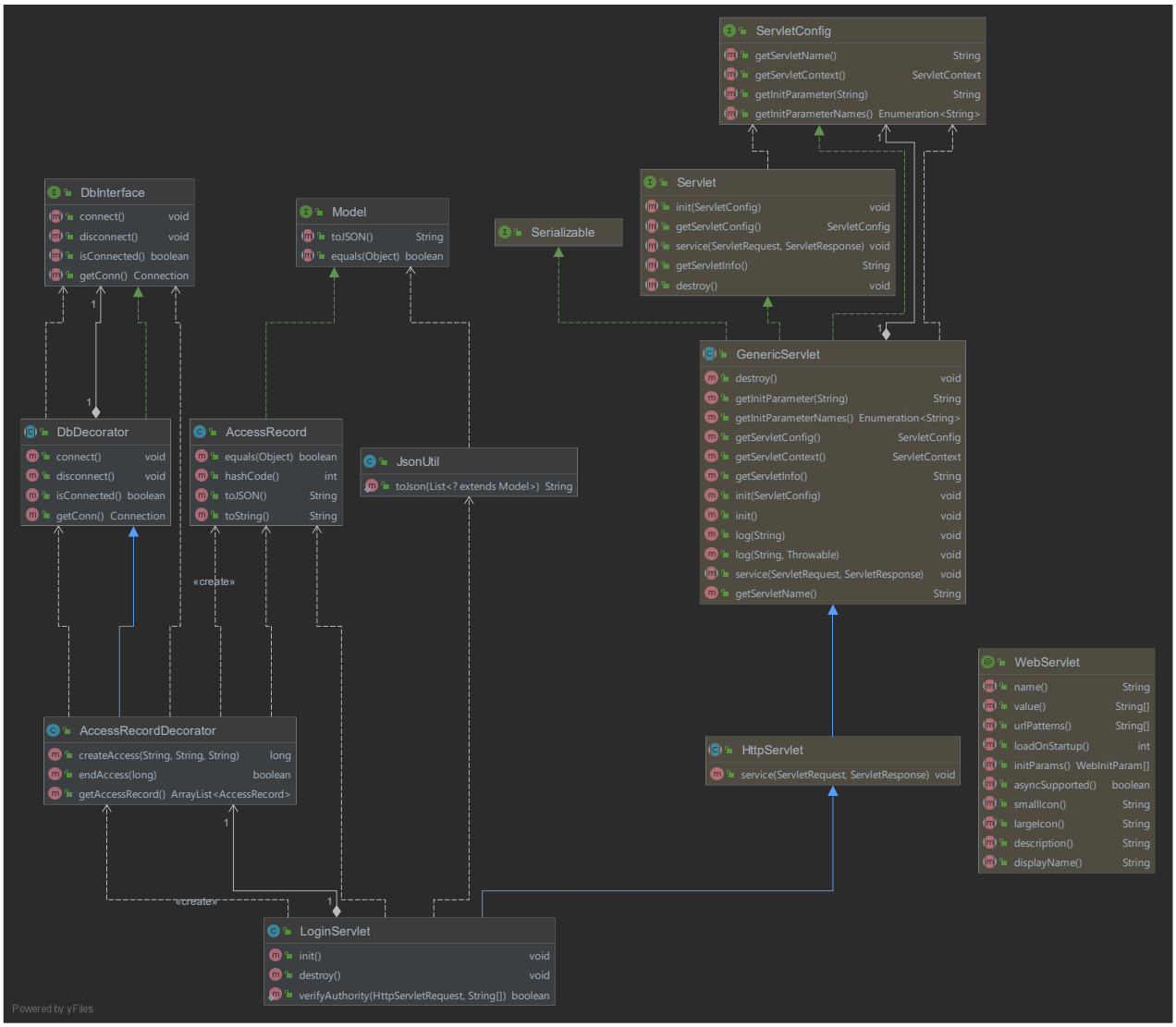


Figure 5.6 - Reverse engineering representing UML extract related to access record data. (https://github.com/apepe108/seProgectTeam3/blob/main/backEnd/Documentation_v2.0/Flow%20example%201.svg)

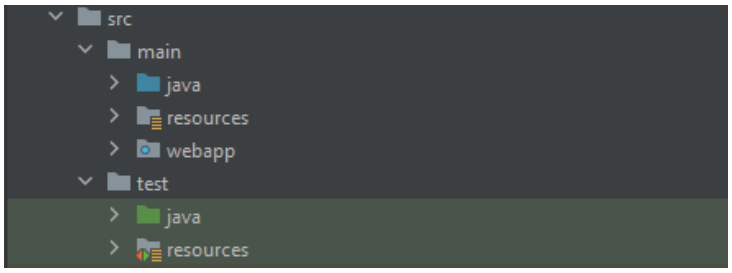


Figure 5.7 - Virtual directory IntelliJ Idea

Knowing that the memorization of the access, with relative login and logout time, once occurred, takes place in the servlet with the name Login. Through the latter, it is possible to request the access list in JSON format to the URL */access-record*, defined with the annotation `@WebServlet`.

The access request then arrives at the servlet, which is managing it calls `DbInterface` instantiating it through `PostgresDb`, decorates it with the `AccessRecordDecorator` class and this returns a list of `AccessRecord` objects (which extend Model) to the servlet. To return the object in JSON format, the servlet calls the `toJson` function of `JsonUtil` and then returns the string to the client that requested it.

5.3 Testing

As previously mentioned, testing was carried out using the Junit5 framework.

Having used IntelliJ Idea to develop the application, the work has been simplified as this IDE allows you to have a separate view of the application code and the test code (due to virtual directories), as shown in Figure 5.7.

The tests were carried out as follows:

- In the Model classes, we have tested the expected result of the methods in various situations.
- For the tests of the `dbinteract` classes, two setups and tear down methods have been set up to load and download the desired data from the database. We then go to test the methods of data visualization or modification of these, evaluating the results. Regarding the visualization of the data, it is assumed that the errors have already been detected and handled for the Model classes and that therefore the responsibility for errors in this point is linked to the classes in question.
- To test the servlet classes, in addition to the setup and teardown methods, additional methods of this type have been set up, before class and after class, which use a class written specifically to test the servlets. In practice, the servlets are mounted on a Jetty server and post and get requests are made to this, then going to read the results directly from the page.

We want to report some code extracts that relate a test function to a standard test documentation module. These, done for a small piece of code, will demonstrate that it is not necessary to document the code as this is self-explanatory thanks to the use of the library.

For example, we can look at the code in Figure 5.8. On line 25, the string expected by the `toJson` method is declared, on line 20 a model object representing that data is instantiated, and on line 32 the `toJson` method is used to obtain the current result. We expect the current and the expected to be the same, so we verify the correctness with the `assertEquals` method.

```java
@Test
void toJSON() {
    // Make expected
    String expected = "{\"email\":\"tizio@email.com\",\"name\":\"tizio\",\"role\":\"SysAdmin\",\"login_date\":" +
            "\"13-12-2021 12:45:05\",\"logout_date\":\"13-12-2021 15:40:44\"}";

    // Make actual
    AccessRecord c = new AccessRecord( email: "tizio@email.com",  name: "tizio",  role: "SysAdmin",
            LocalDateTime.of( year: 2021,  month: 12,  dayOfMonth: 13,  hour: 12,  minute: 45,  second: 5),
            LocalDateTime.of( year: 2021,  month: 12,  dayOfMonth: 13,  hour: 15,  minute: 40,  second: 44));
    String actual = c.toJSON();

    // Test
    assertEquals(expected, actual);
}
```

Figure 5.8 - toJson test of AccessRecord class.

CLASS TEST REPORT		
Classe to test: AccessRecord		
User stories: *10*		
Method to test: *toJSON*		
Target: Check the correctness of the translation from java object to Json string of the modal implementing class.		
TEST DATA		
Input	**Expected result**	**Output**
this *(an AccessRecord object whit the following parameters: email: tizio@email.com, name: tizio, role: SysAdmin, loginDate: 13-12-2021 11:45:05, logoutDate: 13-12-2021 15:40:44)*	`{"email":"tizio@email.com","name":"tizio","role":"SysAdmin","login_date":"13-12-2021 12:45:05","logout_date":"13-12-2021 15:40:44"}`	`{"email":"tizio@email.com","name":"tizio","role":"SysAdmin","login_date":"13-12-2021 12:45:05","logout_date":"13-12-2021 15:40:44"}`

Table 5.1 – AccessRecord toJson method test.

```java
@Test
void getAccessRecord() {
    // Actual
    List<AccessRecord> actual = db.getAccessRecord();

    // Expected
    ArrayList<AccessRecord> expected = new ArrayList<>();
    expected.add(new AccessRecord( email: "tizio@email.com",  name: "tizio",  role: "SysAdmin",
            LocalDateTime.of( year: 2021, month: 12, dayOfMonth: 13, hour: 12, minute: 45, second: 5),
            LocalDateTime.of( year: 2021,  month: 12,  dayOfMonth: 13,  hour: 15,  minute: 40,  second: 44)));

    // Match
    assertEquals(expected, actual);
}
```

Figure 5.9 - getAccessRecord test of AccessRecord class.

CLASS TEST REPORT		
Classe to test: AccessRecordDecorator		
User stories: *10*		
Method to test: *getAccess*		
Target: Verification of connection to the database and correct reception of data.		
TEST DATA		
Input	**Expected result**	**Output**
this *(an AccessRecordDecorator object connect to a db populated like the expected result)*	`[{"email":"tizio@email.com","name":"tizio","role":"SysAdmin","login_date":"13-12-2021 12:45:05","logout_date":"13-12-2021 15:40:44"}]`	`[{"email":"tizio@email.com","name":"tizio","role":"SysAdmin","login_date":"13-12-2021 12:45:05","logout_date":"13-12-2021 15:40:44"}]`

Table 5.2 - AccessRecordDecorator getAccess method test

Figure 5.10 - doGet test of Login servlet

CLASS TEST REPORT		
Classe to test: LoginServlet		
User stories: *10*		
Method to test: *doGet*		
Target: Verify that the data is returned correctly after invoking the servlet by means of a get call.		
TEST DATA		
Input	**Expected result**	**Output**
this *(an LoginServlet object connect to a db populated like the expected result)*	`[{"email":"tizio@email.com","name":"tizio","role":"SysAdmin","login_date":"13-12-2021 12:45:05","logout_date":"13-12-2021 15:40:44"}]`	`[{"email":"tizio@email.com","name":"tizio","role":"SysAdmin","login_date":"13-12-2021 12:45:05","logout_date":"13-12-2021 15:40:44"}]`

Table 5.3 - Login servlet doGet method test

It should be noted that, compared to what would have been formal documentation of the test, represented in Table 5.1, the content of the test is understandable even to those who do not know well what the application is, precisely because of the well-defined standards imposed from the library used. Indeed, it can be noted that it is almost more intuitive to verify the correctness through the code than from the documentation table.

The same can be verified for the methods of the decorators and servlets, of which the methods related to the visualization of the accesses are reported as an example (Figure 5.8 -> Table 5.2, Figure 5.9 -> Table 5.3).

A total of 160 were produced, all successfully passed, of which the results on coverage, already anticipated earlier, are shown in Figure 5.11.

100% classes, 91% lines covered in package 'it.unisa.diem.se.team3'

Element	Class, %	Method, %	Line, %
dbinteract	100% (14/14)	98% (73/74)	91% (727/794)
models	100% (16/16)	100% (56/56)	100% (261/261)
servlet	100% (16/16)	95% (109/114)	88% (652/733)

Figure 5.11 - Coverage test result.

6 Front End Application

6.1 Technologies

As far as the client-side is concerned, the web pages of this application were developed using the HTML5, CSS and JavaScript technologies. It was decided to make use of the framework "Bootstrap", a front-end framework created by Mark Otto and Jacob Thornton. It provides many CSS class for an effortless styling that makes use of their JavaScript plugin as well as Popper (a tooltip & popover positioning engine) to build dynamic and well-optimized web pages. As a result of that, Bootstrap allows developing pages that are entirely responsive to the user preferences in terms of pages dimension and position, in a minimal time.

At each HTML page, a JavaScript class was associated with adding dynamicity and responsiveness to the user's action. The JavaScript class makes use of jQuery, a JavaScript library intended for the manipulation of HTML pages and event handling. The use of this library makes the execution faster since it is optimized for this very purpose and more comfortable to implement. In addition to this, it enhances the versatility of the web application due to its cross-platform property.

As for the exchange of data with the API Server, was used the JSON (JavaScript Object Notation) data format. To get, and send as well, JSON data to the servlet, an AJAX request is performed. AJAX that stands for Asynchronous JavaScript and XML is a web technology that can update part of a page without reloading it, granting a more seamless user experience. This AJAX request is simplified as well by the jQuery library.

6.2 Pages Structure

Each page was developed following a standard structure. All pages present a navbar that allows navigating across all the others, a central body with a table intended to display the requested information, and some buttons that display modals for the creation, editing or deletion of data. In each page, the navbar is also present a button for logging out of the application.

Each JavaScript class presents similar structures as well. In the constructor, all the servlet URL services are defined. Some standard methods like "fillTable" and "renderGUI" are meant to render the HTML page with the JSON data obtained through the servlet service. Other functions deal with the view of the pop-up modals and the sending of data to the servlets. In addition to this, each class presents some specific utility methods.

A JavaScript file for the configuration of the servlet URL is defined as well.

6.3 Web Pages

6.3.1 Login Page

When the application is launched, the login page showed in Figure 6.1 will appear. From this page, you can insert the email and password assigned to you and then submit them through the "Login"

button if one of that field is left blank or filled out wrong (i.e., when the email field does not in the form).

6.3.2 Welcome Pages

When the login succeeds, you will be redirected to one of the pages showed in Figure 6.2 and Figure 6.3, according to your specific role. You can use the navbar to access to the other pages.

Figure 6.1 – Login Access Page

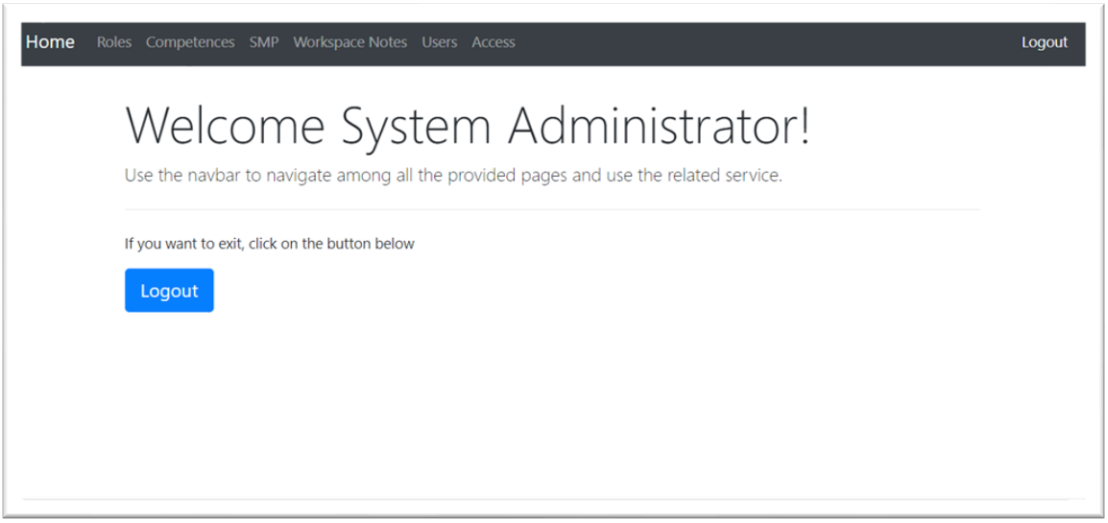


Figure 6.2 – System Administrator Welcome Page

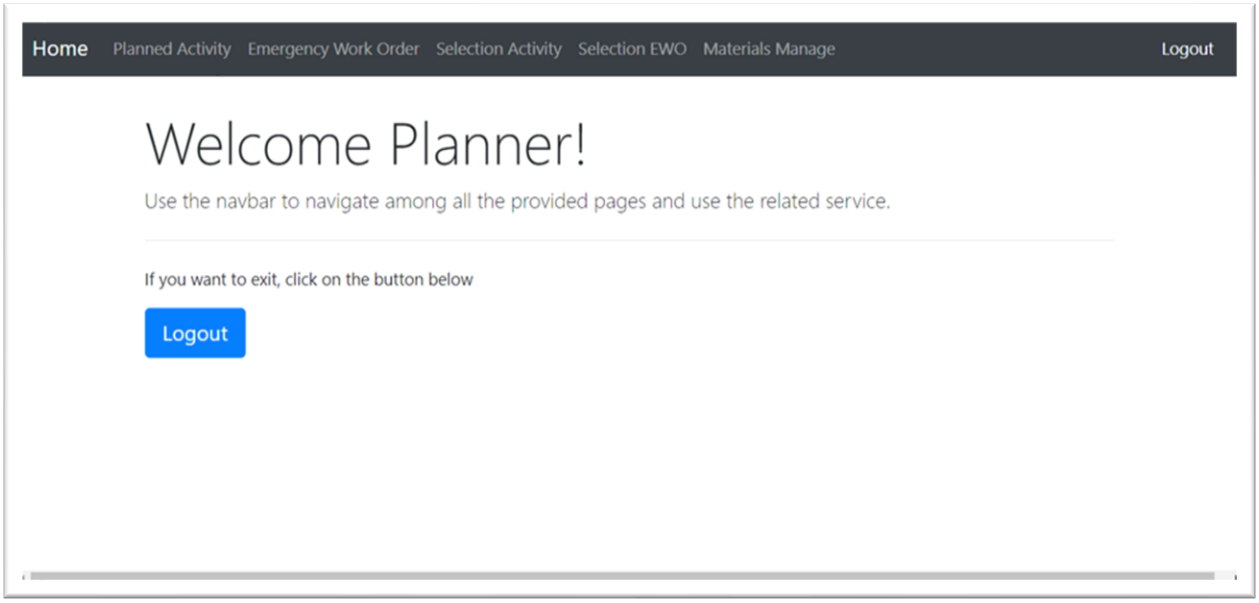


Figure 6.3 - Planner Welcome Page

6.3.3 Maintainer Role Page

The page that allows to view, create, edit, and delete the roles of maintainers is shown in Figure 6.4.

Clicking on the button “Insert”, a modal for the insertion of a new role will appear, as shown in Figure 6.5. Clicking on the “Close” button or outside of the modal will make it close without changes while filling out the required fields and clicking on the “Save Changes” will confirm the insertion. A banner will then appear on the top of the screen, showing the result of the insertion.

Clicking on the edit button will make appear a modal for editing, similar to the one shown in Figure 6.5 but filled out with the data of the corresponding row. Any edit of those data will be saved after clicking on the “Save Changes” button.

Clicking on the delete button will make appear a modal for deletion. Then click on the “Confirm” button will confirm the deletion of the element.

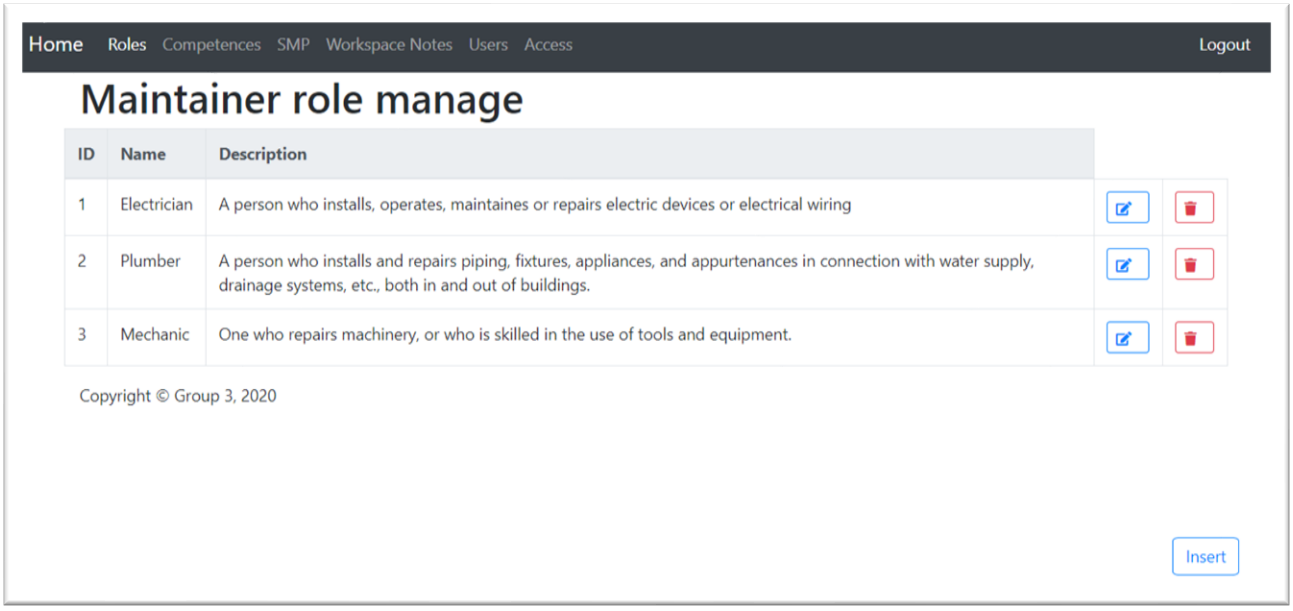


Figure 6.4 - Maintainer Role Manage Main Page

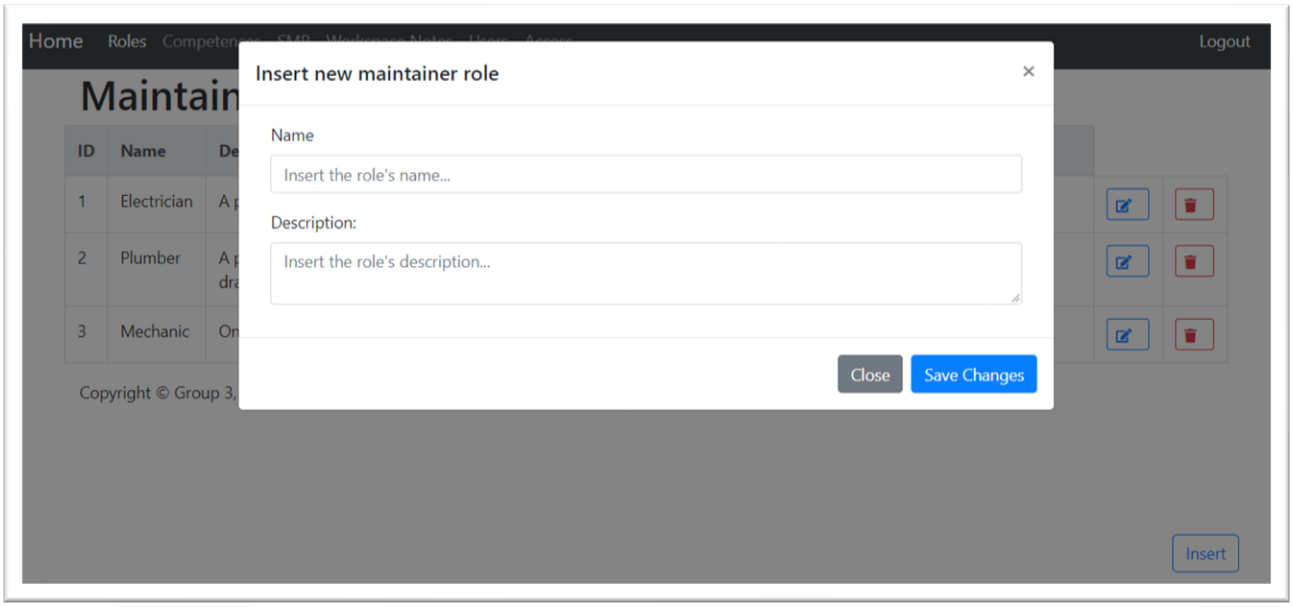


Figure 6.5 - Maintainer Role Manage Insert Modal

6.3.4 Role Competences Page

The page that allows editing the competences associated to the roles of maintainers is shown in Figure 6.6.

A modal for editing, shown in Figure 6.7, will appear after clicking on the edit button. It is possible to edit the competences associated to the related maintainer role selecting and unselecting the checkboxes. Any edit of those data will be saved after clicking on the "Save Changes" button. A banner will then appear on the top of the screen, showing the result of the edit.

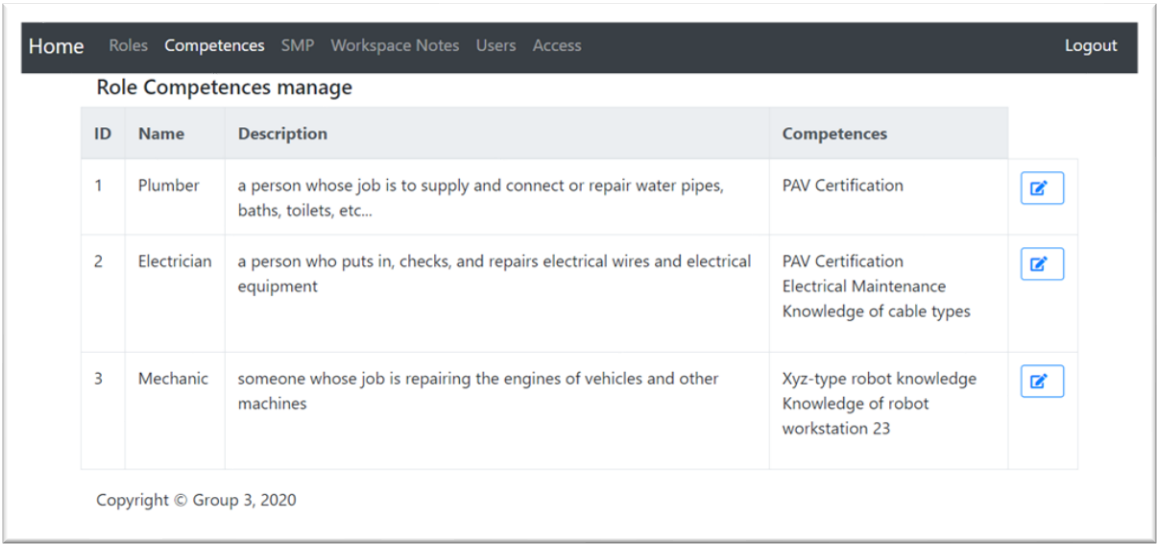


Figure 6.6 - Role Competences Manage Main Page

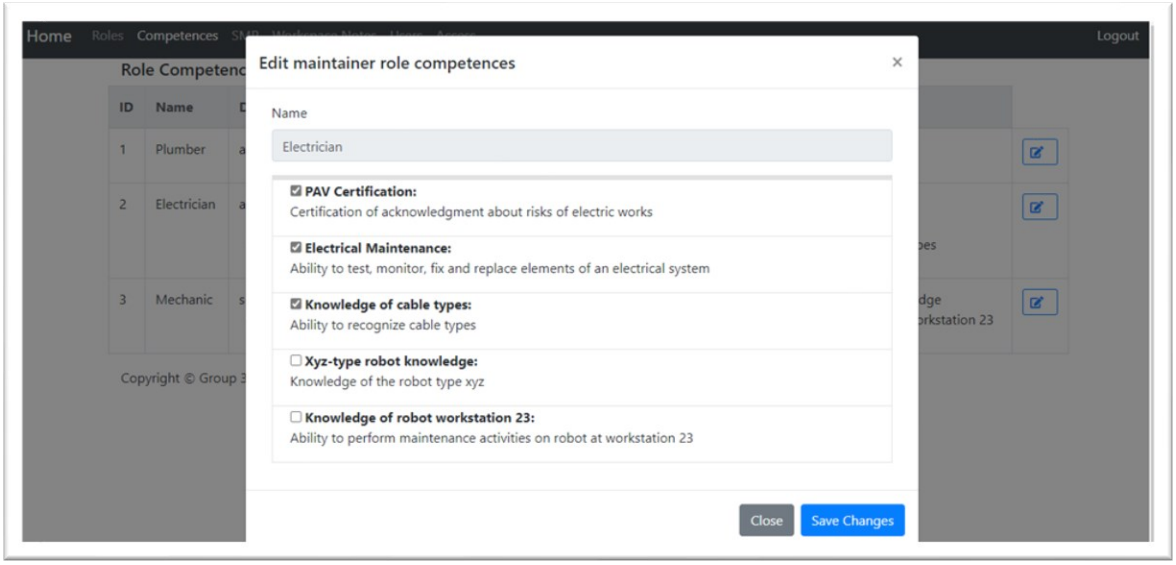


Figure 6.7 - Role Competences Manage Edit Modal

6.3.5 SMP Page

This page allows editing Standard Maintenance Procedures (SMP) associated with a specific procedure.

Clicking on the download button will make download the related SMP file and display it in this page.

A modal for editing, shown in Figure 6.8, will appear to click on the edit button. In the modal, a button that allows the user to select a file from the local machine and upload it. Any edit of the file will be saved after clicking on the "Submit" button. A banner will then appear on the top of the screen, showing the result of the edit, as shown in Figure 6.9.

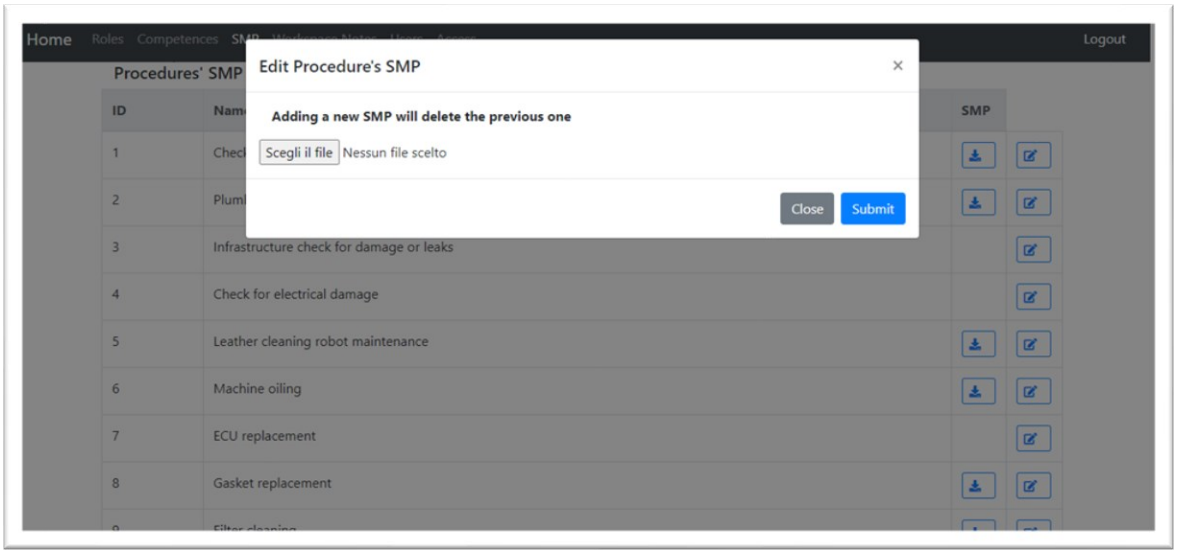


Figure 6.8 - Standard Maintenance Procedure Edit Modal

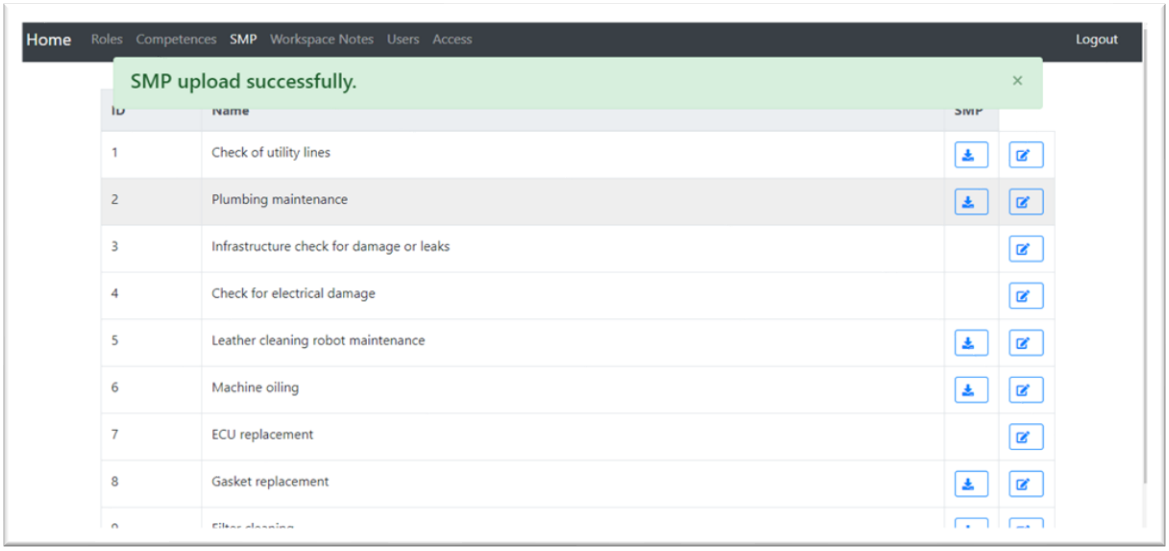


Figure 6.9 - Standard Maintenance Procedure Main Page. This Figure shows the banner obtained after the validation of the edit.

6.3.6 Workspace Notes Page

The page that allows to view, create, edit, and delete the workspace notes is shown in Figure 6.10.

Clicking on the button "Insert", a modal for the insertion of a new workspace note will appear, as shown in Figure 6.11. Filling out the required fields and clicking on the "Save Changes" will confirm the insertion while clicking on the "Close" button or outside of the modal will make it close without changes. A banner with the result of the insertion action will then appear on the top of the screen. All the entity with the related checkbox marked will be de-associated from their previous note and associated with this one.

Clicking on the edit button will make appear a modal for editing, similar to the one shown in Figure 6.11, filled out with the data of the corresponding row. Any edit of those data will be saved after clicking on the "Save Changes" button.

Clicking on the delete button will make appear a modal for deletion. Then click on the "Confirm" button will confirm the deletion of the element.

6.3.7 Access Page

The page that allows to view the list of all the accesses into this application is shown in Figure 6.12. Clicking on the "CSV" button it is possible to download those data as a csv file. An example file is shown in Figure 6.13.

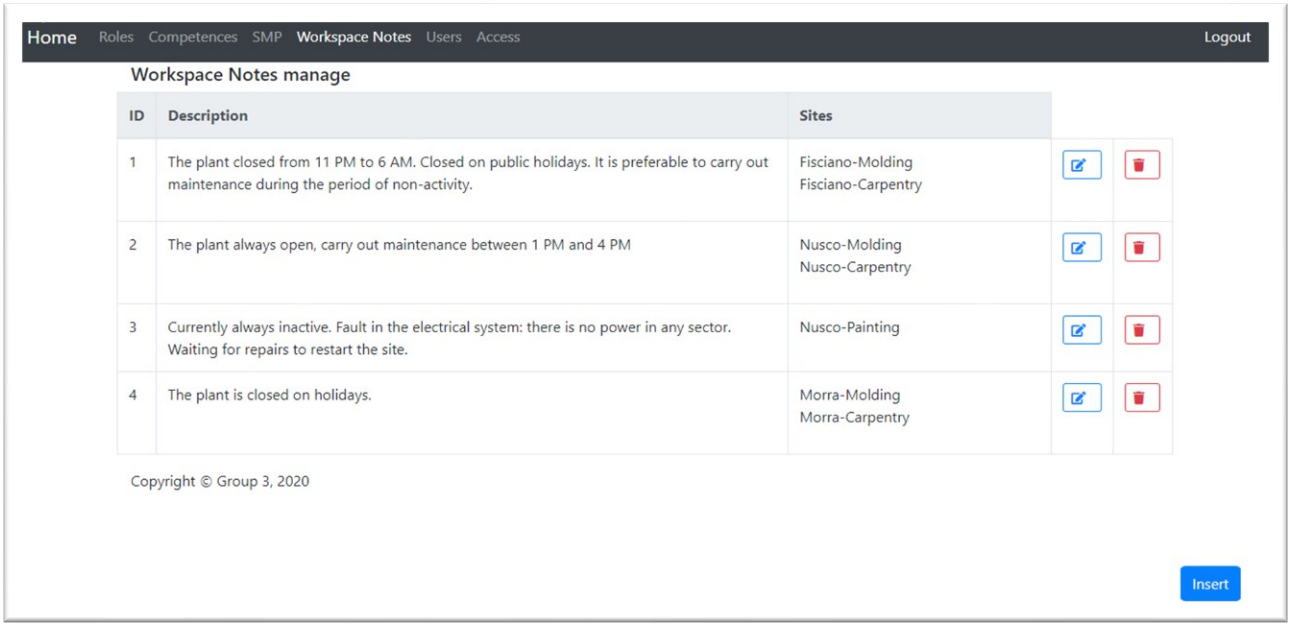


Figure 6.10 - Workspace Notes Manage Main Page

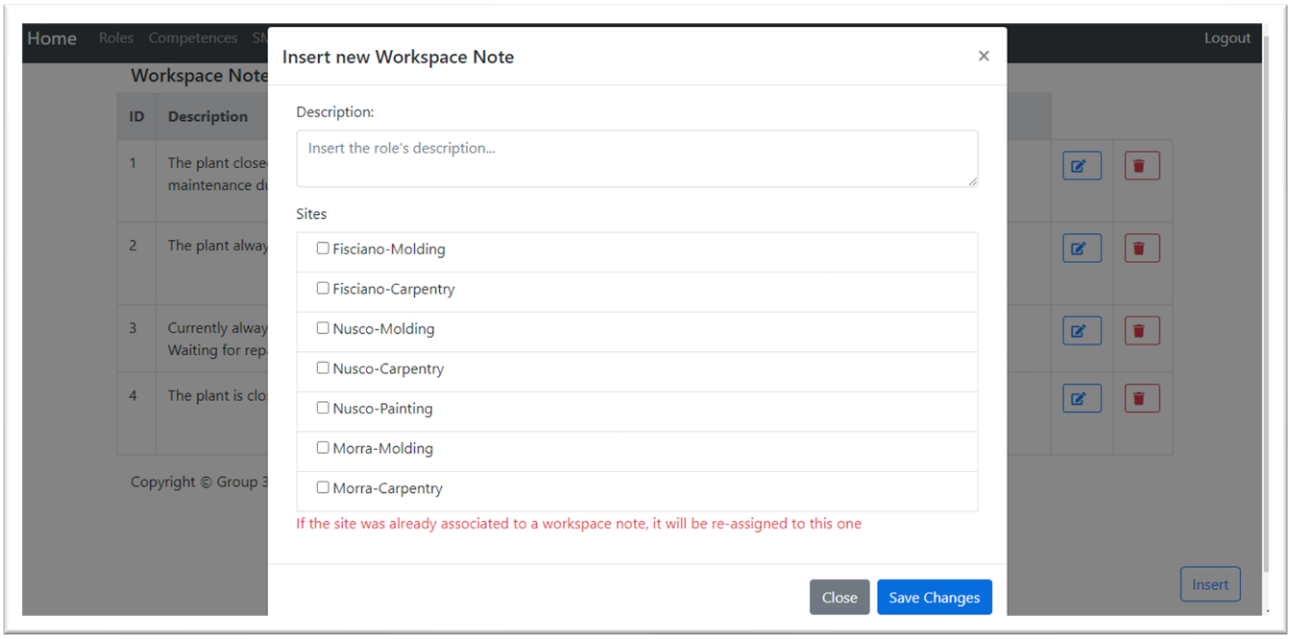
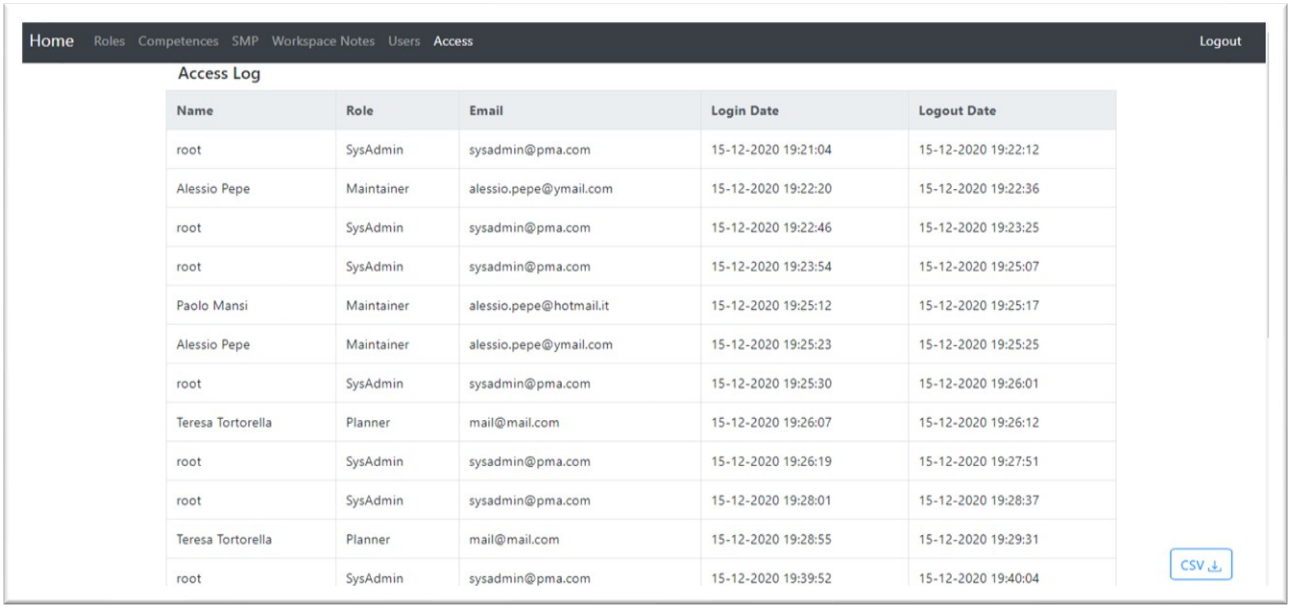


Figure 6.11 - Workspace Notes Mange Insert Modal



Name	Role	Email	Login Date	Logout Date
root	SysAdmin	sysadmin@pma.com	15-12-2020 19:21:04	15-12-2020 19:22:12
Alessio Pepe	Maintainer	alessio.pepe@ymail.com	15-12-2020 19:22:20	15-12-2020 19:22:36
root	SysAdmin	sysadmin@pma.com	15-12-2020 19:22:46	15-12-2020 19:23:25
root	SysAdmin	sysadmin@pma.com	15-12-2020 19:23:54	15-12-2020 19:25:07
Paolo Mansi	Maintainer	alessio.pepe@hotmail.it	15-12-2020 19:25:12	15-12-2020 19:25:17
Alessio Pepe	Maintainer	alessio.pepe@ymail.com	15-12-2020 19:25:23	15-12-2020 19:25:25
root	SysAdmin	sysadmin@pma.com	15-12-2020 19:25:30	15-12-2020 19:26:01
Teresa Tortorella	Planner	mail@mail.com	15-12-2020 19:26:07	15-12-2020 19:26:12
root	SysAdmin	sysadmin@pma.com	15-12-2020 19:26:19	15-12-2020 19:27:51
root	SysAdmin	sysadmin@pma.com	15-12-2020 19:28:01	15-12-2020 19:28:37
Teresa Tortorella	Planner	mail@mail.com	15-12-2020 19:28:55	15-12-2020 19:29:31
root	SysAdmin	sysadmin@pma.com	15-12-2020 19:39:52	15-12-2020 19:40:04

Figure 6.12 – Access Page

	A	B	C	D	E
1	email	name	role	login_date	logout_date
2	sysadmin@pma.com	root	SysAdmin	15/12/2020 19:21	15/12/2020 19:22
3	alessio.pepe@ymail.com	Alessio Pepe	Maintainer	15/12/2020 19:22	15/12/2020 19:22
4	sysadmin@pma.com	root	SysAdmin	15/12/2020 19:22	15/12/2020 19:23
5	sysadmin@pma.com	root	SysAdmin	15/12/2020 19:23	15/12/2020 19:25
6	alessio.pepe@hotmail.it	Paolo Mansi	Maintainer	15/12/2020 19:25	15/12/2020 19:25
7	alessio.pepe@ymail.com	Alessio Pepe	Maintainer	15/12/2020 19:25	15/12/2020 19:25
8	sysadmin@pma.com	root	SysAdmin	15/12/2020 19:25	15/12/2020 19:26
9	mail@mail.com	Teresa Tortorella	Planner	15/12/2020 19:26	15/12/2020 19:26
10	sysadmin@pma.com	root	SysAdmin	15/12/2020 19:26	15/12/2020 19:27
11	sysadmin@pma.com	root	SysAdmin	15/12/2020 19:28	15/12/2020 19:28
12	mail@mail.com	Teresa Tortorella	Planner	15/12/2020 19:28	15/12/2020 19:29
13	sysadmin@pma.com	root	SysAdmin	15/12/2020 19:39	15/12/2020 19:40
14	sysadmin@pma.com	root	SysAdmin	15/12/2020 19:50	15/12/2020 19:50
15	mail@mail.com	Teresa Tortorella	Planner	15/12/2020 19:50	15/12/2020 19:51
16	alessio.pepe@ymail.com	Alessio Pepe	Maintainer	15/12/2020 19:51	15/12/2020 19:51
17	alessio.pepe@hotmail.it	Paolo Mansi	Maintainer	15/12/2020 19:51	15/12/2020 19:51
18	mail@mail.com	Teresa Tortorella	Planner	15/12/2020 19:52	15/12/2020 19:52
19	mail@mail.com	Teresa Tortorella	Planner	15/12/2020 19:52	15/12/2020 19:52
20	sysadmin@pma.com	root	SysAdmin	15/12/2020 19:52	15/12/2020 19:52
21	sysadmin@pma.com	root	SysAdmin	15/12/2020 19:52	15/12/2020 19:54
22	mail@mail.com	Teresa Tortorella	Planner	15/12/2020 19:54	15/12/2020 19:55
23	sysadmin@pma.com	root	SysAdmin	15/12/2020 20:01	15/12/2020 20:01
24	mail@mail.com	Teresa Tortorella	Planner	15/12/2020 20:01	15/12/2020 20:02
25					

Figure 6.13 - Downloaded CSV File

Planned Activity Manage Page

The page that allows to view, create, edit, and delete the planned activities is shown in Figure 6.14.

Clicking on the button “Insert”, a modal for the insertion of new planned activity will appear, as shown in Figure 6.15. Filling out the required fields and clicking on the “Save Changes” button will confirm the insertion while clicking on the “Close” button or outside of the modal will make it close without changes. A banner with the result of the insertion action will then appear on the top of the screen.

Clicking on the edit button will make appear a modal for editing, similar to the one shown in Figure 6.15, filled out with the data of the related row. Any edit of those data will be saved after clicking on the “Save Changes” button.

Clicking on the delete button will make appear a modal for deletion. Then click on the “Confirm” button will confirm the deletion of the element.

6.3.8 Planned Activity Selection Page

The page that allows to view planned activities by the week and select them is shown in Figure 6.16.

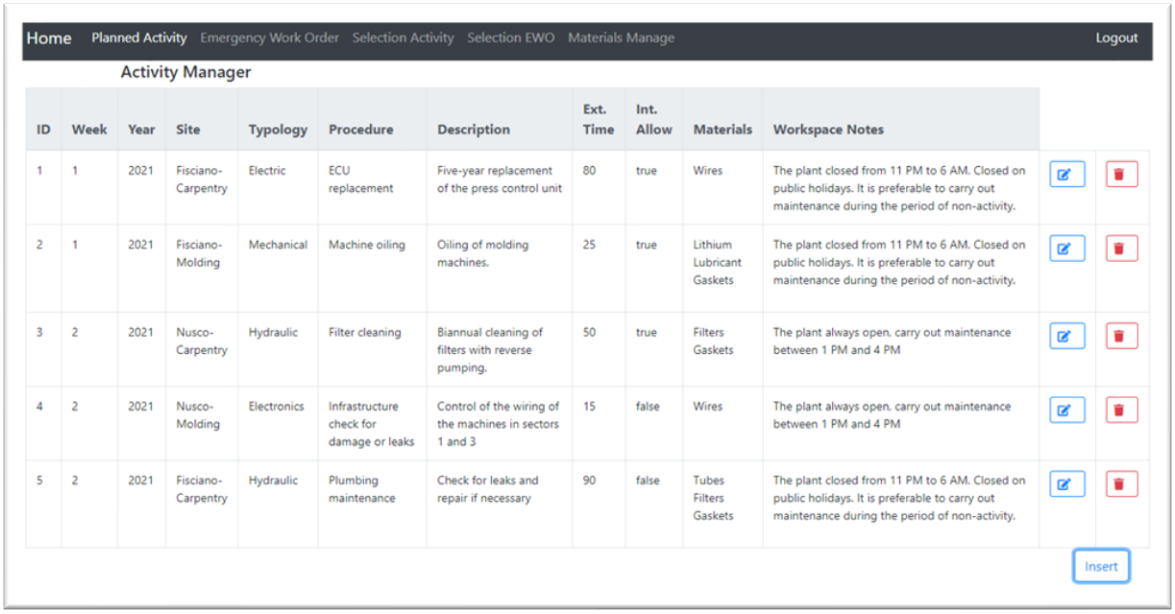


Figure 6.14 - Planned Activity Manage Main Page

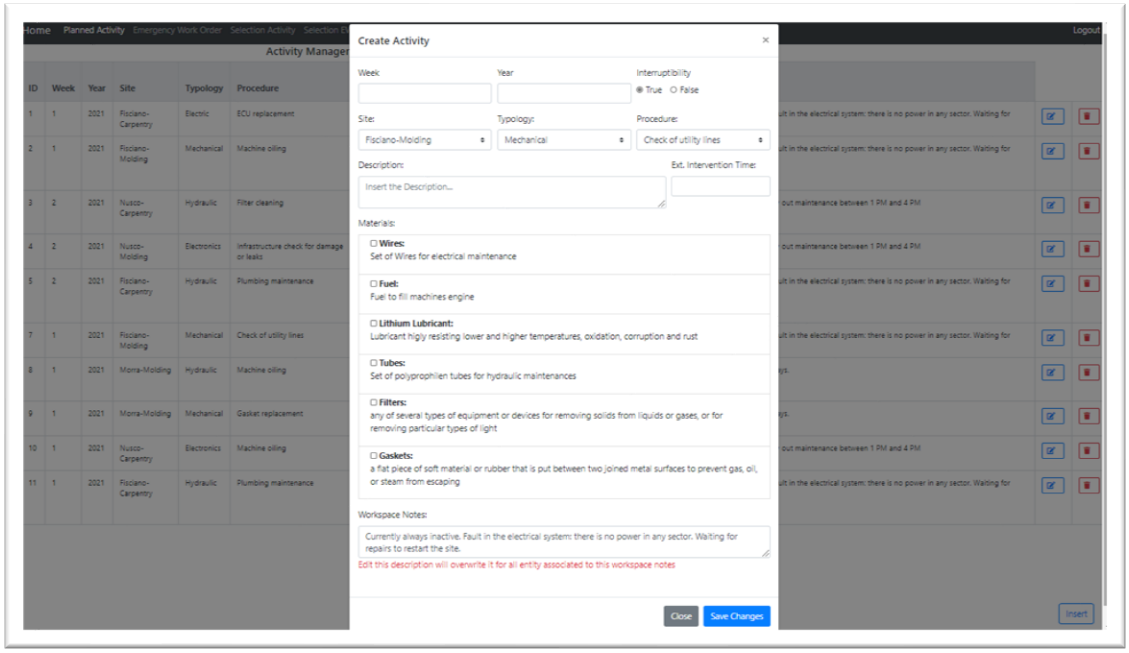


Figure 6.15 - Planned Activity Manage Insert Modal

Clicking on the button "Select", a modal displaying all the information of the related planned activity will appear, as shown in Figure 6.17, where only the workspace note's field is editable. From this modal it is also possible to download the SMP file associated to the procedure, clicking on the file icon.

Clicking on the "Forward" button, the system will redirect you to the page of assignment of the activity to a maintainer.

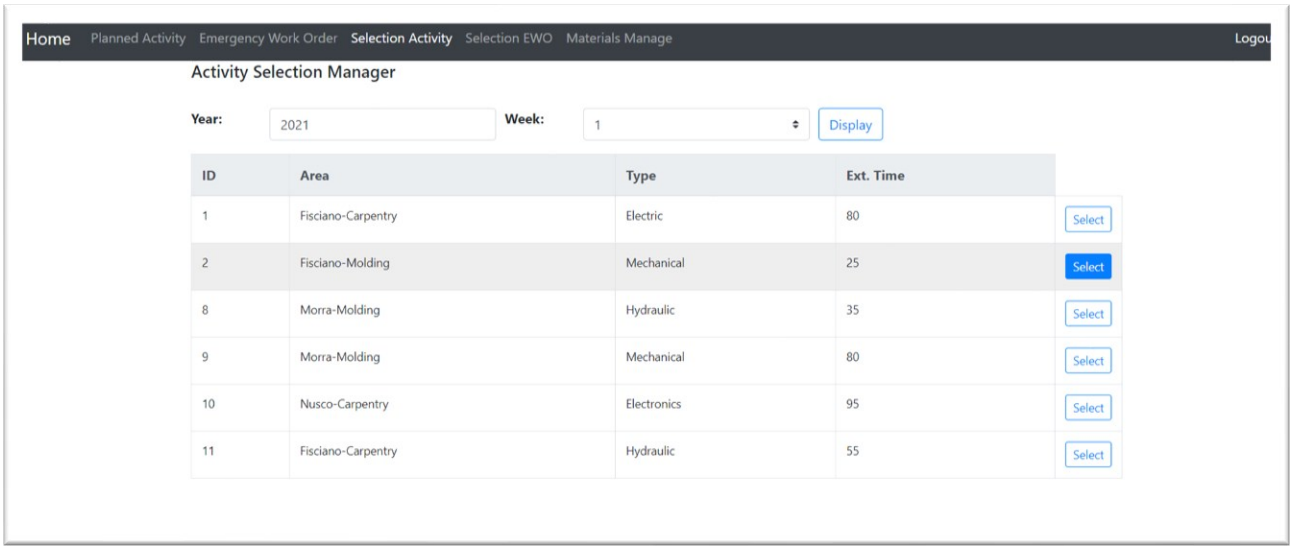


Figure 6.16 - Planned Activity Selection Main Page

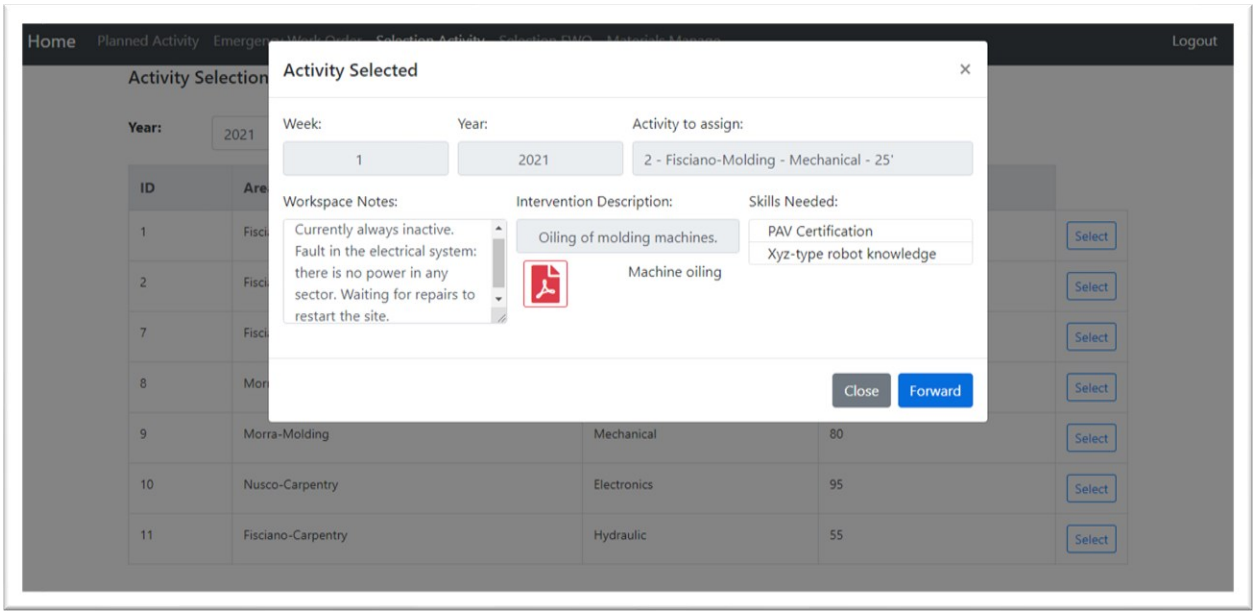


Figure 6.17 - Planned Activity Selection Select Modal

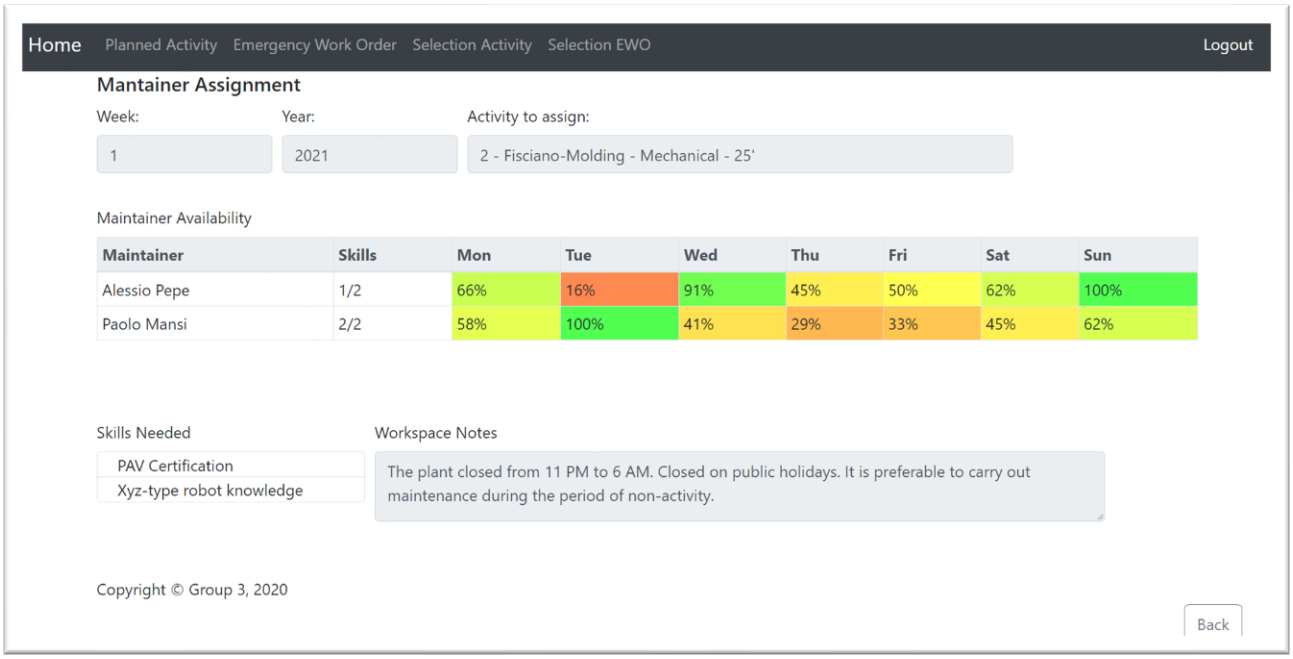


Figure 6.18 - Planned Activity Maintainer Assignment Week Table

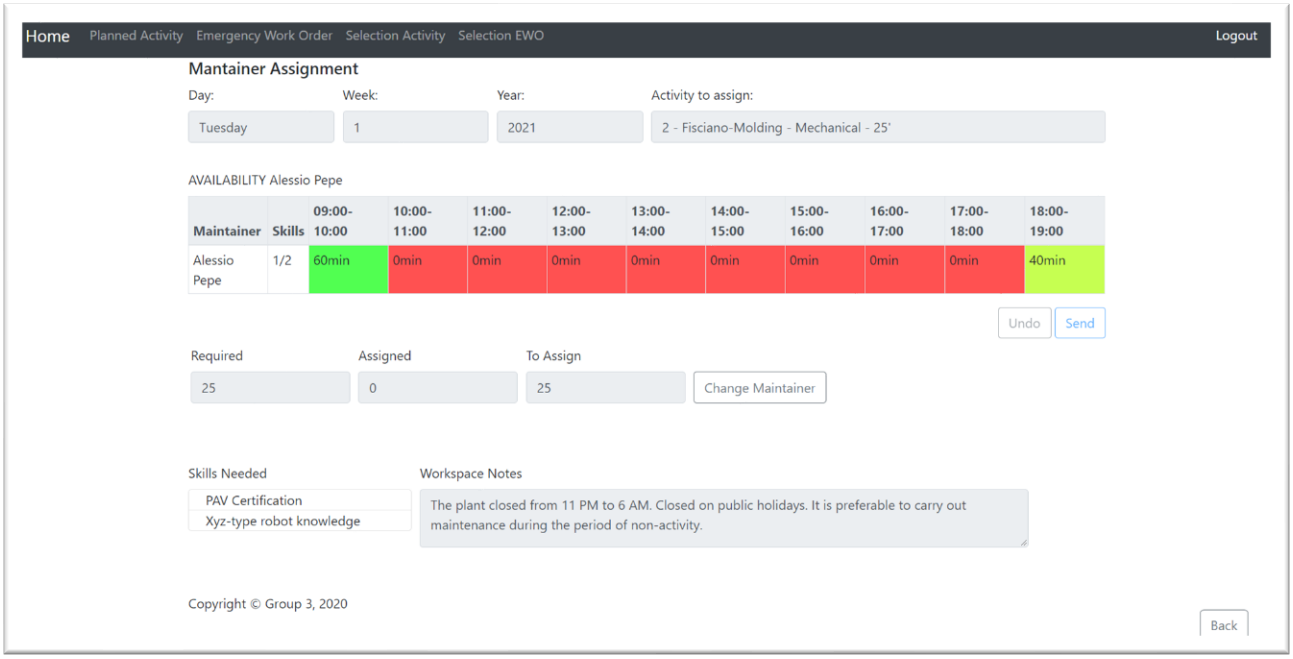


Figure 6.19 - Planned Activity Maintainer Assignment Daily Table (1): This Table is shown after the day "Tuesday" of the Maintainer "Alessio Pepe" is selected.

6.3.9 Maintainer Assignment of Planned Activities Page

This page allows assigning a maintainer to a planned activity. When the page is opened, the weekly availability of all the maintainers is displayed, as shown in Figure 6.18. Clicking on a day of a specific maintainer, its daily availability will be displayed, as shown in Figure 6.19. The total amount of minutes to assign can be selected among the displayed time slots. Once all the minutes have been assigned (Figure 6.20), clicking on the button "Send" will confirm the assignment and redirect to the previous page. Clicking on the "Undo" button will let you reassign the minutes differently. Clicking on the "Change Maintainer" button will allow you to change the selected day and maintainer. When an activity is assigned to a maintainer, the system will send him an email, as shown in Figure 6.30 - Prototype of the email sent to the maintainer

6.3.10 EWO Manage Page

The page that allows to view, create, edit, and delete the tickets of the EWO activities is shown in Figure 6.21.

Clicking on the button "Insert", a modal for the insertion of a new ticket will appear, as shown in Figure 6.22. Filling out the required fields and clicking on the "Save Changes" button will confirm the insertion while click on the "Close" button or outside of the modal will make it close without changes. A banner with the result of the insertion action will then appear on the top of the screen.

Clicking on the edit button will make appear a modal for editing, similar to the one shown in Figure 6.22 but filled out with the data of the related row. Any edit of those data will be saved after clicking on the “Save Changes” button.

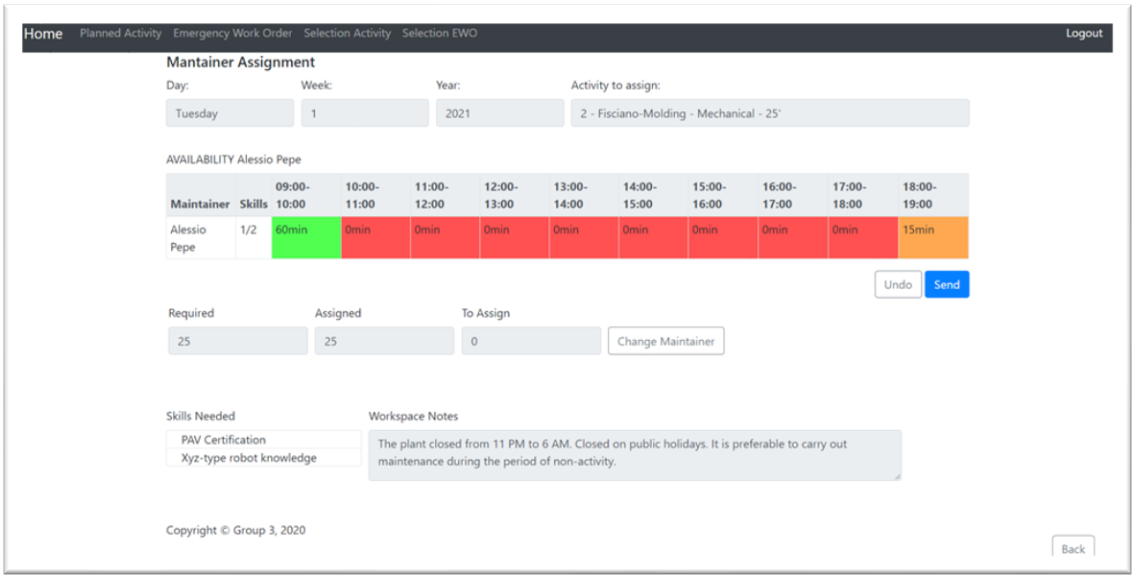


Figure 6.20 - Planned Activity Maintainer Assignment Daily Table (2): This Table is shown after the "18:00-19:00" slot is selected.

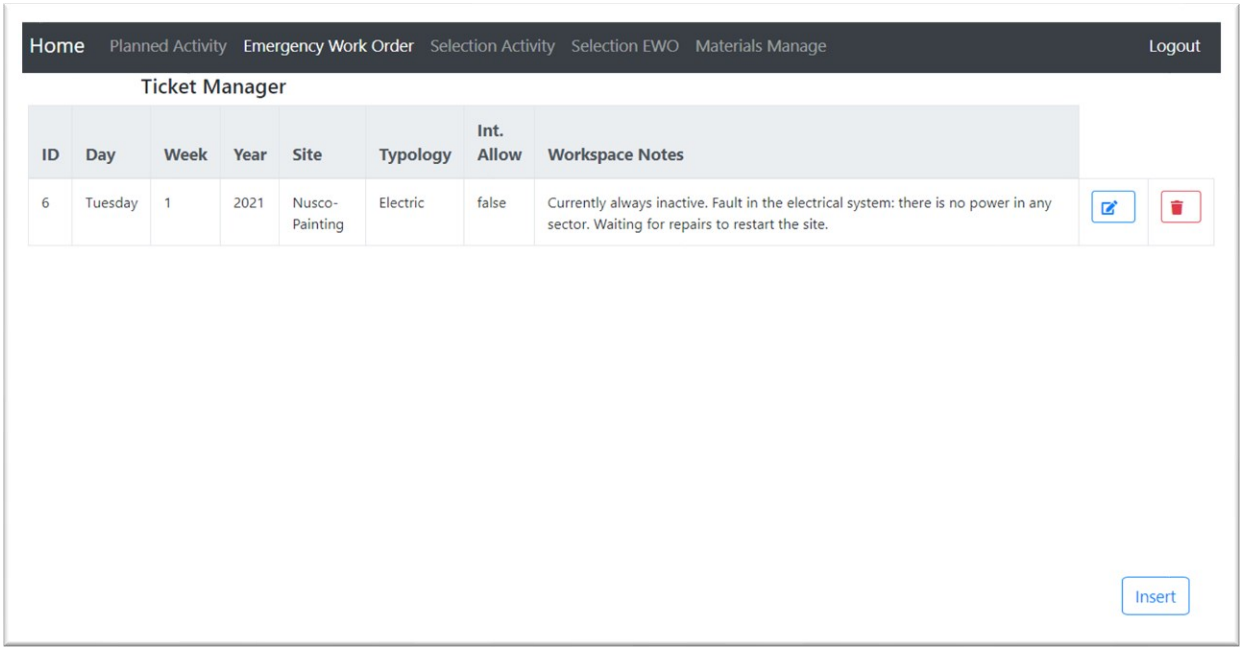


Figure 6.21 - EWO Ticket Manage Main Page

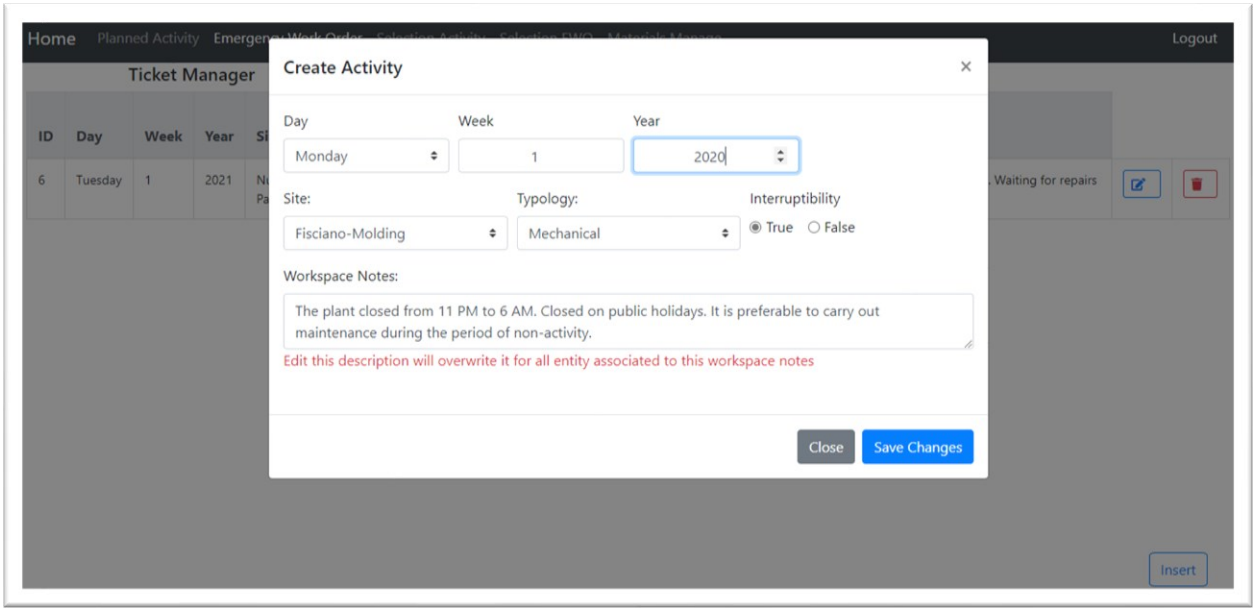


Figure 6.22 - EWO Ticket Manage Insert Modal

6.3.11 EWO Activity Selection Page

The page that allows to view Emergency Work Orders by the day and select them is shown in Figure 6.23.

It is possible to select a specific day, week, and year to display all the emergency work orders associated to that day.

Clicking on the button “Select”, a modal displaying all the information of the related EWO activity will appear, as shown in Figure 6.24. From this modal, it is possible to modify the Intervention Description, the Estimated Intervention Time, the skills needed, the interruptibility and the materials required.

Clicking on the “Forward” button, the system will redirect you to the page of assignment of the activity to a maintainer.

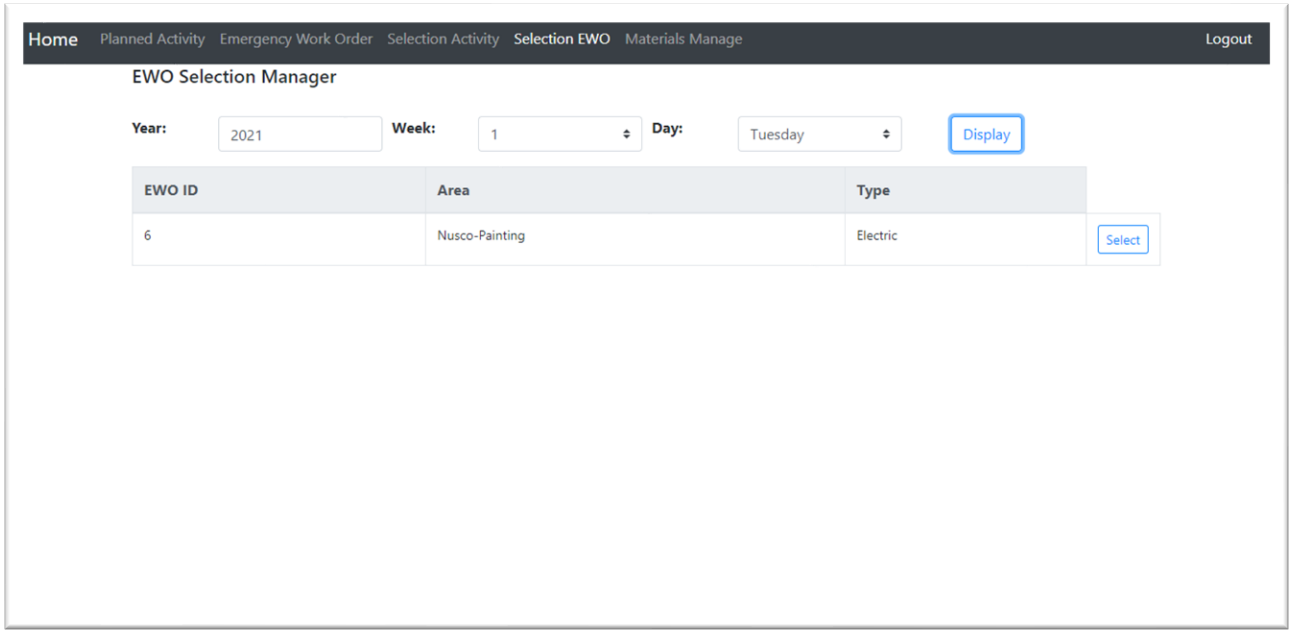


Figure 6.23 – EWO Selection Main Page

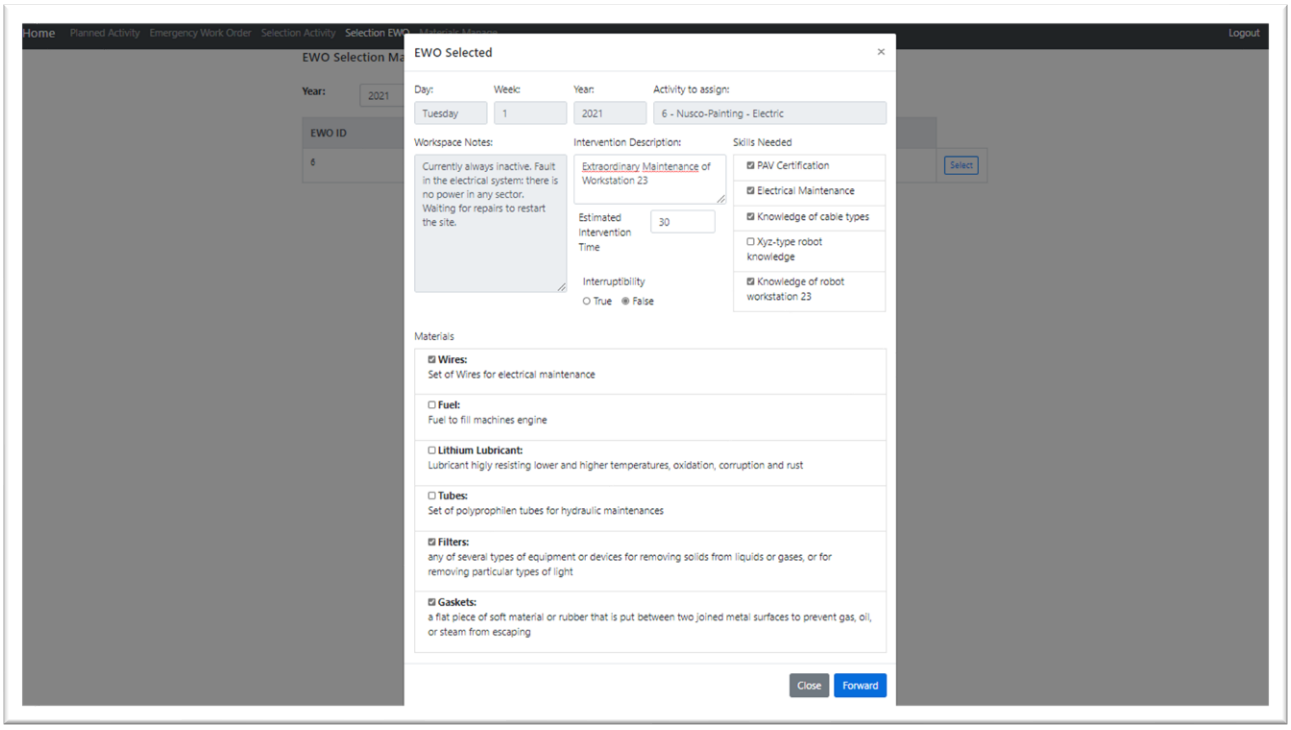


Figure 6.24 - EWO Selection Select Modal

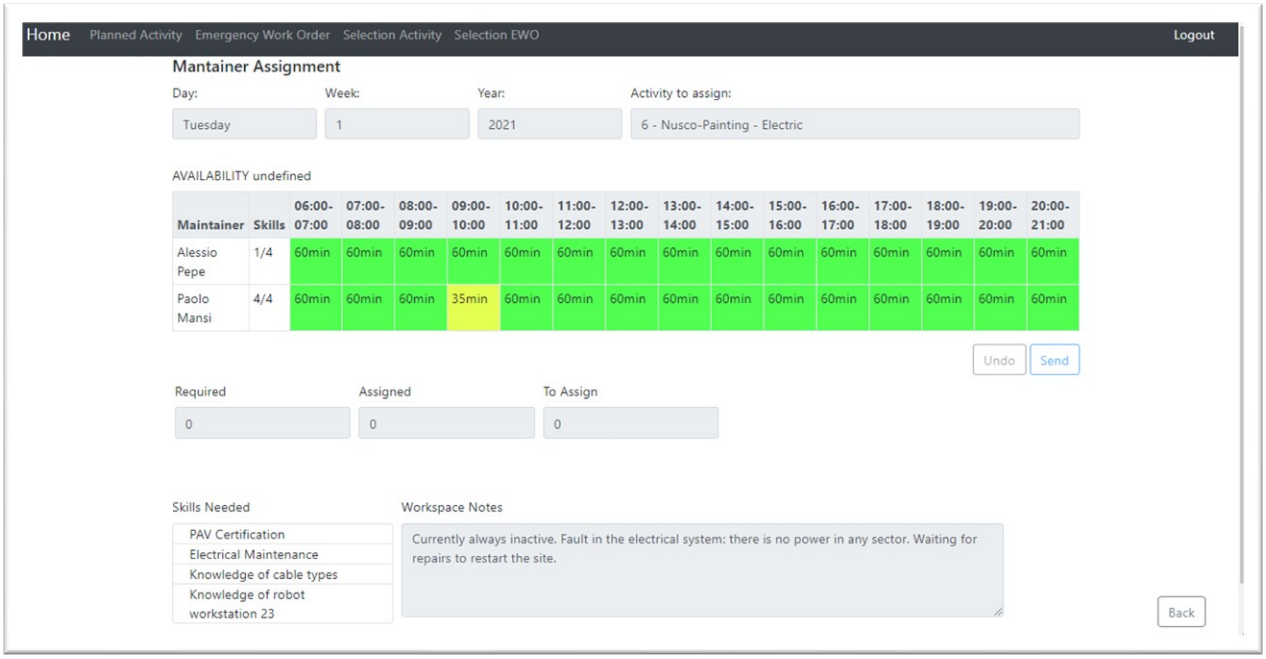


Figure 6.25 - EWO Activity Maintainer Assignment Daily Table (1)

6.3.12 Maintainer Assignment of EWO Activity Page

This page allows assigning a maintainer to an EWO activity. When the page is opened, the daily availability of all the maintainers is displayed, as shown in Figure 6.25. The total amount of minutes to assign can be selected among the displayed time slots. Once all the minutes have been assigned (Figure 6.26 - Figure 6.27), clicking on the button “Send” will confirm the assignment and redirect to the previous page. Clicking on the “Undo” button will let you reassign the minutes differently. When an activity is assigned to a maintainer, the system will send him an email, as shown in Figure 6.30 - Prototype of the email sent to the maintainer

6.3.13 Materials Manage Page

The page that allows to view, create, edit, and delete the materials required for the activity is shown in Figure 6.28.

Clicking on the button “Insert”, a modal for the insertion of new material will appear, as shown in Figure 6.29. Filling out the required fields and clicking on the “Save Changes” will confirm the insertion while clicking on the “Close” button or outside of the modal will make it close without changes. A banner will then appear on the top of the screen, showing the result of the insertion.

Clicking on the edit button will make appear a modal for editing, similar to the one shown in Figure 6.29, filled out with the data of the related row. Any edit of those data will be saved after clicking on the “Save Changes” button.

Clicking on the delete button will make appear a modal for deletion. Then click on the “Confirm” button will confirm the deletion of the element.

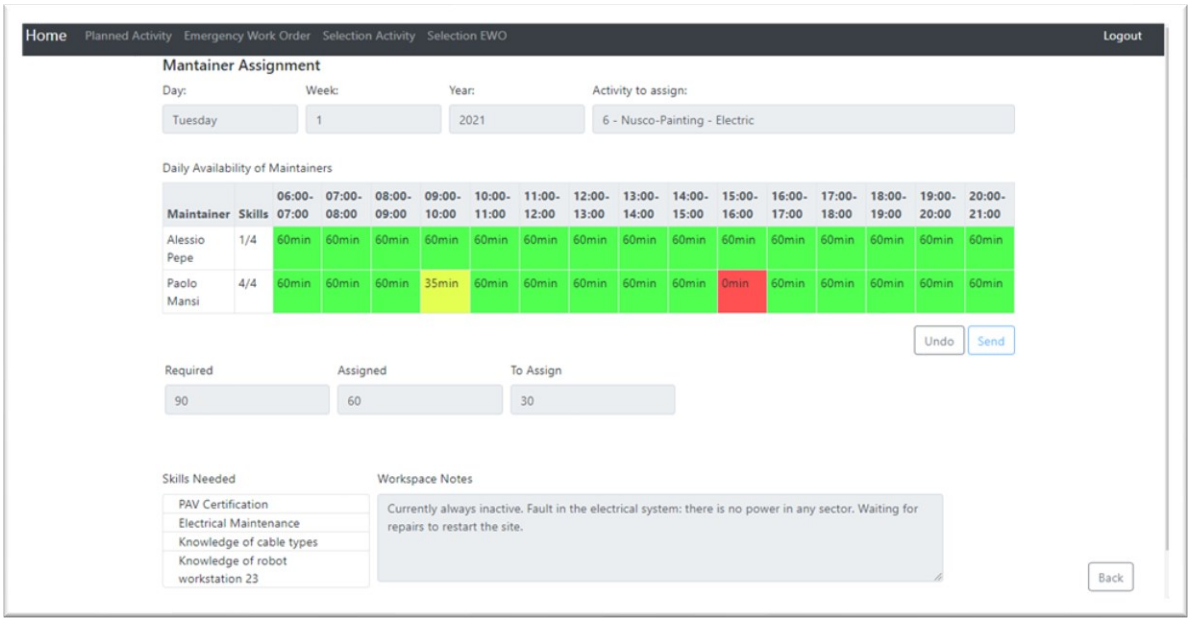


Figure 6.26 - EWO Activity Maintainer Assignment Daily Table (2): This Table is shown after the “15:00 – 16:00” slot of the maintainer “Paolo Mansi” is selected.

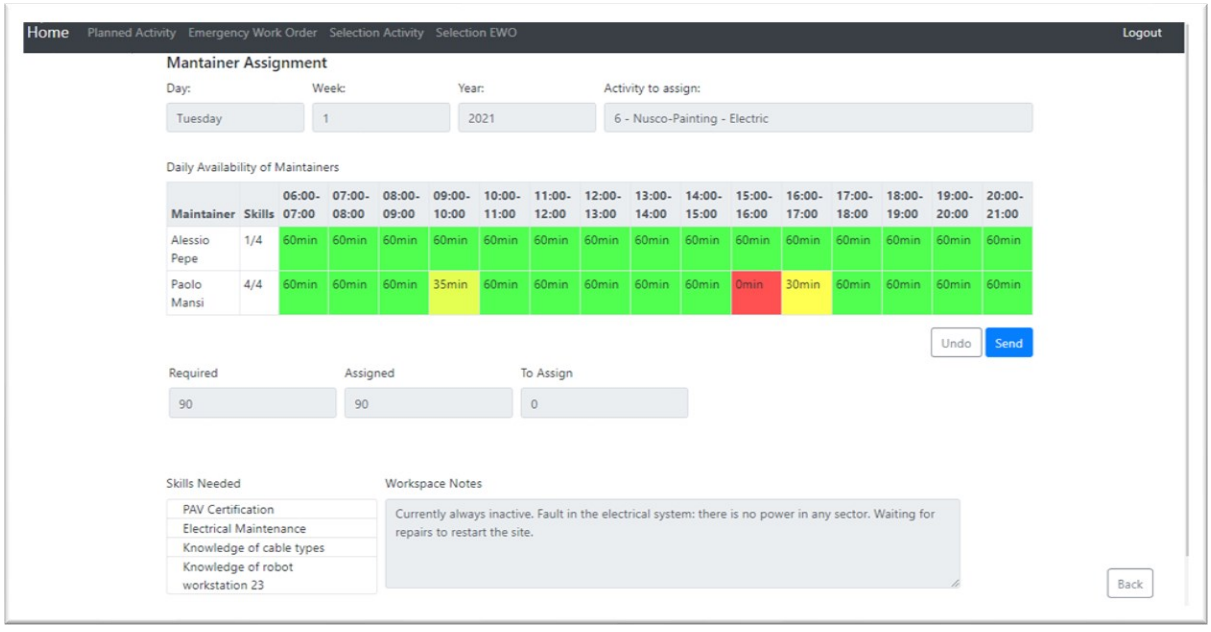


Figure 6.27 - EWO Activity Maintainer Assignment Daily Table (3): This Table is shown after the “16:00 – 17:00” slot of the maintainer “Paolo Mansi” is selected.

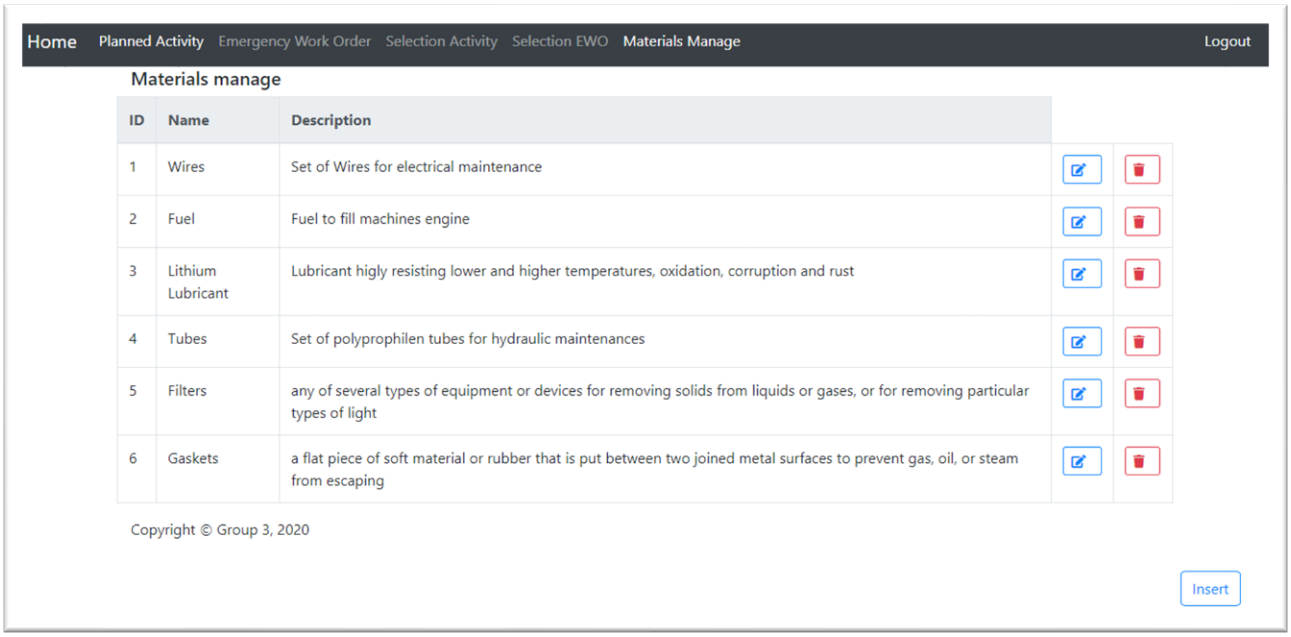


Figure 6.28 - Materials Manage Main Page

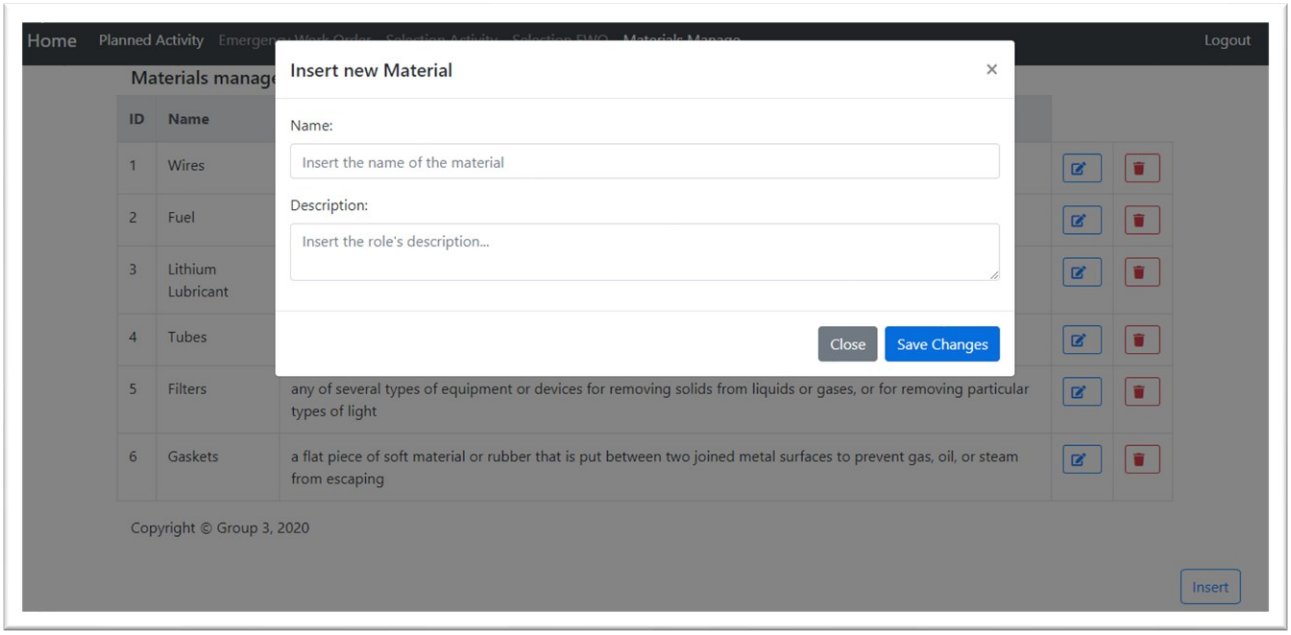


Figure 6.29 - Materials Manage Insert Modal

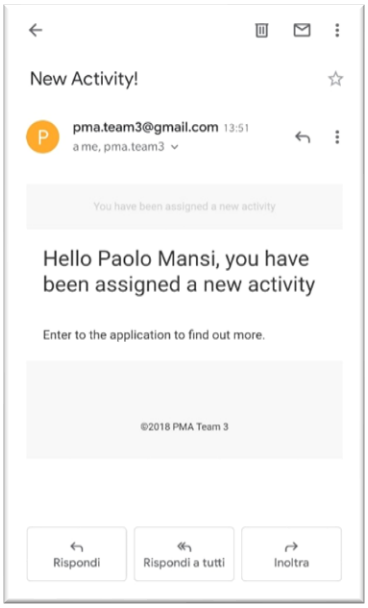


Figure 6.30 - Prototype of the email sent to the maintainer