Fall 2020

RETAIL HEALTH & SAFETY DIGITAL MANAGEMENT SYSTEM

Anderson Oliveira Gomes

David Carvalho

Henrique Atanasio Wegner

cct | College Dublin
Computing • IT • Business

RETAIL HEALTH & SAFETY DIGITAL MANAGEMENT SYSTEM

Supervisor: Mark Morrissey

Students: Anderson Oliveira Gomes (2017330), David Carvalho (2017368), Henrique Atanasio Wegner (2017403)

Date: 19th of May 2020.

ABSTRACT

Project: Retail health & safety digital management system.

Students: Anderson Oliveira Gomes, David Carvalho, Henrique Atanasio Wegner.

Degree: Bachelor of Science in Information Technology.

CCT College Dublin

Health and Safety procedures and documentations are one of the many tasks that need to be done by a small retail shop, they are also one of the most difficult things to manage daily as all those can build up during the day/week. Inside Health and Safety there are many subcategories that need to be administered (maintenances, trainings and checklists for example) which can be very overwhelming for any management team notably in small retail shops as they do not have the permanent personnel to be able to maintain an organised system. .

This project aims to create a management software to assist a business to conduct all those procedures in an easier and more efficient manner.

Key Words: retail, health and safety, management system, management software.

ACKNOWLEDGMENTS

We would like to express our gratitude to our supervisor Mark Morrissey for guiding and encouraging us through our final project. We would also like to thank the Spar Ellis Quay management team, Alan Kealy, Camila Oliveira and Siti Bakri, who assisted us and let us do the project based on the shop. For last, we want to thank our friends and family for all the support during these hard times, especially the year 2020 which created many barriers for everyone's lives.

TABLE OF CONTENTS

Chapter 1 - Introduction

1.0 Health and Safety Control System

The legislation concerning Health and Safety (H&S) in the workplace has evolved along the years. It is imperative that the business owners take all necessary measures to keep the health and safety of their employees as well as their customers or any person that may be affected by their work activities. The government agencies responsible for policing H&S have been developing and improving the best practices, tools and standards that companies must adopt according to their necessity and specialty, also which customized trainings their employees must take, so that the companies may control workplace hazards providing a safe environment to their employees and customers. As Fitzgerald (2015) states:

> Since 1989, the fatal workplace accident rate has fallen from around 6 per 100,000 to about 2.5, a testament to the efforts of the Health and Safety Authority, and to a greater sense of awareness and education within workplaces.

Although every business must abide by the H&S legislation, it is clear that a great number of companies still underestimate the value of an effective H&S management. According to Coleman (2018) 137 workers died in the workplace in 2017, the number of injuries to employees reported totalled 70,116 and the number of working days lost due to work-related illness and workplace injury were 31.2m. The numbers show us how much H&S is undervalued or overlooked. Furthermore, not failing to mention the loss of profits due to indirect costs because of poor H&S procedures. The indirect costs may include, but not limited to time away from the job not paid by the insurance, damaged equipment or properties involved in the accident, medical bills paid by the employer, legal fees with prosecution fines due to enforcement action, increase in insurance premium, cost of hiring and training new staff or recruiting temporary agency staff as a result of employee absenteeism, damage to the companies public image, loss of customers due to negative publicity. (OHS, 2008)

There is clear evidence that H&R management contributes to overall business success. It protects the employers and their staff from injury and illness that may occur due to activities in the workplace. It reduces absences as work-related illness and injuries are diminished. It improves productivity as employer and employees work in a safe environment. It saves money with a low absenteeism rate and turnover as the company will not be spending on recruitment processes and training to new employees. Also, it prevents business owners from being prosecuted due to the occurrence of accidents in the business premises causing damage to their brand and reputation as well as losses with increased insurance premiums or compensation claims.

According to Citizen Information the safety, health and welfare at work Act 2005 under Section 8:

Employer's duties

Under Section 8 of the Act the employer has a duty to ensure employee's safety, health and welfare at work as far as is reasonably practicable. To prevent workplace injuries and ill-health the employer is required, among other things, to:

- Provide and maintain a safe workplace which uses plant and equipment.
- Prevent risks from use of any article or substance and from exposure to physical agents, noise, and vibration.
- Provide instruction and training to employees on health and safety.
- Provide protective clothing and equipment to employees.
- Appointing a competent person as the organization's Safety Officer.

Employees' duties

The duties of employees while at work are set out in Section 13 of the Act. These include the following:

- To take reasonable care to protect the health and safety of themselves and of other people in the workplace.
- Not to engage in improper behaviour that will endanger themselves or others.
- Not to be under the influence of drink or drugs in the workplace.
- To undergo any reasonable medical or other assessment if requested to do so by the employer.
- To report any defects in the place of work or equipment which might be a danger to health a safety (Citizen Information, 2018).

Now that we have an idea of how important H&S management for business success is, we may discuss the purpose of this project. One of our team members works for a private owner who owns three units of the largest convenience retail groups in Ireland and the largest grocery chain in the world well known as SPAR. It is a franchise that manages independently owned food retail stores. Once a SPAR franchise is bought, the franchisee must abide by the standards and certain controls set by the franchisor and by the local zoning rules, but how to manage and keep the standards is totally up to the franchise owner. One of the standards, also legally required, that the franchise owner must keep is a good H&S management. Currently all the documentation related to the H&S is paper-based handled and a system is required to facilitate their work. It will be discussed in more details below in the problem area and solution.

1.1 Problem Area

Although SPAR is a big company, its franchises are small convenience stores independently managed by their private owner. Usually, the largest companies have a dedicated department that handles Hazard Analysis & Critical Control Point (HACCP) responsible for health and safety. There is a lot of paperwork that must be done and presented to the government agency responsible for H&S. It needs people and time to keep everything well controlled. This controlling may or may not cost a lot of money for a big company, but certainly it is a lot of money for a small business. So, the franchisee must keep track of the H&S documentation, but in a cost-effective way without a dedicated department.

There are many little details and procedures that the company need to comply with to be able to keep high H&S standards but there are many other parts of the shop that too need to be addressed such as sales, security, stock control and Human Resources which makes it a lot difficult to be able to manage every single detail from all of these areas especially when most of them are all done in paper.

According to the HSA (Health and Safety Authority, 2019) “The retail sector is Ireland’s largest, employing more than 270,000 people. Hazards associated with a retail environment are often thought to be low risk, but accidents remain common and can inflict significant extra costs on a business”.

The legal requirement for employers to manage health and safety applies across the retail sector, from large distribution centres to self-employed shopkeepers. As in any workplace the hazards faced by employees must be identified, assessed and controlled. Every single place must abide by these requirements and is no different for a small retail business and it needs to be assessed by its own employees.

There are no digital H&S management systems now, all is written and controlled on paper. Time management services and calendars (like Google Calendar) are good for your daily life but not to manage a whole shop, principally when there are so many areas and procedures to be reminded of every day/week/month. There are trainings that can be done online, but they are many and sourced from different websites and companies which becomes very complicated to organise and make sure that all trainings are done.

1.2 Solution

One of our group members is an assistant manager in a Spar shop and is responsible for the H&S of the shop. With these problem in mind we decided to design/create a program in which makes it easier for the staff and management to keep track of all these H&S tasks that need to be done. It will work as a task manager/schedule, in which we can control what has been done and what needs to be done.

The system would include the main areas that need to be managed (trainings, checklists, maintenance, daily tasks) as separate menus and each of them would have their own functions and ways to be managed. An extra menu for reports would be created to gather all the information from these main areas so the management team can customise and keep track of the procedures.

At the end of the day/week/month the system would create a report with everything that still needs to be done to comply with H&S standards that might have been forgotten or simply needs to be reminded of it.

1.3 Project Goals

The main goal of the project is to create a retail H&S management system in the form of a local desktop application that will be used commercially in a retail shop in Ireland, in which it can be scalable for other shops. The system will help the management team and the owner to keep track of the most important health and safety procedures that need to be assessed.

1.4 Project Objectives

Milestones are needed so we can achieve the main goal of the project. Our project objectives are:

- Analyse the requirements.
- Designing and planning of the desktop application.
- Building the software prototype using Java (Main programming language), Spring 5(Framework), JavaFX 2(GUI) and MySQL (Database).

1.5 Project Deliverables

This is what we will be delivering at the end of this project:

- A prototype of the project.
- Documentation for the development of the project.
- User manual.
- Screencast presentation of the program working.

1.6 Risk Assessment

According to Zolfaghari and Mousavi (2018, P.639) “Project risk management is an essential activity that refers to identify and analyse potential risks to take precautionary responses to mitigate their negative impacts or intensify their positive effects.”. Risk assessment is essential for any project, and building a software is no different, meaning that we need to plan for any risks that can occur during the planning and building of our software.

Barki, Rivard and Talbot (1993, p.204) states that: “The term risk is associated with many human endeavours be it space exploration, nuclear reactor construction, company acquisition, security evaluations of information systems, or information systems development.”. Which means that risk is in every project and it needs to be assessed from smaller projects to big ones as space exploration.

For a successful project Lavanya and Malarvizhi (2008) suggests that a structured risk assessment framework is in place which will help the way the team will handle the risks through many different steps, from risk identification, to risk evaluation, risk handling and risk controlling. There are many other guidelines between those, but this are the four most important and general ones.

From here we will start assessing the potential risks to our project through these table:

No.	Risk	Probability	Impact	Effect	Risk Reduction	Actions if it happens
1	Health problems in the team.	Medium	High	It can cause delays, changes in the requirements of the project or a member of the team leaving.	If getting sick, or in case anything happens communicate to the other team members.	In case of health emergencies: Sharing of responsibilities between remaining team members.
2	Disagreement in the team.	High	Medium	It can cause delays and changes in the requirements of the project.	Good communication between team members.	In case disagreement happens, the team members need to respectfully talk and get to a middle ground.
3	Changes in the project requirements.	High	Medium	Delay in the project deliverables.	Try to maintain a stable idea.	Fit changes in the project and dissect how it will affect the project.
4	Poor communication within the group.	High	Medium	Mistakes in research and development.	Keep everyone up to date on everything we do and how we need to do it.	Improve the communication in the areas where the mistake was made.
5	Change of supervisor.	Low	High	Delay in development/research.	Make sure that the supervisor is the right one for the project and	Look for another supervisor with CCT management.

					that everyone is ok with it.	
6	Changes in deliverable dates.	Low	Medium	Delivering earlier than expected or later than expected.	Keep the project on development in schedule.	Do extra hours or extra work for the respective cases.
7	Change in the H&S related laws.	Low	High	Change in procedures of H&S.	Keep up to date with H&S related changes in law.	Research how the changes will affect our project.
8	Losing documentation due to hardware/software failure	High	High	Need to re-do parts of the project.	Backing up all documents every time anything is done and uploading and saving to multiple places.	Use the backups done to recover work lost.
9	Shop owner disagreeing to the project.	Low	High	Project needs to be cancelled or modified.	Give updates to the business owner and maintaining contact with him throughout the project development.	Modify our project in a way we can reuse assets and research.
10	Users mishandling the software.	High	Low	Software crashes/bugs/glitches or deletion.	Make sure every part of the software is user friendly and working properly.	Identify if it is our fault or user fault and fix it.
11	CAs in excess.	High	Medium	Delays in development.	Keep on time with all CA’s deliverable dates and do them in advance.	Prioritise the one that needs to be done first and complete it as soon as possible.

Table 1 - Project Risk Assessment

2.0 System Requirements

2.0.1 Security

- The system will need a login page to ensure that only authorized people can access the information stored in it. The login would also control who is accessing, if it’s the owner, the manager, assistant manager or supervisor. It would also set permissions for whoever is accessing.

- If the login page is used, it would lead you to the Main Menu page.

2.0.2 Menu

- A menu with the most used pages of the system and maybe a summary of things that need to be done.

- From the main Menu you can go to the secondary Menus through the click of an icon.

2.0.3 Secondary Menus

We still have not decided which are going to be the secondary Menus, but we can have an idea. Some of them might become part of others and there might be new ones but for now we will have:

- Tasks
- Maintenance
- Checklists
- Trainings
- Reports
- Configuration

2.0.4 Tasks

- This would be a page that would do a summary of all the other pages, with everything that needs to be done (current day, week and month) and that were already done (current day, week, month and by date).
- It mainly would gather all data from the rest of the system and bring to this page as a summary in topics, so we can see it clearly instead of going through all pages and checking one by one.
- This page and the reports one would probably be the most important ones as it is the main objective of the system. To gather all these things and put it clearly in a simpler way.

2.0.5 Maintenance

- This page would involve the control of all maintenance done in the shop. From last time done, next time it needs to be done, name of the company and if it was already paid or not.
- Some of them are in automatic payment, and they are too far apart to check, so we end up paying for something we are not receiving.

2.0.6 Checklists

- We have a LOT of checklists that need to be done around the shop, from fire safety, to H&S, food safety etc.
- It is quite hard to keep track of all them as not everyone in the shop knows how to do them and even where they are. Checklists can be daily, weekly or monthly, making it even harder to remember all of them.

2.0.7 Trainings

- Trainings can be a nightmare to keep track of as staff changes constantly and there are many trainings in different areas (Deli, Floor, Chemicals, Manual Handling, Machinery etc).
- Trainings can be quick or can take up to 3 hours to be done, making it hard to keep track of some of them, or to find a good moment to do them.
- Some of them have a higher priority, and not everyone needs the same trainings (deli staff/floor staff/ management) making it even harder to have some control over it when some of them are online and some of them are made in paper/person.

2.0.8 Reports

- The report menu would be similar to the tasks one, but it would be customisable to choose only the areas that you want so you can print it or save as pdf.
- From this menu you can also set the reports that will be send to the email and when do you want it to be sent.

2.0.9 Configuration

- The configurations page can serve for both configuring the system and preferences. An example would be setting passwords and accounts and customising the interface.

2.1 Functionalities

2.1.0 Login

When the users start the application, two windows will open simultaneously. The main menu and the login window. The user will have access to the application only if it gets passed the authentication. The password must match the minimum of eight characters, at least 1 upper case, at least 1 lower case and at least one special character.

Flowchart login process

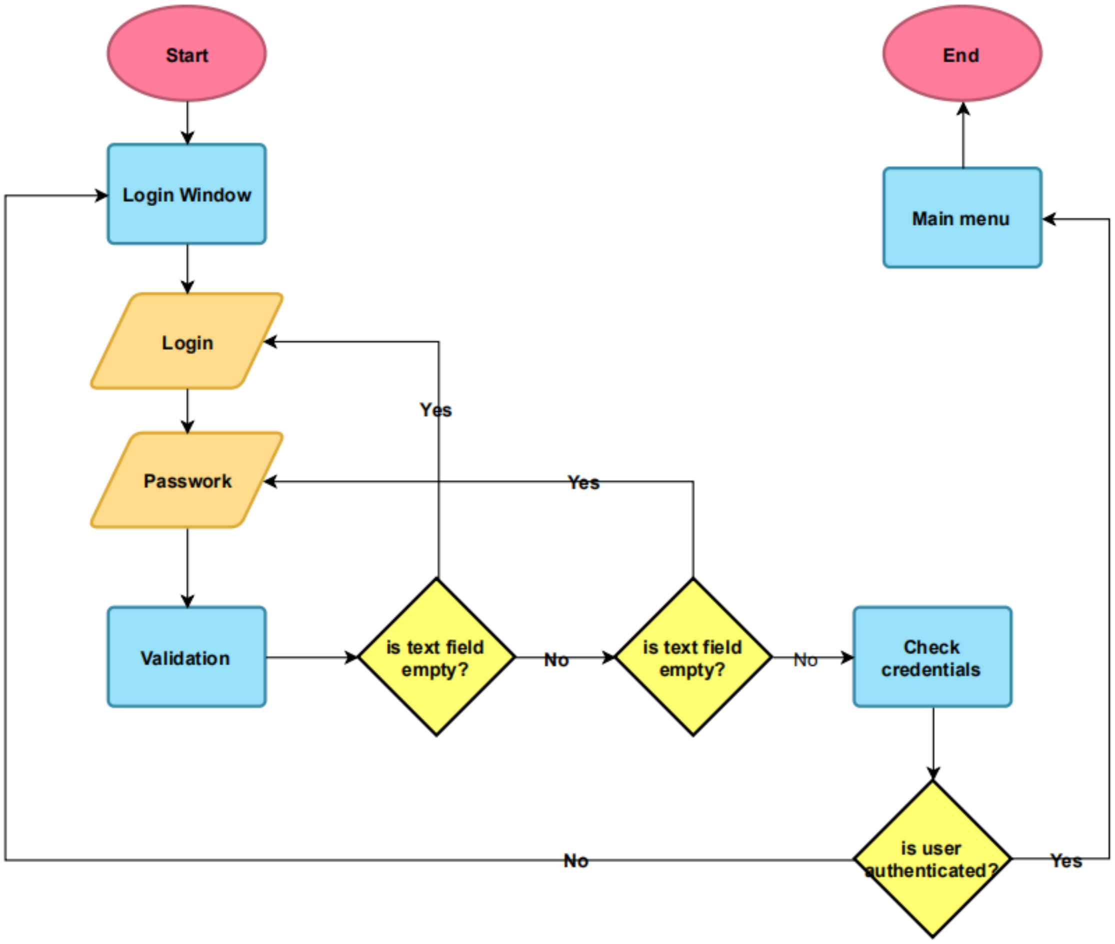


Figure 1 - Login flowchart

State diagram login process

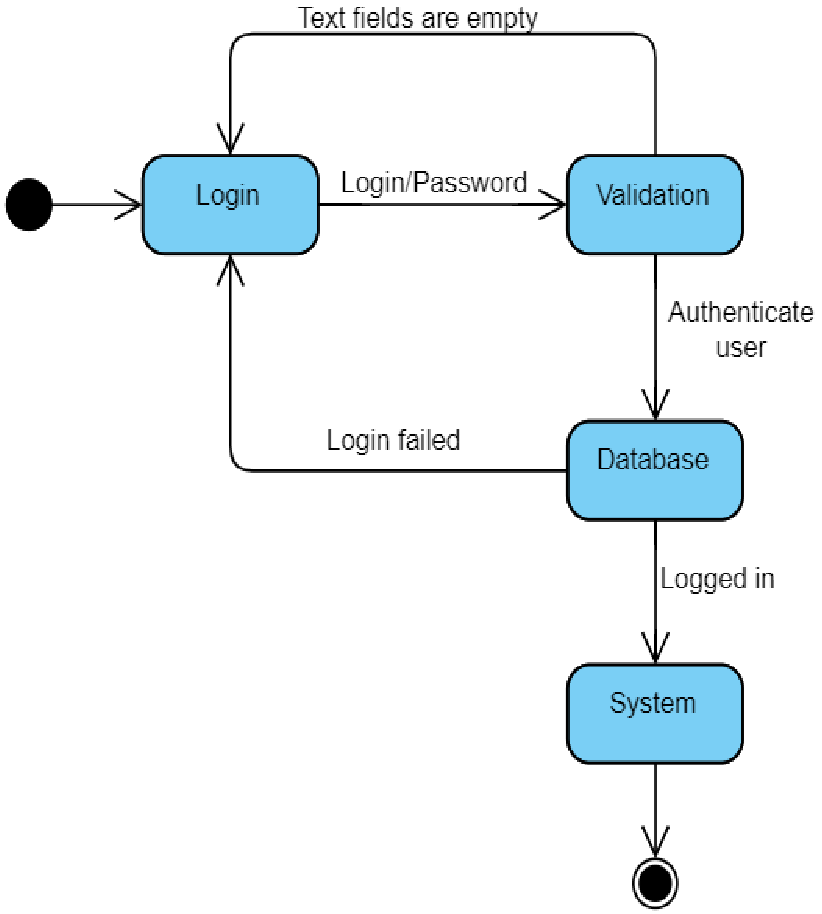


Figure 2 - Login state diagram

2.1.1 Add employee

The system must have an option that allows the manager/supervisor to add employees. These employees will be added to each training that they must take while working in the company. The employee must be composed of first name, last name, address, email, mobile, telephone, city/county, and date of birth. All fields are required except telephone.

Flowchart add employee process

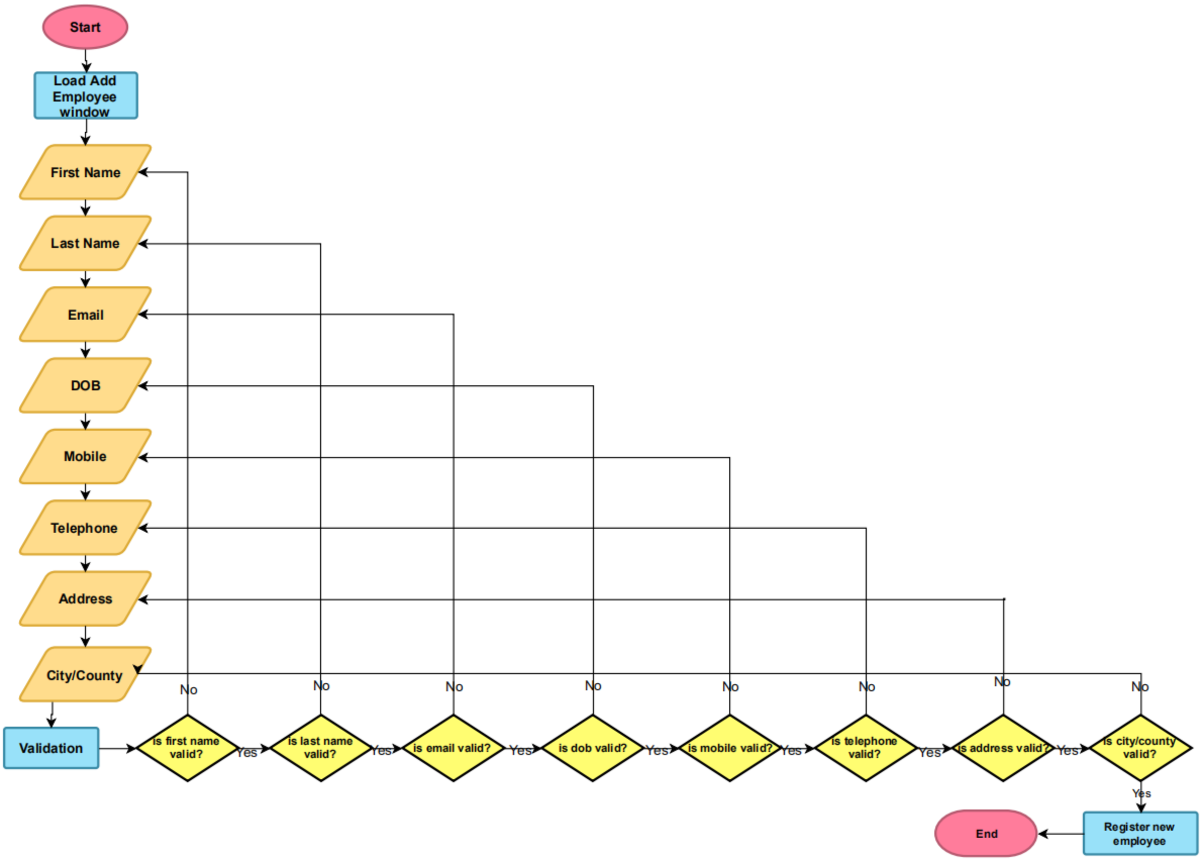


Figure 3 - Add employee flowchart

State diagram add employee

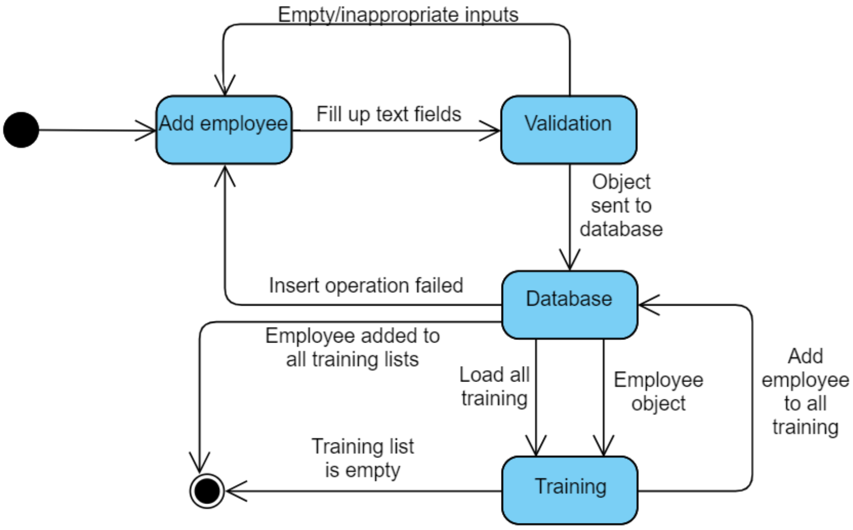


Figure 4 - Add employee state diagram

2.1.2 Update employee

There must be an option that allows the employee information to be updated. Update flowchart for this process can be seen in figure 7.

State diagram update employee process

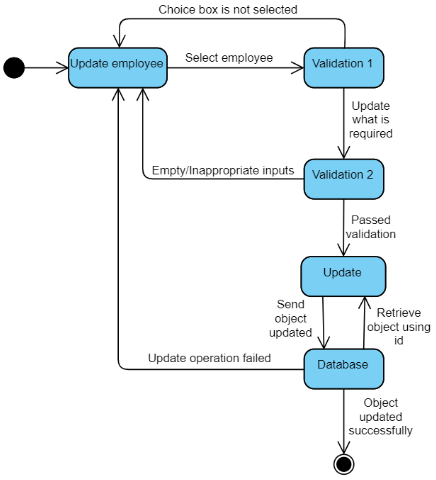


Figure 5 - Update employee state diagram

2.1.3 Delete employee

There must be an option to remove the employee from the system. Delete flowchart can be seen in figure7.

State diagram delete employee process

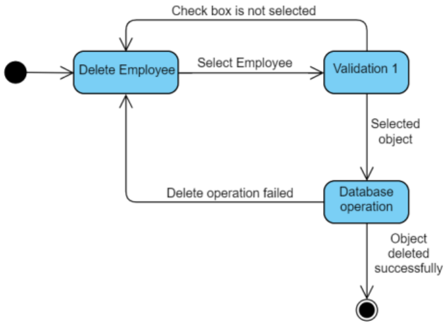


Figure 6 - Delete employee state diagram

Flowchart update and delete processes

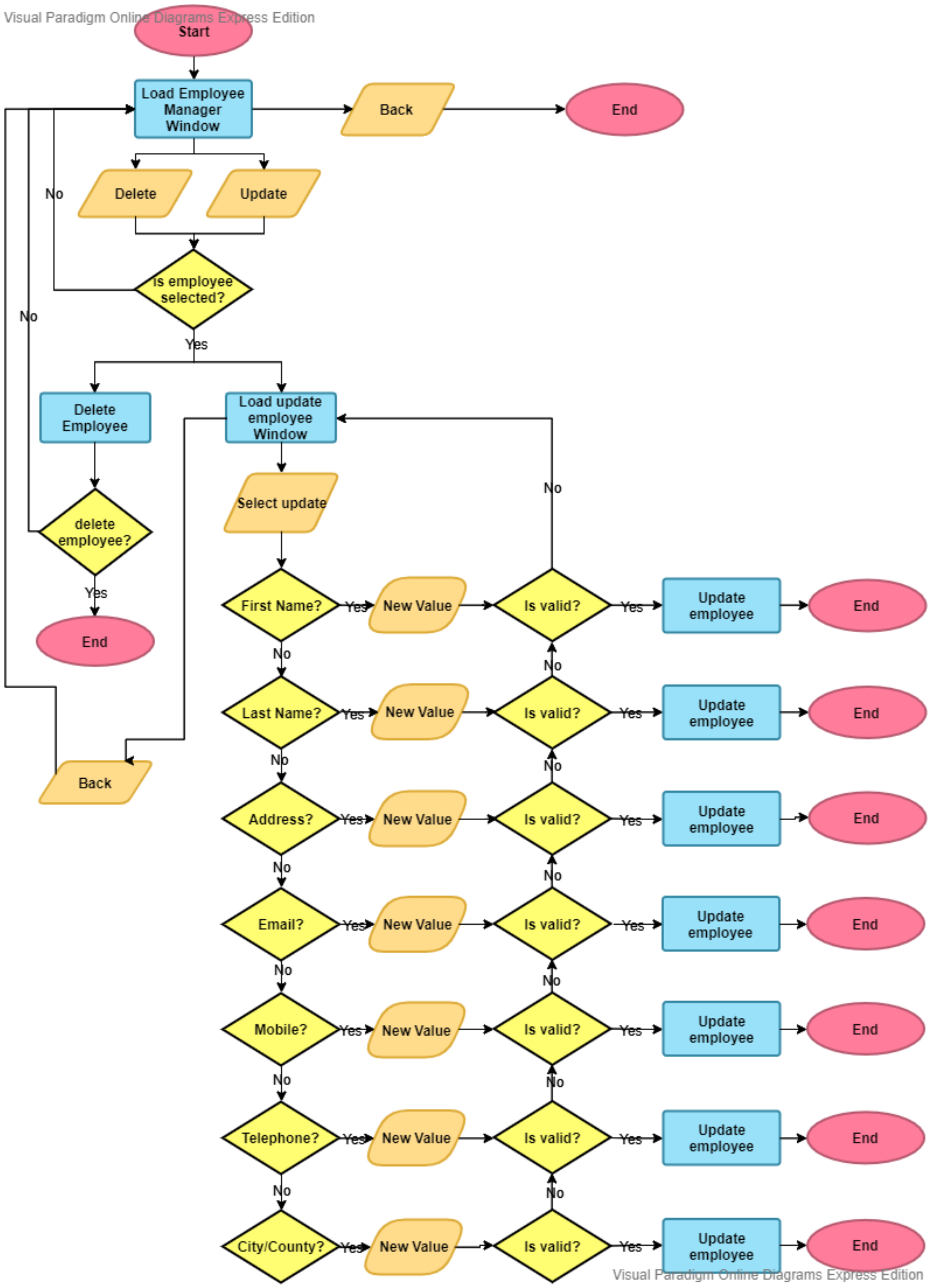


Figure 7 - Update and delete employee flowchart

2.1.4 Add task

There must be an option that allows the manager/supervisor to add a task in the system. A task is something that might or might not happen in the store. For example, the coffee machine is leaking water and one should call the maintenance to have a look at it. Once the machine is fixed the task is marked as done. This is not something that has to be done every day. There must be fields to enter the task description and select the date that the task should be completed.

Flowchart add task process

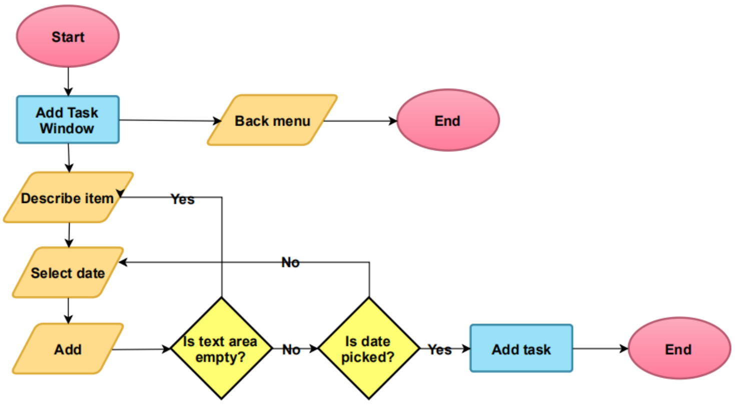


Figure 8 - Add task flowchart

State diagram add task process

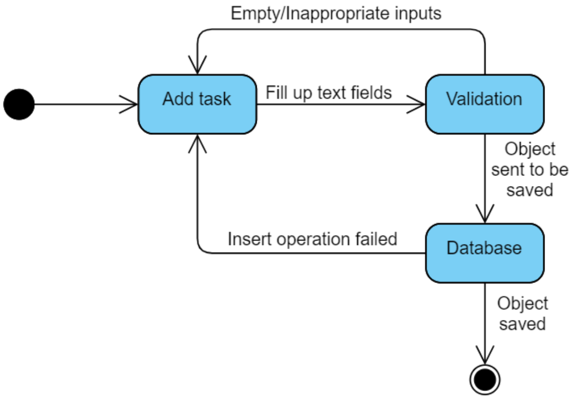


Figure 9 - Add task state diagram

2.1.5 Update task

There must be an option to update the task. Once the task is completed, the manager/supervisor will update the task checking it as done. There must be a field that will be entered the action taken to solve the task. Flowchart for this process can be seen in figure 12.

State diagram update task process

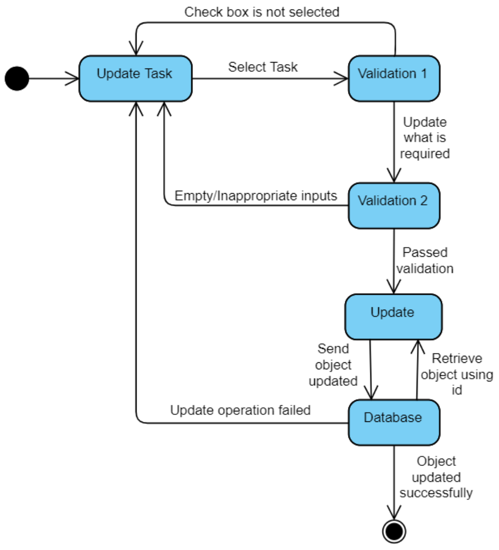


Figure 10 - Update task state diagram

2.1.6 Delete task

The system must have an option to delete the task if it is required. Flowchart for this process can be seen in figure 12.

State diagram delete task process

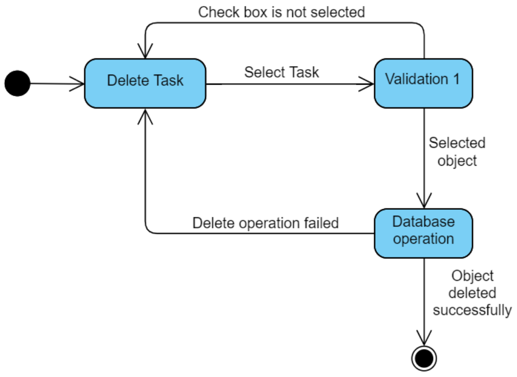


Figure 11 - Delete task state diagram

Flowchart update and delete processes

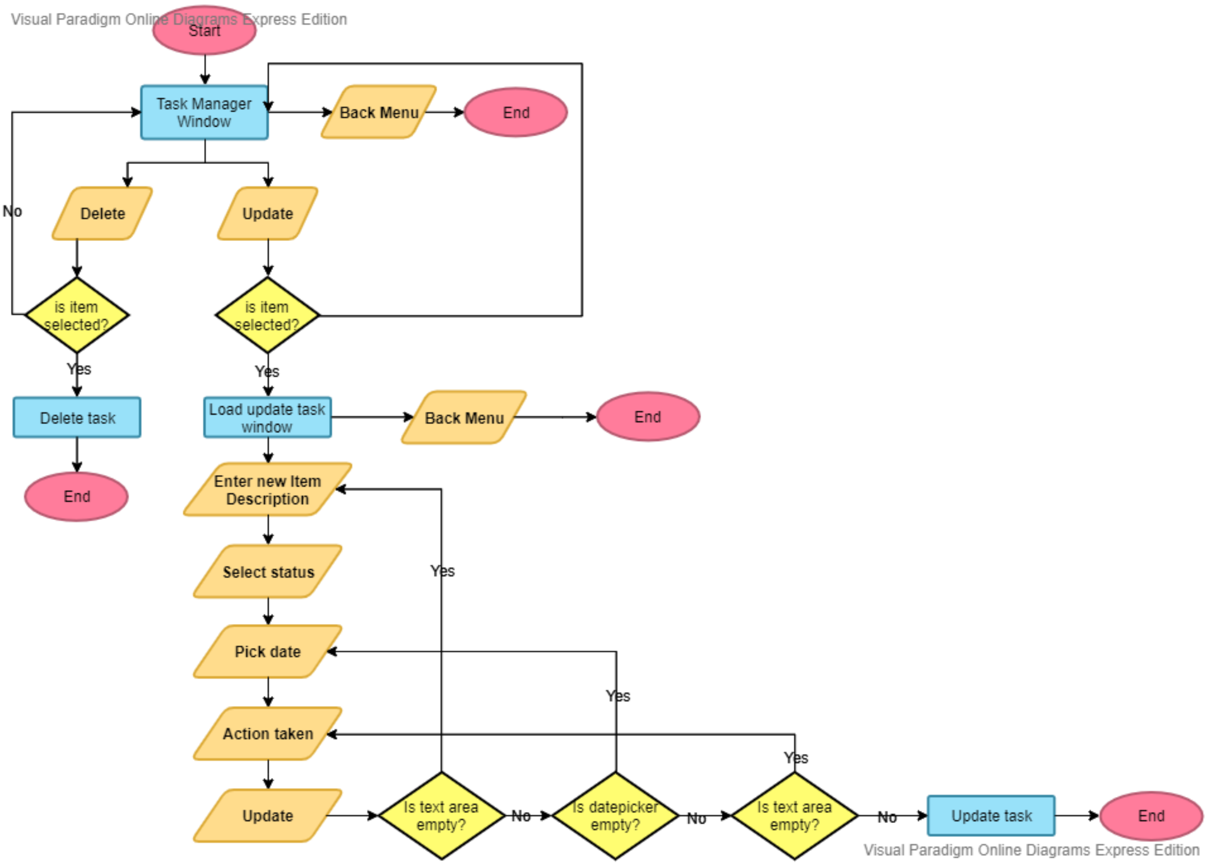


Figure 12 - Update and delete task flowchart

2.1.7 Add checklist item

There must be an option to add items to the checklists in the system. There are a lot of things that must be checked periodically. Some of them are required by law such as the fire warden checklist that contains biweekly and monthly items to be checked such as the fire doors, extinguishers, accessibility, etc. It must have fields to select the type of checklist, the frequency that includes the options daily, weekly, biweekly, and monthly, and a field to enter the description of the item.

Flowchart add checklist item process

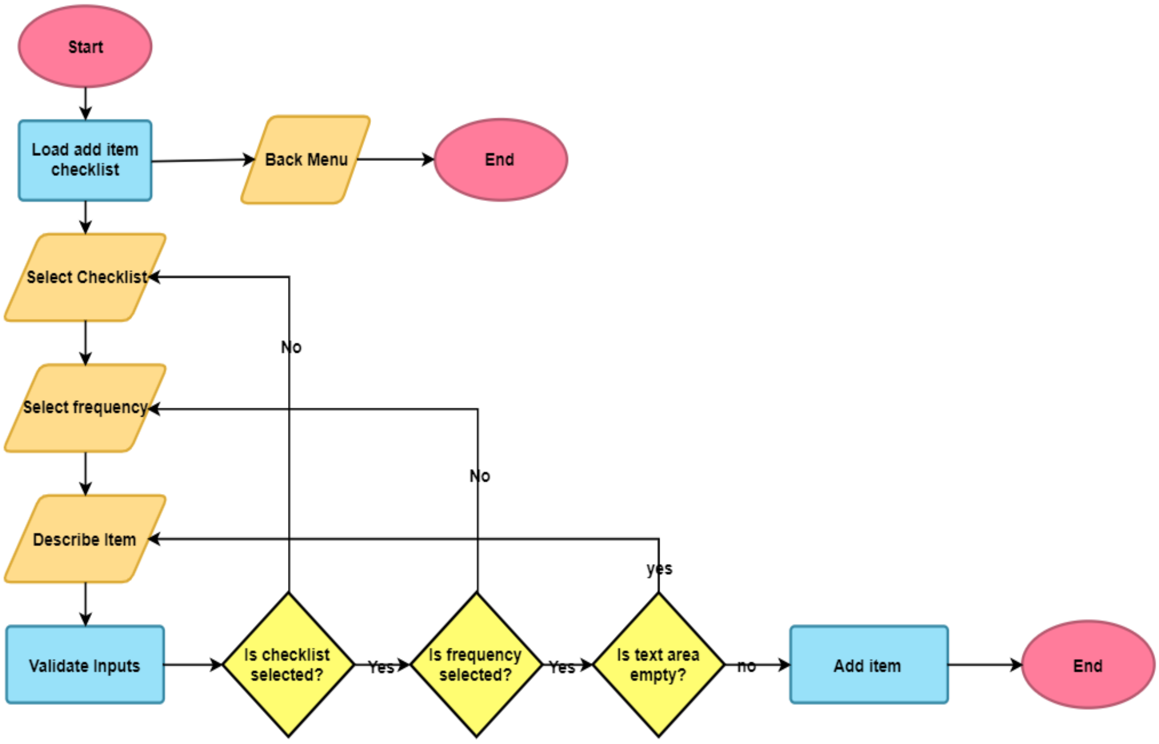


Figure 13 - Add checklist item flowchart

State diagram Add checklist item process

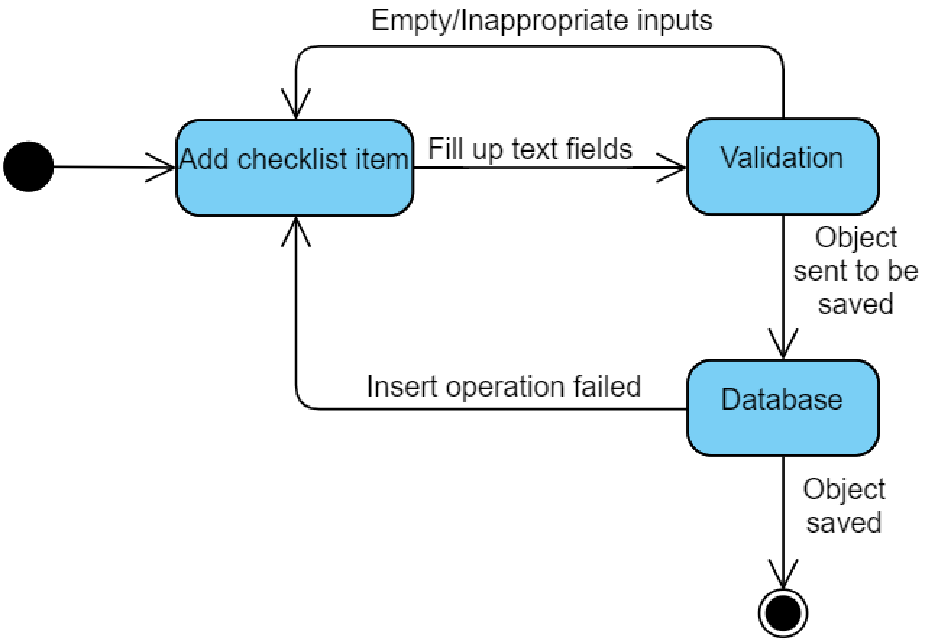


Figure 14 - Add checklist item state diagram

2.1.8 Update checklist item

The system must have two options to update the checklist items in the system. The first option is a window that only updates the status of the items from ‘pending’ to ‘done’. It will facilitate the users to change the status of the items quickly. The second option will allow the users to update the items in the system including the item description, frequency, date that the item was checked and status.

Flowchart update item option 1

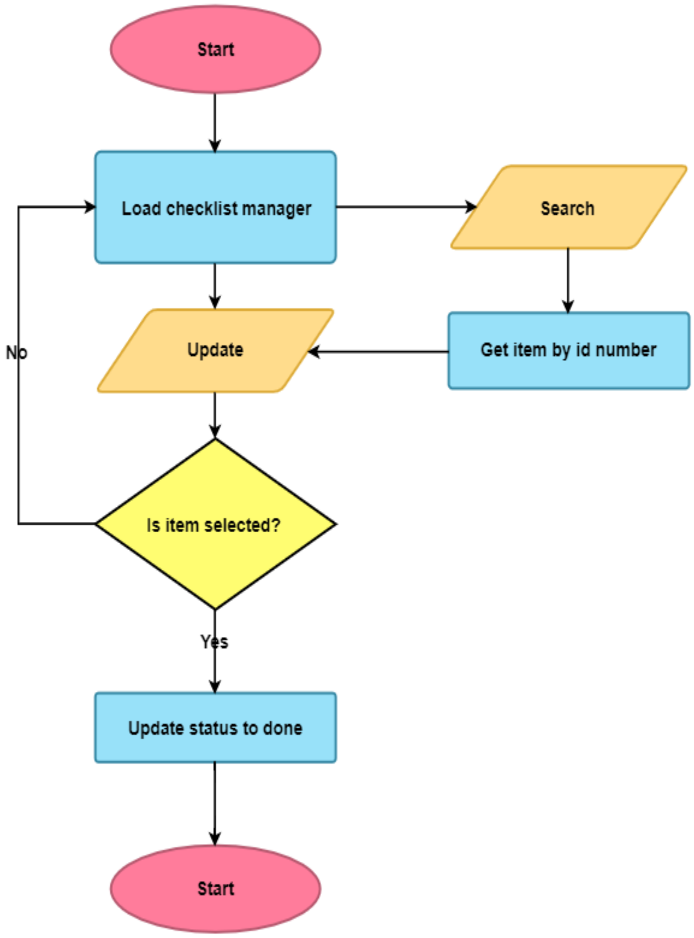


Figure 15 - Update checklist item flowchart

Flowchart update item option 2

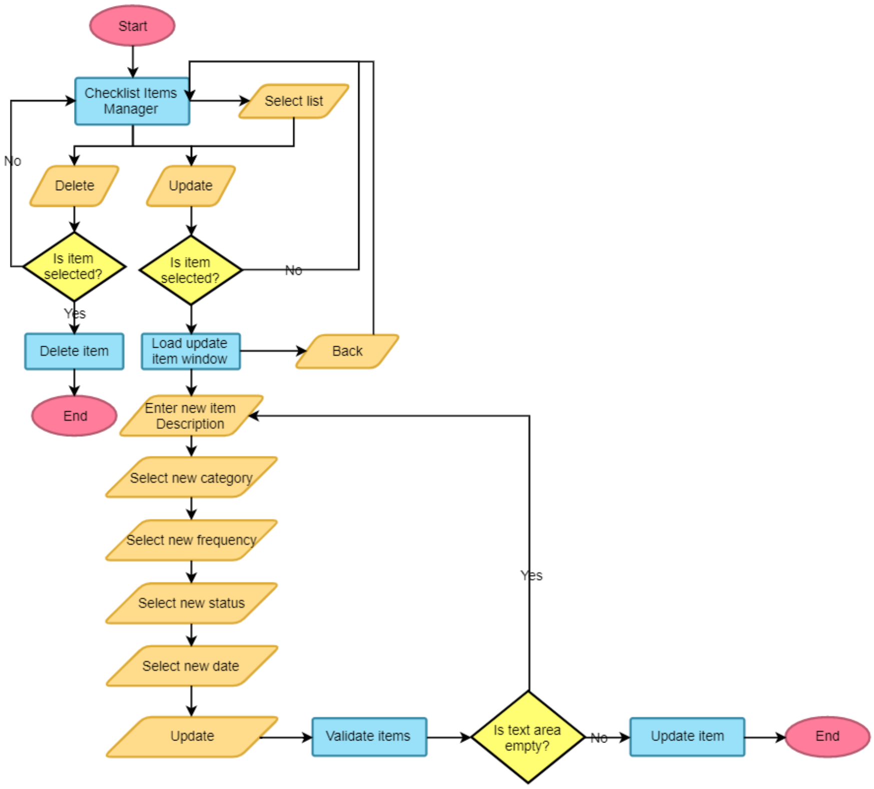


Figure 16 - Flowchart update checklist item option 2

State diagram update checklist item

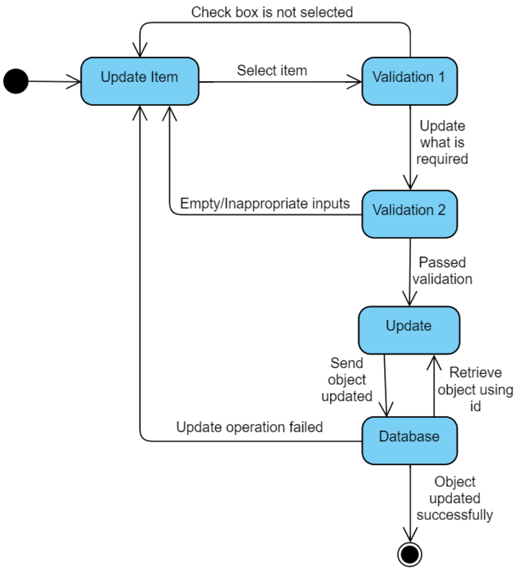


Figure 17 - Update checklist item state diagram

2.1.9 Delete checklist item

There must be an option to delete items from the checklists. The user must tick the check box and delete the item. Flowchart for delete option is together with the flowchart for update items since they operate in the same window.

State diagram delete checklist item process

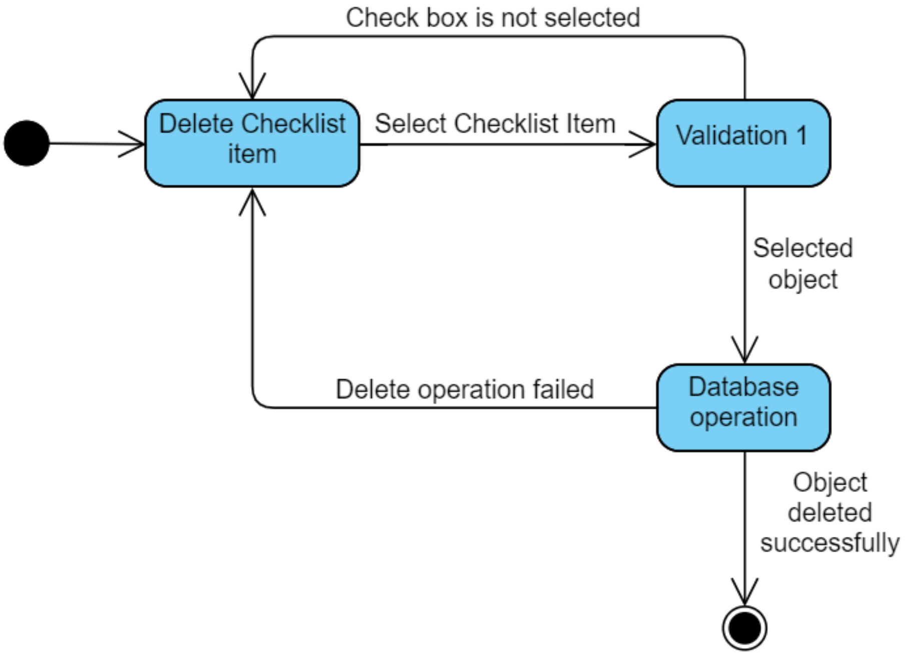


Figure 18 - Delete checklist item

2.1.10 Add training

There must be an option to add training. There are a few training types that all the employees must take. Some of them are required by law such as Manual Handling, Food Safety, etc. The user must be provided with an option to select the training type and a field to type the training description.

Flowchart add training process

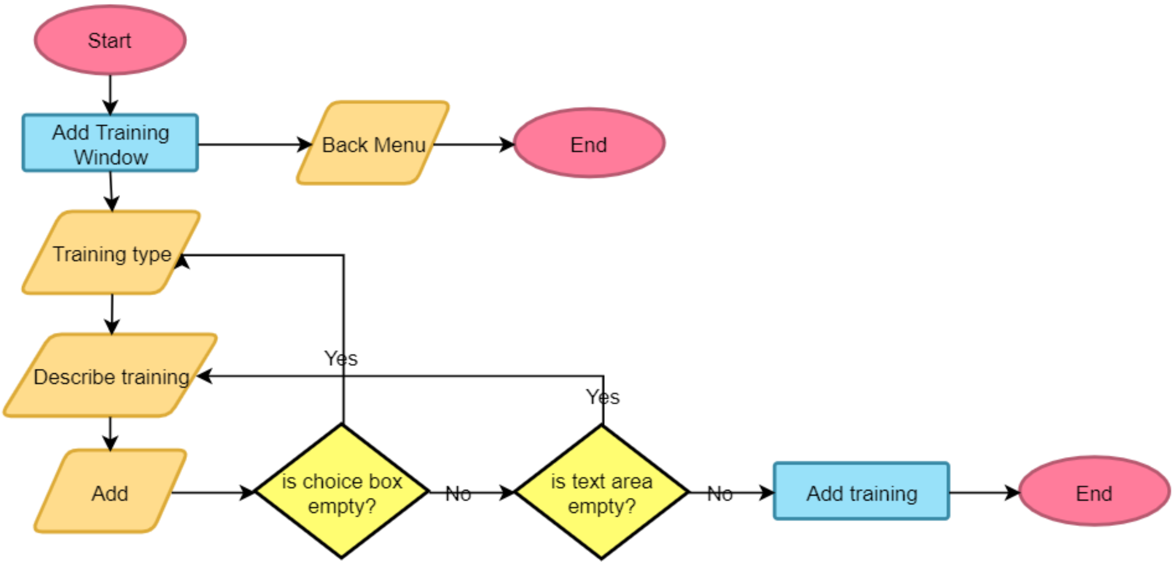


Figure 19 Add training flowchart

State diagram add training

Once the training object is created, the system automatically adds all the registered employees to the training.

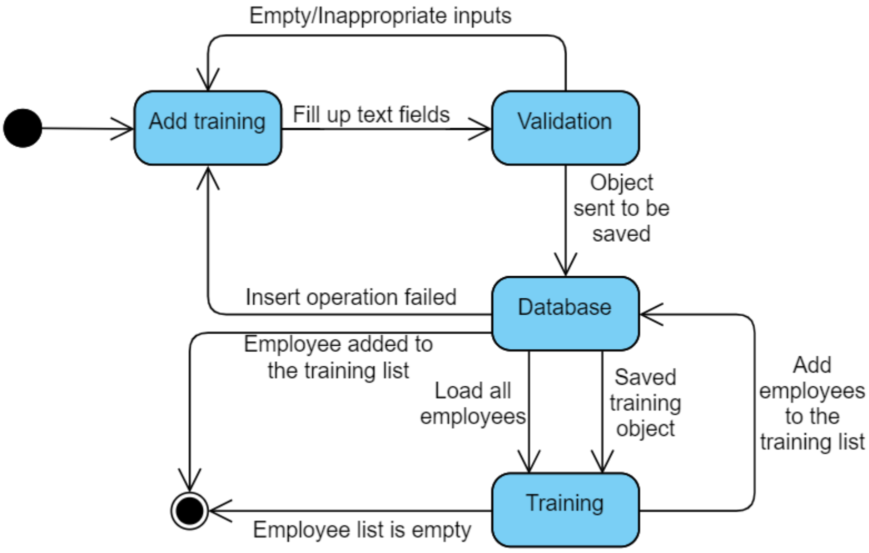


Figure 20 - Add training state diagram

2.1.11 Update Training

The system must have one update feature that will allow the user to update the training type and description, and another update feature that will allow the user to prompt an employee and see all the training related to that employee. The user must be able to change the training status from 'pending' to 'done' and enter the date that the training is updated.

Flowchart update/delete training I

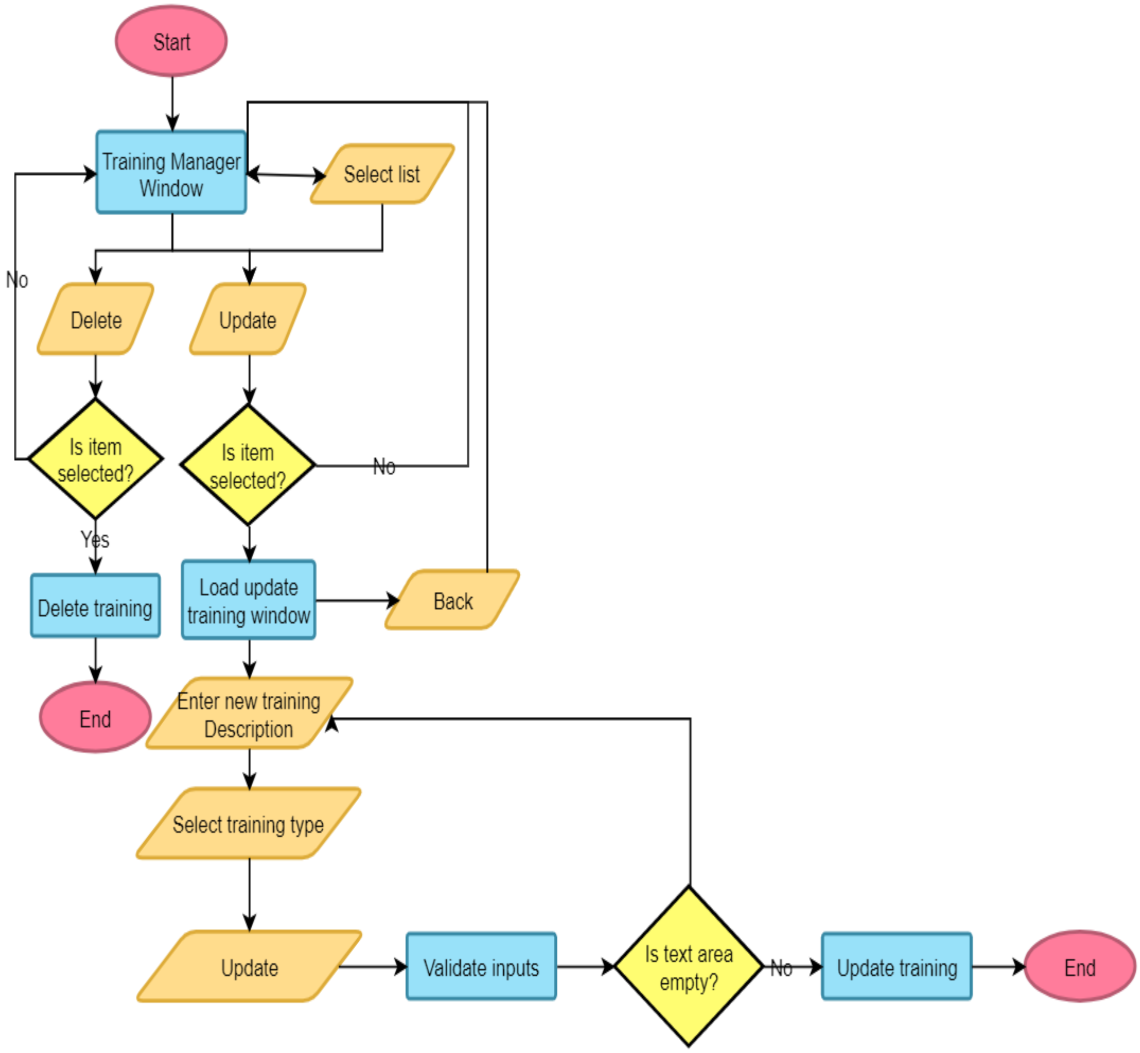


Figure 21 Update delete training

Flowchart update training II

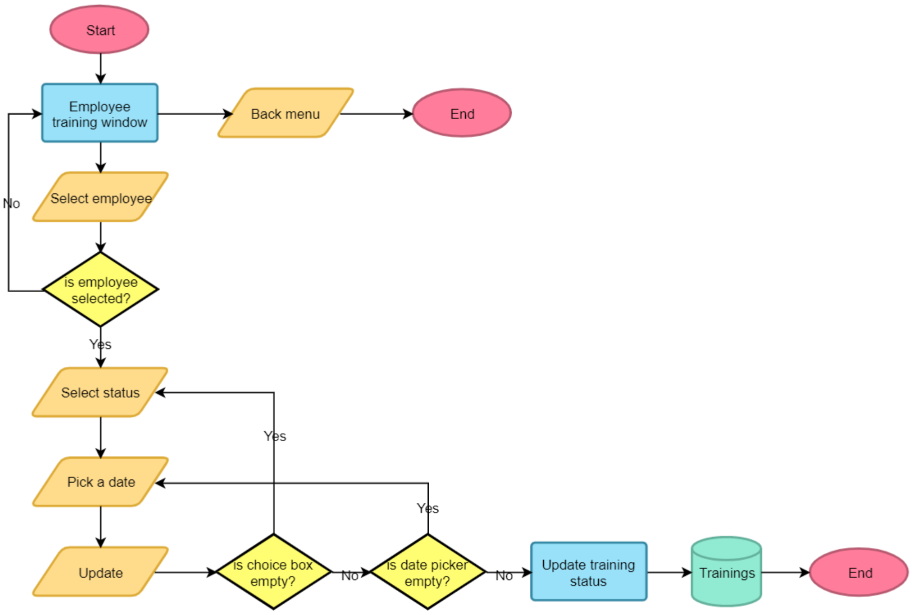


Figure 22 - Employee training window

State diagram update training

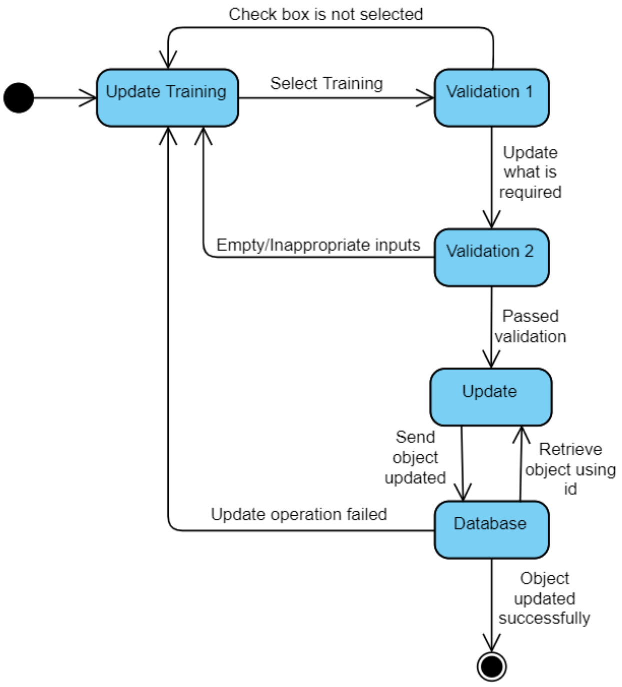


Figure 23 - Update training state diagram

State diagram update employee's training

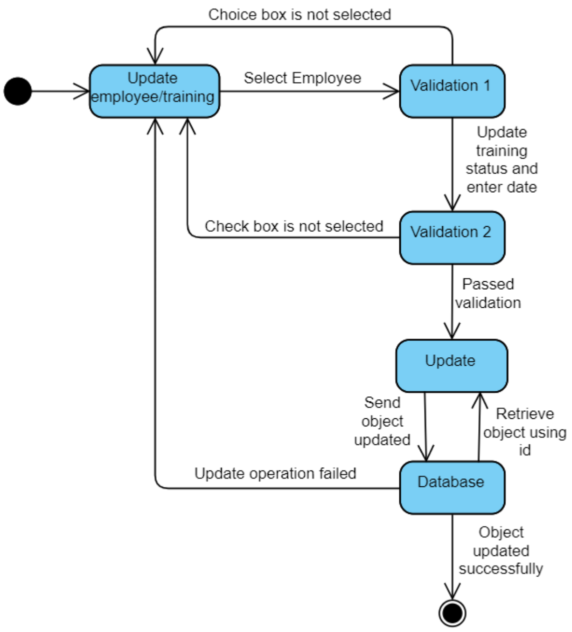


Figure 24 - Update employee training state diagram

2.1.12 Delete training

There must be an option to delete a training from the training lists. When the training is deleted it also must be deleted from the employee/training list.

State diagram delete training

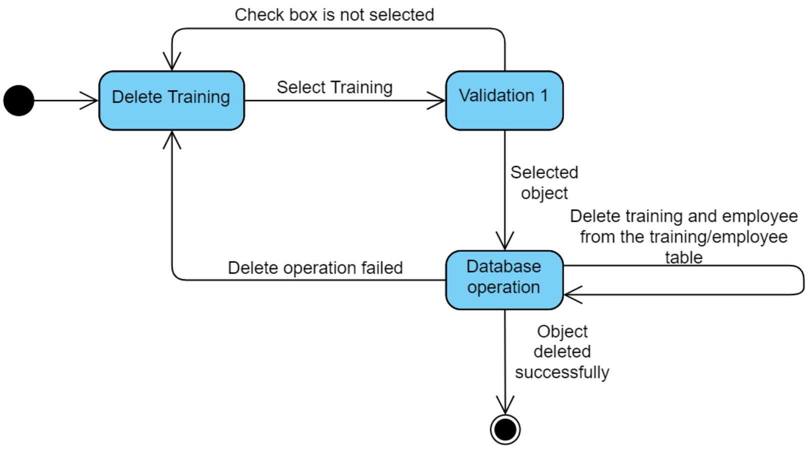

Check box is not selected
Delete Training
Select Training
Validation 1
Selected object
Delete training and employee from the training/employee table
Delete operation failed
Database operation
Object deleted successfully

Figure 25 Delete training state diagram

2.1.13 Add maintenance item

The system must have an option to allow the user to add a maintenance item. The company have some contractors that periodically go the store to render services such as fire system inspection, among others. It must a have a field to enter the contractor name, description of the service, last time that the service was rendered and the next time that the service must be done.

Flowchart add contractor process

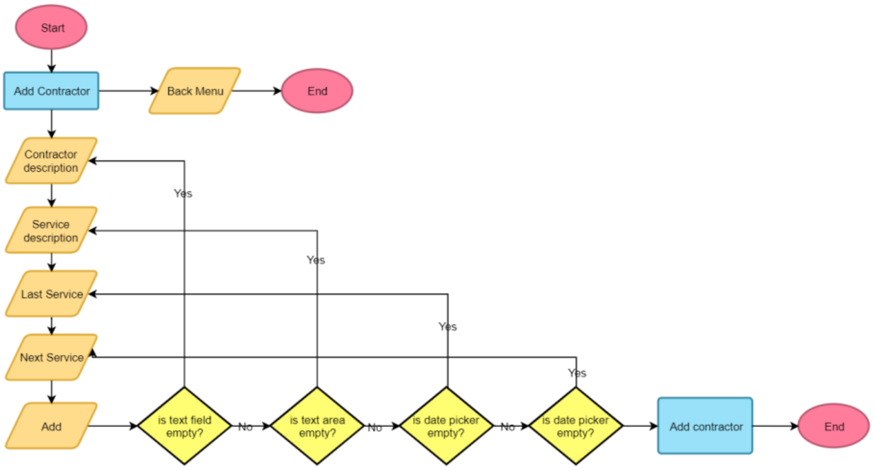


Figure 26 - Add contractor flowchart

State diagram add maintenance item

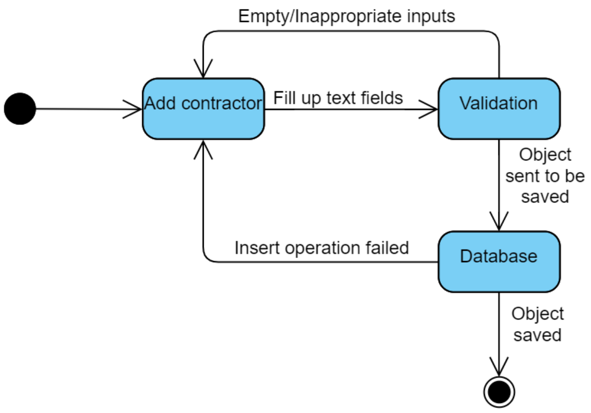


Figure 27 - Add contractor state diagram

2.1.14 Update maintenance item

There must be an option to update the contractor details in the system. Update the contractor name, service description, last and next service.

Flowchart update/delete maintenance item process

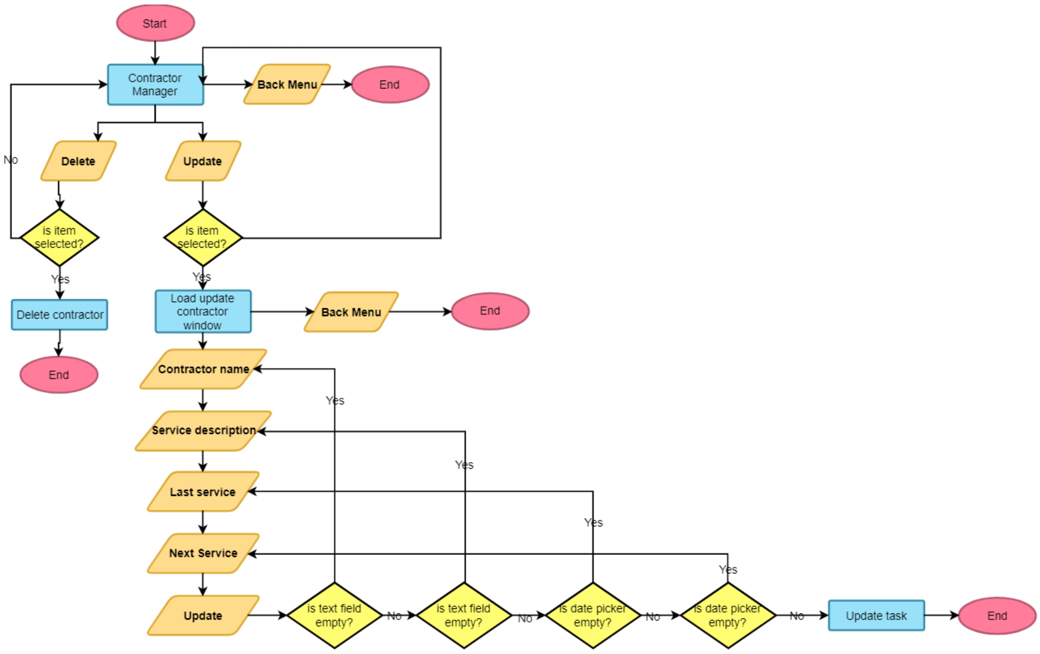


Figure 28 - Update/delete contractor flowchart

State diagram update maintenance item

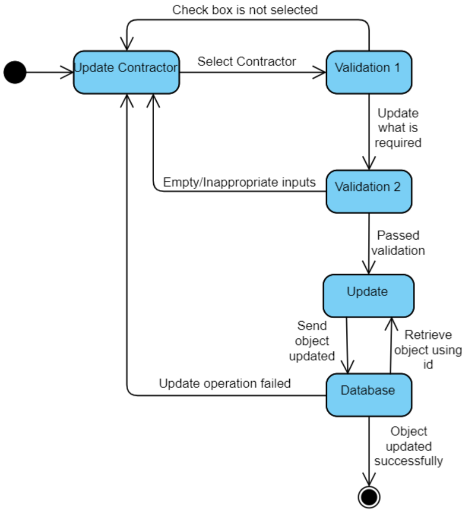


Figure 29 - Update contractor state diagram

2.1.15 Delete maintenance item

There must be an option in the system that allows the user to delete a contractor from the system. Flowchart for this process can be seen in figure 28.

State diagram delete maintenance item

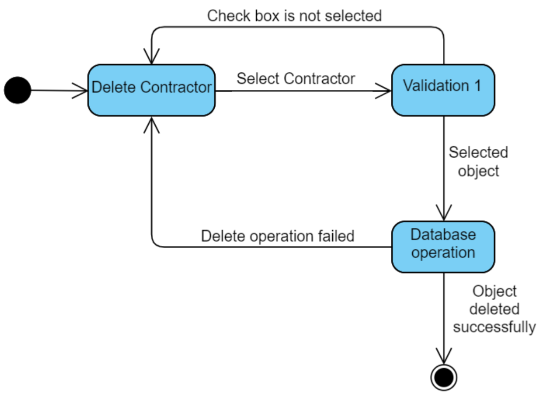


Figure 30 - Delete contractor state diagram

2.1.16 Generate task report

The system must have an option that generates a report of the tasks sorted by status (pending, done, both).

Flowchart task report process

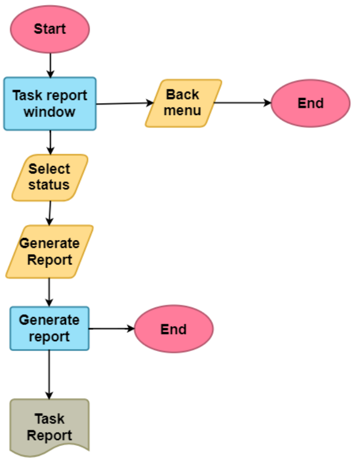


Figure 31 - Generate task report

2.1.17 Generate checklist report

The system must generate a report of all checklist items by checklist type and status. Also, generate a report with all checklist items that must be done in that day along with all pending items from past days.

Flowchart checklist report process

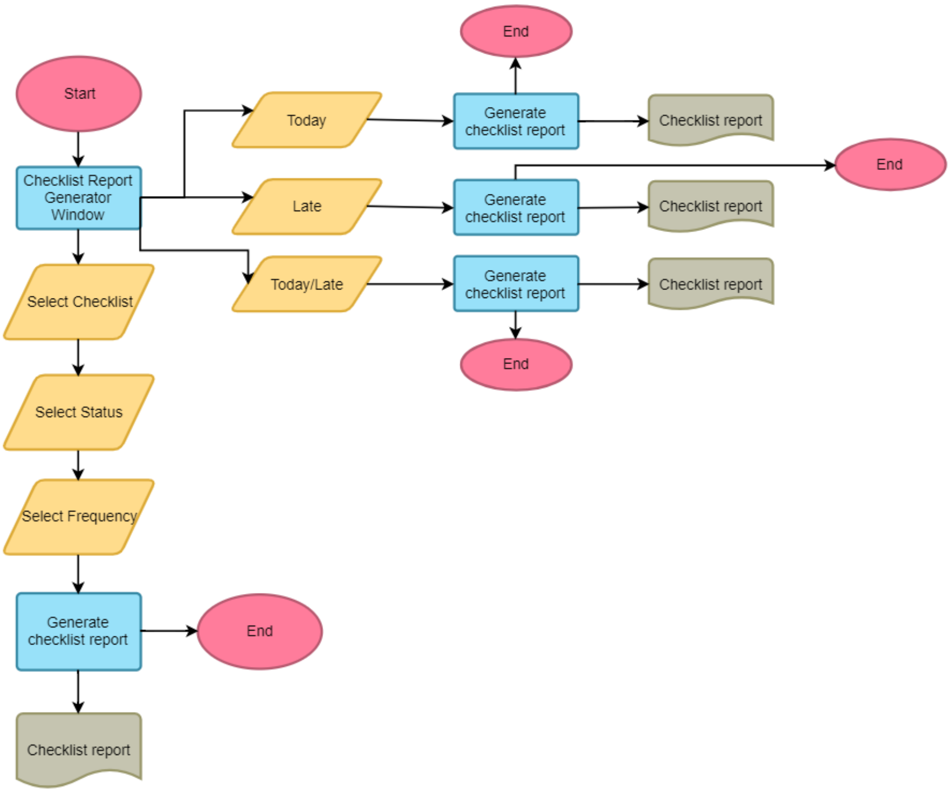


Figure 32 - Generate checklist report flowchart

2.1.18 Generate training report

There must be an option to generate a training report that be sorted by employee, training type and status.

Flowchart training report process

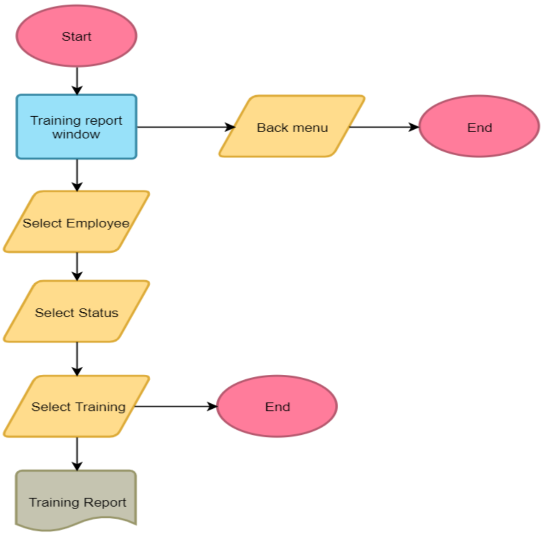


Figure 33 - Generate training report flowchart

2.1.19 Generate maintenance report

There must be an option that allows the user to generate a maintenance report sorted by all items, status, or next service.

Flowchart maintenance report process

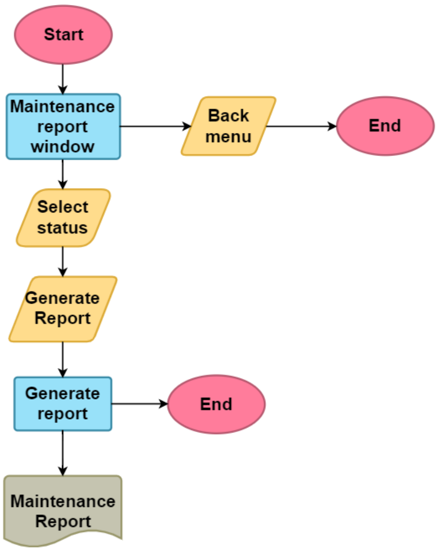
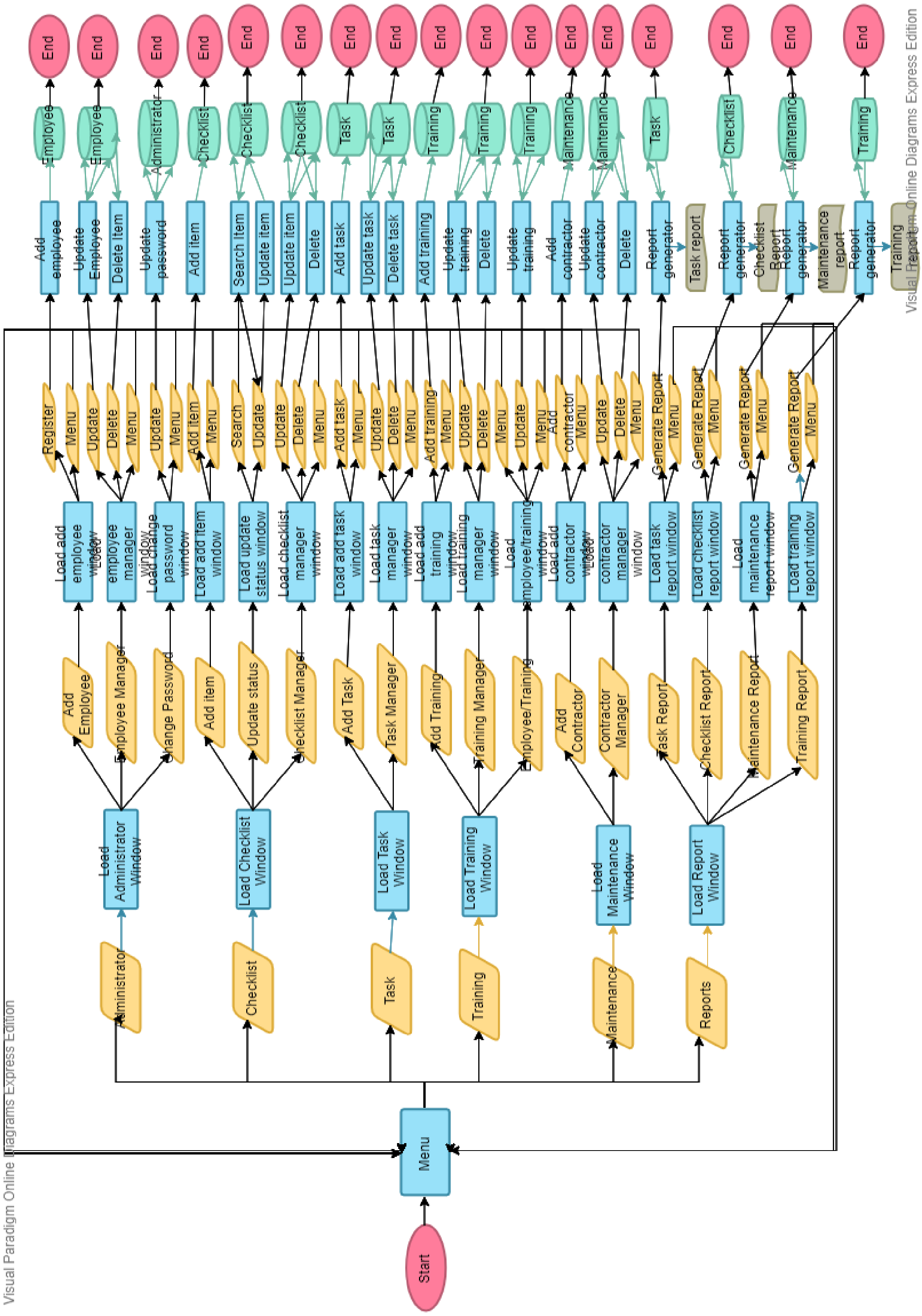


Figure 34 - Generate maintenance report flowchart

2.2 Overall Flowchart Diagram



2.3 Use Case Scenarios

Use case scenarios and diagrams are used to understand what your application will make and how the flow of the program would work. It is used to understand how the program is supposed to be seen from the end user perspective. It is mostly used to summarize how the program would work and what it is going to do. (Visual Paradigm, 2020).

2.3.0 Scenario I

A new staff starts to work on the shop and needs to be trained. The manager needs to add a new staff to the training list and then print a report of all trainings needed for each staff, including the new one so he/she can see who can be trained together. The process can be seen in figure 35.

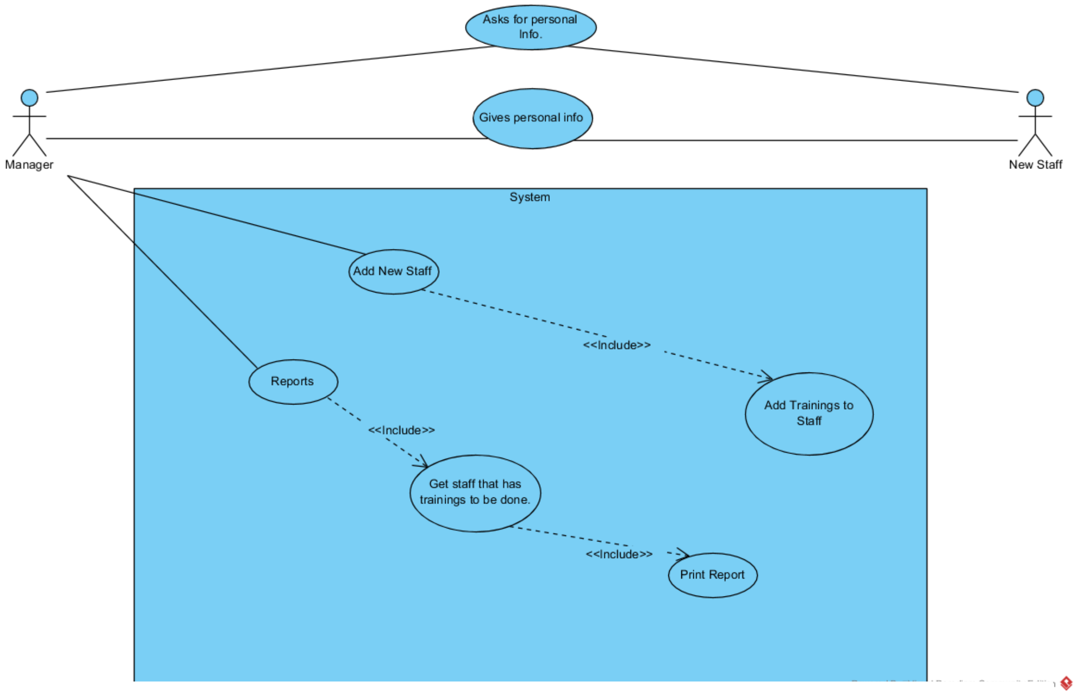


Figure 35 - Case scenario I

2.3.1 Scenario II

A lamp broke in the café area and needs to be changed until next week but there is no lamp available in the shop (needs to be ordered). The manager needs to enter this task into the system so he/she can be remembered later as the shop is busy at the moment. Some other tasks have been done meanwhile that weren’t completed in the system in which the manager can mark as completed. The process can be seen in figure 36.

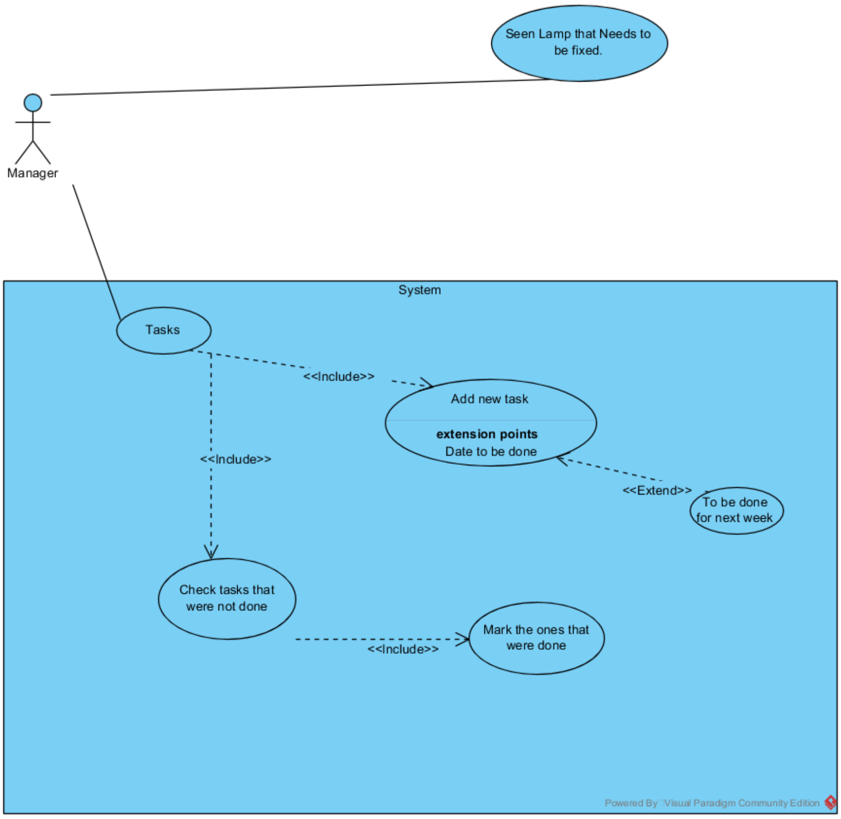

Seen Lamp that Needs to be fixed.
Manager
System
Tasks
<<Include>>
Add new task
extension points
Date to be done
<<Include>>
<<Extend>>
To be done for next week
Check tasks that were not done
<<Include>>
Mark the ones that were done
Powered By Visual Paradigm Community Edition

Figure 36 - Case scenario II

2.3.2 Scenario III

A report of everything that needs to be done for this week and month needs to be printed at the end of every week automatically. It includes tasks, checklists, trainings, and maintenance. The process can be seen in picture 37.

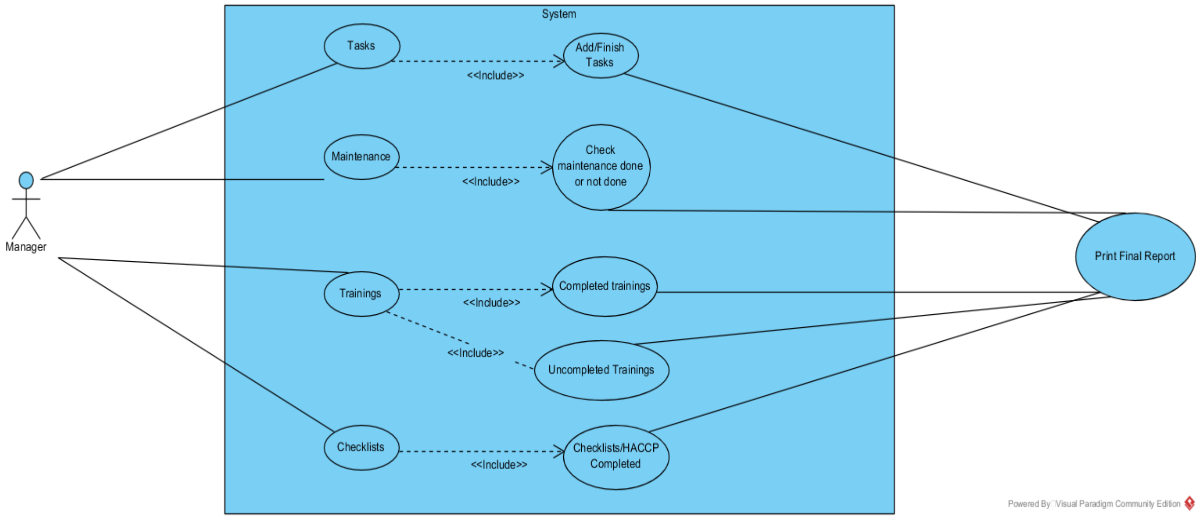


Figure 37 - Case scenario III

2.3.3 Scenario IV

The system automatically was printing a report and uploading backup data to the cloud every day. The manager does not think this is necessary and wants to change it. The manager goes to the settings and change it for once a week. The process can be seen in picture 38.

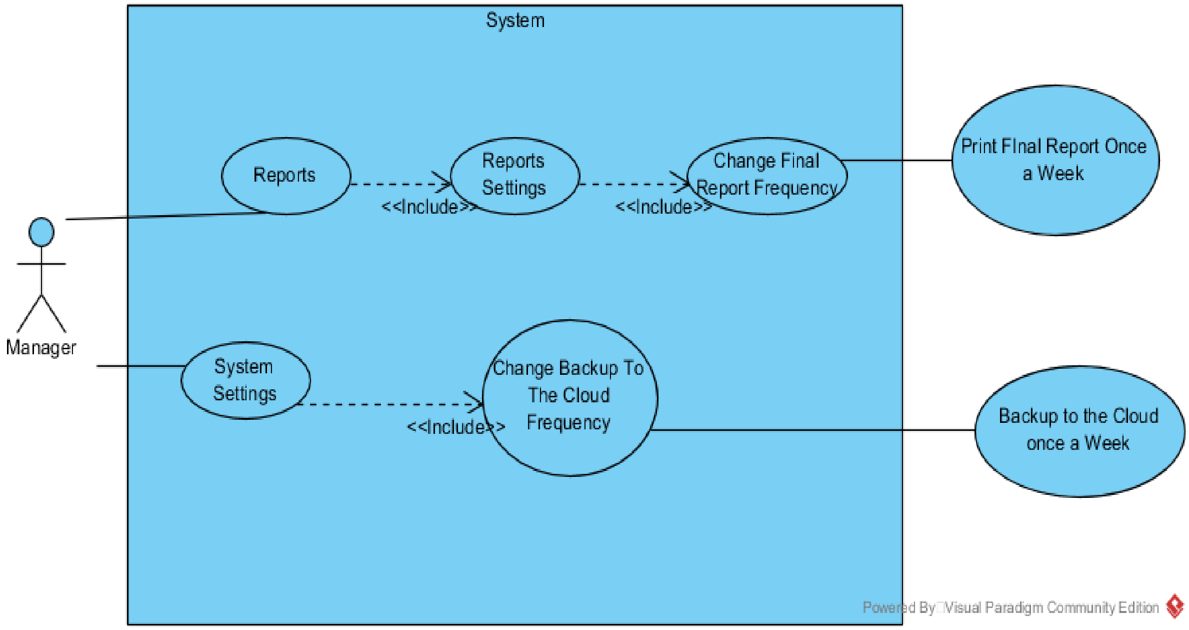

System
Reports
Reports Settings
Change Final Report Frequency
<<Include>>
<<Include>>
Print FInal Report Once a Week
Manager
System Settings
Change Backup To The Cloud Frequency
<<Include>>
Backup to the Cloud once a Week
Powered By Visual Paradigm Community Edition

Figure 38 - Use case scenario IV

2.4 Sequence Diagram

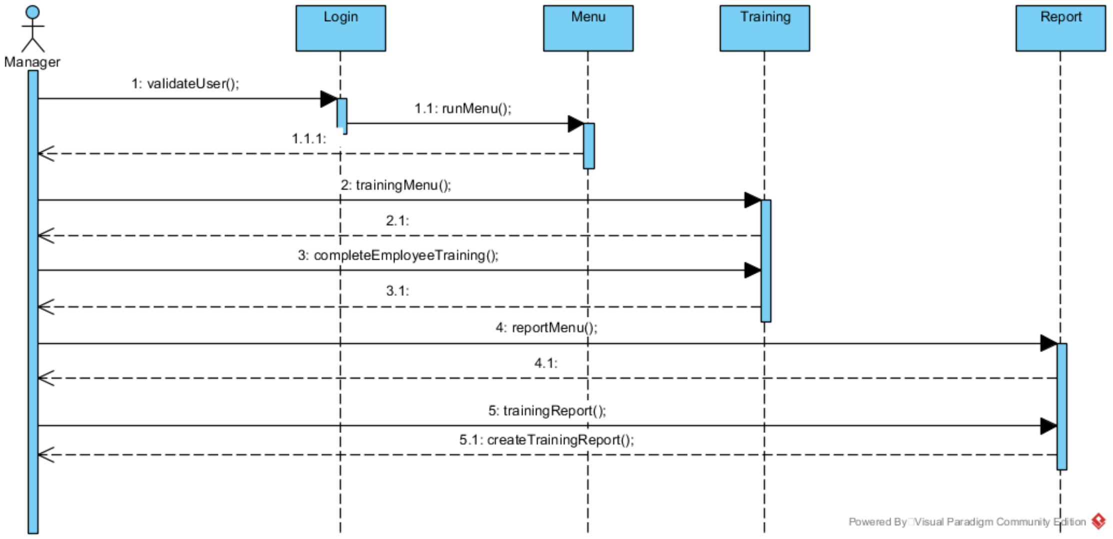
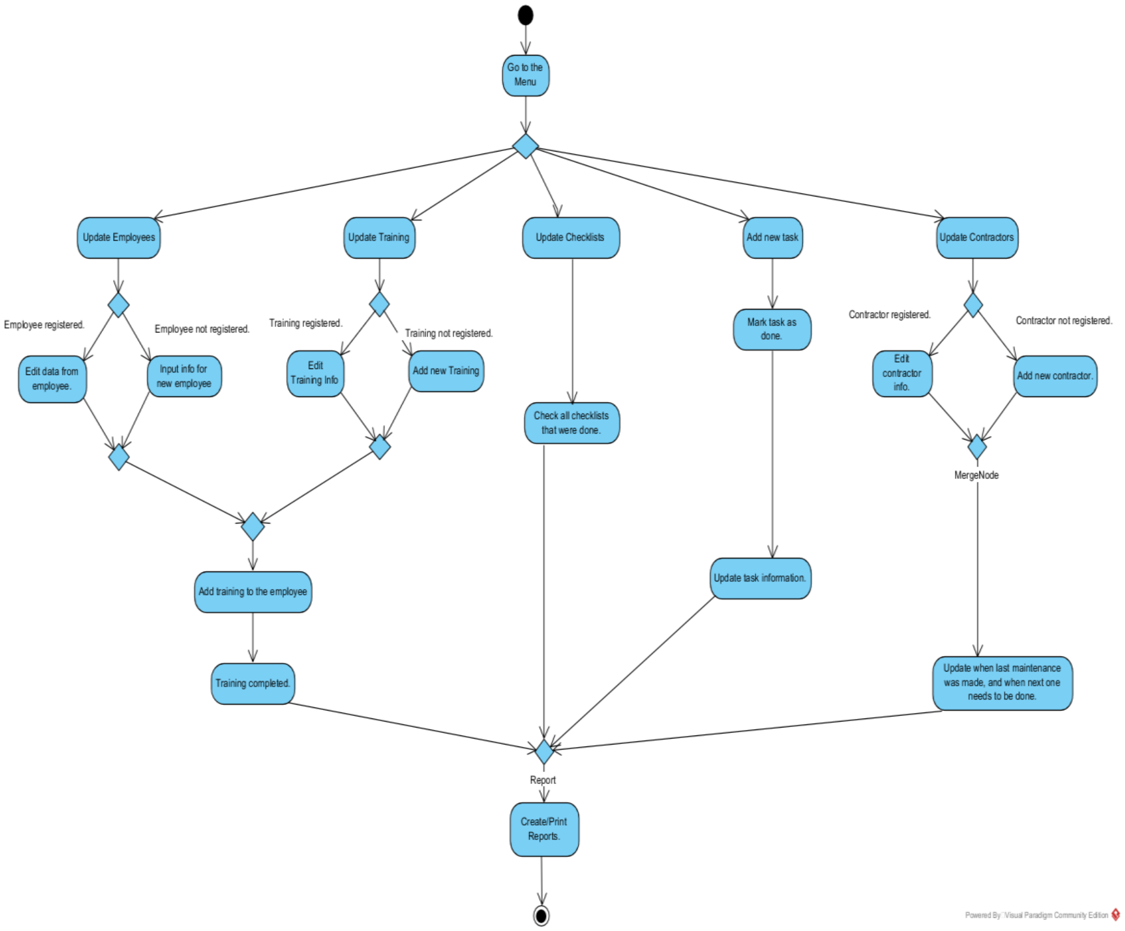


Figure 39 - Sequence diagram

2.5 Activity Diagram



2.6 Checklist State Diagram

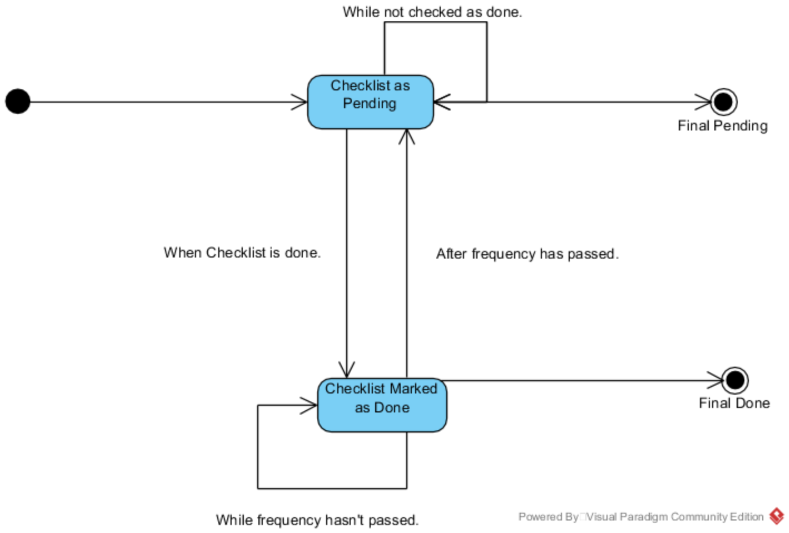

While not checked as done.
Checklist as Pending
Final Pending
When Checklist is done.
After frequency has passed.
Checklist Marked as Done
Final Done
While frequency hasn't passed.
Powered By Visual Paradigm Community Edition

Figure 41 - Checklist state diagram

2.7 Class Diagrams

Main class

This is the main class of the project that runs the application. It extends Application that is the entry point of the application in JavaFX.

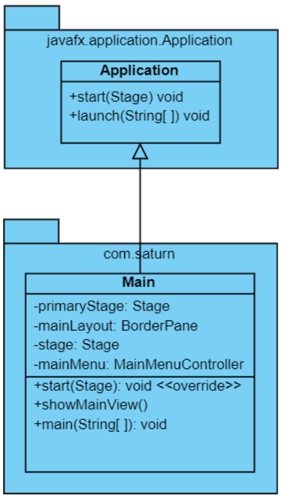
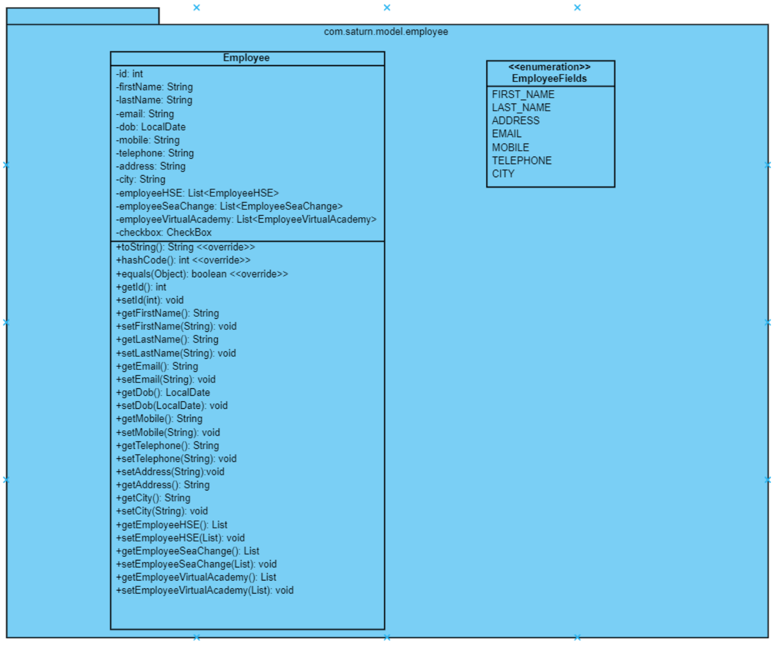


Figure 42 - Main class

Employee class / Enum

It is the Employee class that allows the creation of an employee object

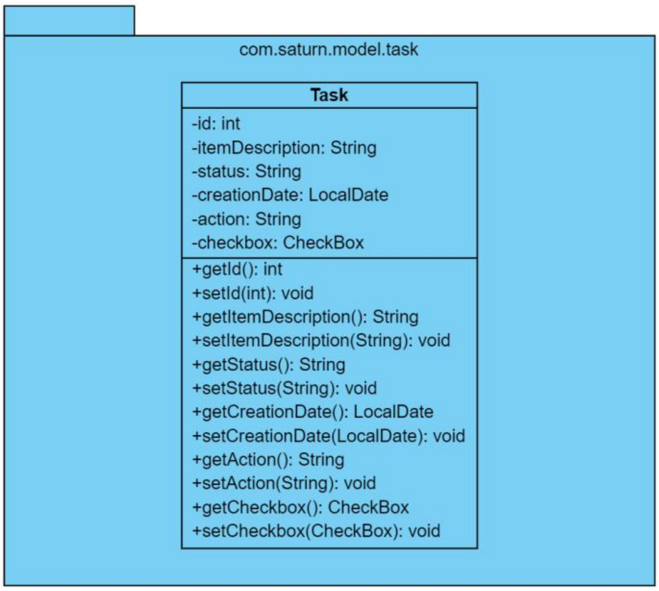


```
com.saturn.model.employee

Employee
-id: int
-firstName: String
-lastName: String
-email: String
-dob: LocalDate
-mobile: String
-telephone: String
-address: String
-city: String
-employeeHSE: List<EmployeeHSE>
-employeeSeaChange: List<EmployeeSeaChange>
-employeeVirtualAcademy: List<EmployeeVirtualAcademy>
-checkbox: CheckBox
+toString(): String <<override>>
+hashCode(): int <<override>>
+equals(Object): boolean <<override>>
+getId(): int
+setId(int): void
+getFirstName(): String
+setFirstName(String): void
+getLastName(): String
+setLastName(String): void
+getEmail(): String
+setEmail(String): void
+getDob(): LocalDate
+setDob(LocalDate): void
+getMobile(): String
+setMobile(String): void
+getTelephone(): String
+setTelephone(String): void
+setAddress(String):void
+getAddress(): String
+getCity(): String
+setCity(String): void
+getEmployeeHSE(): List
+setEmployeeHSE(List): void
+getEmployeeSeaChange(): List
+setEmployeeSeaChange(List): void
+getEmployeeVirtualAcademy(): List
+setEmployeeVirtualAcademy(List): void

<<enumeration>>
EmployeeFields
FIRST_NAME
LAST_NAME
ADDRESS
EMAIL
MOBILE
TELEPHONE
CITY
```

Figure 43 - Employee package

Task class

Task class is the blueprint for the creation of a task object.



```
com.saturn.model.task

Task
-id: int
-itemDescription: String
-status: String
-creationDate: LocalDate
-action: String
-checkbox: CheckBox
+getId(): int
+setId(int): void
+getItemDescription(): String
+setItemDescription(String): void
+getStatus(): String
+setStatus(String): void
+getCreationDate(): LocalDate
+setCreationDate(LocalDate): void
+getAction(): String
+setAction(String): void
+getCheckbox(): CheckBox
+setCheckbox(CheckBox): void
```

Figure 44 - Task package

Maintenance class

Maintenance class is the blueprint for a maintenance object.

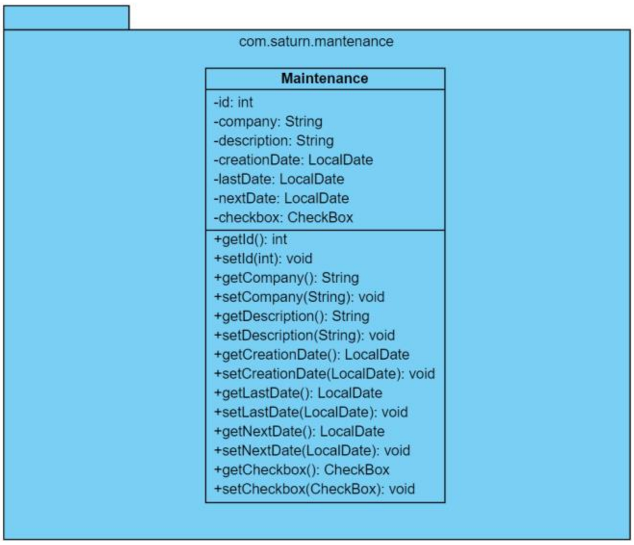

com.saturn.mantenance
Maintenance
-id: int
-company: String
-description: String
-creationDate: LocalDate
-lastDate: LocalDate
-nextDate: LocalDate
-checkbox: CheckBox
+getId(): int
+setId(int): void
+getCompany(): String
+setCompany(String): void
+getDescription(): String
+setDescription(String): void
+getCreationDate(): LocalDate
+setCreationDate(LocalDate): void
+getLastDate(): LocalDate
+setLastDate(LocalDate): void
+getNextDate(): LocalDate
+setNextDate(LocalDate): void
+getCheckbox(): CheckBox
+setCheckbox(CheckBox): void

Figure 45 - Maintenance package

Views class

Views class is a final class with private constructor. It is used as constants that provides a path to the fxml files. In case that the paths to the files change, it will not be necessary going through the whole code, all the paths are in this class.

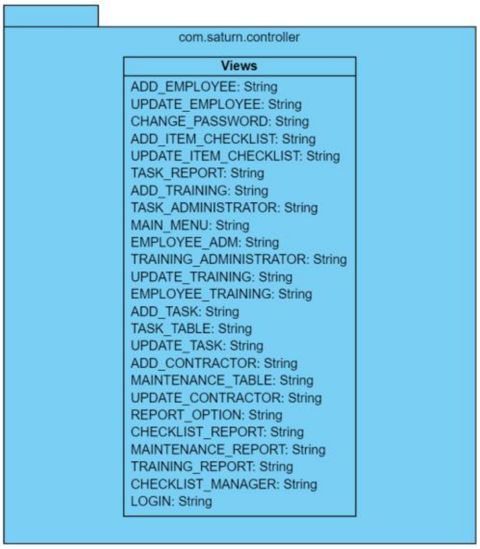


Figure 46 - View class

Administrator/Validation classes

Administrator class is the blueprint for the administrator object that holds the login and password to access the system. Validation class is a final with private constructor. This class provides validation for the user's inputs.

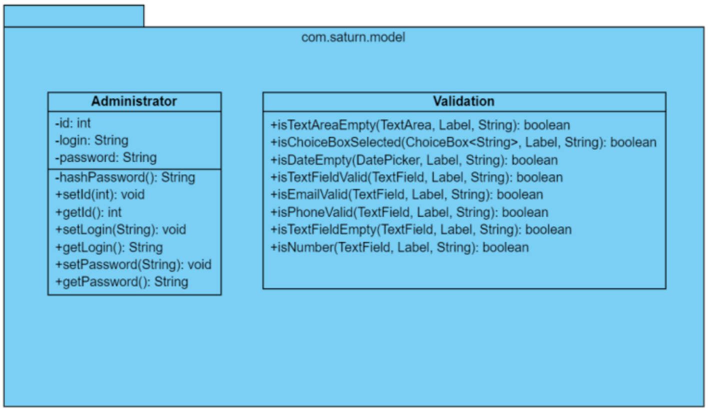

com.saturn.model
Administrator
-id: int
-login: String
-password: String
-hashPassword(): String
+setId(int): void
+getId(): int
+setLogin(String): void
+getLogin(): String
+setPassword(String): void
+getPassword(): String
Validation
+isTextAreaEmpty(TextArea, Label, String): boolean
+isChoiceBoxSelected(ChoiceBox<String>, Label, String): boolean
+isDateEmpty(DatePicker, Label, String): boolean
+isTextFieldValid(TextField, Label, String): boolean
+isEmailValid(TextField, Label, String): boolean
+isPhoneValid(TextField, Label, String): boolean
+isTextFieldEmpty(TextField, Label, String): boolean
+isNumber(TextField, Label, String): boolean

Figure 47 Administrator and Validation classes

Training classes

Inside this package there are five classes. TrainingSuperClass is an abstract class that holds all necessary common instances variables and methods for HSETraining, SeaChangeTraining and VirtualAcademyTraining classes. TrainingFactory is the class responsible for the creation of a training object.

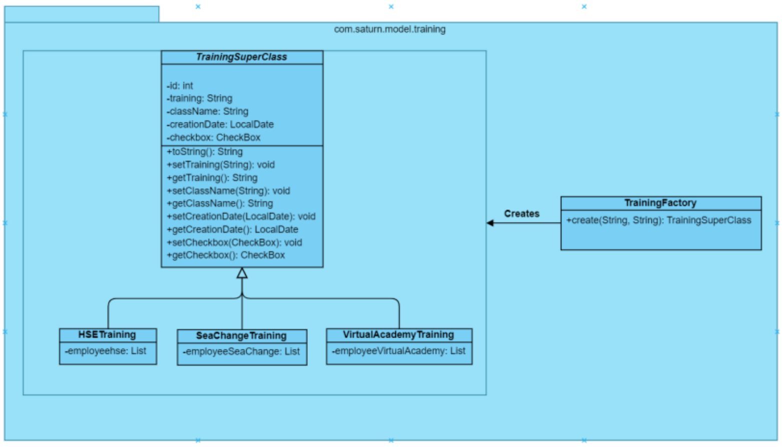


Figure 48 Training package

Checklist classes/Enums

There are seven classes and two Enums in this package. ChecklistSuperClass holds all necessary instances variables and methods for its child classes. ChecklistFactory class is responsible for the creation of a checklist object.

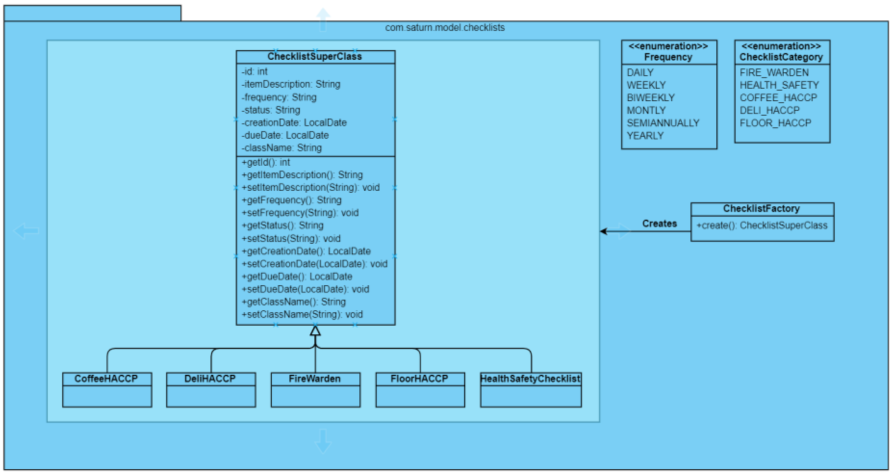


Figure 49 - Checklist package

Login controller class

LoginController class processes the login inputs from the user with the validation logic and determines if the user will be authenticated or not. It interacts with Dao class responsible for the interaction with the database.

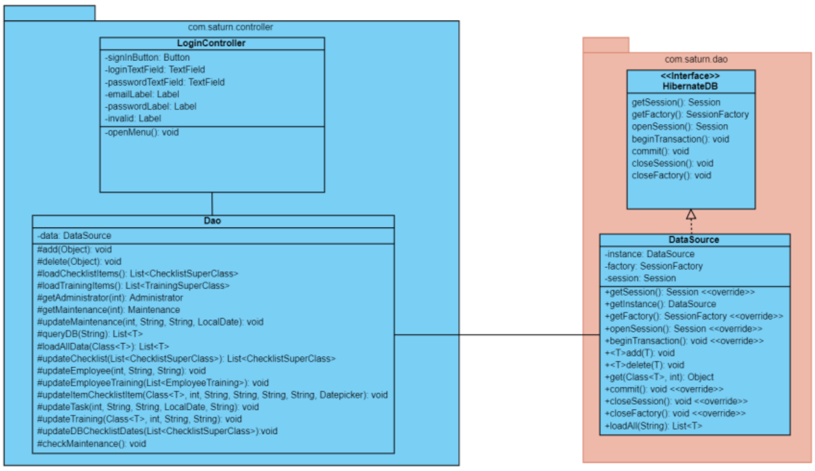
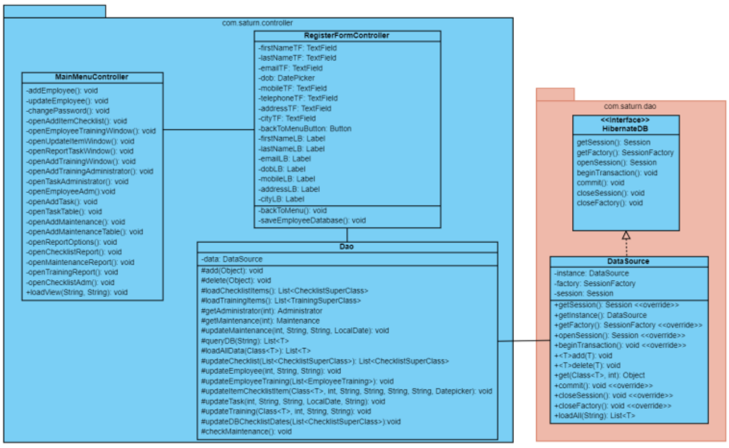

com.saturn.controller
LoginController
-signInButton: Button
-loginTextField: TextField
-passwordTextField: TextField
-emailLabel: Label
-passwordLabel: Label
-invalid: Label
-openMenu(): void
Dao
-data: DataSource
#add(Object): void
#delete(Object): void
#loadChecklistItems(): List<ChecklistSuperClass>
#loadTrainingItems(): List<TrainingSuperClass>
#getAdministrator(int): Administrator
#getMaintenance(int): Maintenance
#updateMaintenance(int, String, String, LocalDate): void
#queryDB(String): List<T>
#loadAllData(Class<T>): List<T>
#updateChecklist(List<ChecklistSuperClass>): List<ChecklistSuperClass>
#updateEmployee(int, String, String): void
#updateEmployeeTraining(List<EmployeeTraining>): void
#updateItemChecklistItem(Class<T>, int, String, String, String, String, Datepicker): void
#updateTask(int, String, String, LocalDate, String): void
#updateTraining(Class<T>, int, String, String): void
#updateDBChecklistDates(List<ChecklistSuperClass>):void
#checkMaintenance(): void
com.saturn.dao
<<Interface>>
HibernateDB
getSession(): Session
getFactory(): SessionFactory
openSession(): Session
beginTransaction(): void
commit(): void
closeSession(): void
closeFactory(): void
DataSource
-instance: DataSource
-factory: SessionFactory
-session: Session
+getSession(): Session <<override>>
+getInstance(): DataSource
+getFactory(): SessionFactory <<override>>
+openSession(): Session <<override>>
+beginTransaction(): void <<override>>
+<T>add(T): void
+<T>delete(T): void
+get(Class<T>, int): Object
+commit(): void <<override>>
+closeSession(): void <<override>>
+closeFactory(): void <<override>>
+loadAll(String): List<T>

Figure 50 - Login controller class

Register employee controller class

RegisterFormController is the controller responsible for the creation of an employee object. It validates the user inputs and save the object to the database through the Dao class.

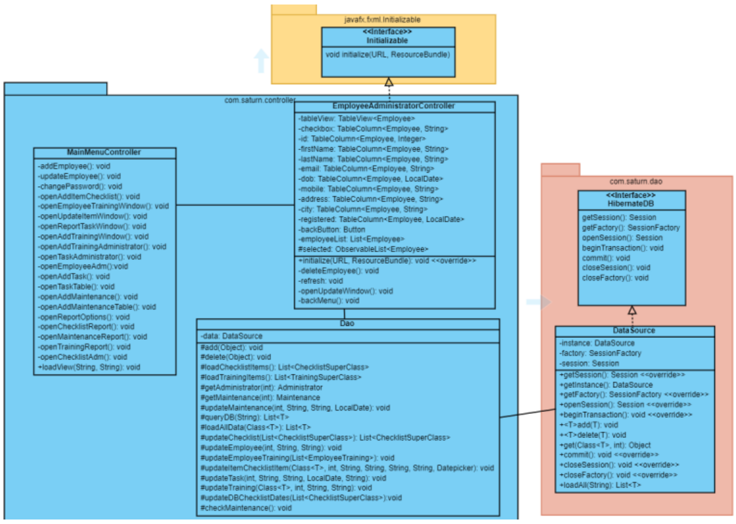


com.saturn.controller
MainMenuController
-addEmployee(): void
-updateEmployee(): void
-changePassword(): void
-openAddItemChecklist(): void
-openEmployeeTrainingWindow(): void
-openUpdateItemWindow(): void
-openReportTaskWindow(): void
-openAddTrainingWindow(): void
-openAddTrainingAdministrator(): void
-openTaskAdministrator(): void
-openEmployeeAdm():void
-openAddTask(): void
-openTaskTable(): void
-openAddMaintenance(): void
-openAddMaintenanceTable(): void
-openReportOptions(): void
-openChecklistReport(): void
-openMaintenanceReport(): void
-openTrainingReport(): void
-openChecklistAdm(): void
+loadView(String, String): void
RegisterFormController
-firstNameTF: TextField
-lastNameTF: TextField
-emailTF: TextField
-dob: DatePicker
-mobileTF: TextField
-telephoneTF: TextField
-addressTF: TextField
-cityTF: TextField
-backToMenuButton: Button
-firstNameLB: Label
-lastNameLB: Label
-emailLB: Label
-dobLB: Label
-mobileLB: Label
-addressLB: Label
-cityLB: Label
-backToMenu(): void
-saveEmployeeDatabase(): void
Dao
-data: DataSource
#add(Object): void
#delete(Object): void
#loadChecklistItems(): List<ChecklistSuperClass>
#loadTrainingItems(): List<TrainingSuperClass>
#getAdministrator(int): Administrator
#getMaintenance(int): Maintenance
#updateMaintenance(int, String, String, LocalDate): void
#queryDB(String): List<T>
#loadAllData(Class<T>): List<T>
#updateChecklist(List<ChecklistSuperClass>): List<ChecklistSuperClass>
#updateEmployee(int, String, String): void
#updateEmployeeTraining(List<EmployeeTraining>): void
#updateItemChecklistItem(Class<T>, int, String, String, String, String, Datepicker): void
#updateTask(int, String, String, LocalDate, String): void
#updateTraining(Class<T>, int, String, String): void
#updateDBChecklistDates(List<ChecklistSuperClass>):void
#checkMaintenance(): void
com.saturn.dao
<<interface>>
HibernateDB
getSession(): Session
getFactory(): SessionFactory
openSession(): Session
beginTransaction(): void
commit(): void
closeSession(): void
closeFactory(): void
DataSource
-instance: DataSource
-factory: SessionFactory
-session: Session
+getSession(): Session <<override>>
+getInstance(): DataSource
+getFactory(): SessionFactory <<override>>
+openSession(): Session <<override>>
+beginTransaction(): void <<override>>
+<T>add(T): void
+<T>delete(T): void
+get(Class<T>, int): Object
+commit(): void <<override>>
+closeSession(): void <<override>>
+closeFactory(): void <<override>>
+loadAll(String): List<T>

Figure 51 - Register employee class

Employee manager class

EmployeeAdministratorController is the controller responsible for deletion of an employee object. It fills up the table view with employee objects from the database through the Dao class.



javafx.fxml.Initializable
<<Interface>>
Initializable
void initialize(URL, ResourceBundle)
com.saturn.controller
MainMenuController
-addEmployee(): void
-updateEmployee(): void
-changePassword(): void
-openAddItemChecklist(): void
-openEmployeeTrainingWindow(): void
-openUpdateItemWindow(): void
-openReportTaskWindow(): void
-openAddTrainingWindow(): void
-openAddTrainingAdministrator(): void
-openTaskAdministrator(): void
-openEmployeeAdm():void
-openAddTask(): void
-openTaskTable(): void
-openAddMaintenance(): void
-openAddMaintenanceTable(): void
-openReportOptions(): void
-openChecklistReport(): void
-openMaintenanceReport(): void
-openTrainingReport(): void
-openChecklistAdm(): void
+loadView(String, String): void
EmployeeAdministratorController
-tableView: TableView<Employee>
-checkbox: TableColumn<Employee, String>
-id: TableColumn<Employee, Integer>
-firstName: TableColumn<Employee, String>
-lastName: TableColumn<Employee, String>
-email: TableColumn<Employee, String>
-dob: TableColumn<Employee, LocalDate>
-mobile: TableColumn<Employee, String>
-address: TableColumn<Employee, String>
-city: TableColumn<Employee, String>
-registered: TableColumn<Employee, LocalDate>
-backButton: Button
-employeeList: List<Employee>
#selected: ObservableList<Employee>
+initialize(URL, ResourceBundle): void <<override>>
-deleteEmployee(): void
-refresh: void
-openUpdateWindow(): void
-backMenu(): void
Dao
-data: DataSource
#add(Object): void
#delete(Object): void
#loadChecklistItems(): List<ChecklistSuperClass>
#loadTrainingItems(): List<TrainingSuperClass>
#getAdministrator(int): Administrator
#getMaintenance(int): Maintenance
#updateMaintenance(int, String, String, LocalDate): void
#queryDB(String): List<T>
#loadAllData(Class<T>): List<T>
#updateChecklist(List<ChecklistSuperClass>): List<ChecklistSuperClass>
#updateEmployee(int, String, String): void
#updateEmployeeTraining(List<EmployeeTraining>): void
#updateItemChecklistItem(Class<T>, int, String, String, String, String, Datepicker): void
#updateTask(int, String, String, LocalDate, String): void
#updateTraining(Class<T>, int, String, String): void
#updateDBChecklistDates(List<ChecklistSuperClass>):void
#checkMaintenance(): void
com.saturn.dao
<<Interface>>
HibernateDB
getSession(): Session
getFactory(): SessionFactory
openSession(): Session
beginTransaction(): void
commit(): void
closeSession(): void
closeFactory(): void
DataSource
-instance: DataSource
-factory: SessionFactory
-session: Session
+getSession(): Session <<override>>
+getInstance(): DataSource
+getFactory(): SessionFactory <<override>>
+openSession(): Session <<override>>
+beginTransaction(): void <<override>>
+<T>add(T): void
+<T>delete(T): void
+get(Class<T>, int): Object
+commit(): void <<override>>
+closeSession(): void <<override>>
+closeFactory(): void <<override>>
+loadAll(String): List<T>

Figure 52 - Employee manager controller class

Update employee class

UpdateEmployeeController is responsible for the updates of an employee object. It interacts with the database through the Dao class.

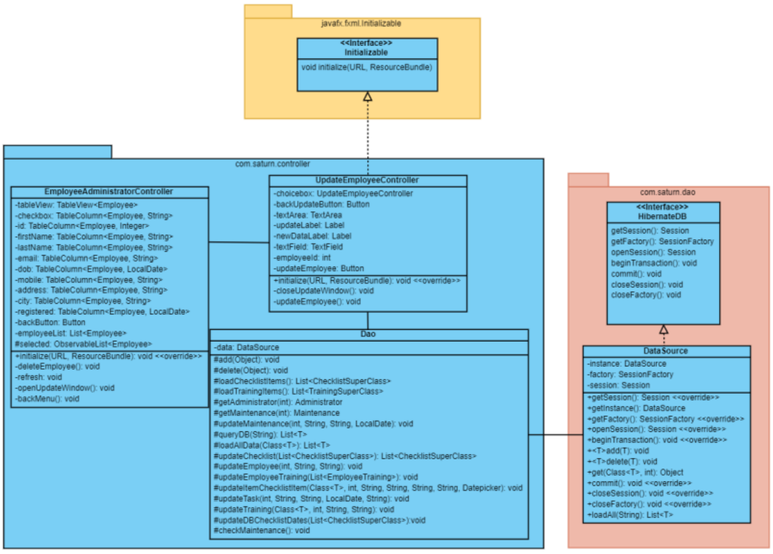

javafx.fxml.Initializable
<<Interface>>
Initializable
void initialize(URL, ResourceBundle)
com.saturn.controller
EmployeeAdministratorController
-tableView: TableView<Employee>
-checkbox: TableColumn<Employee, String>
-id: TableColumn<Employee, Integer>
-firstName: TableColumn<Employee, String>
-lastName: TableColumn<Employee, String>
-email: TableColumn<Employee, String>
-dob: TableColumn<Employee, LocalDate>
-mobile: TableColumn<Employee, String>
-address: TableColumn<Employee, String>
-city: TableColumn<Employee, String>
-registered: TableColumn<Employee, LocalDate>
-backButton: Button
-employeeList: List<Employee>
#selected: ObservableList<Employee>
+initialize(URL, ResourceBundle): void <<override>>
-deleteEmployee(): void
-refresh: void
-openUpdateWindow(): void
-backMenu(): void
UpdateEmployeeController
-choicebox: UpdateEmployeeController
-backUpdateButton: Button
-textArea: TextArea
-updateLabel: Label
-newDataLabel: Label
-textField: TextField
-employeeId: int
-updateEmployee: Button
+initialize(URL, ResourceBundle): void <<override>>
-closeUpdateWindow(): void
-updateEmployee(): void
Dao
-data: DataSource
#add(Object): void
#delete(Object): void
#loadChecklistItems(): List<ChecklistSuperClass>
#loadTrainingItems(): List<TrainingSuperClass>
#getAdministrator(int): Administrator
#getMaintenance(int): Maintenance
#updateMaintenance(int, String, String, LocalDate): void
#queryDB(String): List<T>
#loadAllData(Class<T>): List<T>
#updateChecklist(List<ChecklistSuperClass>): List<ChecklistSuperClass>
#updateEmployee(int, String, String): void
#updateEmployeeTraining(List<EmployeeTraining>): void
#updateItemChecklistItem(Class<T>, int, String, String, String, String, Datepicker): void
#updateTask(int, String, String, LocalDate, String): void
#updateTraining(Class<T>, int, String, String): void
#updateDBChecklistDates(List<ChecklistSuperClass>):void
#checkMaintenance(): void
com.saturn.dao
<<Interface>>
HibernateDB
getSession(): Session
getFactory(): SessionFactory
openSession(): Session
beginTransaction(): void
commit(): void
closeSession(): void
closeFactory(): void
DataSource
-instance: DataSource
-factory: SessionFactory
-session: Session
+getSession(): Session <<override>>
+getInstance(): DataSource
+getFactory(): SessionFactory <<override>>
+openSession(): Session <<override>>
+beginTransaction(): void <<override>>
+<T>add(T): void
+<T>delete(T): void
+get(Class<T>, int): Object
+commit(): void <<override>>
+closeSession(): void <<override>>
+closeFactory(): void <<override>>
+loadAll(String): List<T>

Figure 53 - Update employee controller

Add checklist item class

AddChecklistController class is responsible for processing inputs and creating a checklist object.

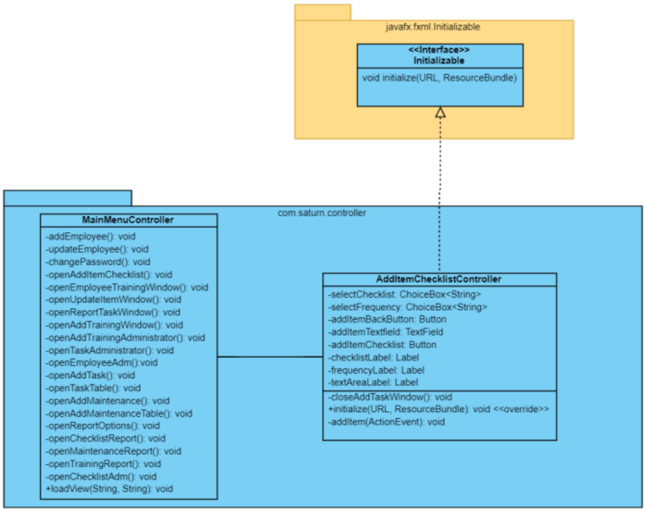

javafx.fxml.Initializable
<<Interface>>
Initializable
void initialize(URL, ResourceBundle)
com.saturn.controller
MainMenuController
-addEmployee(): void
-updateEmployee(): void
-changePassword(): void
-openAddItemChecklist(): void
-openEmployeeTrainingWindow(): void
-openUpdateItemWindow(): void
-openReportTaskWindow(): void
-openAddTrainingWindow(): void
-openAddTrainingAdministrator(): void
-openTaskAdministrator(): void
-openEmployeeAdm():void
-openAddTask(): void
-openTaskTable(): void
-openAddMaintenance(): void
-openAddMaintenanceTable(): void
-openReportOptions(): void
-openChecklistReport(): void
-openMaintenanceReport(): void
-openTrainingReport(): void
-openChecklistAdm(): void
+loadView(String, String): void
AddItemChecklistController
-selectChecklist: ChoiceBox<String>
-selectFrequency: ChoiceBox<String>
-addItemBackButton: Button
-addItemTextfield: TextField
-addItemChecklist: Button
-checklistLabel: Label
-frequencyLabel: Label
-textAreaLabel: Label
-closeAddTaskWindow(): void
+initialize(URL, ResourceBundle): void <<override>>
-addItem(ActionEvent): void

Figure 54 - Add checklist controller

Checklist manager class

ChecklistManagerController is responsible for update the status of a checklist object. It changes the status of a checklist item from ‘pending’ to ‘done’. It interacts with Dao class that is the bridge to the database.

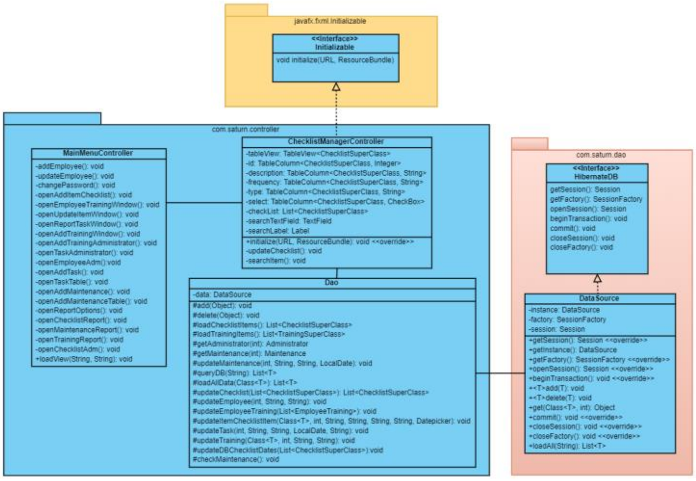

javafx.fxml.Initializable
<<Interface>>
Initializable
void initialize(URL, ResourceBundle)
com.saturn.controller
MainMenuController
-addEmployee(): void
-updateEmployee(): void
-changePassword(): void
-openAddItemChecklist(): void
-openEmployeeTrainingWindow(): void
-openUpdateItemWindow(): void
-openReportTaskWindow(): void
-openAddTrainingWindow(): void
-openAddTrainingAdministrator(): void
-openTaskAdministrator(): void
-openEmployeeAdm():void
-openAddTask(): void
-openTaskTable(): void
-openAddMaintenance(): void
-openAddMaintenanceTable(): void
-openReportOptions(): void
-openChecklistReport(): void
-openMaintenanceReport(): void
-openTrainingReport(): void
-openChecklistAdm(): void
+loadView(String, String): void
ChecklistManagerController
-tableView: TableView<ChecklistSuperClass>
-id: TableColumn<ChecklistSuperClass, Integer>
-description: TableColumn<ChecklistSuperClass, String>
-frequency: TableColumn<ChecklistSuperClass, String>
-type: TableColumn<ChecklistSuperClass, String>
-select: TableColumn<ChecklistSuperClass, CheckBox>
-checkList: List<ChecklistSuperClass>
-searchTextField: TextField
-searchLabel: Label
+initialize(URL, ResourceBundle): void <<override>>
-updateChecklist(): void
-searchItem(): void
Dao
-data: DataSource
#add(Object): void
#delete(Object): void
#loadChecklistItems(): List<ChecklistSuperClass>
#loadTrainingItems(): List<TrainingSuperClass>
#getAdministrator(int): Administrator
#getMaintenance(int): Maintenance
#updateMaintenance(int, String, String, LocalDate): void
#queryDB(String): List<T>
#loadAllData(Class<T>): List<T>
#updateChecklist(List<ChecklistSuperClass>): List<ChecklistSuperClass>
#updateEmployee(int, String, String): void
#updateEmployeeTraining(List<EmployeeTraining>): void
#updateItemChecklistItem(Class<T>, int, String, String, String, String, Datepicker): void
#updateTask(int, String, String, LocalDate, String): void
#updateTraining(Class<T>, int, String, String): void
#updateDBChecklistDates(List<ChecklistSuperClass>):void
#checkMaintenance(): void
com.saturn.dao
<<Interface>>
HibernateDB
getSession(): Session
getFactory(): SessionFactory
openSession(): Session
beginTransaction(): void
commit(): void
closeSession(): void
closeFactory(): void
DataSource
-instance: DataSource
-factory: SessionFactory
-session: Session
+getSession(): Session <<override>>
+getInstance(): DataSource
+getFactory(): SessionFactory <<override>>
+openSession(): Session <<override>>
+beginTransaction(): void <<override>>
+<T>add(T): void
+<T>delete(T): void
+get(Class<T>, int): Object
+commit(): void <<override>>
+closeSession(): void <<override>>
+closeFactory(): void <<override>>
+loadAll(String): List<T>

Figure 55 - Checklist manager controller

Checklist manager class II

TaskAdministratorController is responsible for the deletion of a checklist object. This class fills up the table view with all checklist objects stored in the database.

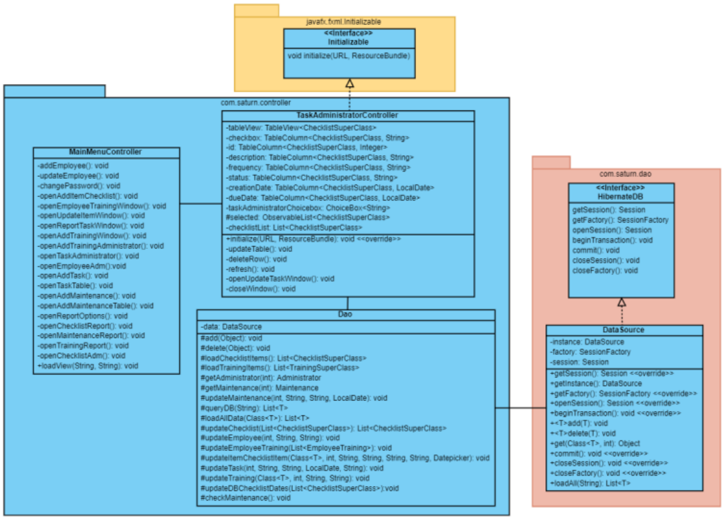

javafx.fxml.Initializable
<<Interface>>
Initializable
void initialize(URL, ResourceBundle)
com.saturn.controller
MainMenuController
-addEmployee(): void
-updateEmployee(): void
-changePassword(): void
-openAddItemChecklist(): void
-openEmployeeTrainingWindow(): void
-openUpdateItemWindow(): void
-openReportTaskWindow(): void
-openAddTrainingWindow(): void
-openAddTrainingAdministrator(): void
-openTaskAdministrator(): void
-openEmployeeAdm():void
-openAddTask(): void
-openTaskTable(): void
-openAddMaintenance(): void
-openAddMaintenanceTable(): void
-openReportOptions(): void
-openChecklistReport(): void
-openMaintenanceReport(): void
-openTrainingReport(): void
-openChecklistAdm(): void
+loadView(String, String): void
TaskAdministratorController
-tableView: TableView<ChecklistSuperClass>
-checkbox: TableColumn<ChecklistSuperClass, String>
-id: TableColumn<ChecklistSuperClass, Integer>
-description: TableColumn<ChecklistSuperClass, String>
-frequency: TableColumn<ChecklistSuperClass, String>
-status: TableColumn<ChecklistSuperClass, String>
-creationDate: TableColumn<ChecklistSuperClass, LocalDate>
-dueDate: TableColumn<ChecklistSuperClass, LocalDate>
-taskAdministratorChoicebox: ChoiceBox<String>
#selected: ObservableList<ChecklistSuperClass>
-checklistList: List<ChecklistSuperClass>
+initialize(URL, ResourceBundle): void <<override>>
-updateTable(): void
-deleteRow(): void
-refresh(): void
-openUpdateTaskWindow(): void
-closeWindow(): void
Dao
-data: DataSource
#add(Object): void
#delete(Object): void
#loadChecklistItems(): List<ChecklistSuperClass>
#loadTrainingItems(): List<TrainingSuperClass>
#getAdministrator(int): Administrator
#getMaintenance(int): Maintenance
#updateMaintenance(int, String, String, LocalDate): void
#queryDB(String): List<T>
#loadAllData(Class<T>): List<T>
#updateChecklist(List<ChecklistSuperClass>): List<ChecklistSuperClass>
#updateEmployee(int, String, String): void
#updateEmployeeTraining(List<EmployeeTraining>): void
#updateItemChecklistItem(Class<T>, int, String, String, String, String, Datepicker): void
#updateTask(int, String, String, LocalDate, String): void
#updateTraining(Class<T>, int, String, String): void
#updateDBChecklistDates(List<ChecklistSuperClass>):void
#checkMaintenance(): void
com.saturn.dao
<<Interface>>
HibernateDB
getSession(): Session
getFactory(): SessionFactory
openSession(): Session
beginTransaction(): void
commit(): void
closeSession(): void
closeFactory(): void
DataSource
-instance: DataSource
-factory: SessionFactory
-session: Session
+getSession(): Session <<override>>
+getInstance(): DataSource
+getFactory(): SessionFactory <<override>>
+openSession(): Session <<override>>
+beginTransaction(): void <<override>>
+<T>add(T): void
+<T>delete(T): void
+get(Class<T>, int): Object
+commit(): void <<override>>
+closeSession(): void <<override>>
+closeFactory(): void <<override>>
+loadAll(String): List<T>

Figure 56 - Checklist manager

Update checklist items

UpdateItemChecklistController is responsible for the updates of a checklist object. It processes the inputs and interacts with the database through the Dao class.

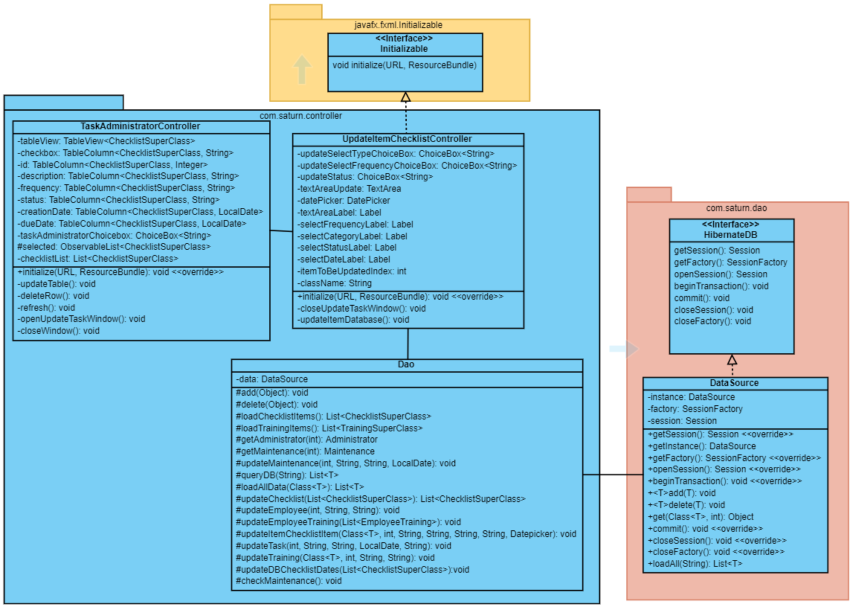

javafx.fxml.Initializable
<<Interface>>
Initializable
void initialize(URL, ResourceBundle)
com.saturn.controller
TaskAdministratorController
-tableView: TableView<ChecklistSuperClass>
-checkbox: TableColumn<ChecklistSuperClass, String>
-id: TableColumn<ChecklistSuperClass, Integer>
-description: TableColumn<ChecklistSuperClass, String>
-frequency: TableColumn<ChecklistSuperClass, String>
-status: TableColumn<ChecklistSuperClass, String>
-creationDate: TableColumn<ChecklistSuperClass, LocalDate>
-dueDate: TableColumn<ChecklistSuperClass, LocalDate>
-taskAdministratorChoicebox: ChoiceBox<String>
#selected: ObservableList<ChecklistSuperClass>
-checklistList: List<ChecklistSuperClass>
+initialize(URL, ResourceBundle): void <<override>>
-updateTable(): void
-deleteRow(): void
-refresh(): void
-openUpdateTaskWindow(): void
-closeWindow(): void
UpdateItemChecklistController
-updateSelectTypeChoiceBox: ChoiceBox<String>
-updateSelectFrequencyChoiceBox: ChoiceBox<String>
-updateStatus: ChoiceBox<String>
-textAreaUpdate: TextArea
-datePicker: DatePicker
-textAreaLabel: Label
-selectFrequencyLabel: Label
-selectCategoryLabel: Label
-selectStatusLabel: Label
-selectDateLabel: Label
-itemToBeUpdatedIndex: int
-className: String
+initialize(URL, ResourceBundle): void <<override>>
-closeUpdateTaskWindow(): void
-updateItemDatabase(): void
Dao
-data: DataSource
#add(Object): void
#delete(Object): void
#loadChecklistItems(): List<ChecklistSuperClass>
#loadTrainingItems(): List<TrainingSuperClass>
#getAdministrator(int): Administrator
#getMaintenance(int): Maintenance
#updateMaintenance(int, String, String, LocalDate): void
#queryDB(String): List<T>
#loadAllData(Class<T>): List<T>
#updateChecklist(List<ChecklistSuperClass>): List<ChecklistSuperClass>
#updateEmployee(int, String, String): void
#updateEmployeeTraining(List<EmployeeTraining>): void
#updateItemChecklistItem(Class<T>, int, String, String, String, String, Datepicker): void
#updateTask(int, String, String, LocalDate, String): void
#updateTraining(Class<T>, int, String, String): void
#updateDBChecklistDates(List<ChecklistSuperClass>):void
#checkMaintenance(): void
com.saturn.dao
<<Interface>>
HibernateDB
getSession(): Session
getFactory(): SessionFactory
openSession(): Session
beginTransaction(): void
commit(): void
closeSession(): void
closeFactory(): void
DataSource
-instance: DataSource
-factory: SessionFactory
-session: Session
+getSession(): Session <<override>>
+getInstance(): DataSource
+getFactory(): SessionFactory <<override>>
+openSession(): Session <<override>>
+beginTransaction(): void <<override>>
+<T>add(T): void
+<T>delete(T): void
+get(Class<T>, int): Object
+commit(): void <<override>>
+closeSession(): void <<override>>
+closeFactory(): void <<override>>
+loadAll(String): List<T>

Figure 57 - Update checklist item controller

Add task class

AddTaskController is responsible for processing inputs for the creation of a task object. It interacts with the database through the Dao class.

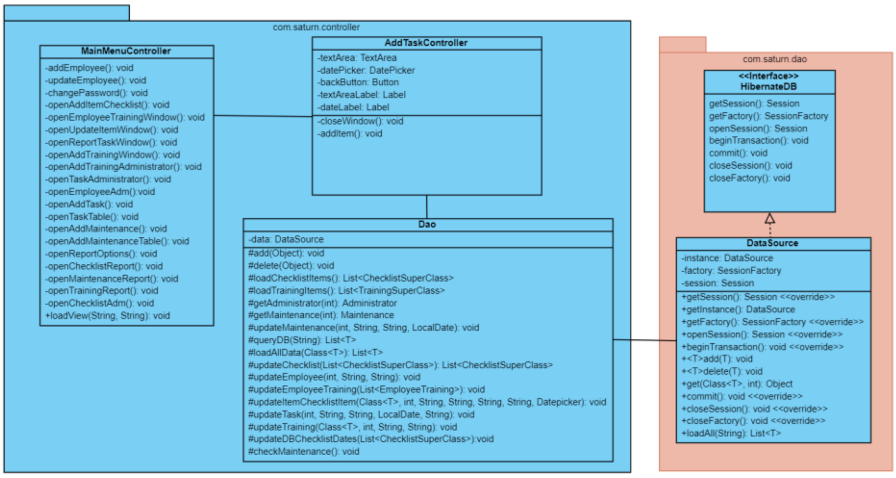

com.saturn.controller
MainMenuController
-addEmployee(): void
-updateEmployee(): void
-changePassword(): void
-openAddItemChecklist(): void
-openEmployeeTrainingWindow(): void
-openUpdateItemWindow(): void
-openReportTaskWindow(): void
-openAddTrainingWindow(): void
-openAddTrainingAdministrator(): void
-openTaskAdministrator(): void
-openEmployeeAdm():void
-openAddTask(): void
-openTaskTable(): void
-openAddMaintenance(): void
-openAddMaintenanceTable(): void
-openReportOptions(): void
-openChecklistReport(): void
-openMaintenanceReport(): void
-openTrainingReport(): void
-openChecklistAdm(): void
+loadView(String, String): void
AddTaskController
-textArea: TextArea
-datePicker: DatePicker
-backButton: Button
-textAreaLabel: Label
-dateLabel: Label
-closeWindow(): void
-addItem(): void
Dao
-data: DataSource
#add(Object): void
#delete(Object): void
#loadChecklistItems(): List<ChecklistSuperClass>
#loadTrainingItems(): List<TrainingSuperClass>
#getAdministrator(int): Administrator
#getMaintenance(int): Maintenance
#updateMaintenance(int, String, String, LocalDate): void
#queryDB(String): List<T>
#loadAllData(Class<T>): List<T>
#updateChecklist(List<ChecklistSuperClass>): List<ChecklistSuperClass>
#updateEmployee(int, String, String): void
#updateEmployeeTraining(List<EmployeeTraining>): void
#updateItemChecklistItem(Class<T>, int, String, String, String, String, Datepicker): void
#updateTask(int, String, String, LocalDate, String): void
#updateTraining(Class<T>, int, String, String): void
#updateDBChecklistDates(List<ChecklistSuperClass>):void
#checkMaintenance(): void
com.saturn.dao
<<Interface>>
HibernateDB
getSession(): Session
getFactory(): SessionFactory
openSession(): Session
beginTransaction(): void
commit(): void
closeSession(): void
closeFactory(): void
DataSource
-instance: DataSource
-factory: SessionFactory
-session: Session
+getSession(): Session <<override>>
+getInstance(): DataSource
+getFactory(): SessionFactory <<override>>
+openSession(): Session <<override>>
+beginTransaction(): void <<override>>
+<T>add(T): void
+<T>delete(T): void
+get(Class<T>, int): Object
+commit(): void <<override>>
+closeSession(): void <<override>>
+closeFactory(): void <<override>>
+loadAll(String): List<T>

Figure 58 - Add task controller class

Task manager

TaskTableController is responsible for the deletion of a task object. Also, it fills up the table view with all task objects stored in the database. It interacts with the database through the Dao class.

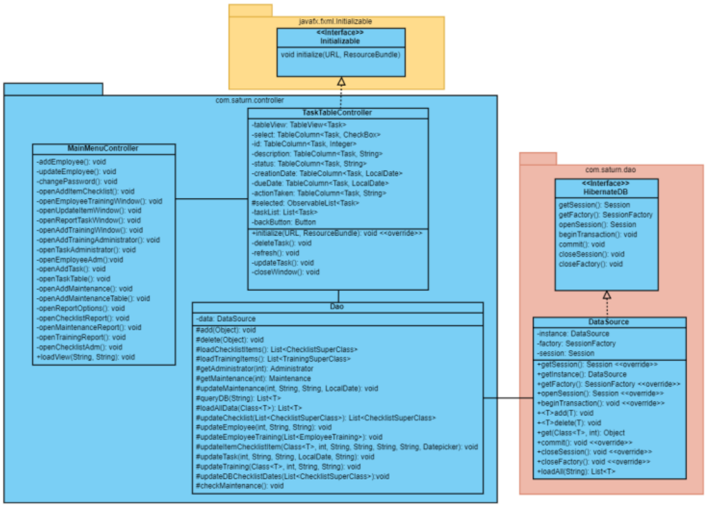

javafx.fxml.Initializable
<<Interface>>
Initializable
void initialize(URL, ResourceBundle)
com.saturn.controller
TaskTableController
-tableView: TableView<Task>
-select: TableColumn<Task, CheckBox>
-id: TableColumn<Task, Integer>
-description: TableColumn<Task, String>
-status: TableColumn<Task, String>
-creationDate: TableColumn<Task, LocalDate>
-dueDate: TableColumn<Task, LocalDate>
-actionTaken: TableColumn<Task, String>
#selected: ObservableList<Task>
-taskList: List<Task>
-backButton: Button
+initialize(URL, ResourceBundle): void <<override>>
-deleteTask(): void
-refresh(): void
-updateTask(): void
-closeWindow(): void
MainMenuController
-addEmployee(): void
-updateEmployee(): void
-changePassword(): void
-openAddItemChecklist(): void
-openEmployeeTrainingWindow(): void
-openUpdateItemWindow(): void
-openReportTaskWindow(): void
-openAddTrainingWindow(): void
-openAddTrainingAdministrator(): void
-openTaskAdministrator(): void
-openEmployeeAdm():void
-openAddTask(): void
-openTaskTable(): void
-openAddMaintenance(): void
-openAddMaintenanceTable(): void
-openReportOptions(): void
-openChecklistReport(): void
-openMaintenanceReport(): void
-openTrainingReport(): void
-openChecklistAdm(): void
+loadView(String, String): void
Dao
-data: DataSource
#add(Object): void
#delete(Object): void
#loadChecklistItems(): List<ChecklistSuperClass>
#loadTrainingItems(): List<TrainingSuperClass>
#getAdministrator(int): Administrator
#getMaintenance(int): Maintenance
#updateMaintenance(int, String, String, LocalDate): void
#queryDB(String): List<T>
#loadAllData(Class<T>): List<T>
#updateChecklist(List<ChecklistSuperClass>): List<ChecklistSuperClass>
#updateEmployee(int, String, String): void
#updateEmployeeTraining(List<EmployeeTraining>): void
#updateItemChecklistItem(Class<T>, int, String, String, String, String, Datepicker): void
#updateTask(int, String, String, LocalDate, String): void
#updateTraining(Class<T>, int, String, String): void
#updateDBChecklistDates(List<ChecklistSuperClass>):void
#checkMaintenance(): void
com.saturn.dao
<<Interface>>
HibernateDB
getSession(): Session
getFactory(): SessionFactory
openSession(): Session
beginTransaction(): void
commit(): void
closeSession(): void
closeFactory(): void
DataSource
-instance: DataSource
-factory: SessionFactory
-session: Session
+getSession(): Session <<override>>
+getInstance(): DataSource
+getFactory(): SessionFactory <<override>>
+openSession(): Session <<override>>
+beginTransaction(): void <<override>>
+<T>add(T): void
+<T>delete(T): void
+get(Class<T>, int): Object
+commit(): void <<override>>
+closeSession(): void <<override>>
+closeFactory(): void <<override>>
+loadAll(String): List<T>

Figure 59 - Task manager controller class

Update task

UpdateTaskController is responsible for the updates of a task object. It interacts with the database through the Dao class.

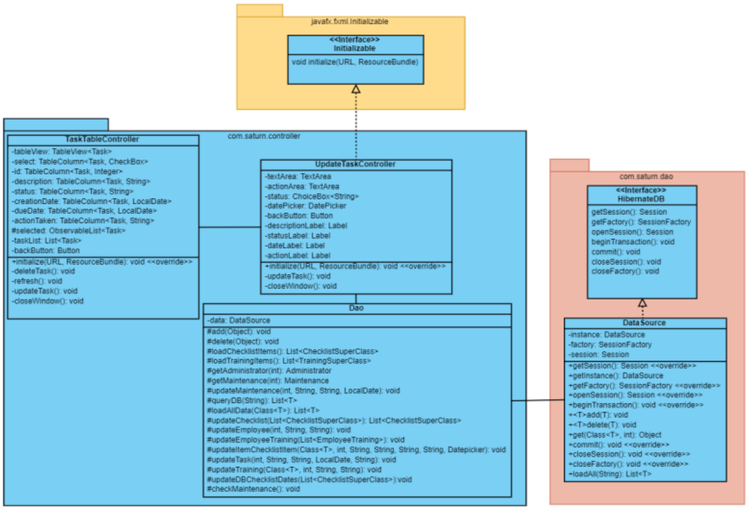

javafx.fxml.Initializable
<<Interface>>
Initializable
void initialize(URL, ResourceBundle)
com.saturn.controller
TaskTableController
-tableView: TableView<Task>
-select: TableColumn<Task, CheckBox>
-id: TableColumn<Task, Integer>
-description: TableColumn<Task, String>
-status: TableColumn<Task, String>
-creationDate: TableColumn<Task, LocalDate>
-dueDate: TableColumn<Task, LocalDate>
-actionTaken: TableColumn<Task, String>
#selected: ObservableList<Task>
-taskList: List<Task>
-backButton: Button
+initialize(URL, ResourceBundle): void <<override>>
-deleteTask(): void
-refresh(): void
-updateTask(): void
-closeWindow(): void
UpdateTaskController
-textArea: TextArea
-actionArea: TextArea
-status: ChoiceBox<String>
-datePicker: DatePicker
-backButton: Button
-descriptionLabel: Label
-statusLabel: Label
-dateLabel: Label
-actionLabel: Label
+initialize(URL, ResourceBundle): void <<override>>
-updateTask(): void
-closeWindow(): void
Dao
-data: DataSource
#add(Object): void
#delete(Object): void
#loadChecklistItems(): List<ChecklistSuperClass>
#loadTrainingItems(): List<TrainingSuperClass>
#getAdministrator(int): Administrator
#getMaintenance(int): Maintenance
#updateMaintenance(int, String, String, LocalDate): void
#queryDB(String): List<T>
#loadAllData(Class<T>): List<T>
#updateChecklist(List<ChecklistSuperClass>): List<ChecklistSuperClass>
#updateEmployee(int, String, String): void
#updateEmployeeTraining(List<EmployeeTraining>): void
#updateItemChecklistItem(Class<T>, int, String, String, String, String, Datepicker): void
#updateTask(int, String, String, LocalDate, String): void
#updateTraining(Class<T>, int, String, String): void
#updateDBChecklistDates(List<ChecklistSuperClass>):void
#checkMaintenance(): void
com.saturn.dao
<<Interface>>
HibernateDB
getSession(): Session
getFactory(): SessionFactory
openSession(): Session
beginTransaction(): void
commit(): void
closeSession(): void
closeFactory(): void
DataSource
-instance: DataSource
-factory: SessionFactory
-session: Session
+getSession(): Session <<override>>
+getInstance(): DataSource
+getFactory(): SessionFactory <<override>>
+openSession(): Session <<override>>
+beginTransaction(): void <<override>>
+<T>add(T): void
+<T>delete(T): void
+get(Class<T>, int): Object
+commit(): void <<override>>
+closeSession(): void <<override>>
+closeFactory(): void <<override>>
+loadAll(String): List<T>

Figure 60 - Update task controller class

Add training

AddTrainingController is responsible for processing inputs for the creation of a training object. It interacts with the database through the Dao class.

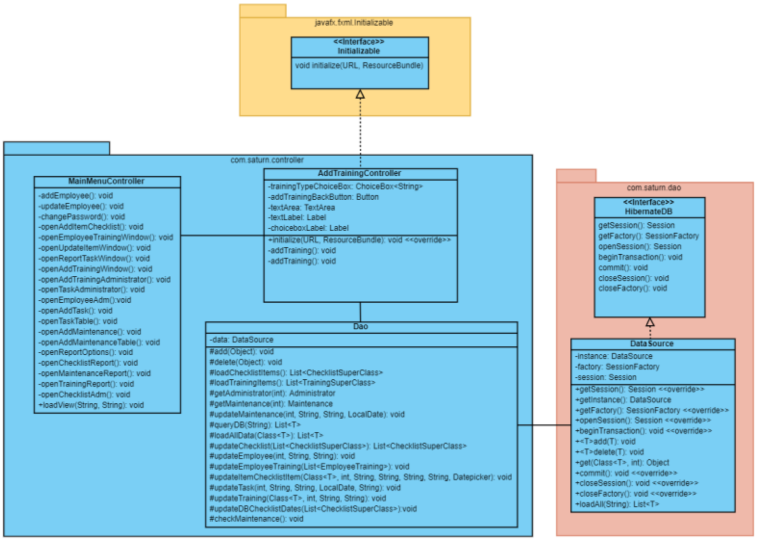

javafx.fxml.Initializable
<<Interface>>
Initializable
void initialize(URL, ResourceBundle)
com.saturn.controller
MainMenuController
-addEmployee(): void
-updateEmployee(): void
-changePassword(): void
-openAddItemChecklist(): void
-openEmployeeTrainingWindow(): void
-openUpdateItemWindow(): void
-openReportTaskWindow(): void
-openAddTrainingWindow(): void
-openAddTrainingAdministrator(): void
-openTaskAdministrator(): void
-openEmployeeAdm():void
-openAddTask(): void
-openTaskTable(): void
-openAddMaintenance(): void
-openAddMaintenanceTable(): void
-openReportOptions(): void
-openChecklistReport(): void
-openMaintenanceReport(): void
-openTrainingReport(): void
-openChecklistAdm(): void
+loadView(String, String): void
AddTrainingController
-trainingTypeChoiceBox: ChoiceBox<String>
-addTrainingBackButton: Button
-textArea: TextArea
-textLabel: Label
-choiceboxLabel: Label
+initialize(URL, ResourceBundle): void <<override>>
-addTraining(): void
-addTraining(): void
Dao
-data: DataSource
#add(Object): void
#delete(Object): void
#loadChecklistItems(): List<ChecklistSuperClass>
#loadTrainingItems(): List<TrainingSuperClass>
#getAdministrator(int): Administrator
#getMaintenance(int): Maintenance
#updateMaintenance(int, String, String, LocalDate): void
#queryDB(String): List<T>
#loadAllData(Class<T>): List<T>
#updateChecklist(List<ChecklistSuperClass>): List<ChecklistSuperClass>
#updateEmployee(int, String, String): void
#updateEmployeeTraining(List<EmployeeTraining>): void
#updateItemChecklistItem(Class<T>, int, String, String, String, String, Datepicker): void
#updateTask(int, String, String, LocalDate, String): void
#updateTraining(Class<T>, int, String, String): void
#updateDBChecklistDates(List<ChecklistSuperClass>):void
#checkMaintenance(): void
com.saturn.dao
<<Interface>>
HibernateDB
getSession(): Session
getFactory(): SessionFactory
openSession(): Session
beginTransaction(): void
commit(): void
closeSession(): void
closeFactory(): void
DataSource
-instance: DataSource
-factory: SessionFactory
-session: Session
+getSession(): Session <<override>>
+getInstance(): DataSource
+getFactory(): SessionFactory <<override>>
+openSession(): Session <<override>>
+beginTransaction(): void <<override>>
+<T>add(T): void
+<T>delete(T): void
+get(Class<T>, int): Object
+commit(): void <<override>>
+closeSession(): void <<override>>
+closeFactory(): void <<override>>
+loadAll(String): List<T>

Figure 61 - Add training controller class

Training manager

TrainingAdministratorController is responsible for filling up the table view with training objects and deletion. It interacts with the database through the Dao class.

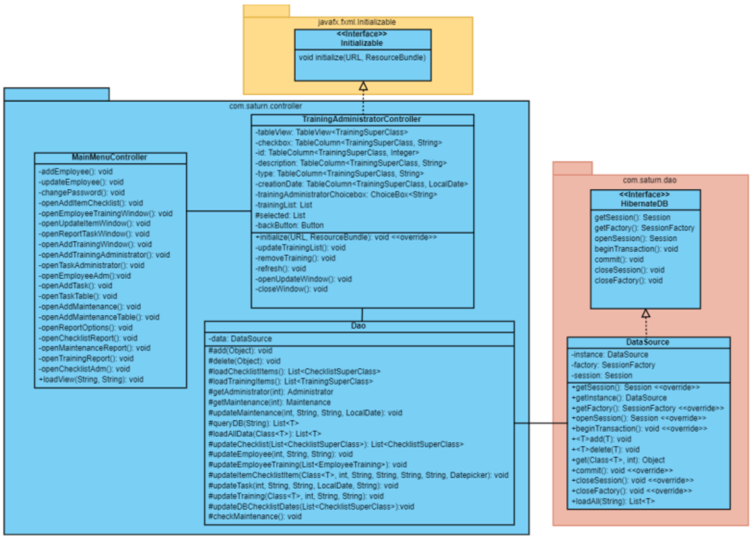

javafx.fxml.Initializable
<<Interface>>
Initializable
void initialize(URL, ResourceBundle)
com.saturn.controller
MainMenuController
-addEmployee(): void
-updateEmployee(): void
-changePassword(): void
-openAddItemChecklist(): void
-openEmployeeTrainingWindow(): void
-openUpdateItemWindow(): void
-openReportTaskWindow(): void
-openAddTrainingWindow(): void
-openAddTrainingAdministrator(): void
-openTaskAdministrator(): void
-openEmployeeAdm():void
-openAddTask(): void
-openTaskTable(): void
-openAddMaintenance(): void
-openAddMaintenanceTable(): void
-openReportOptions(): void
-openChecklistReport(): void
-openMaintenanceReport(): void
-openTrainingReport(): void
-openChecklistAdm(): void
+loadView(String, String): void
TrainingAdministratorController
-tableView: TableView<TrainingSuperClass>
-checkbox: TableColumn<TrainingSuperClass, String>
-id: TableColumn<TrainingSuperClass, Integer>
-description: TableColumn<TrainingSuperClass, String>
-type: TableColumn<TrainingSuperClass, String>
-creationDate: TableColumn<TrainingSuperClass, LocalDate>
-trainingAdministratorChoicebox: ChoiceBox<String>
-trainingList: List
#selected: List
-backButton: Button
+initialize(URL, ResourceBundle): void <<override>>
-updateTrainingList(): void
-removeTraining(): void
-refresh(): void
-openUpdateWindow(): void
-closeWindow(): void
Dao
-data: DataSource
#add(Object): void
#delete(Object): void
#loadChecklistItems(): List<ChecklistSuperClass>
#loadTrainingItems(): List<TrainingSuperClass>
#getAdministrator(int): Administrator
#getMaintenance(int): Maintenance
#updateMaintenance(int, String, String, LocalDate): void
#queryDB(String): List<T>
#loadAllData(Class<T>): List<T>
#updateChecklist(List<ChecklistSuperClass>): List<ChecklistSuperClass>
#updateEmployee(int, String, String): void
#updateEmployeeTraining(List<EmployeeTraining>): void
#updateItemChecklistItem(Class<T>, int, String, String, String, String, Datepicker): void
#updateTask(int, String, String, LocalDate, String): void
#updateTraining(Class<T>, int, String, String): void
#updateDBChecklistDates(List<ChecklistSuperClass>):void
#checkMaintenance(): void
com.saturn.dao
<<Interface>>
HibernateDB
getSession(): Session
getFactory(): SessionFactory
openSession(): Session
beginTransaction(): void
commit(): void
closeSession(): void
closeFactory(): void
DataSource
-instance: DataSource
-factory: SessionFactory
-session: Session
+getSession(): Session <<override>>
+getInstance(): DataSource
+getFactory(): SessionFactory <<override>>
+openSession(): Session <<override>>
+beginTransaction(): void <<override>>
+<T>add(T): void
+<T>delete(T): void
+get(Class<T>, int): Object
+commit(): void <<override>>
+closeSession(): void <<override>>
+closeFactory(): void <<override>>
+loadAll(String): List<T>

Figure 62 - Training manager controller class

Update training

UpdateTrainingController is responsible for the updates of a training object. It interacts with the database through the Dao class.

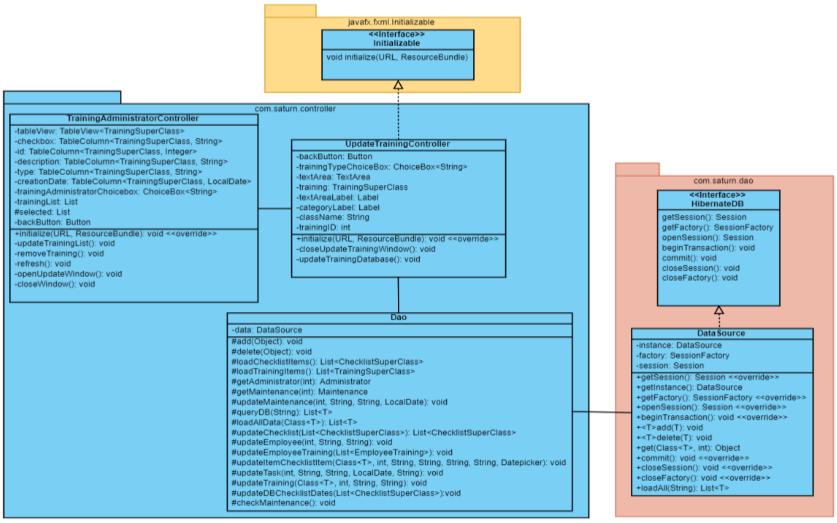

javafx.fxml.Initializable
<<Interface>>
Initializable
void initialize(URL, ResourceBundle)
com.saturn.controller
TrainingAdministratorController
-tableView: TableView<TrainingSuperClass>
-checkbox: TableColumn<TrainingSuperClass, String>
-id: TableColumn<TrainingSuperClass, Integer>
-description: TableColumn<TrainingSuperClass, String>
-type: TableColumn<TrainingSuperClass, String>
-creationDate: TableColumn<TrainingSuperClass, LocalDate>
-trainingAdministratorChoicebox: ChoiceBox<String>
-trainingList: List
#selected: List
-backButton: Button
+initialize(URL, ResourceBundle): void <<override>>
-updateTrainingList(): void
-removeTraining(): void
-refresh(): void
-openUpdateWindow(): void
-closeWindow(): void
UpdateTrainingController
-backButton: Button
-trainingTypeChoiceBox: ChoiceBox<String>
-textArea: TextArea
-training: TrainingSuperClass
-textAreaLabel: Label
-categoryLabel: Label
-className: String
-trainingID: int
+initialize(URL, ResourceBundle): void <<override>>
-closeUpdateTrainingWindow(): void
-updateTrainingDatabase(): void
Dao
-data: DataSource
#add(Object): void
#delete(Object): void
#loadChecklistItems(): List<ChecklistSuperClass>
#loadTrainingItems(): List<TrainingSuperClass>
#getAdministrator(int): Administrator
#getMaintenance(int): Maintenance
#updateMaintenance(int, String, String, LocalDate): void
#queryDB(String): List<T>
#loadAllData(Class<T>): List<T>
#updateChecklist(List<ChecklistSuperClass>): List<ChecklistSuperClass>
#updateEmployee(int, String, String): void
#updateEmployeeTraining(List<EmployeeTraining>): void
#updateItemChecklistItem(Class<T>, int, String, String, String, String, Datepicker): void
#updateTask(int, String, String, LocalDate, String): void
#updateTraining(Class<T>, int, String, String): void
#updateDBChecklistDates(List<ChecklistSuperClass>):void
#checkMaintenance(): void
com.saturn.dao
<<Interface>>
HibernateDB
getSession(): Session
getFactory(): SessionFactory
openSession(): Session
beginTransaction(): void
commit(): void
closeSession(): void
closeFactory(): void
DataSource
-instance: DataSource
-factory: SessionFactory
-session: Session
+getSession(): Session <<override>>
+getInstance(): DataSource
+getFactory(): SessionFactory <<override>>
+openSession(): Session <<override>>
+beginTransaction(): void <<override>>
+<T>add(T): void
+<T>delete(T): void
+get(Class<T>, int): Object
+commit(): void <<override>>
+closeSession(): void <<override>>
+closeFactory(): void <<override>>
+loadAll(String): List<T>

Figure 63 - Update training controller class

Employee training

EmployeeHSECompositeID, EmployeeSeaChangeCompositeID and EmployeeVirtualAcademyCompositeID classes take an employee object and a training object and create a composite primary key using Hibernate annotation. EmployeeHSE, EmployeeSeaChange and EmployeeVirtualAcademy classes take the primary key and save the object in the database in a join operation using Hibernate annotation.

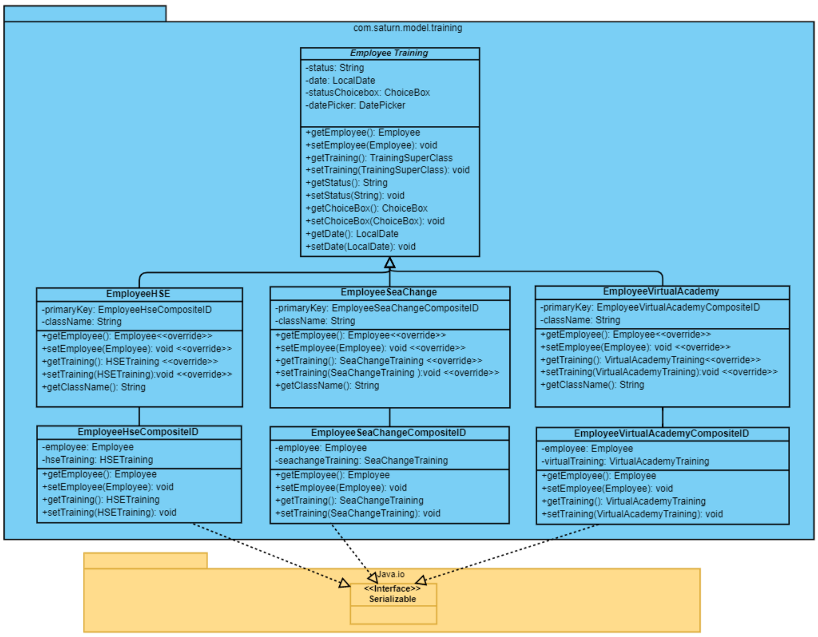

com.saturn.model.training
Employee Training
-status: String
-date: LocalDate
-statusChoicebox: ChoiceBox
-datePicker: DatePicker
+getEmployee(): Employee
+setEmployee(Employee): void
+getTraining(): TrainingSuperClass
+setTraining(TrainingSuperClass): void
+getStatus(): String
+setStatus(String): void
+getChoiceBox(): ChoiceBox
+setChoiceBox(ChoiceBox): void
+getDate(): LocalDate
+setDate(LocalDate): void
EmployeeHSE
-primaryKey: EmployeeHseCompositeID
-className: String
+getEmployee(): Employee<<override>>
+setEmployee(Employee): void <<override>>
+getTraining(): HSETraining <<override>>
+setTraining(HSETraining):void <<override>>
+getClassName(): String
EmployeeSeaChange
-primaryKey: EmployeeSeaChangeCompositeID
-className: String
+getEmployee(): Employee<<override>>
+setEmployee(Employee): void <<override>>
+getTraining(): SeaChangeTraining <<override>>
+setTraining(SeaChangeTraining):void <<override>>
+getClassName(): String
EmployeeVirtualAcademy
-primaryKey: EmployeeVirtualAcademyCompositeID
-className: String
+getEmployee(): Employee<<override>>
+setEmployee(Employee): void <<override>>
+getTraining(): VirtualAcademyTraining<<override>>
+setTraining(VirtualAcademyTraining):void <<override>>
+getClassName(): String
EmployeeHseCompositeID
-employee: Employee
-hseTraining: HSETraining
+getEmployee(): Employee
+setEmployee(Employee): void
+getTraining(): HSETraining
+setTraining(HSETraining): void
EmployeeSeaChangeCompositeID
-employee: Employee
-seachangeTraining: SeaChangeTraining
+getEmployee(): Employee
+setEmployee(Employee): void
+getTraining(): SeaChangeTraining
+setTraining(SeaChangeTraining): void
EmployeeVirtualAcademyCompositeID
-employee: Employee
-virtualTraining: VirtualAcademyTraining
+getEmployee(): Employee
+setEmployee(Employee): void
+getTraining(): VirtualAcademyTraining
+setTraining(VirtualAcademyTraining): void
Java.io
<<Interface>>
Serializable

Figure 64 - Training package

Employee/training manager

EmployeeTrainingController is responsible for filling up a table view with employee object and all training objects this employee is in it. This controller is also responsible for the update of the status of the training from ‘pending’ to ‘done’ and vice-versa.

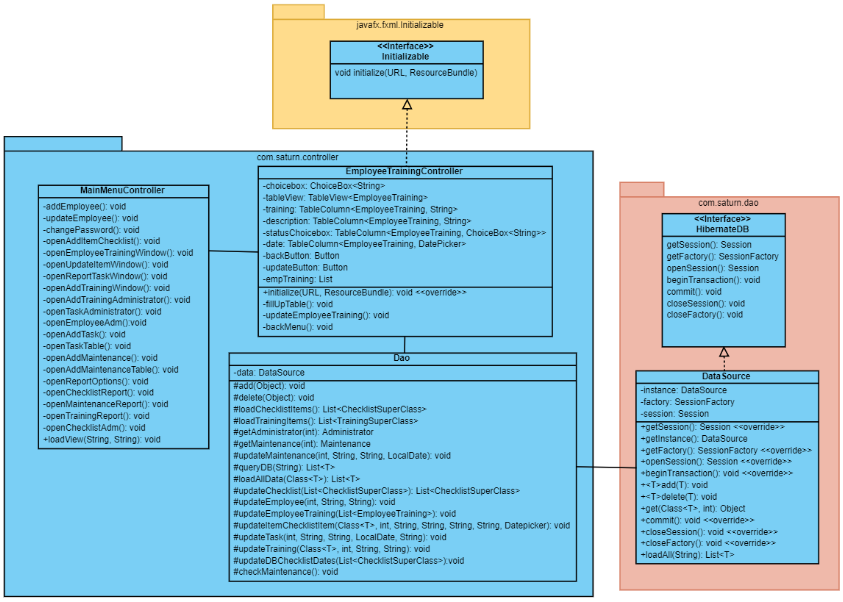

javafx.fxml.Initializable
<<Interface>>
Initializable
void initialize(URL, ResourceBundle)
com.saturn.controller
MainMenuController
-addEmployee(): void
-updateEmployee(): void
-changePassword(): void
-openAddItemChecklist(): void
-openEmployeeTrainingWindow(): void
-openUpdateItemWindow(): void
-openReportTaskWindow(): void
-openAddTrainingWindow(): void
-openAddTrainingAdministrator(): void
-openTaskAdministrator(): void
-openEmployeeAdm():void
-openAddTask(): void
-openTaskTable(): void
-openAddMaintenance(): void
-openAddMaintenanceTable(): void
-openReportOptions(): void
-openChecklistReport(): void
-openMaintenanceReport(): void
-openTrainingReport(): void
-openChecklistAdm(): void
+loadView(String, String): void
EmployeeTrainingController
-choicebox: ChoiceBox<String>
-tableView: TableView<EmployeeTraining>
-training: TableColumn<EmployeeTraining, String>
-description: TableColumn<EmployeeTraining, String>
-statusChoicebox: TableColumn<EmployeeTraining, ChoiceBox<String>>
-date: TableColumn<EmployeeTraining, DatePicker>
-backButton: Button
-updateButton: Button
-empTraining: List
+initialize(URL, ResourceBundle): void <<override>>
-fillUpTable(): void
-updateEmployeeTraining(): void
-backMenu(): void
Dao
-data: DataSource
#add(Object): void
#delete(Object): void
#loadChecklistItems(): List<ChecklistSuperClass>
#loadTrainingItems(): List<TrainingSuperClass>
#getAdministrator(int): Administrator
#getMaintenance(int): Maintenance
#updateMaintenance(int, String, String, LocalDate): void
#queryDB(String): List<T>
#loadAllData(Class<T>): List<T>
#updateChecklist(List<ChecklistSuperClass>): List<ChecklistSuperClass>
#updateEmployee(int, String, String): void
#updateEmployeeTraining(List<EmployeeTraining>): void
#updateItemChecklistItem(Class<T>, int, String, String, String, String, Datepicker): void
#updateTask(int, String, String, LocalDate, String): void
#updateTraining(Class<T>, int, String, String): void
#updateDBChecklistDates(List<ChecklistSuperClass>):void
#checkMaintenance(): void
com.saturn.dao
<<Interface>>
HibernateDB
getSession(): Session
getFactory(): SessionFactory
openSession(): Session
beginTransaction(): void
commit(): void
closeSession(): void
closeFactory(): void
DataSource
-instance: DataSource
-factory: SessionFactory
-session: Session
+getSession(): Session <<override>>
+getInstance(): DataSource
+getFactory(): SessionFactory <<override>>
+openSession(): Session <<override>>
+beginTransaction(): void <<override>>
+<T>add(T): void
+<T>delete(T): void
+get(Class<T>, int): Object
+commit(): void <<override>>
+closeSession(): void <<override>>
+closeFactory(): void <<override>>
+loadAll(String): List<T>

Figure 65 - Employee training controller class

Add maintenance item

AddMaintenanceController is responsible for the creation of maintenance object and save it in the database through the Dao class.

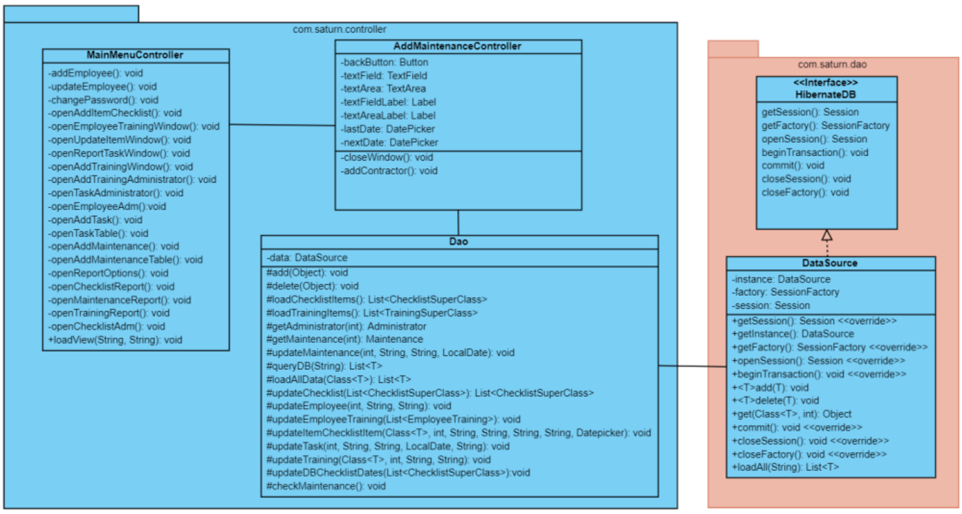

com.saturn.controller
MainMenuController
-addEmployee(): void
-updateEmployee(): void
-changePassword(): void
-openAddItemChecklist(): void
-openEmployeeTrainingWindow(): void
-openUpdateItemWindow(): void
-openReportTaskWindow(): void
-openAddTrainingWindow(): void
-openAddTrainingAdministrator(): void
-openTaskAdministrator(): void
-openEmployeeAdm():void
-openAddTask(): void
-openTaskTable(): void
-openAddMaintenance(): void
-openAddMaintenanceTable(): void
-openReportOptions(): void
-openChecklistReport(): void
-openMaintenanceReport(): void
-openTrainingReport(): void
-openChecklistAdm(): void
+loadView(String, String): void
AddMaintenanceController
-backButton: Button
-textField: TextField
-textArea: TextArea
-textFieldLabel: Label
-textAreaLabel: Label
-lastDate: DatePicker
-nextDate: DatePicker
-closeWindow(): void
-addContractor(): void
Dao
-data: DataSource
#add(Object): void
#delete(Object): void
#loadChecklistItems(): List<ChecklistSuperClass>
#loadTrainingItems(): List<TrainingSuperClass>
#getAdministrator(int): Administrator
#getMaintenance(int): Maintenance
#updateMaintenance(int, String, String, LocalDate): void
#queryDB(String): List<T>
#loadAllData(Class<T>): List<T>
#updateChecklist(List<ChecklistSuperClass>): List<ChecklistSuperClass>
#updateEmployee(int, String, String): void
#updateEmployeeTraining(List<EmployeeTraining>): void
#updateItemChecklistItem(Class<T>, int, String, String, String, String, Datepicker): void
#updateTask(int, String, String, LocalDate, String): void
#updateTraining(Class<T>, int, String, String): void
#updateDBChecklistDates(List<ChecklistSuperClass>):void
#checkMaintenance(): void
com.saturn.dao
<<Interface>>
HibernateDB
getSession(): Session
getFactory(): SessionFactory
openSession(): Session
beginTransaction(): void
commit(): void
closeSession(): void
closeFactory(): void
DataSource
-instance: DataSource
-factory: SessionFactory
-session: Session
+getSession(): Session <<override>>
+getInstance(): DataSource
+getFactory(): SessionFactory <<override>>
+openSession(): Session <<override>>
+beginTransaction(): void <<override>>
+<T>add(T): void
+<T>delete(T): void
+get(Class<T>, int): Object
+commit(): void <<override>>
+closeSession(): void <<override>>
+closeFactory(): void <<override>>
+loadAll(String): List<T>

Figure 66 - Add contractor controller class

Maintenance manager

MaintenanceTableController is responsible for filling up the table view with maintenance objects stored in the database and for the deletion of a maintenance object.

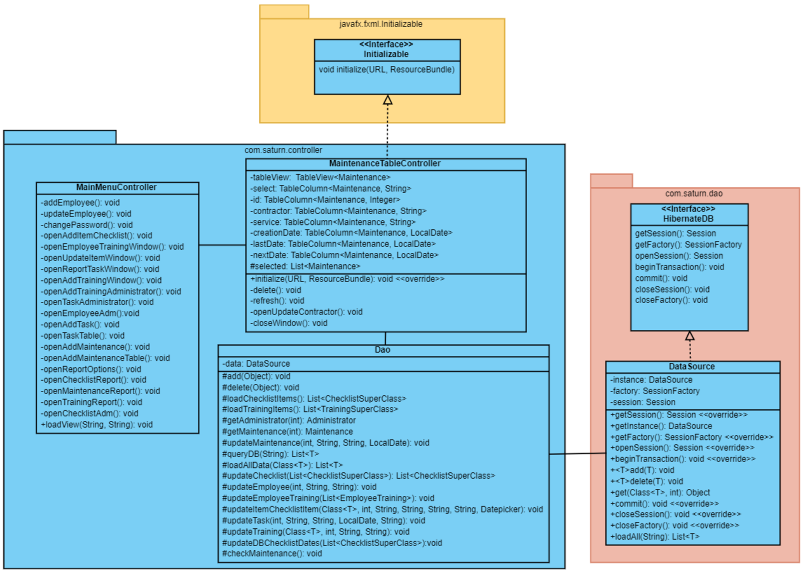

javafx.fxml.Initializable
<<Interface>>
Initializable
void initialize(URL, ResourceBundle)
com.saturn.controller
MaintenanceTableController
-tableView: TableView<Maintenance>
-select: TableColumn<Maintenance, String>
-id: TableColumn<Maintenance, Integer>
-contractor: TableColumn<Maintenance, String>
-service: TableColumn<Maintenance, String>
-creationDate: TableColumn<Maintenance, LocalDate>
-lastDate: TableColumn<Maintenance, LocalDate>
-nextDate: TableColumn<Maintenance, LocalDate>
#selected: List<Maintenance>
+initialize(URL, ResourceBundle): void <<override>>
-delete(): void
-refresh(): void
-openUpdateContractor(): void
-closeWindow(): void
MainMenuController
-addEmployee(): void
-updateEmployee(): void
-changePassword(): void
-openAddItemChecklist(): void
-openEmployeeTrainingWindow(): void
-openUpdateItemWindow(): void
-openReportTaskWindow(): void
-openAddTrainingWindow(): void
-openAddTrainingAdministrator(): void
-openTaskAdministrator(): void
-openEmployeeAdm():void
-openAddTask(): void
-openTaskTable(): void
-openAddMaintenance(): void
-openAddMaintenanceTable(): void
-openReportOptions(): void
-openChecklistReport(): void
-openMaintenanceReport(): void
-openTrainingReport(): void
-openChecklistAdm(): void
+loadView(String, String): void
Dao
-data: DataSource
#add(Object): void
#delete(Object): void
#loadChecklistItems(): List<ChecklistSuperClass>
#loadTrainingItems(): List<TrainingSuperClass>
#getAdministrator(int): Administrator
#getMaintenance(int): Maintenance
#updateMaintenance(int, String, String, LocalDate): void
#queryDB(String): List<T>
#loadAllData(Class<T>): List<T>
#updateChecklist(List<ChecklistSuperClass>): List<ChecklistSuperClass>
#updateEmployee(int, String, String): void
#updateEmployeeTraining(List<EmployeeTraining>): void
#updateItemChecklistItem(Class<T>, int, String, String, String, String, Datepicker): void
#updateTask(int, String, String, LocalDate, String): void
#updateTraining(Class<T>, int, String, String): void
#updateDBChecklistDates(List<ChecklistSuperClass>):void
#checkMaintenance(): void
com.saturn.dao
<<Interface>>
HibernateDB
getSession(): Session
getFactory(): SessionFactory
openSession(): Session
beginTransaction(): void
commit(): void
closeSession(): void
closeFactory(): void
DataSource
-instance: DataSource
-factory: SessionFactory
-session: Session
+getSession(): Session <<override>>
+getInstance(): DataSource
+getFactory(): SessionFactory <<override>>
+openSession(): Session <<override>>
+beginTransaction(): void <<override>>
+<T>add(T): void
+<T>delete(T): void
+get(Class<T>, int): Object
+commit(): void <<override>>
+closeSession(): void <<override>>
+closeFactory(): void <<override>>
+loadAll(String): List<T>

Figure 67 - Maintenance manager controller

Update maintenance item

UpdateContractorController is responsible for the updates of a maintenance object. It interacts with the database through the Dao class.

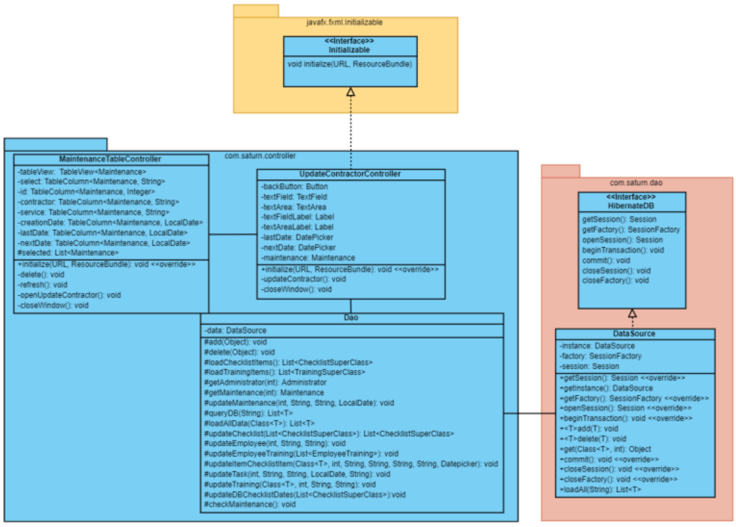

javafx.fxml.Initializable
<<Interface>>
Initializable
void initialize(URL, ResourceBundle)
com.saturn.controller
MaintenanceTableController
-tableView: TableView<Maintenance>
-select: TableColumn<Maintenance, String>
-id: TableColumn<Maintenance, Integer>
-contractor: TableColumn<Maintenance, String>
-service: TableColumn<Maintenance, String>
-creationDate: TableColumn<Maintenance, LocalDate>
-lastDate: TableColumn<Maintenance, LocalDate>
-nextDate: TableColumn<Maintenance, LocalDate>
#selected: List<Maintenance>
+initialize(URL, ResourceBundle): void <<override>>
-delete(): void
-refresh(): void
-openUpdateContractor(): void
-closeWindow(): void
UpdateContractorController
-backButton: Button
-textField: TextField
-textArea: TextArea
-textFieldLabel: Label
-textAreaLabel: Label
-lastDate: DatePicker
-nextDate: DatePicker
-maintenance: Maintenance
+initialize(URL, ResourceBundle): void <<override>>
-updateContractor(): void
-closeWindow(): void
Dao
-data: DataSource
#add(Object): void
#delete(Object): void
#loadChecklistItems(): List<ChecklistSuperClass>
#loadTrainingItems(): List<TrainingSuperClass>
#getAdministrator(int): Administrator
#getMaintenance(int): Maintenance
#updateMaintenance(int, String, String, LocalDate): void
#queryDB(String): List<T>
#loadAllData(Class<T>): List<T>
#updateChecklist(List<ChecklistSuperClass>): List<ChecklistSuperClass>
#updateEmployee(int, String, String): void
#updateEmployeeTraining(List<EmployeeTraining>): void
#updateItemChecklistItem(Class<T>, int, String, String, String, String, Datepicker): void
#updateTask(int, String, String, LocalDate, String): void
#updateTraining(Class<T>, int, String, String): void
#updateDBChecklistDates(List<ChecklistSuperClass>):void
#checkMaintenance(): void
com.saturn.dao
<<Interface>>
HibernateDB
getSession(): Session
getFactory(): SessionFactory
openSession(): Session
beginTransaction(): void
commit(): void
closeSession(): void
closeFactory(): void
DataSource
-instance: DataSource
-factory: SessionFactory
-session: Session
+getSession(): Session <<override>>
+getInstance(): DataSource
+getFactory(): SessionFactory <<override>>
+openSession(): Session <<override>>
+beginTransaction(): void <<override>>
+<T>add(T): void
+<T>delete(T): void
+get(Class<T>, int): Object
+commit(): void <<override>>
+closeSession(): void <<override>>
+closeFactory(): void <<override>>
+loadAll(String): List<T>

Figure 68 - Update contractor controller class

Report/Database

Report package is responsible for the creation of pdf report and dao package stores the singleton pattern database class.

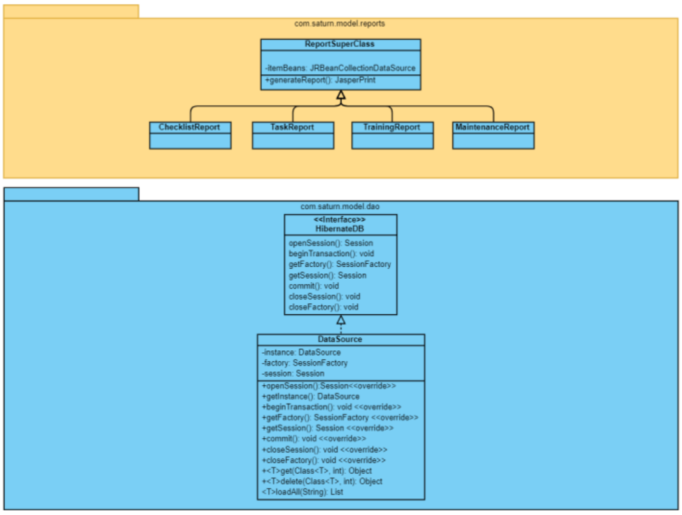

com.saturn.model.reports
ReportSuperClass
-itemBeans: JRBeanCollectionDataSource
+generateReport(): JasperPrint
ChecklistReport
TaskReport
TrainingReport
MaintenanceReport
com.saturn.model.dao
<<Interface>>
HibernateDB
openSession(): Session
beginTransaction(): void
getFactory(): SessionFactory
getSession(): Session
commit(): void
closeSession(): void
closeFactory(): void
DataSource
-instance: DataSource
-factory: SessionFactory
-session: Session
+openSession():Session<<override>>
+getInstance(): DataSource
+beginTransaction(): void <<override>>
+getFactory(): SessionFactory <<override>>
+getSession(): Session <<override>>
+commit(): void <<override>>
+closeSession(): void <<override>>
+closeFactory(): void <<override>>
+<T>get(Class<T>, int): Object
+<T>delete(Class<T>, int): Object
<T>loadAll(String): List

Figure 69 - Report package/ Database package

Report option

ReportOptionController processes the user input about which report it wants to generate redirects to the responsible class.

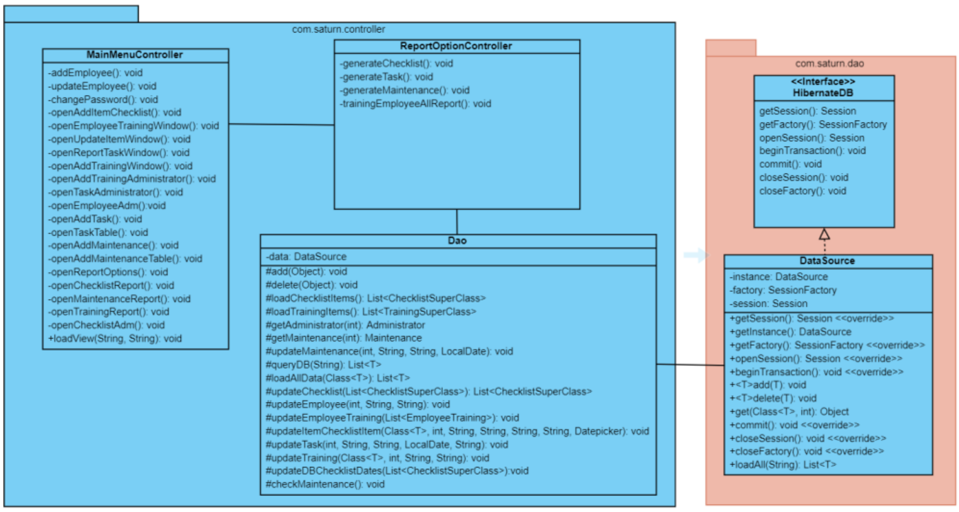

com.saturn.controller
MainMenuController
-addEmployee(): void
-updateEmployee(): void
-changePassword(): void
-openAddItemChecklist(): void
-openEmployeeTrainingWindow(): void
-openUpdateItemWindow(): void
-openReportTaskWindow(): void
-openAddTrainingWindow(): void
-openAddTrainingAdministrator(): void
-openTaskAdministrator(): void
-openEmployeeAdm():void
-openAddTask(): void
-openTaskTable(): void
-openAddMaintenance(): void
-openAddMaintenanceTable(): void
-openReportOptions(): void
-openChecklistReport(): void
-openMaintenanceReport(): void
-openTrainingReport(): void
-openChecklistAdm(): void
+loadView(String, String): void
ReportOptionController
-generateChecklist(): void
-generateTask(): void
-generateMaintenance(): void
-trainingEmployeeAllReport(): void
Dao
-data: DataSource
#add(Object): void
#delete(Object): void
#loadChecklistItems(): List<ChecklistSuperClass>
#loadTrainingItems(): List<TrainingSuperClass>
#getAdministrator(int): Administrator
#getMaintenance(int): Maintenance
#updateMaintenance(int, String, String, LocalDate): void
#queryDB(String): List<T>
#loadAllData(Class<T>): List<T>
#updateChecklist(List<ChecklistSuperClass>): List<ChecklistSuperClass>
#updateEmployee(int, String, String): void
#updateEmployeeTraining(List<EmployeeTraining>): void
#updateItemChecklistItem(Class<T>, int, String, String, String, String, Datepicker): void
#updateTask(int, String, String, LocalDate, String): void
#updateTraining(Class<T>, int, String, String): void
#updateDBChecklistDates(List<ChecklistSuperClass>):void
#checkMaintenance(): void
com.saturn.dao
<<Interface>>
HibernateDB
getSession(): Session
getFactory(): SessionFactory
openSession(): Session
beginTransaction(): void
commit(): void
closeSession(): void
closeFactory(): void
DataSource
-instance: DataSource
-factory: SessionFactory
-session: Session
+getSession(): Session <<override>>
+getInstance(): DataSource
+getFactory(): SessionFactory <<override>>
+openSession(): Session <<override>>
+beginTransaction(): void <<override>>
+<T>add(T): void
+<T>delete(T): void
+get(Class<T>, int): Object
+commit(): void <<override>>
+closeSession(): void <<override>>
+closeFactory(): void <<override>>
+loadAll(String): List<T>

Figure 70 - Report shortcut controller class

Maintenance report

MaintenanceReportController is responsible for processing the user input and generate a custom report according to the needs of the user.

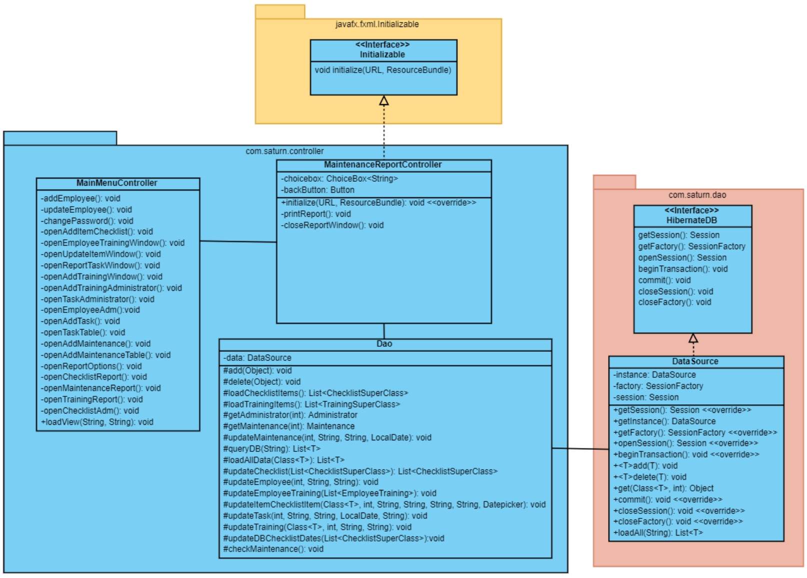

javafx.fxml.Initializable
<<Interface>>
Initializable
void initialize(URL, ResourceBundle)
com.saturn.controller
MainMenuController
-addEmployee(): void
-updateEmployee(): void
-changePassword(): void
-openAddItemChecklist(): void
-openEmployeeTrainingWindow(): void
-openUpdateItemWindow(): void
-openReportTaskWindow(): void
-openAddTrainingWindow(): void
-openAddTrainingAdministrator(): void
-openTaskAdministrator(): void
-openEmployeeAdm():void
-openAddTask(): void
-openTaskTable(): void
-openAddMaintenance(): void
-openAddMaintenanceTable(): void
-openReportOptions(): void
-openChecklistReport(): void
-openMaintenanceReport(): void
-openTrainingReport(): void
-openChecklistAdm(): void
+loadView(String, String): void
MaintenanceReportController
-choicebox: ChoiceBox<String>
-backButton: Button
+initialize(URL, ResourceBundle): void <<override>>
-printReport(): void
-closeReportWindow(): void
Dao
-data: DataSource
#add(Object): void
#delete(Object): void
#loadChecklistItems(): List<ChecklistSuperClass>
#loadTrainingItems(): List<TrainingSuperClass>
#getAdministrator(int): Administrator
#getMaintenance(int): Maintenance
#updateMaintenance(int, String, String, LocalDate): void
#queryDB(String): List<T>
#loadAllData(Class<T>): List<T>
#updateChecklist(List<ChecklistSuperClass>): List<ChecklistSuperClass>
#updateEmployee(int, String, String): void
#updateEmployeeTraining(List<EmployeeTraining>): void
#updateItemChecklistItem(Class<T>, int, String, String, String, String, Datepicker): void
#updateTask(int, String, String, LocalDate, String): void
#updateTraining(Class<T>, int, String, String): void
#updateDBChecklistDates(List<ChecklistSuperClass>):void
#checkMaintenance(): void
com.saturn.dao
<<Interface>>
HibernateDB
getSession(): Session
getFactory(): SessionFactory
openSession(): Session
beginTransaction(): void
commit(): void
closeSession(): void
closeFactory(): void
DataSource
-instance: DataSource
-factory: SessionFactory
-session: Session
+getSession(): Session <<override>>
+getInstance(): DataSource
+getFactory(): SessionFactory <<override>>
+openSession(): Session <<override>>
+beginTransaction(): void <<override>>
+<T>add(T): void
+<T>delete(T): void
+get(Class<T>, int): Object
+commit(): void <<override>>
+closeSession(): void <<override>>
+closeFactory(): void <<override>>
+loadAll(String): List<T>

Figure 71 - Maintenance report controller class

Checklist report

ChecklistReportController is responsible for processing the user input and generate a custom report according to the needs of the user.

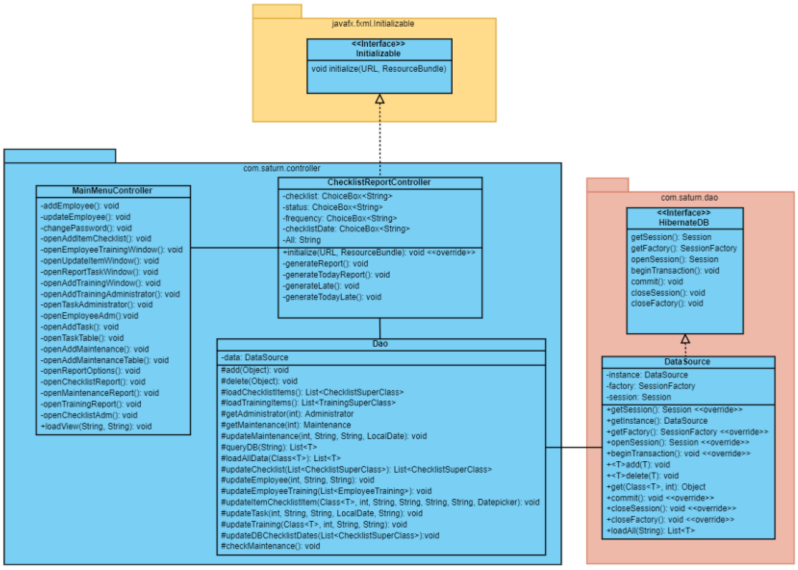

javafx.fxml.Initializable
<<Interface>>
Initializable
void initialize(URL, ResourceBundle)
com.saturn.controller
MainMenuController
-addEmployee(): void
-updateEmployee(): void
-changePassword(): void
-openAddItemChecklist(): void
-openEmployeeTrainingWindow(): void
-openUpdateItemWindow(): void
-openReportTaskWindow(): void
-openAddTrainingWindow(): void
-openAddTrainingAdministrator(): void
-openTaskAdministrator(): void
-openEmployeeAdm():void
-openAddTask(): void
-openTaskTable(): void
-openAddMaintenance(): void
-openAddMaintenanceTable(): void
-openReportOptions(): void
-openChecklistReport(): void
-openMaintenanceReport(): void
-openTrainingReport(): void
-openChecklistAdm(): void
+loadView(String, String): void
ChecklistReportController
-checklist: ChoiceBox<String>
-status: ChoiceBox<String>
-frequency: ChoiceBox<String>
-checklistDate: ChoiceBox<String>
-All: String
+initialize(URL, ResourceBundle): void <<override>>
-generateReport(): void
-generateTodayReport(): void
-generateLate(): void
-generateTodayLate(): void
Dao
-data: DataSource
#add(Object): void
#delete(Object): void
#loadChecklistItems(): List<ChecklistSuperClass>
#loadTrainingItems(): List<TrainingSuperClass>
#getAdministrator(int): Administrator
#getMaintenance(int): Maintenance
#updateMaintenance(int, String, String, LocalDate): void
#queryDB(String): List<T>
#loadAllData(Class<T>): List<T>
#updateChecklist(List<ChecklistSuperClass>): List<ChecklistSuperClass>
#updateEmployee(int, String, String): void
#updateEmployeeTraining(List<EmployeeTraining>): void
#updateItemChecklistItem(Class<T>, int, String, String, String, String, Datepicker): void
#updateTask(int, String, String, LocalDate, String): void
#updateTraining(Class<T>, int, String, String): void
#updateDBChecklistDates(List<ChecklistSuperClass>):void
#checkMaintenance(): void
com.saturn.dao
<<Interface>>
HibernateDB
getSession(): Session
getFactory(): SessionFactory
openSession(): Session
beginTransaction(): void
commit(): void
closeSession(): void
closeFactory(): void
DataSource
-instance: DataSource
-factory: SessionFactory
-session: Session
+getSession(): Session <<override>>
+getInstance(): DataSource
+getFactory(): SessionFactory <<override>>
+openSession(): Session <<override>>
+beginTransaction(): void <<override>>
+<T>add(T): void
+<T>delete(T): void
+get(Class<T>, int): Object
+commit(): void <<override>>
+closeSession(): void <<override>>
+closeFactory(): void <<override>>
+loadAll(String): List<T>

Figure 72 - Checklist report controller

Task report

TaskReportController is responsible for processing the user input and generate a custom report according to the needs of the user.

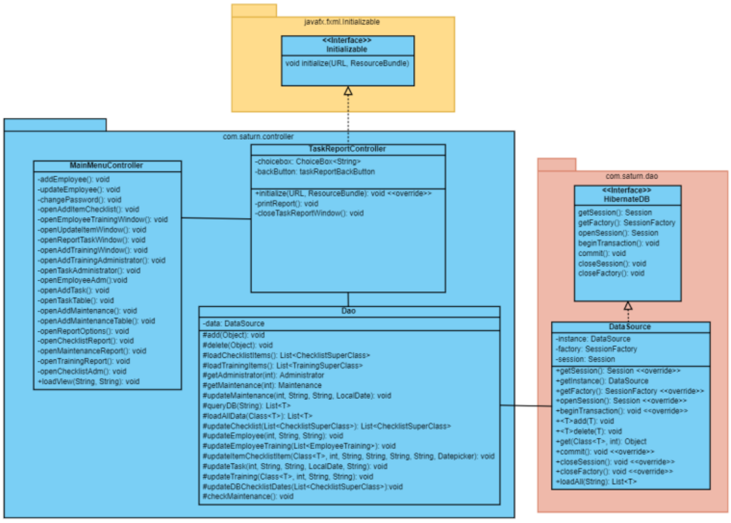
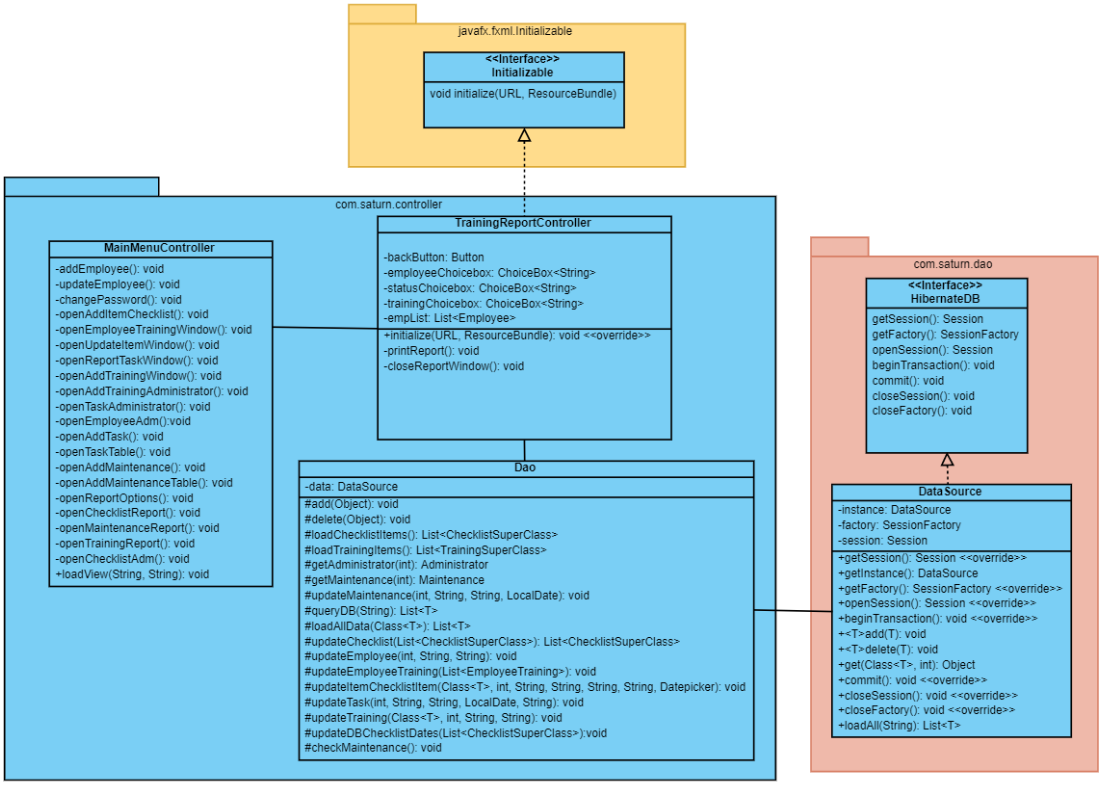

javafx.fxml.Initializable
<<Interface>>
Initializable
void initialize(URL, ResourceBundle)
com.saturn.controller
MainMenuController
-addEmployee(): void
-updateEmployee(): void
-changePassword(): void
-openAddItemChecklist(): void
-openEmployeeTrainingWindow(): void
-openUpdateItemWindow(): void
-openReportTaskWindow(): void
-openAddTrainingWindow(): void
-openAddTrainingAdministrator(): void
-openTaskAdministrator(): void
-openEmployeeAdm():void
-openAddTask(): void
-openTaskTable(): void
-openAddMaintenance(): void
-openAddMaintenanceTable(): void
-openReportOptions(): void
-openChecklistReport(): void
-openMaintenanceReport(): void
-openTrainingReport(): void
-openChecklistAdm(): void
+loadView(String, String): void
TaskReportController
-choicebox: ChoiceBox<String>
-backButton: taskReportBackButton
+initialize(URL, ResourceBundle): void <<override>>
-printReport(): void
-closeTaskReportWindow(): void
Dao
-data: DataSource
#add(Object): void
#delete(Object): void
#loadChecklistItems(): List<ChecklistSuperClass>
#loadTrainingItems(): List<TrainingSuperClass>
#getAdministrator(int): Administrator
#getMaintenance(int): Maintenance
#updateMaintenance(int, String, String, LocalDate): void
#queryDB(String): List<T>
#loadAllData(Class<T>): List<T>
#updateChecklist(List<ChecklistSuperClass>): List<ChecklistSuperClass>
#updateEmployee(int, String, String): void
#updateEmployeeTraining(List<EmployeeTraining>): void
#updateItemChecklistItem(Class<T>, int, String, String, String, String, Datepicker): void
#updateTask(int, String, String, LocalDate, String): void
#updateTraining(Class<T>, int, String, String): void
#updateDBChecklistDates(List<ChecklistSuperClass>):void
#checkMaintenance(): void
com.saturn.dao
<<Interface>>
HibernateDB
getSession(): Session
getFactory(): SessionFactory
openSession(): Session
beginTransaction(): void
commit(): void
closeSession(): void
closeFactory(): void
DataSource
-instance: DataSource
-factory: SessionFactory
-session: Session
+getSession(): Session <<override>>
+getInstance(): DataSource
+getFactory(): SessionFactory <<override>>
+openSession(): Session <<override>>
+beginTransaction(): void <<override>>
+<T>add(T): void
+<T>delete(T): void
+get(Class<T>, int): Object
+commit(): void <<override>>
+closeSession(): void <<override>>
+closeFactory(): void <<override>>
+loadAll(String): List<T>

Figure 73 - Task report controller

Training report

TrainingReportController is responsible for processing the user input and generate a custom report according to the needs of the user.


javafx.fxml.Initializable
<<Interface>>
Initializable
void initialize(URL, ResourceBundle)
com.saturn.controller
MainMenuController
-addEmployee(): void
-updateEmployee(): void
-changePassword(): void
-openAddItemChecklist(): void
-openEmployeeTrainingWindow(): void
-openUpdateItemWindow(): void
-openReportTaskWindow(): void
-openAddTrainingWindow(): void
-openAddTrainingAdministrator(): void
-openTaskAdministrator(): void
-openEmployeeAdm():void
-openAddTask(): void
-openTaskTable(): void
-openAddMaintenance(): void
-openAddMaintenanceTable(): void
-openReportOptions(): void
-openChecklistReport(): void
-openMaintenanceReport(): void
-openTrainingReport(): void
-openChecklistAdm(): void
+loadView(String, String): void
TrainingReportController
-backButton: Button
-employeeChoicebox: ChoiceBox<String>
-statusChoicebox: ChoiceBox<String>
-trainingChoicebox: ChoiceBox<String>
-empList: List<Employee>
+initialize(URL, ResourceBundle): void <<override>>
-printReport(): void
-closeReportWindow(): void
Dao
-data: DataSource
#add(Object): void
#delete(Object): void
#loadChecklistItems(): List<ChecklistSuperClass>
#loadTrainingItems(): List<TrainingSuperClass>
#getAdministrator(int): Administrator
#getMaintenance(int): Maintenance
#updateMaintenance(int, String, String, LocalDate): void
#queryDB(String): List<T>
#loadAllData(Class<T>): List<T>
#updateChecklist(List<ChecklistSuperClass>): List<ChecklistSuperClass>
#updateEmployee(int, String, String): void
#updateEmployeeTraining(List<EmployeeTraining>): void
#updateItemChecklistItem(Class<T>, int, String, String, String, String, Datepicker): void
#updateTask(int, String, String, LocalDate, String): void
#updateTraining(Class<T>, int, String, String): void
#updateDBChecklistDates(List<ChecklistSuperClass>):void
#checkMaintenance(): void
com.saturn.dao
<<Interface>>
HibernateDB
getSession(): Session
getFactory(): SessionFactory
openSession(): Session
beginTransaction(): void
commit(): void
closeSession(): void
closeFactory(): void
DataSource
-instance: DataSource
-factory: SessionFactory
-session: Session
+getSession(): Session <<override>>
+getInstance(): DataSource
+getFactory(): SessionFactory <<override>>
+openSession(): Session <<override>>
+beginTransaction(): void <<override>>
+<T>add(T): void
+<T>delete(T): void
+get(Class<T>, int): Object
+commit(): void <<override>>
+closeSession(): void <<override>>
+closeFactory(): void <<override>>
+loadAll(String): List<T>

Overall class diagram

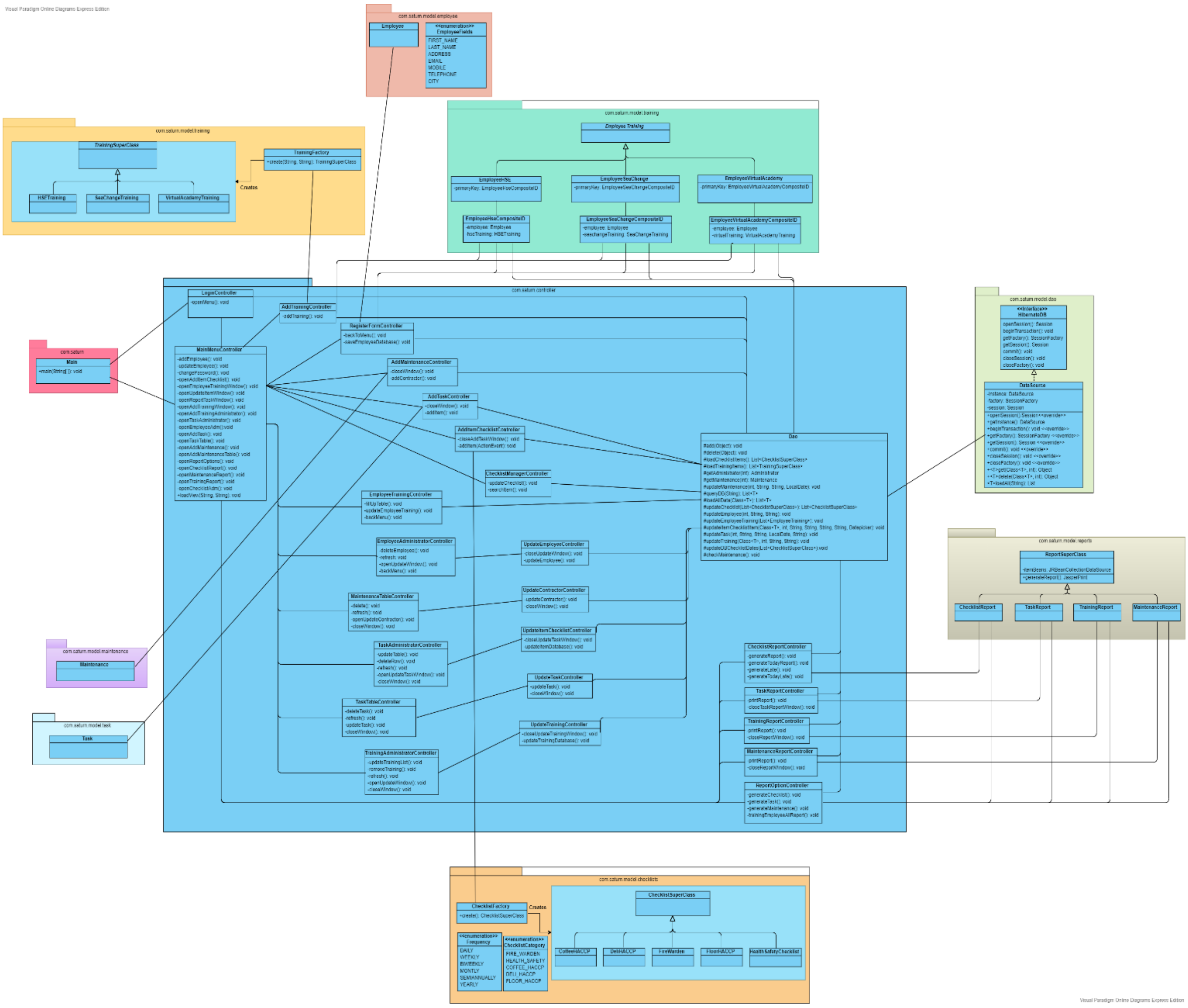

Visual Paradigm Online Diagrams Express Edition
com.saturn.model.employee
Employee
<<enumeration>>
EmployeeFields
FIRST_NAME
LAST_NAME
ADDRESS
EMAIL
MOBILE
TELEPHONE
CITY
com.saturn.model.training
TrainingSuperClass
HSETraining
SeaChangeTraining
VirtualAcademyTraining
TrainingFactory
+create(String, String): TrainingSuperClass
Creates
com.saturn.model.training
Employee Training
EmployeeHSE
-primaryKey: EmployeeHseCompositeID
EmployeeSeaChange
-primaryKey: EmployeeSeaChangeCompositeID
EmployeeVirtualAcademy
-primaryKey: EmployeeVirtualAcademyCompositeID
EmployeeHseCompositeID
-employee: Employee
-hseTraining: HSETraining
EmployeeSeaChangeCompositeID
-employee: Employee
-seachangeTraining: SeaChangeTraining
EmployeeVirtualAcademyCompositeID
-employee: Employee
-virtualTraining: VirtualAcademyTraining
com.saturn
Main
+main(String[ ]): void
com.saturn.controller
LoginController
-openMenu(): void
AddTrainingController
-addTraining(): void
RegisterFormController
-backToMenu(): void
-saveEmployeeDatabase(): void
MainMenuController
-addEmployee(): void
-updateEmployee(): void
-changePassword(): void
-openAddItemChecklist(): void
-openEmployeeTrainingWindow(): void
-openUpdateItemWindow(): void
-openReportTaskWindow(): void
-openAddTrainingWindow(): void
-openAddTrainingAdministrator(): void
-openTaskAdministrator(): void
-openEmployeeAdm():void
-openAddTask(): void
-openTaskTable(): void
-openAddMaintenance(): void
-openAddMaintenanceTable(): void
-openReportOptions(): void
-openChecklistReport(): void
-openMaintenanceReport(): void
-openTrainingReport(): void
-openChecklistAdm(): void
+loadView(String, String): void
AddMaintenanceController
-closeWindow(): void
-addContractor(): void
AddTaskController
-closeWindow(): void
-addItem(): void
AddItemChecklistController
-closeAddTaskWindow(): void
-addItem(ActionEvent): void
ChecklistManagerController
-updateChecklist(): void
-searchItem(): void
EmployeeTrainingController
-fillUpTable(): void
-updateEmployeeTraining(): void
-backMenu(): void
EmployeeAdministratorController
-deleteEmployee(): void
-refresh: void
-openUpdateWindow(): void
-backMenu(): void
UpdateEmployeeController
-closeUpdateWindow(): void
-updateEmployee(): void
MaintenanceTableController
-delete(): void
-refresh(): void
-openUpdateContractor(): void
-closeWindow(): void
UpdateContractorController
-updateContractor(): void
-closeWindow(): void
UpdateItemChecklistController
-closeUpdateTaskWindow(): void
-updateItemDatabase(): void
TaskAdministratorController
-updateTable(): void
-deleteRow(): void
-refresh(): void
-openUpdateTaskWindow(): void
-closeWindow(): void
UpdateTaskController
-updateTask(): void
-closeWindow(): void
TaskTableController
-deleteTask(): void
-refresh(): void
-updateTask(): void
-closeWindow(): void
UpdateTrainingController
-closeUpdateTrainingWindow(): void
-updateTrainingDatabase(): void
TrainingAdministratorController
-updateTrainingList(): void
-removeTraining(): void
-refresh(): void
-openUpdateWindow(): void
-closeWindow(): void
Dao
#add(Object): void
#delete(Object): void
#loadChecklistItems(): List<ChecklistSuperClass>
#loadTrainingItems(): List<TrainingSuperClass>
#getAdministrator(int): Administrator
#getMaintenance(int): Maintenance
#updateMaintenance(int, String, String, LocalDate): void
#queryDB(String): List<T>
#loadAllData(Class<T>): List<T>
#updateChecklist(List<ChecklistSuperClass>): List<ChecklistSuperClass>
#updateEmployee(int, String, String): void
#updateEmployeeTraining(List<EmployeeTraining>): void
#updateItemChecklistItem(Class<T>, int, String, String, String, String, Datepicker): void
#updateTask(int, String, String, LocalDate, String): void
#updateTraining(Class<T>, int, String, String): void
#updateDBChecklistDates(List<ChecklistSuperClass>):void
#checkMaintenance(): void
ChecklistReportController
-generateReport(): void
-generateTodayReport(): void
-generateLate(): void
-generateTodayLate(): void
TaskReportController
-printReport(): void
-closeTaskReportWindow(): void
TrainingReportController
-printReport(): void
-closeReportWindow(): void
MaintenanceReportController
-printReport(): void
-closeReportWindow(): void
ReportOptionController
-generateChecklist(): void
-generateTask(): void
-generateMaintenence(): void
-trainingEmployeeAllReport(): void
com.saturn.model.dao
<<Interface>>
HibernateDB
openSession(): Session
beginTransaction(): void
getFactory(): SessionFactory
getSession(): Session
commit(): void
closeSession(): void
closeFactory(): void
DataSource
-instance: DataSource
-factory: SessionFactory
-session: Session
+openSession():Session<<override>>
+getInstance(): DataSource
+beginTransaction(): void <<override>>
+getFactory(): SessionFactory <<override>>
+getSession(): Session <<override>>
+commit(): void <<override>>
+closeSession(): void <<override>>
+closeFactory(): void <<override>>
+<T>get(Class<T>, int): Object
+<T>delete(Class<T>, int): Object
+T>loadAll(String): List
com.saturn.model.reports
ReportSuperClass
-itemBeans: JRBeanCollectionDataSource
+generateReport(): JasperPrint
ChecklistReport
TaskReport
TrainingReport
MaintenanceReport
com.saturn.model.maintenance
Maintenance
com.saturn.model.task
Task
com.saturn.model.checklists
ChecklistFactory
+create(): ChecklistSuperClass
Creates
<<enumeration>>
Frequency
DAILY
WEEKLY
BIWEEKLY
MONTLY
SEMIANNUALLY
YEARLY
<<enumeration>>
ChecklistCategory
FIRE_WARDEN
HEALTH_SAFETY
COFFEE_HACCP
DELI_HACCP
FLOOR_HACCP
ChecklistSuperClass
CoffeeHACCP
DeliHACCP
FireWarden
FloorHACCP
HealthSafetyChecklist
Visual Paradigm Online Diagrams Express Edition

Figure 74 - Overall class diagram

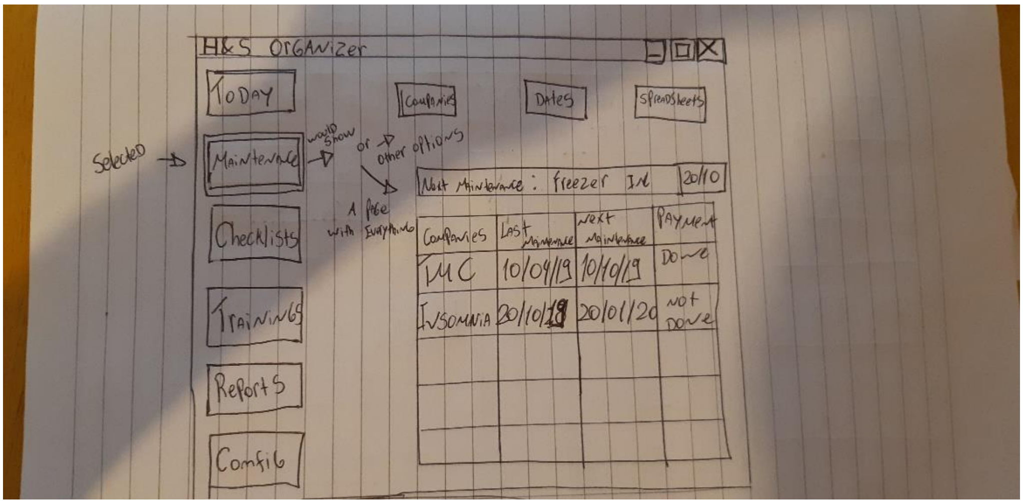
3.0 GUI Design and functionalities

The application relies heavily on a user-friendly GUI making the design of it an important part of the development of the project. The user interface needs to be done in a way where the user can easily navigate and understand what he is doing and what he wants and needs to do. We as the developers know what it is doing and what it is supposed to do but for the user it might not be as obvious as it is to us.

We need to follow best practices on the way we build the interface. Keeping it simple and clean is a good way to not make the user feel overwhelmed and get confused on where he needs to click on. Creating a consistent interface is a great way to make the user comfortable when using it, especially on creating patterns to help make it obvious. Using colours and light to catch the user's attention to certain places helps on the navigation. Also communicating to the user if there are any errors, changes or problems. (Usability.gov, 2020).

We started our project mainly on the GUI design as we needed to know not only what we needed but how we needed it to be done, as said before the application relies heavily on how the user will interact with it. We did a few drawing on how it was supposed to look like so we could have an idea of where to start.

Drawing of how the User Interface is supposed to look like:



Companies	Last Maintenance	Next Maintenance	Payment
IMC	10/04/19	10/10/19	Done
Insomnia	20/10/18	20/01/20	not Done

Figure 77 User interface main menu

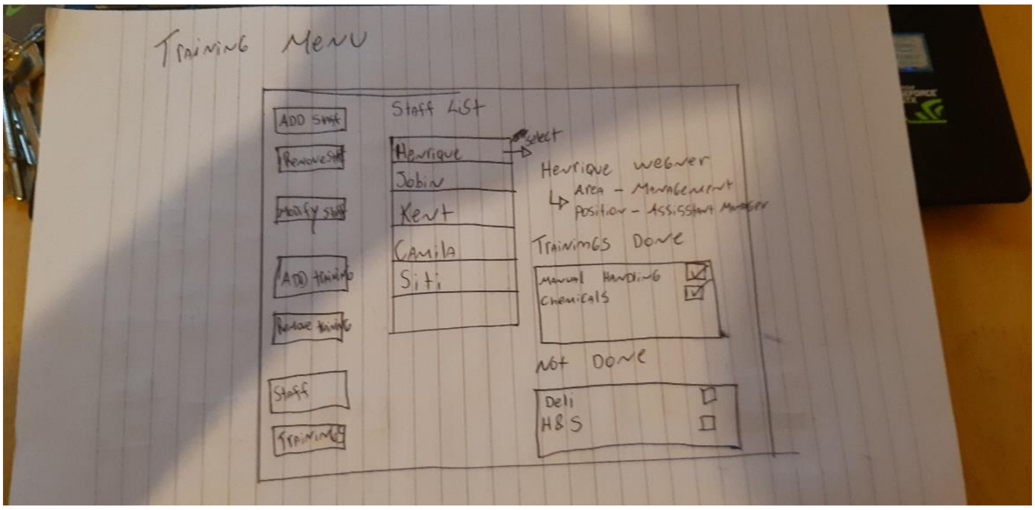


Figure 76 - User interface training menu

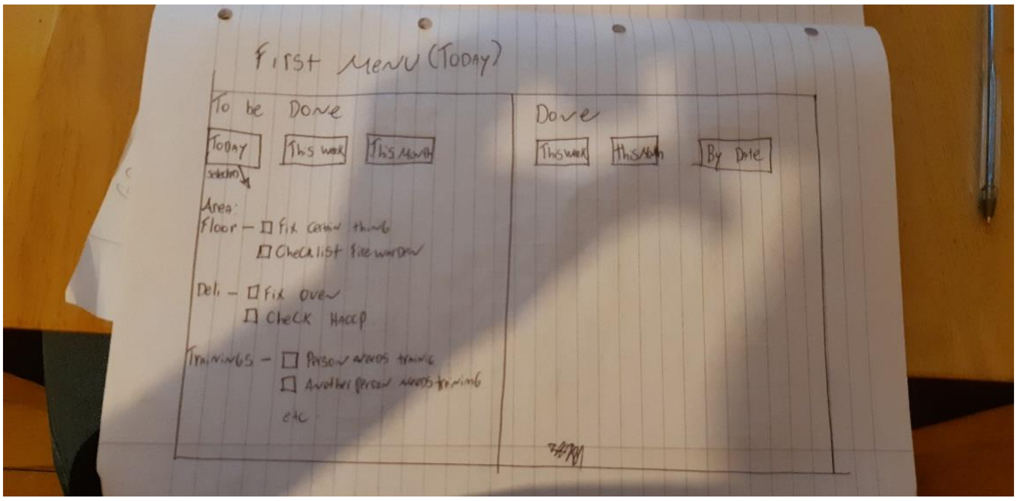


Figure 75 - Report menu

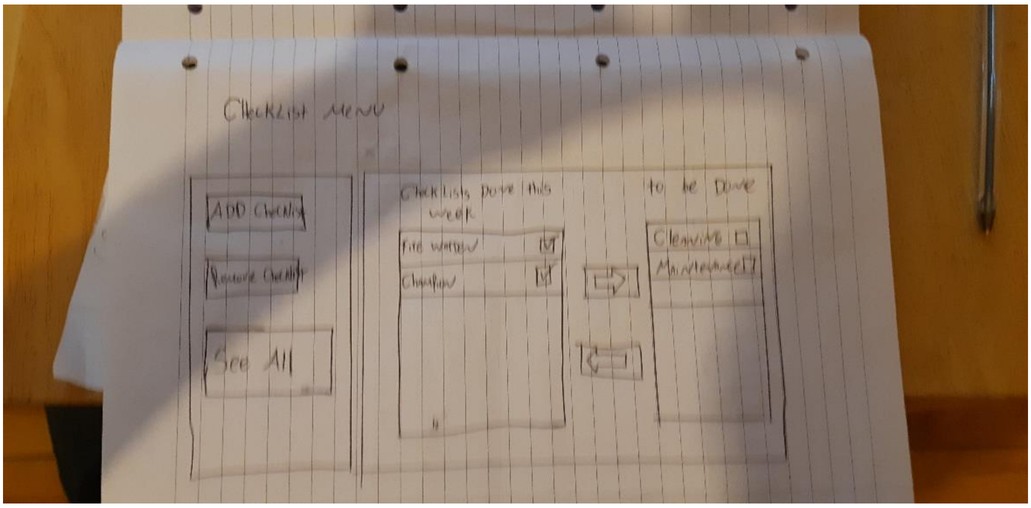


Figure 78 - Checklist menu

We started working with the skeleton of the program and its functionalities, we tried to implement the basic functionalities first like adding staff to the program, training types, checklists and made some generic names so we could test it.

Employee Registration Form:

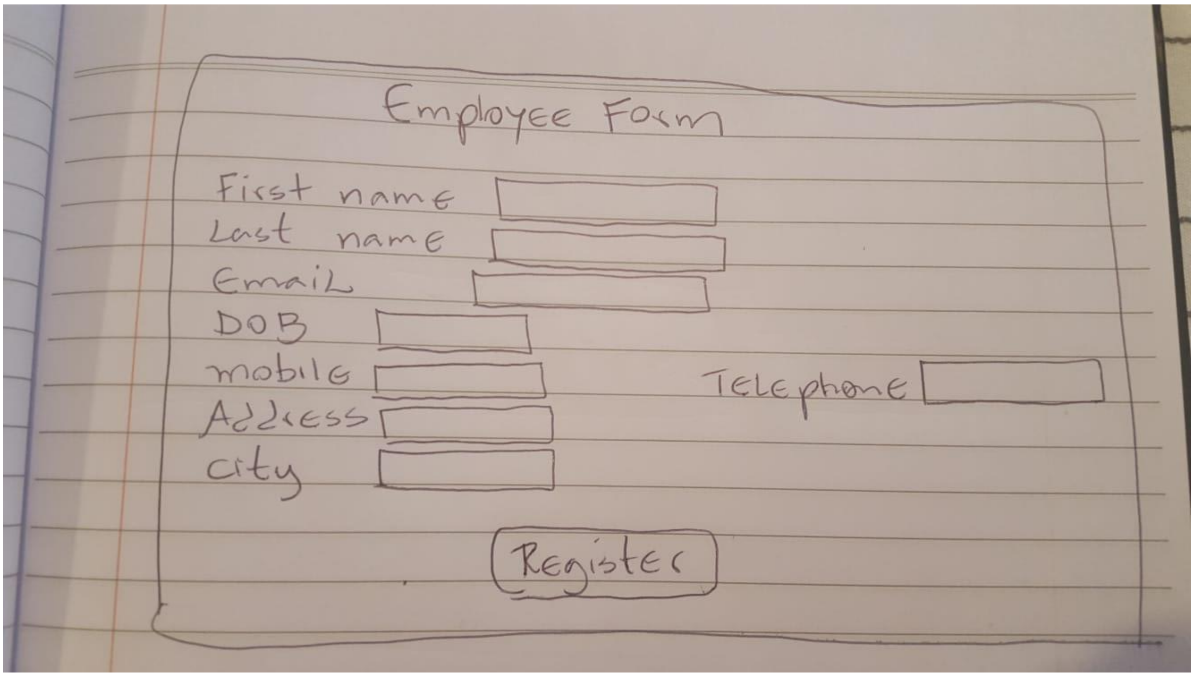


Figure 79 - Employee registration form

Administrator Tab:

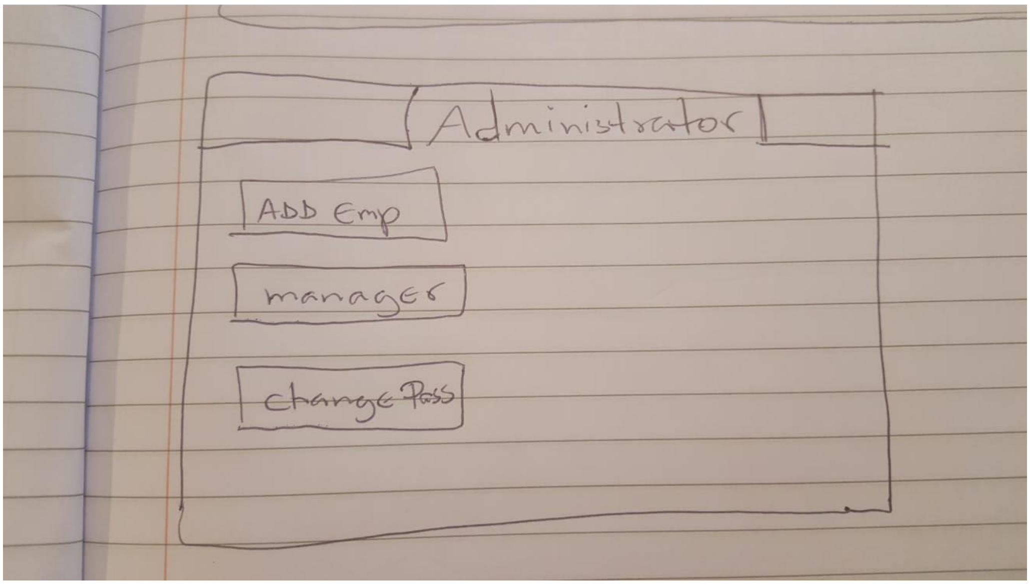


Figure 80 - Administrator tab

Add a new Checklist Form:

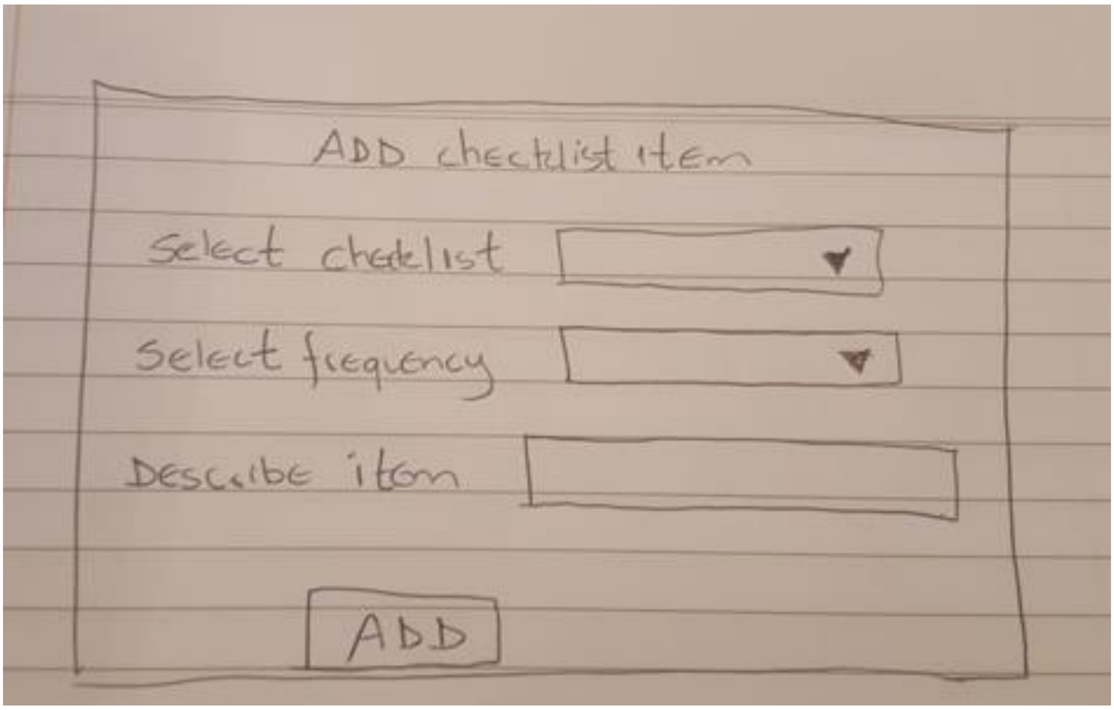


Figure 81 - Add checklist form

Add a new training form:

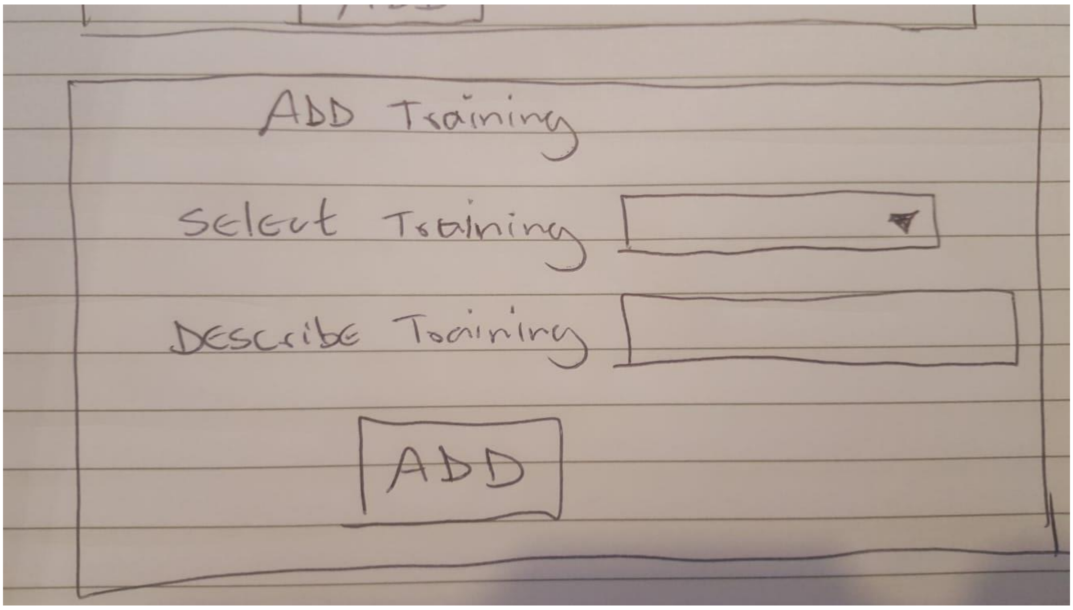


Figure 82 - Add training form

We had to change a few functionalities, for example the checklist became an only a table where you can mark it as done or not done and it will reset depending on its frequency. A new contractor tab was added, and a managing window was created where you can check every training, tasks, staff and contractors that you have, update them or delete them. Trainings were added to the employees, so you can add trainings to them.

A menu was created so the user can have shortcuts to the managing tables of all functionalities, icons were also added as they make it easier to identify the buttons and what they do. The reports were not working on our first basic functionalities, now for the tasks we can create a report of what is pending, done or all of them. You can also modify if tasks were done and what the action taken was, this all show up on the report.

System Menu:

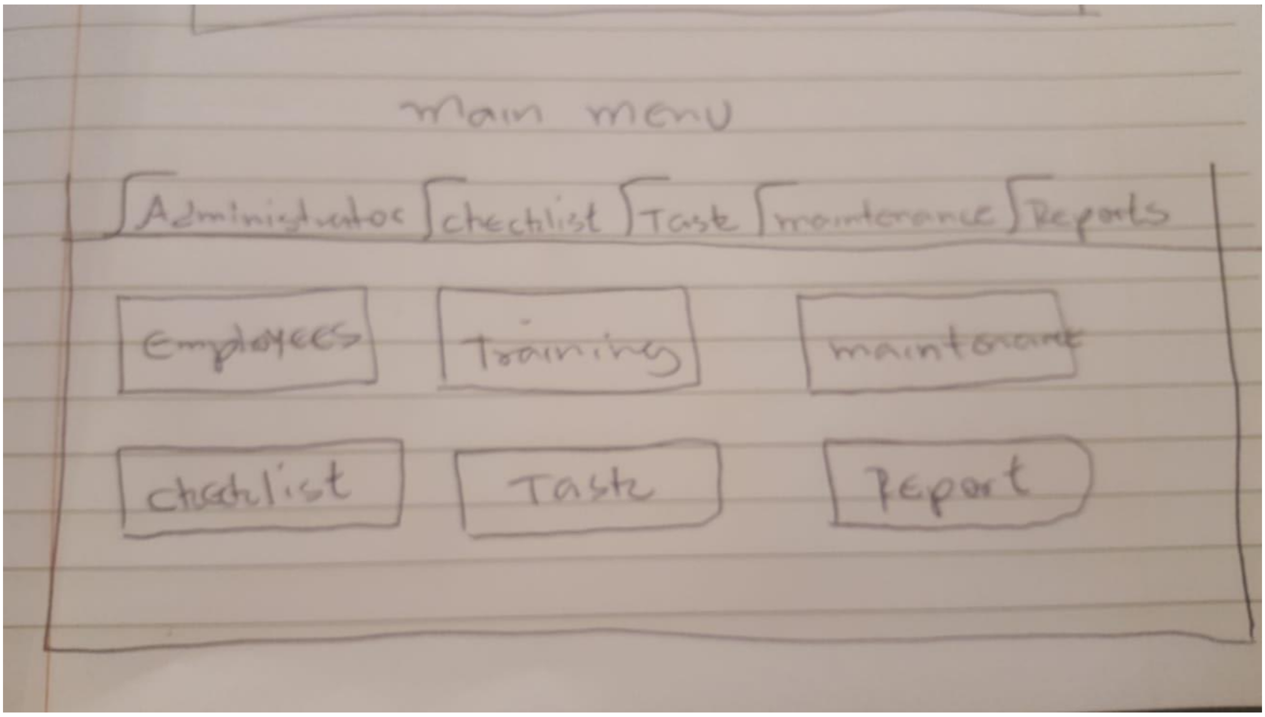
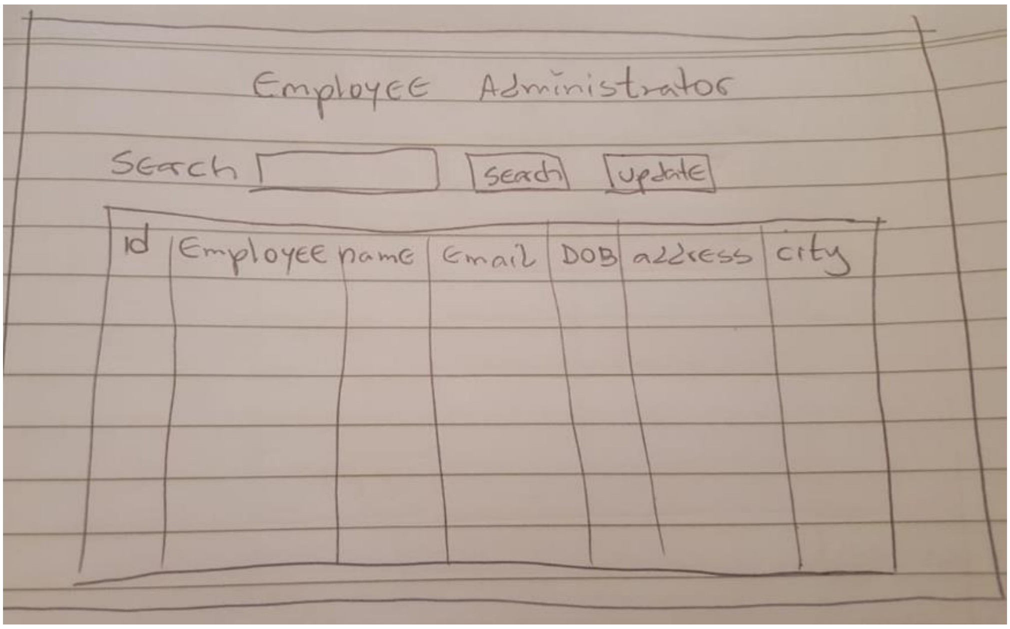


Figure 83 - Main menu

Employee Administrator Tab:



id	Employee name	Email	DOB	address	city

Figure 84 - Employee manager

Training Administrator Table:

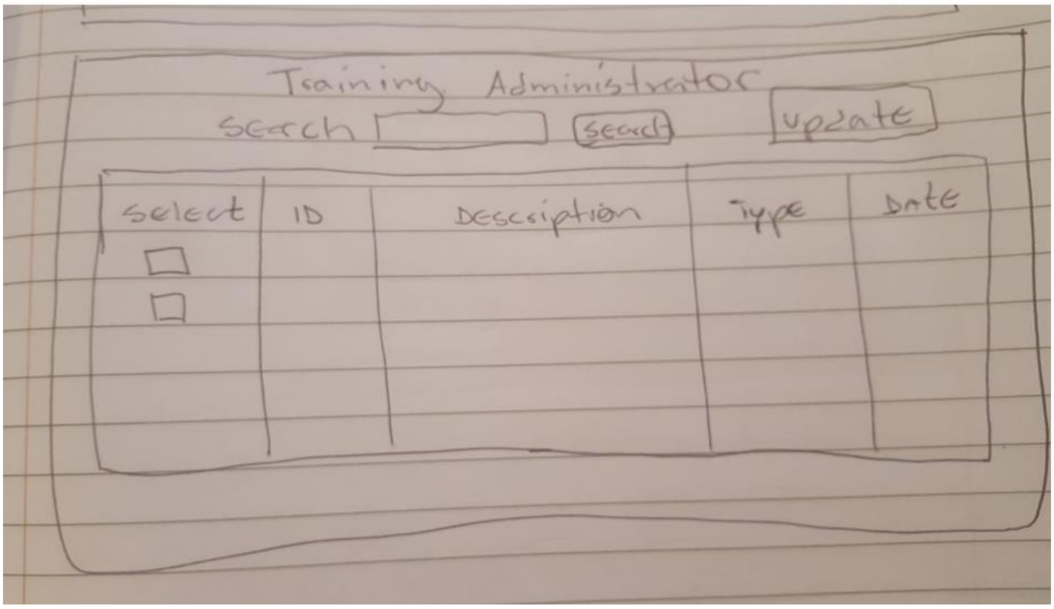


Figure 85 - Training manager window

Employee/Training Administrator Tab:

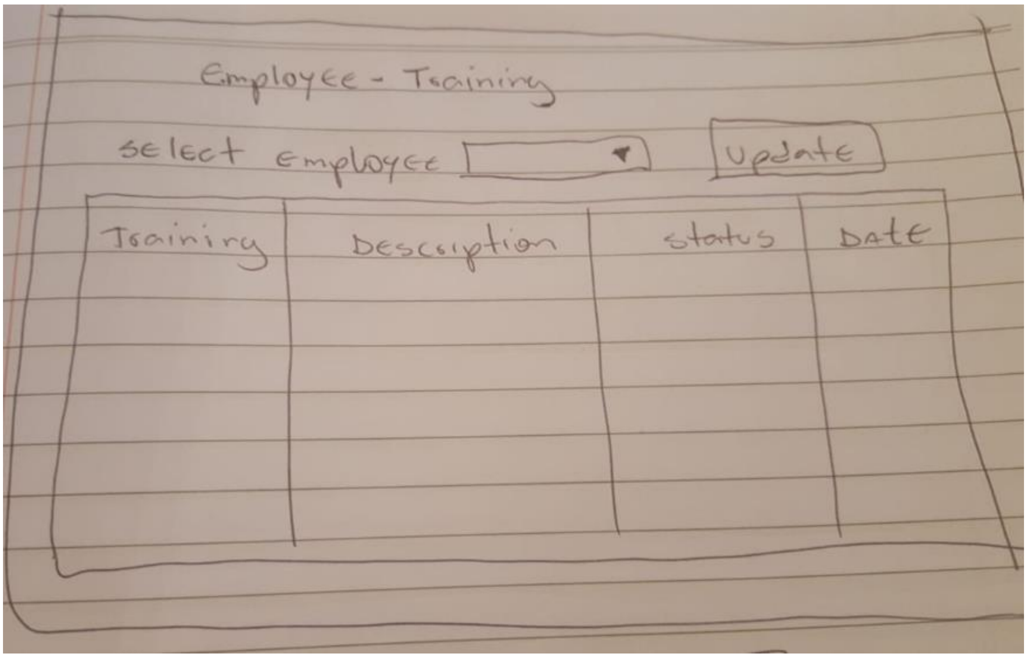


Figure 86 - Employee/training window

Simple Reports Tab:

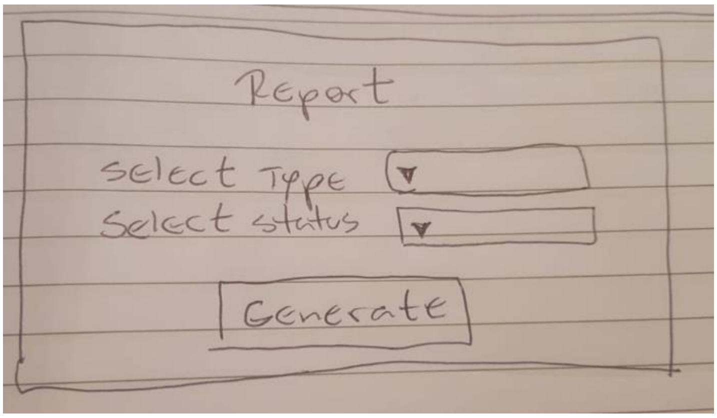


Figure 87 - Report generator

The checklist functionalities were not working the way we wanted, so we had to change for a table where you can just mark if it was done or not and it would reset accordingly to its frequency. We also added a login screen so no one that should not have access to the program can access it. Reports are working for all functionalities now too (Tasks, Checklists, Maintenance and Trainings).

Login Screen:

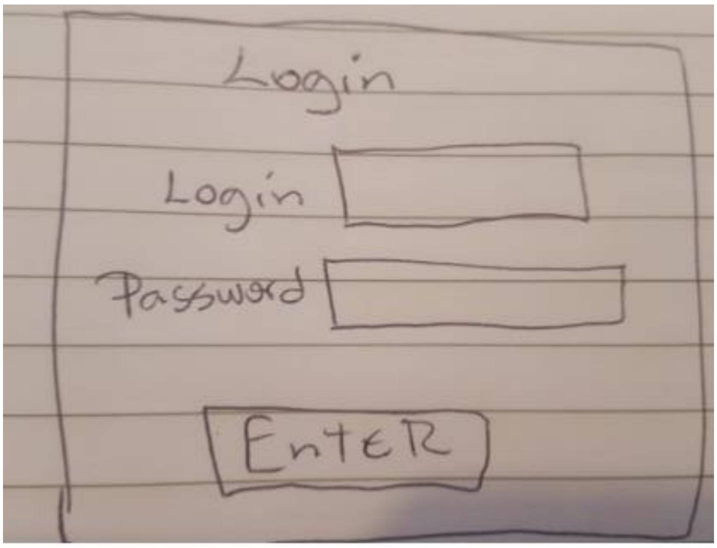


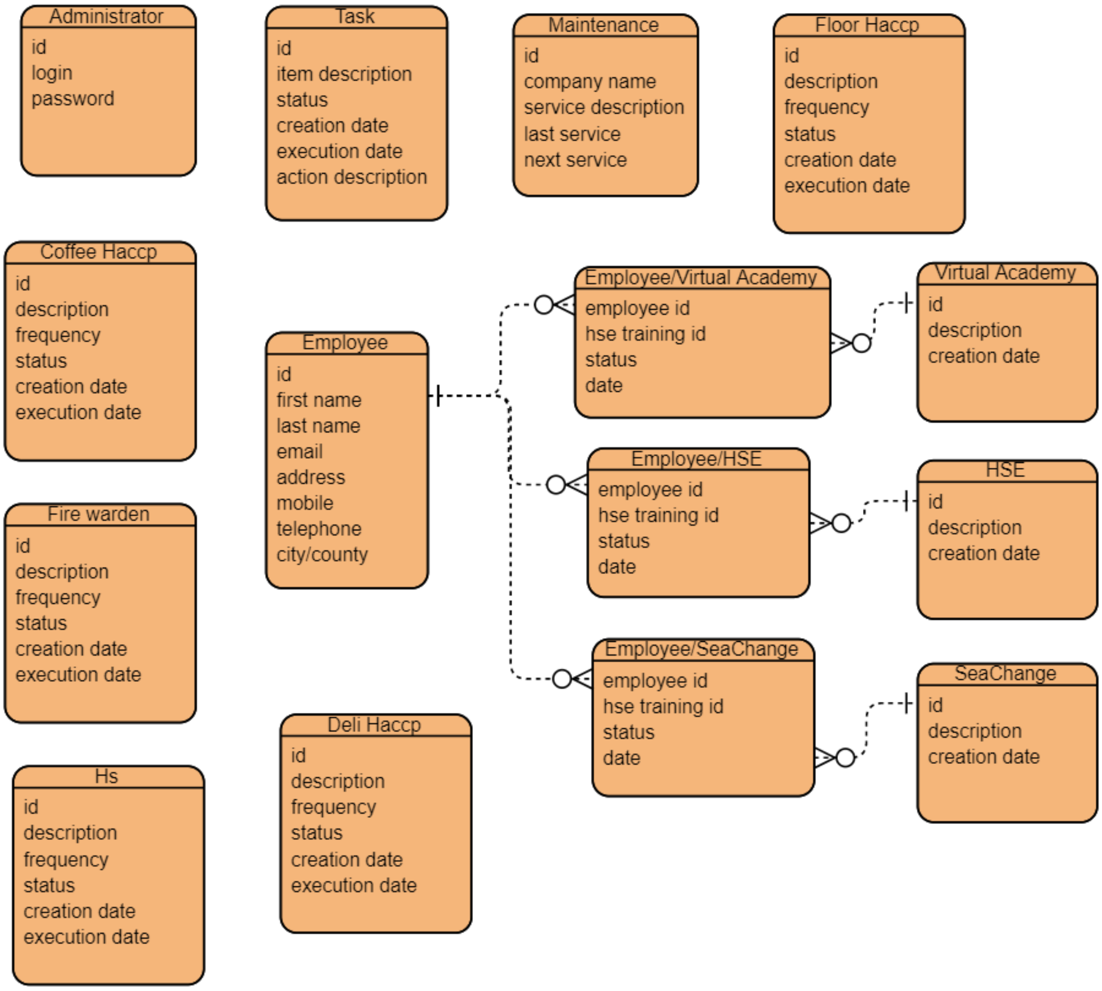
Figure 88 - Login entry

3.1 Data Design

The database which was chosen for this project is MySQL. Although MySQL is a relational database, for this project this database will be treated as object-oriented using Hibernate. Most table will not have a relationship with other tables.

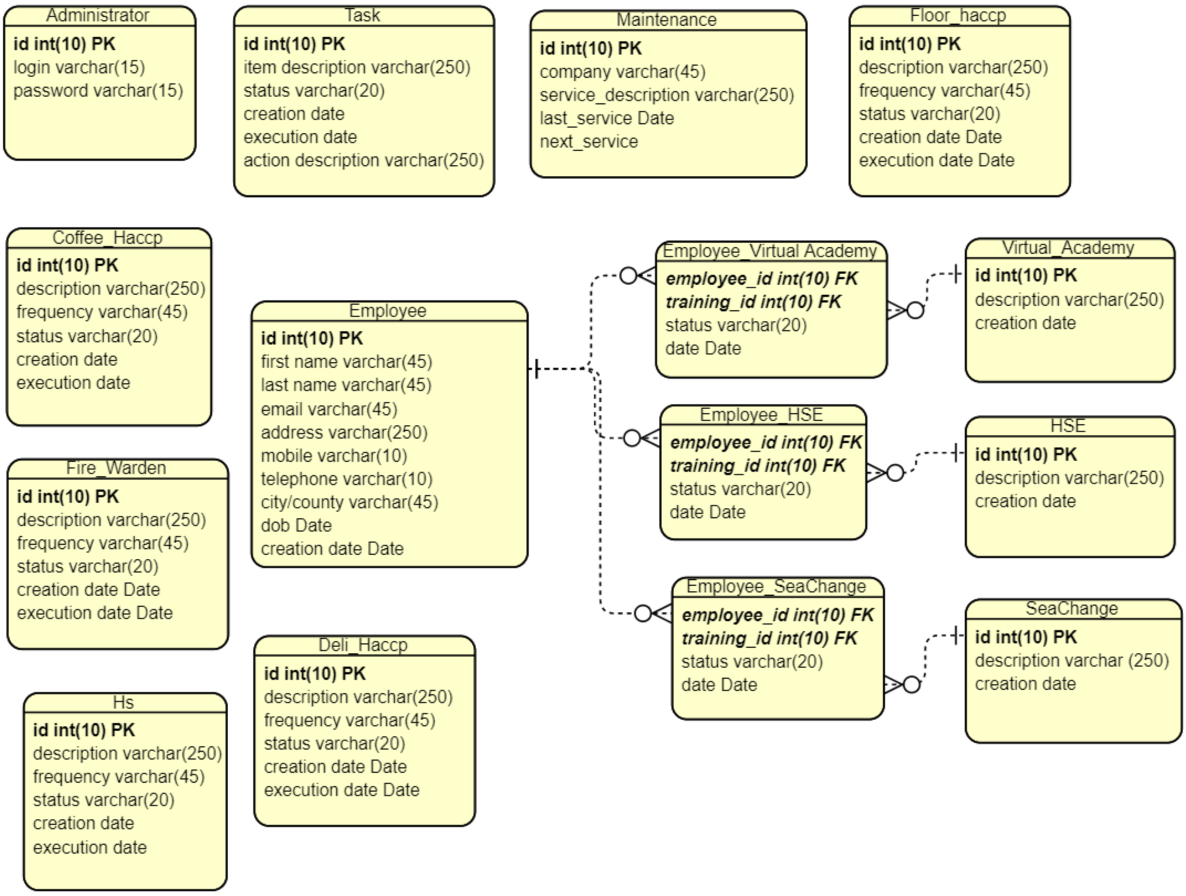
3.1.0 Conceptual Model

It is a simple representation of how the database should support the business needs. It is a concept of how the schema should be.



3.1.1 Physical Model

It is how the database schema must be to accommodate the application's needs. It shows the primary and foreign keys, the variables type and lengths. All primary keys must be auto incremented.



4.0 Technologies

Visual Paradigm - The visual paradigm is a powerful and mostly free cross-platform tool we will be using for UML modelling, manage development processes like USE CASES, database design and maybe code generation.

Eclipse –We used this IDE for programming in Java as we are more familiar with it and it provided us with a library that has a huge scope of plug-ins needed for developing the application.

JavaFX 2 – The reason why we used JavaFX 2 is that it provided us with some advantages over using only swing for GUI for java and here is why:

1- Being able to implement CSS style on JavaFX.

2- JavaFX has 2D chart as built-in data visualization.

3- JavaFX supports Java Lambda expressions in action-triggering events unlike Swing. Using Lambda expressions, JavaFX promotes code readability and just needs fewer lines of code than Swing.

4- JavaFX makes programmer`s life easier by offering JavaFX scene builder which programmers can create GUI controls like buttons or windows via drag and drop. (Oracle, 2013).

Spring framework 5 – Spring framework is one of the most popular frameworks to build java applications and has a lot of pros:

- Lightweight - It doesn’t have to receive any extra interface. In fact, the basic version is around 2 megabytes.

- Well-designed – It’s very easy to configure, as its very efficient it can be a great substitute for other frameworks.
- Integration – Very compatible with other frameworks in case you need it.

In general Spring framework is developer friendly as its organized, lightweight and easily testable and due to its capability for adopting changes it gets better day by day which enhances the development process and assists developers for rapid development. (Tutorial Point, 2019)

MySQL/phpMyAdmin/Hibernate – After a research on what kind of DBMS to use we decided to use an OODBMS (Object Oriented Database Management System) which was encouraged by our project supervisor. The main reason of the use of an OODBMS for our project is its consistency, data independency, secure, controlled and extensible data management services to support the object-oriented model where data is stored in form of object, as used in object-oriented programming.

Different from a relational database the OODBMS has the system generated OID or object identifier that is immutable and should not change which avoid the problem faced on relational database where users have to make sure two record do not have the same primary key.

Also the OODBMS supports the hold of complex data and application which is not possible in the other database models. The navigational access and concurrency control are improved by the programmability and performance of the OODBMS. (Academia, 2019).

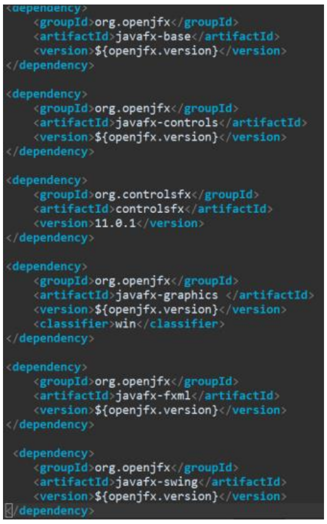
In terms of which OODBMS to use, after research it was clear that for starting, Versant would be the best option. Pronschinske, M (2010) says VOD (Versant object database) uses many concepts on the queries found in the ORM (object relational mapping) which someway is similar to the queries in relational database management systems which we are familiarized with. Although that was the chosen at first, we found it too confusing as we were not a hundred per cent sure on how to design an OODBMS and never used one.

In the end during our development cycle, we ended up deciding that we would use MySQL as we studied it during many semesters, and we were a lot more familiar with it and we ended up using Hibernate ORM (Object/Relational Mapping Framework) to map our relational model to an object oriented database. Hibernate is helpful to maintain data persistence from object-oriented models, it offers a scalable architecture, highly customizable and has great performance (Hibernate, 2020).

4.1 Implementation of the System

The first step to implement the system was to create a Maven project. As it had been previously decided to use JavaFX, MySQL and Hibernate, it would be necessary to use a few external libraries. It could be accomplished either by downloading all the necessary libraries or use Maven to do all the work. The team decided to go with Maven, so it was searched and added all the dependencies that would be necessary for this project.

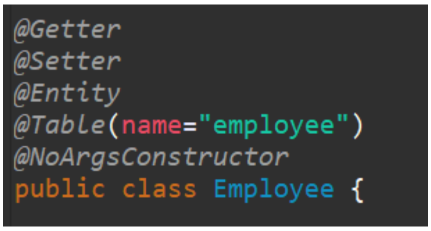
The dependencies below are necessary to create the graphical user interface (GUI) using JavaFX.



```
<dependency>
    <groupId>org.openjfx</groupId>
    <artifactId>javafx-base</artifactId>
    <version>${openjfx.version}</version>
</dependency>

<dependency>
    <groupId>org.openjfx</groupId>
    <artifactId>javafx-controls</artifactId>
    <version>${openjfx.version}</version>
</dependency>

<dependency>
    <groupId>org.controlsfx</groupId>
    <artifactId>controlsfx</artifactId>
    <version>11.0.1</version>
</dependency>

<dependency>
    <groupId>org.openjfx</groupId>
    <artifactId>javafx-graphics </artifactId>
    <version>${openjfx.version}</version>
    <classifier>win</classifier>
</dependency>

<dependency>
    <groupId>org.openjfx</groupId>
    <artifactId>javafx-fxml</artifactId>
    <version>${openjfx.version}</version>
</dependency>

 <dependency>
    <groupId>org.openjfx</groupId>
    <artifactId>javafx-swing</artifactId>
    <version>${openjfx.version}</version>
</dependency>
```

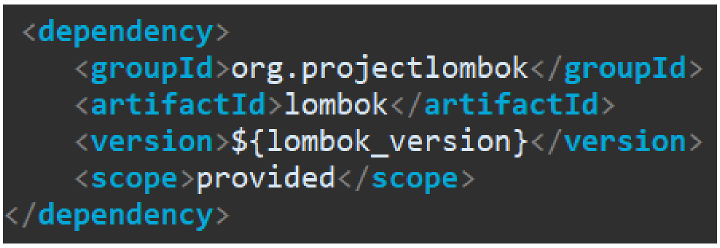
Figure 89 - JavaFX dependencies

As part of the encapsulation process, instances variables are declared as private, so it is necessary to create access to them out of the class where they reside in using setters and getters. It generates a lot of boilerplate code. So, to eliminate a lot of work and lines of code, the Project Lombok dependency was added. Lombok allows the use of setters and getters without writing the code for each variable. Instead @Setter and @Getter annotations are added at the top of the class or the instance variables.



```
@Getter
@Setter
@Entity
@Table(name="employee")
@NoArgsConstructor
public class Employee {
```

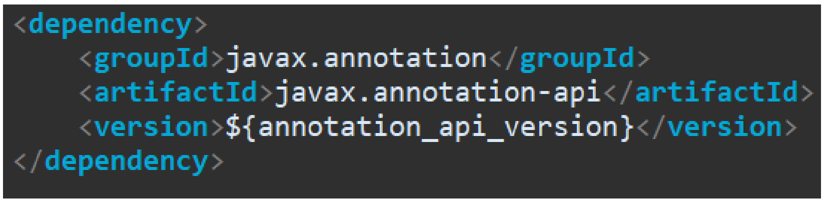
Figure 90 - Lombok and Hibernate annotations



```
 <dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>${lombok_version}</version>
    <scope>provided</scope>
</dependency>
```

Figure 91 - Lombok dependency

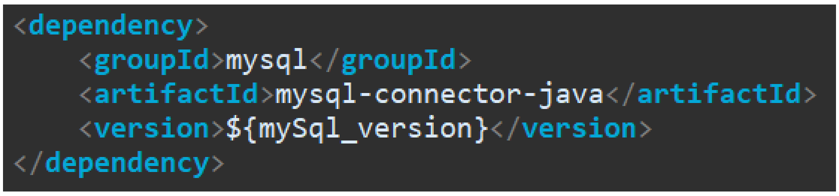
To allow java to make use of annotations, Javax annotation dependency was added.



```
<dependency>
    <groupId>javax.annotation</groupId>
    <artifactId>javax.annotation-api</artifactId>
    <version>${annotation_api_version}</version>
</dependency>
```

Figure 92 - Javax Annotation dependency

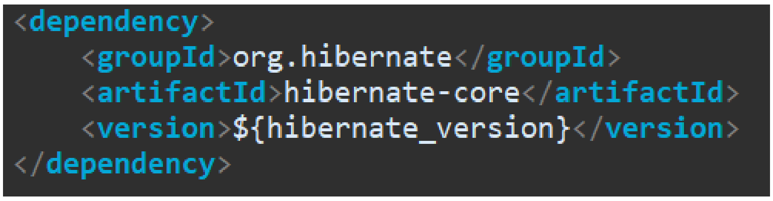
MySQL database was chosen for this project. To make it work with Java it was necessary to add the dependency.



```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>${mySql_version}</version>
</dependency>
```

Figure 93 - MySQL dependency

To make the MySQL connection with Java it was necessary to add the Hibernate dependency.

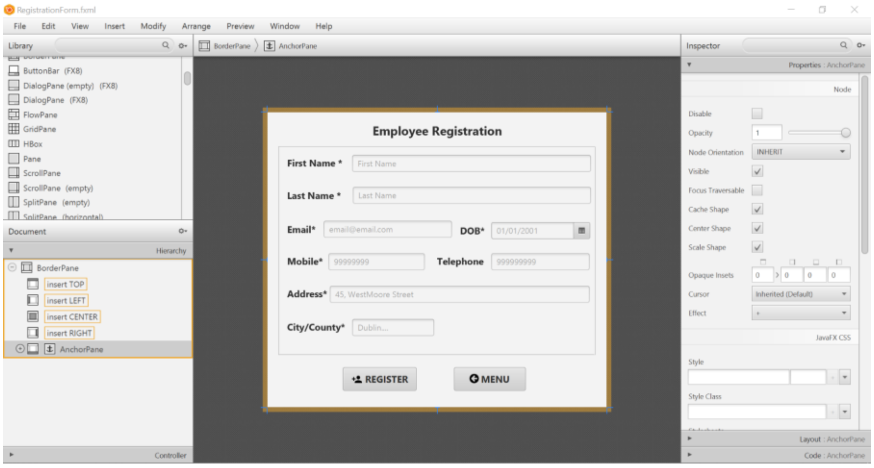


```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>${hibernate_version}</version>
</dependency>
```

Figure 94 - Hibernate dependency

All the GUI was created using Scene Builder that is a tool that allows to easily create components. There is no need to code manually. Scene Builder allows to drag and drop components wherever the developer wants it. Although it is needed previous knowledge of Java Swing to work with this tool.

Scene Builder



Employee Registration

First Name * First Name

Last Name * Last Name

Email* email@email.com DOB* 01/01/2001

Mobile* 99999999 Telephone 999999999

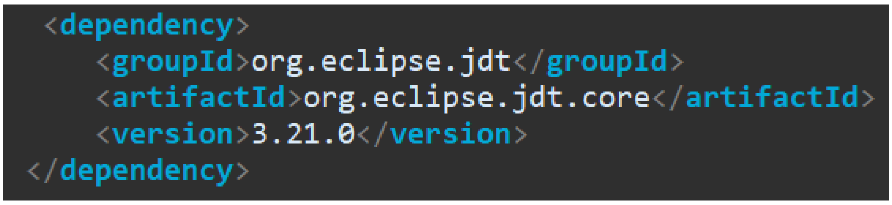
Address* 45, WestMoore Street

City/County* Dublin...

Disable □
Opacity 1
Node Orientation INHERIT
Visible ☑
Focus Traversable □
Cache Shape ☑
Center Shape ☑
Scale Shape ☑
Opaque Insets 0 0 0 0
Cursor Inherited (Default)
Effect +
Style
Style Class

Figure 95 - Scene Builder interface

To use Scene Builder with Eclipse IDE, it was necessary to add the dependency below.



```
  <dependency>
     <groupId>org.eclipse.jdt</groupId>
     <artifactId>org.eclipse.jdt.core</artifactId>
     <version>3.21.0</version>
 </dependency>
```

Figure 96 - Eclipse JDT dependency

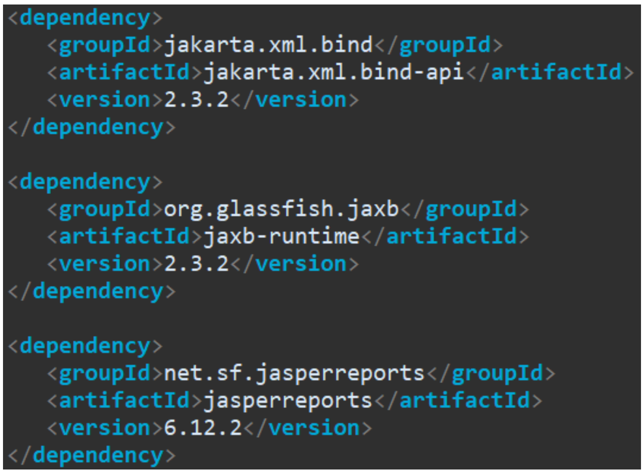
The project is structured based on the Model View Controller (MVC) software design pattern. The project has a package named model that holds other sub-packages which hold the interfaces, Enums, abstract and non-abstract classes responsible for the logic of the software. Inside the package resources there is a folder named view which stores all the FXML files that are used to render the graphics. The package controller holds all the controller classes responsible for processing input data, executing the application logic and determining the view to the user. All the FXML files were generated automatically by Scene Builder. Every FXML file has its controller, the main reason is to avoid extensive controller classes with a lot of lines of code. This way is better to keep the code organized and manageable.

All the GUI control components were created based on the Spar documentation and what was designed by Henrique who is one of the team's member and works for Spar which is where this application will be used.

The purpose of the application is to save time. So, the graphical user interface had to be very friendly and easy to interact. At the coding time some changes had to be done to achieve the application purpose. Some of the layouts in the design was a little confusing. The layouts were adjusted and changed a few times until the team agreed with the final result.

The class structures follow exactly as in the class diagrams in the system analysis chapter. All employees when starting to work in the company they must take trainings that are separated in three categories, SeaChange, HSE and Virtual Academy. From the code perspective they are treated as three different objects with similar attributes. A super class is created with all common attributes and these training objects share these attributes as they became children of the super class. The same structure is used with the checklist objects that are separated in six categories. All share attributes from their parent class. We made the best use of inheritance principles to keep the code shorter and more readable.

Other feature of the application is to generate pdf reports. To accomplish it some external libraries are required.

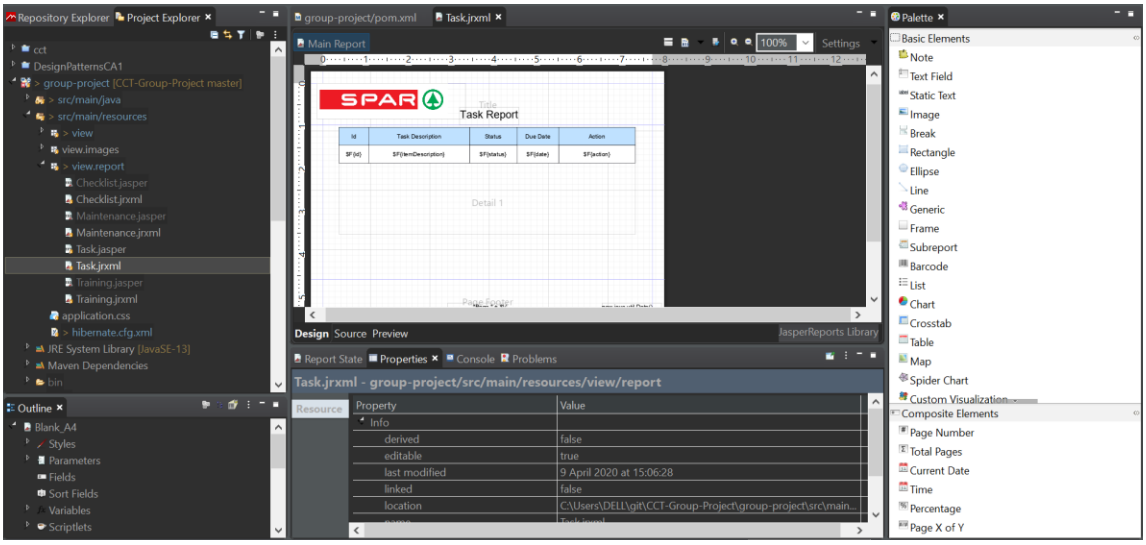


```
<dependency>
   <groupId>jakarta.xml.bind</groupId>
   <artifactId>jakarta.xml.bind-api</artifactId>
   <version>2.3.2</version>
</dependency>

<dependency>
   <groupId>org.glassfish.jaxb</groupId>
   <artifactId>jaxb-runtime</artifactId>
   <version>2.3.2</version>
</dependency>

<dependency>
   <groupId>net.sf.jasperreports</groupId>
   <artifactId>jasperreports</artifactId>
   <version>6.12.2</version>
</dependency>
```

Figure 97 - Jasper report dependencies

Jasper report is a platform for designing and delivering custom reports. Jaspersoft offers a plugin that makes possible to use the jasper tools inside the Eclipse IDE. Jasper generates a JRXML file that can be compiled to Jasper extension and be available for the Java application. The tool was used to design four custom reports. One for the task object, the second one for the training objects, the third one for the checklist objects and the fourth for the maintenance objects.

Jasper plugin



Task Report

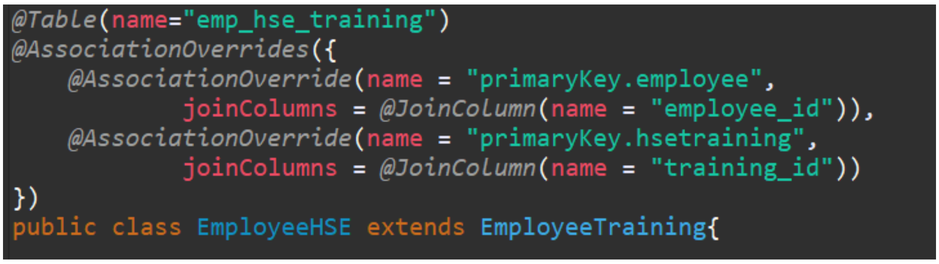
Id	Task Description	Status	Due Date	Action
$F{id}	$F{itemDescription}	$F{status}	$F{date}	$F{action}

Task.jrxml - group-project/src/main/resources/view/report

Property	Value
Info	
derived	false
editable	true
last modified	9 April 2020 at 15:06:28
linked	false
location	C:\Users\DELL\git\CCT-Group-Project\group-project\src\main...

Figure 98 - Jasper Eclipse plugin

MySQL is a relational Database, but for the project we decided to use an Object-Oriented Database (OODB). We had to use the Hibernate framework that provides mapping from object-oriented model to a relational database and vice-versa. Instead of saving the attributes separately as it would be with JDBC, the Hibernate allows to save them as an object. There are a few relations between tables that were necessary for the training section. There is an employee object and a training object that are totally different objects in different tables. As each employee must take all the registered training, it was necessary to create a relational table between them. This table takes the employee id primary key with the training id primary key, a status attribute to check if the employee has finished the training and a date attribute to register when the training was finished. With this table is possible to control all the training. This was accomplished using Hibernate that provides a join table annotation as it is shown below.

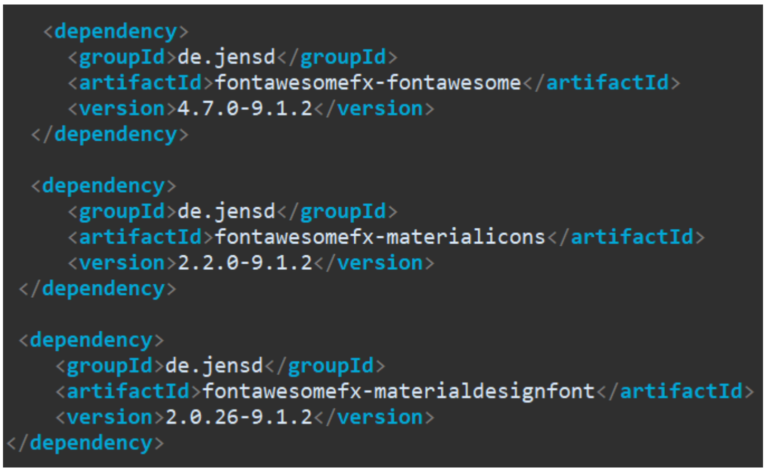


```java
@Table(name="emp_hse_training")
@AssociationOverrides({
    @AssociationOverride(name = "primaryKey.employee",
            joinColumns = @JoinColumn(name = "employee_id")),
    @AssociationOverride(name = "primaryKey.hsetraining",
            joinColumns = @JoinColumn(name = "training_id"))
})
public class EmployeeHSE extends EmployeeTraining{
```

Figure 99 - Hibernate join tables annotation

If a training object or an employee object is deleted from the database, automatically the employee object and training object is deleted from the relational table. Hibernate provides this flexibility.

To create a more friendly user interface, we decided to add icons to the buttons improving the application visual. It was only possible after adding the dependencies below that adds icon libraries to the Scene Builder



```xml
   <dependency>
     <groupId>de.jensd</groupId>
     <artifactId>fontawesomefx-fontawesome</artifactId>
     <version>4.7.0-9.1.2</version>
  </dependency>

  <dependency>
     <groupId>de.jensd</groupId>
     <artifactId>fontawesomefx-materialicons</artifactId>
     <version>2.2.0-9.1.2</version>
 </dependency>

 <dependency>
    <groupId>de.jensd</groupId>
    <artifactId>fontawesomefx-materialdesignfont</artifactId>
    <version>2.0.26-9.1.2</version>
</dependency>
```

Figure 100 - FontAwesome dependencies

There were a few changes in the interfaces design in comparison to the had in the design. They were necessary changes to improve the visual and make the application more friendly to the user.

4.2 Design Patterns

4.2.0 Singleton pattern

The class responsible for the communication with the database was structured using the Singleton pattern. The main reason is that there was no reason to have more than one instance of this class. In the project, an instance of this class is created when the application is started and then it is used the same object with different reference variables as long as the application is running. The singleton pattern guarantees that only one object with a global access point can be created in a given instance of time (Debnath, 2019).

The singleton object dies when the application is ended, then the connection with the database is ended as well. It prevents the application from having multiple connections opened while the application is running.

4.2.1 Factory pattern

According to Pankaj (n.d), The factory design pattern is used when we have a superclass with multiple sub-classes and based on input, we need to return one of the sub-class.

In the checklist package there is a super class with six child classes. They all share the same attributes from their super class. When creating a checklist object, the user selects which object it wants to add to that checklist. Also, there is a Factory class inside the package that creates one object of the six child classes according to the user input. Inside package training there is a training super class with three child classes. Also, there is a Factory class that handles the creation of a training object according to the user input.

4.2.2 Data access pattern (DAO)

According to Shubham (n.d), DAO is used to separate the data persistence logic in a separate layer. The application layer does not know how the low-level operations to access the database is done.

DAO pattern was implemented in this project. There is only one class that interacts directly with the database named DataSource and a DAO class that allows the controller classes to interact with the database indirectly through it. It works as a bridge between the controllers and the DataSource class. It isolates the complexities of database operations from the application logic.

4.3 Problems Encountered

The first problem encountered was to initialize the application from the main class. To use the JavaFX, the main class must extend javafx.application.Application, and it seems that the JavaFX platform must be available as a module and not as a jar file. Every time we ran the main class, we got the Error: JavaFX runtime components are missing and are required to run this application. While searching a way to fix the problem we found an easy fix that solved this problem. We had to add another main class that did not extend javafx.application.Application and call the main class that extends application from it.

The second problem was the lack of knowledge with tools that we had not used. It was the first time we ever used Scene Builder, so we had to learn how to work with it. The same with Jasper report tool. Although we had some basic knowledge to work with Hibernate, we had a lot of trouble trying to join tables with it. All solved searching, learning, and applying.

5.0 Functional correctness and Usability

5.0.0 Functionalities:

After all the main functionalities were incorporated in the application, we tested the program to see if there are any crashes, bugs, glitches or if any of the functionalities are not doing what they are supposed to do.

This is the list of what should be working in the system:

Login screen with user login and password. Validation should only who has an account log in in the system.

There should be a main menu with shortcuts to chosen main functionalities:

- Employee Manager Table.
- Checklist Manager Table.
- Training/Employee Manager Table.
- Task Manager Table.
- Maintenance Manager Table.
- Fast Report Selector, to check any of the pending trainings, checklists, tasks and maintenances.

There are 7 menu/tabs for the management of each area that we separated (Employees, Checklists, Trainings, Tasks, Maintenance and Reports plus the Menu).

Administrator Tab:

- Add Employee: Adds an employee to the system with personal info.
- Employee Manager: You can update already registered employees or delete them from the system.
- Change Password: You can change the password of the user login.

Checklist Tab:

- Add Checklist: Add a checklist to the system. Every checklist has a frequency which are used in the "Update Status" to reset them every time this frequency has finished.
- Update Status: Here you can update if the checklist was done or not, checklists that its frequency time has finished (daily, weekly, monthly) will reset and show up back again in this table.
- Checklist Manager: Here you can delete checklists or update their names and information.

Training Tab:

- Add Training: Here you can add new trainings to the system, every training fit one of these 3 categories: SeaChange, Virtual Academy or HSE.
- Training Manager: In this table you can change the training info or delete old trainings.
- Employee/Training: This is where both training and employee are added together. Whenever a new training is created, all employees inherit that training as everyone needs to do them. So here you can select an employee and check which trainings are pending and which ones are done. You can also change if a training was done by the employee or not and insert the date it was done.

Maintenance Tab:

- Add Contractor: Adds a contractor to the system.
- Contractor Manager: In this table you can edit the contractors, when the last maintenance was done and when the next should be. Whenever a maintenance passes its due date, it gives the user a warning and shows in pending maintenance reports.

Reports Tab:

Here is where we gather all the data that is inserted in the other tabs and let the user create customised reports. The user can choose from many different options:

What has been done, what is pending, all of them together, check only about one of the staff trainings, etc.

There are 4 options of reports to choose and all of them have customisable settings:

- Task Report.
- Checklist Report.
- Maintenance Report.
- Training Report.

5.0.1 Test Scripts:

We did a few test scripts to test our program functionalities and if everything is working correctly behind the scenes. Some of the data was added all through our system(besides the login, as the program already needs an login in the database to be able to work), we compared the input done through the system to the database data to check if everything was working properly.

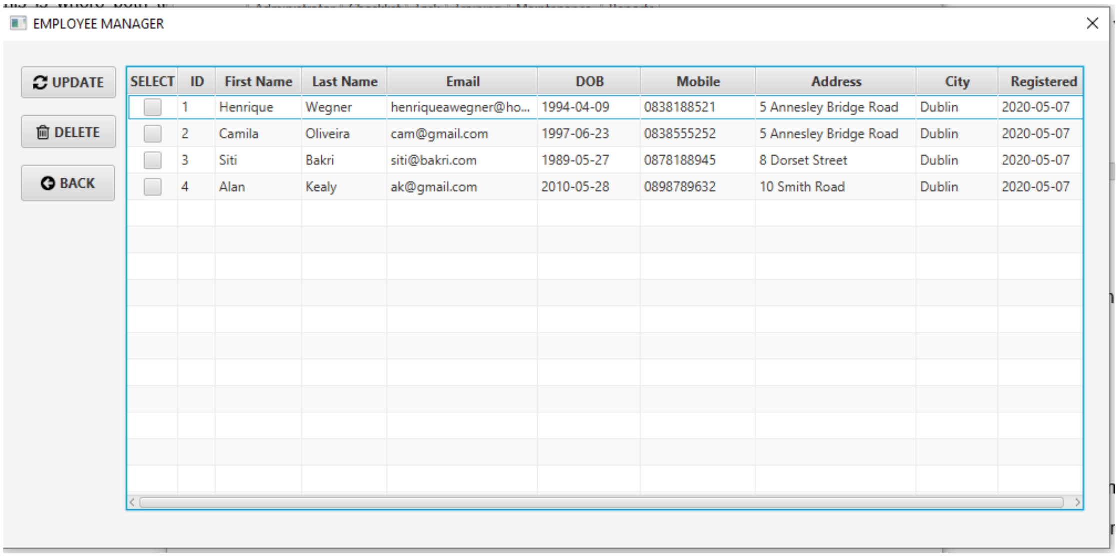
1st Test Script:

The first test script will test the Administrator tab, Employee Tab and Reports Tab.

- Add 3 new employees to the system.
- Delete one of the employees and update another.
- Add 2 new trainings to the system and delete an older one.
- Check if trainings and employees were added and if they are connected.
- Show a report of all pending trainings for employees.

Results:

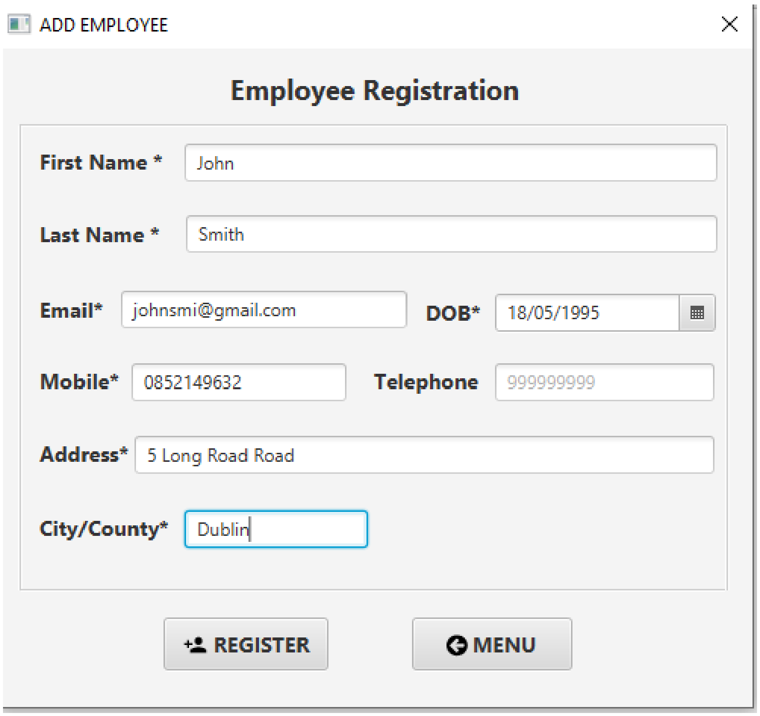
Employee table with already registered employees.



SELECT	ID	First Name	Last Name	Email	DOB	Mobile	Address	City	Registered
□	1	Henrique	Wegner	henriqueawegner@ho...	1994-04-09	0838188521	5 Annesley Bridge Road	Dublin	2020-05-07
□	2	Camila	Oliveira	cam@gmail.com	1997-06-23	0838555252	5 Annesley Bridge Road	Dublin	2020-05-07
□	3	Siti	Bakri	siti@bakri.com	1989-05-27	0878188945	8 Dorset Street	Dublin	2020-05-07
□	4	Alan	Kealy	ak@gmail.com	2010-05-28	0898789632	10 Smith Road	Dublin	2020-05-07

Figure 101 - Employee manager window

Adding a new employee called John Smith. Another 2 (Amanda McDonald and Arthur Morgan) were added but no screenshots were taken to not fill up the documentation with too many images.



Employee Registration

First Name * John

Last Name * Smith

Email* johnsmi@gmail.com

DOB* 18/05/1995

Mobile* 0852149632

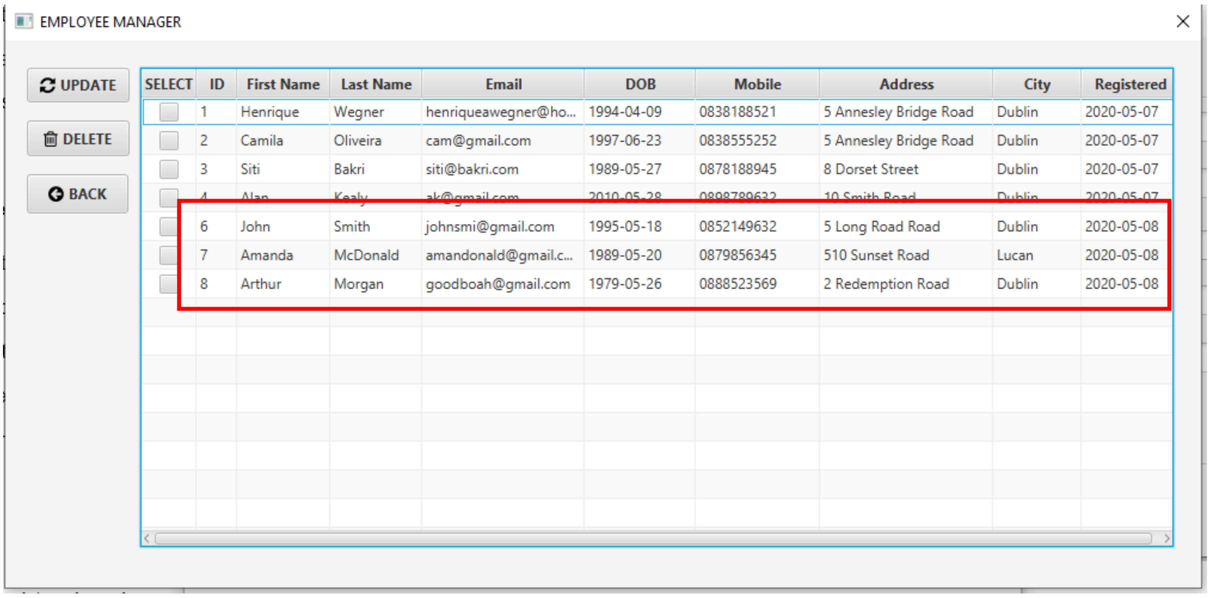
Telephone 999999999

Address* 5 Long Road Road

City/County* Dublin

Figure 102 - Employee registration window

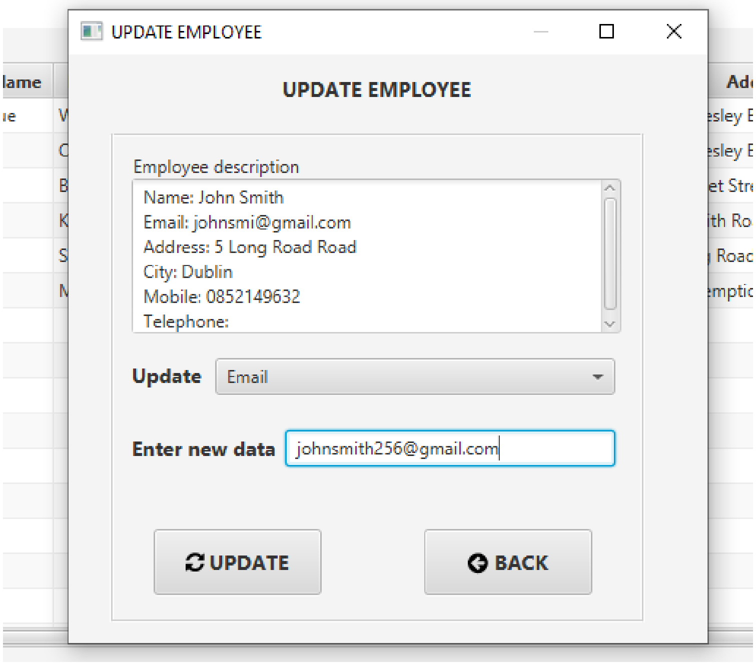
Added new employees to the table.



SELECT	ID	First Name	Last Name	Email	DOB	Mobile	Address	City	Registered
□	1	Henrique	Wegner	henriqueawegner@ho...	1994-04-09	0838188521	5 Annesley Bridge Road	Dublin	2020-05-07
□	2	Camila	Oliveira	cam@gmail.com	1997-06-23	0838555252	5 Annesley Bridge Road	Dublin	2020-05-07
□	3	Siti	Bakri	siti@bakri.com	1989-05-27	0878188945	8 Dorset Street	Dublin	2020-05-07
□	4	Alan	Kealy	ak@gmail.com	2010-05-28	0898789632	10 Smith Road	Dublin	2020-05-07
□	6	John	Smith	johnsmi@gmail.com	1995-05-18	0852149632	5 Long Road Road	Dublin	2020-05-08
□	7	Amanda	McDonald	amandonald@gmail.c...	1989-05-20	0879856345	510 Sunset Road	Lucan	2020-05-08
□	8	Arthur	Morgan	goodboah@gmail.com	1979-05-26	0888523569	2 Redemption Road	Dublin	2020-05-08

Figure 103 - Employees added

We then deleted Amanda and modified John Smith email.



UPDATE EMPLOYEE

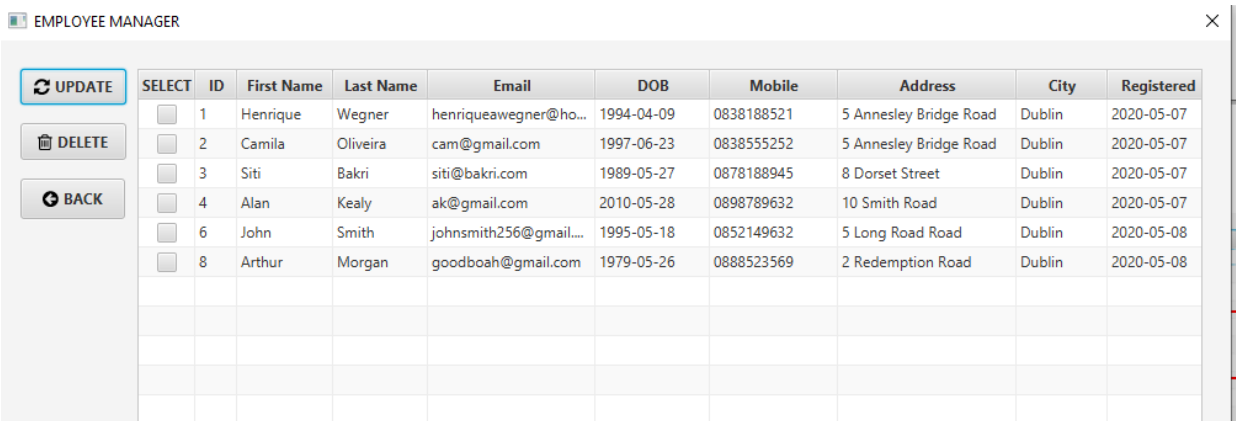
Employee description

Name: John Smith
Email: johnsmi@gmail.com
Address: 5 Long Road Road
City: Dublin
Mobile: 0852149632
Telephone:

Update: Email

Enter new data: johnsmith256@gmail.com

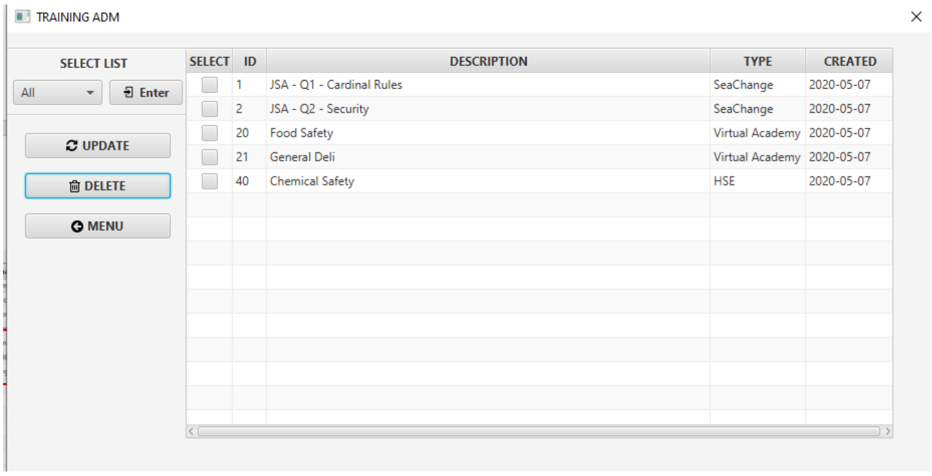
Figure 104 - Update employee window



SELECT	ID	First Name	Last Name	Email	DOB	Mobile	Address	City	Registered
□	1	Henrique	Wegner	henriqueawegner@ho...	1994-04-09	0838188521	5 Annesley Bridge Road	Dublin	2020-05-07
□	2	Camila	Oliveira	cam@gmail.com	1997-06-23	0838555252	5 Annesley Bridge Road	Dublin	2020-05-07
□	3	Siti	Bakri	siti@bakri.com	1989-05-27	0878188945	8 Dorset Street	Dublin	2020-05-07
□	4	Alan	Kealy	ak@gmail.com	2010-05-28	0898789632	10 Smith Road	Dublin	2020-05-07
□	6	John	Smith	johnsmith256@gmail....	1995-05-18	0852149632	5 Long Road Road	Dublin	2020-05-08
□	8	Arthur	Morgan	goodboah@gmail.com	1979-05-26	0888523569	2 Redemption Road	Dublin	2020-05-08

Figure 105 - Employee updated

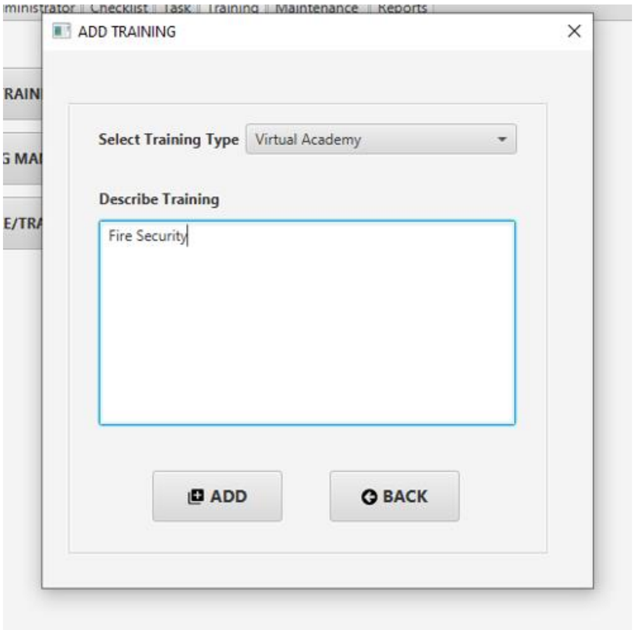
Training Manager with already registered trainings:



SELECT	ID	DESCRIPTION	TYPE	CREATED
□	1	JSA - Q1 - Cardinal Rules	SeaChange	2020-05-07
□	2	JSA - Q2 - Security	SeaChange	2020-05-07
□	20	Food Safety	Virtual Academy	2020-05-07
□	21	General Deli	Virtual Academy	2020-05-07
□	40	Chemical Safety	HSE	2020-05-07

Figure 106 - Training manager window

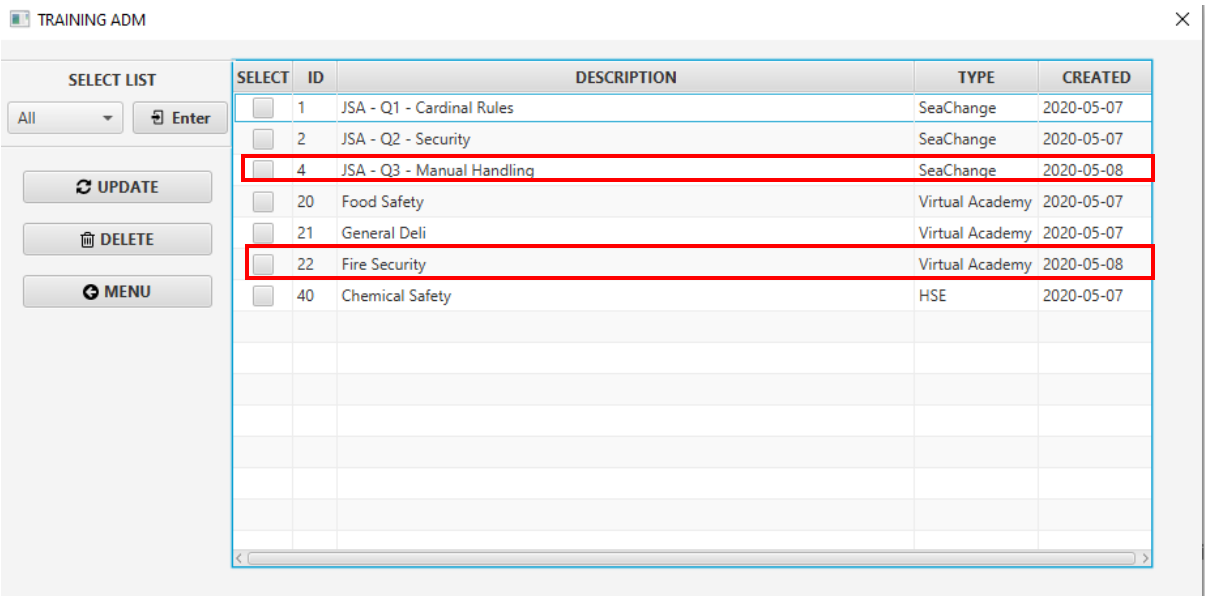
Two new trainings were added, one called JSA Q3 (SeaChange) and one for Fire Safety (Virtual Academy). Only the last one was taken a screenshot.


ministrator | Checklist | Task | Training | Maintenance | Reports
ADD TRAINING
RAIN
G MAI
E/TRA
ADD
BACK

Select Training Type: Virtual Academy

Describe Training: Fire Security

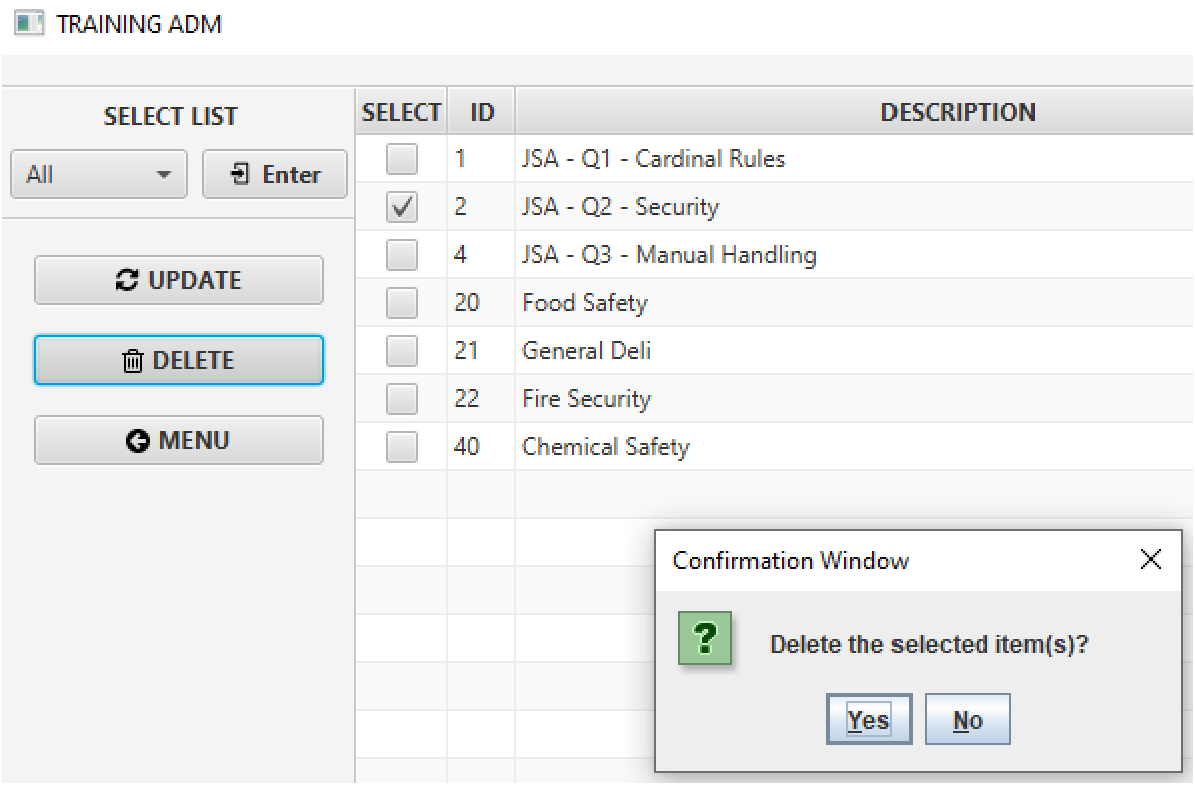
Figure 107 - Add training window



SELECT	ID	DESCRIPTION	TYPE	CREATED
☐	1	JSA - Q1 - Cardinal Rules	SeaChange	2020-05-07
☐	2	JSA - Q2 - Security	SeaChange	2020-05-07
☐	4	JSA - Q3 - Manual Handling	SeaChange	2020-05-08
☐	20	Food Safety	Virtual Academy	2020-05-07
☐	21	General Deli	Virtual Academy	2020-05-07
☐	22	Fire Security	Virtual Academy	2020-05-08
☐	40	Chemical Safety	HSE	2020-05-07

Figure 108 - Training manager

After that we deleted the JSA – Q2 from the table.:

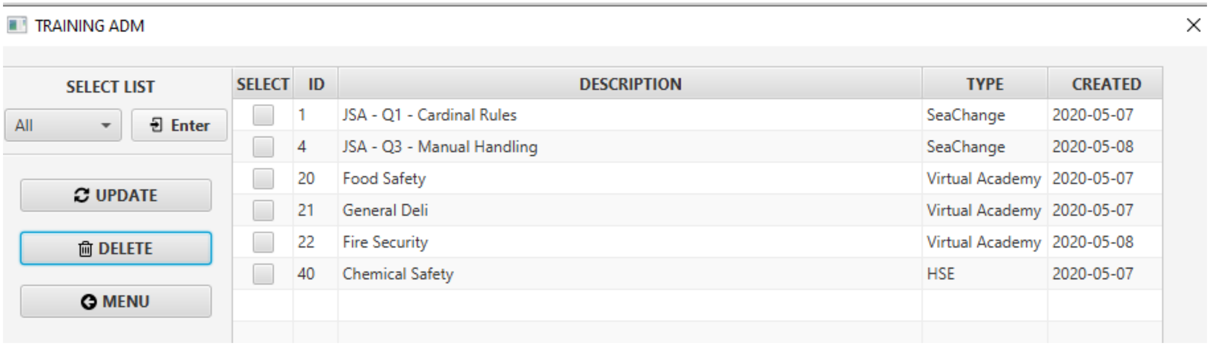


SELECT	ID	DESCRIPTION
□	1	JSA - Q1 - Cardinal Rules
☑	2	JSA - Q2 - Security
□	4	JSA - Q3 - Manual Handling
□	20	Food Safety
□	21	General Deli
□	22	Fire Security
□	40	Chemical Safety

Confirmation Window

Delete the selected item(s)?

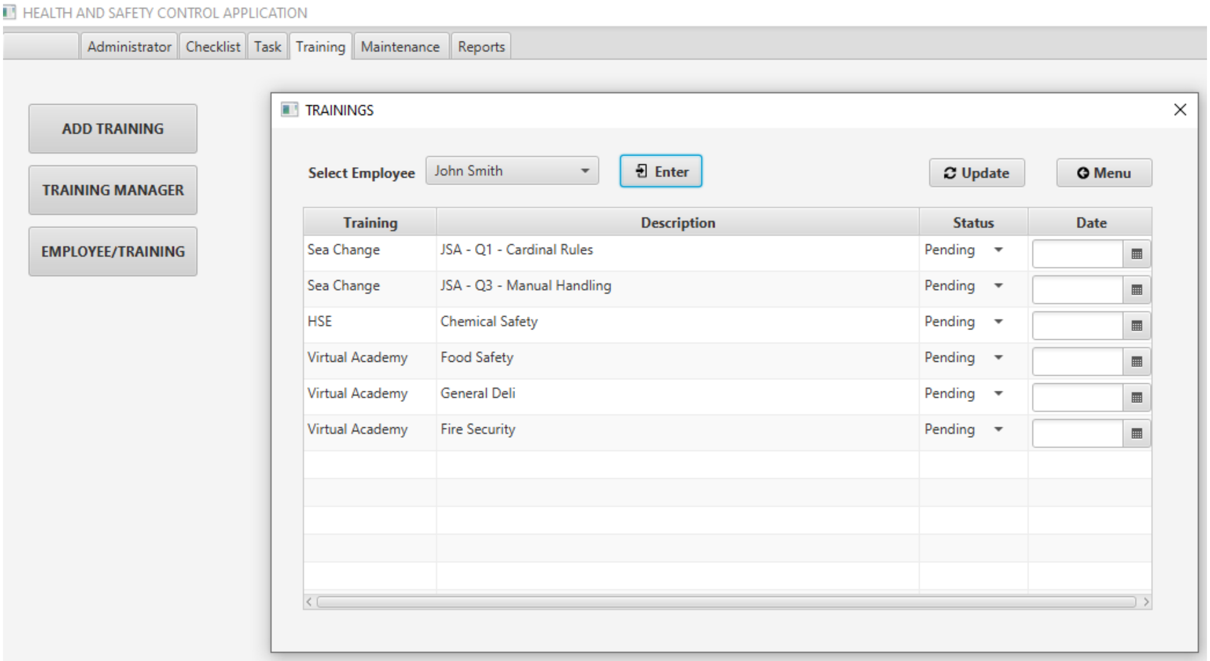
Figure 109 - Deletion of a training



SELECT	ID	DESCRIPTION	TYPE	CREATED
□	1	JSA - Q1 - Cardinal Rules	SeaChange	2020-05-07
□	4	JSA - Q3 - Manual Handling	SeaChange	2020-05-08
□	20	Food Safety	Virtual Academy	2020-05-07
□	21	General Deli	Virtual Academy	2020-05-07
□	22	Fire Security	Virtual Academy	2020-05-08
□	40	Chemical Safety	HSE	2020-05-07

Figure 110 - After deletion

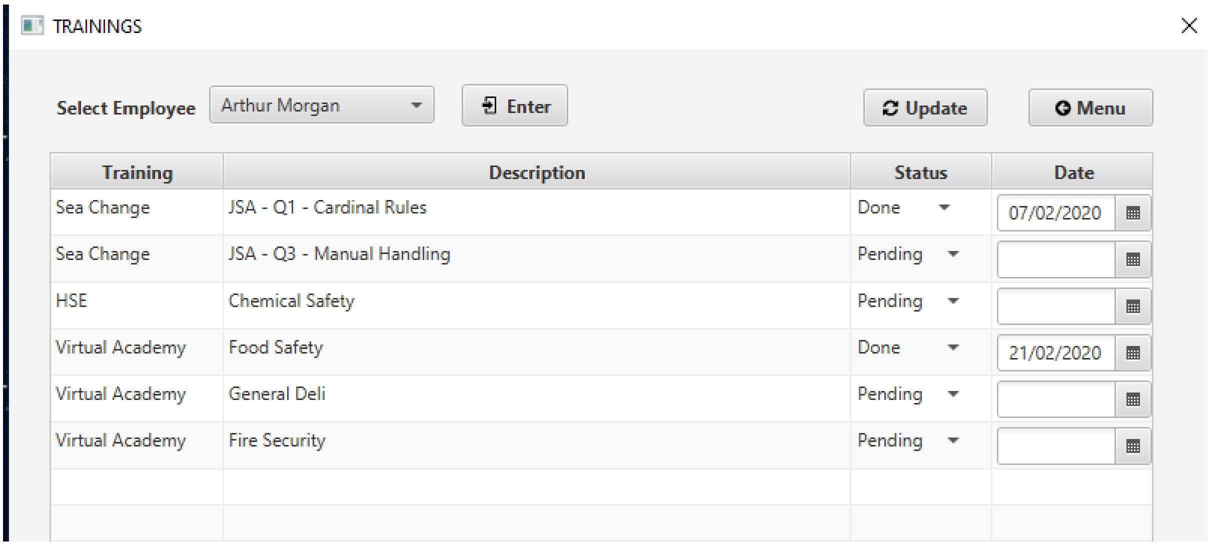
Showing employees and training connected to each other (John Smith and Arthur Morgan), a couple of Arthur's training were modified to done so they will not show up on the final report which will only show the pending trainings:



Select Employee: John Smith

Training	Description	Status	Date
Sea Change	JSA - Q1 - Cardinal Rules	Pending	
Sea Change	JSA - Q3 - Manual Handling	Pending	
HSE	Chemical Safety	Pending	
Virtual Academy	Food Safety	Pending	
Virtual Academy	General Deli	Pending	
Virtual Academy	Fire Security	Pending	

Figure 111 - Pending training

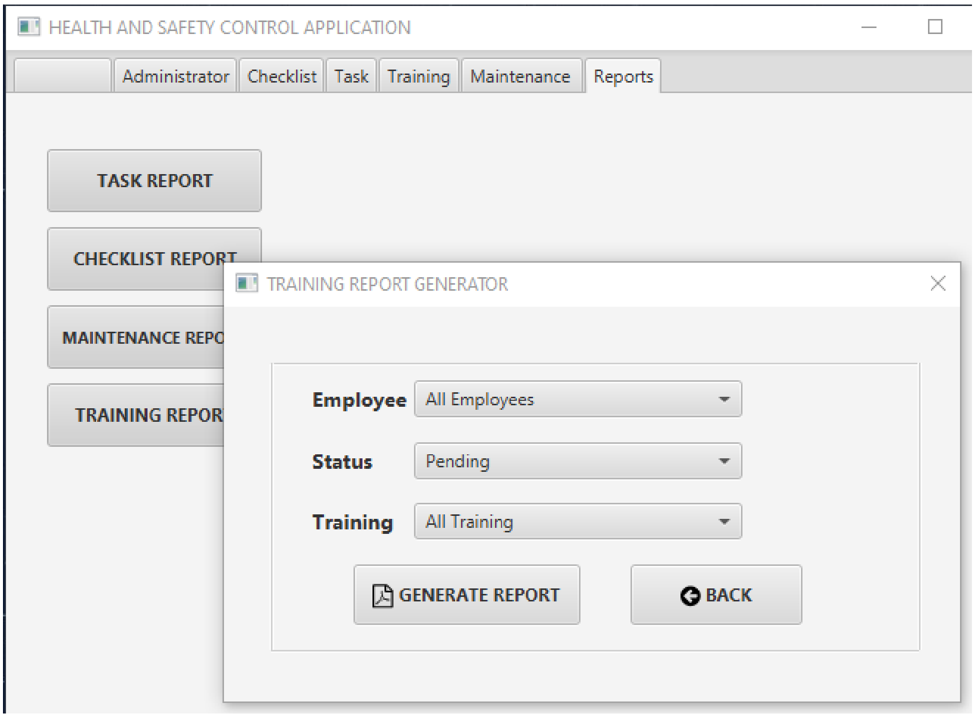


Select Employee: Arthur Morgan

Training	Description	Status	Date
Sea Change	JSA - Q1 - Cardinal Rules	Done	07/02/2020
Sea Change	JSA - Q3 - Manual Handling	Pending	
HSE	Chemical Safety	Pending	
Virtual Academy	Food Safety	Done	21/02/2020
Virtual Academy	General Deli	Pending	
Virtual Academy	Fire Security	Pending	

Figure 112 - Pending and done

Creating a report on all pending trainings for all employees:



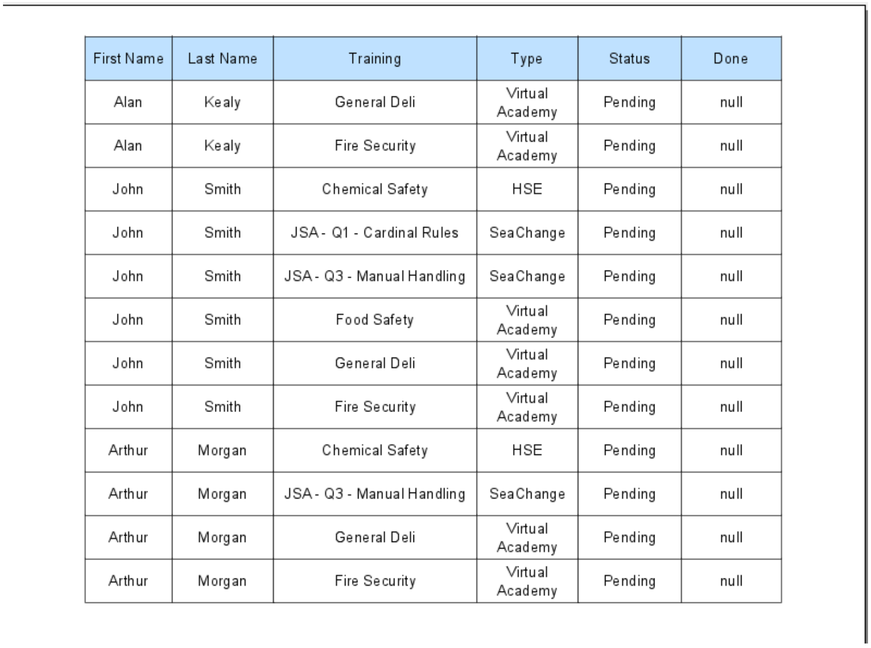
Employee: All Employees

Status: Pending

Training: All Training

Figure 113 – Training report generator

On the second page of the report, we can see the newly added employees and the pending trainings. It does not show the ones that were modified to done as we selected only the pending status.



First Name	Last Name	Training	Type	Status	Done
Alan	Kealy	General Deli	Virtual Academy	Pending	null
Alan	Kealy	Fire Security	Virtual Academy	Pending	null
John	Smith	Chemical Safety	HSE	Pending	null
John	Smith	JSA - Q1 - Cardinal Rules	SeaChange	Pending	null
John	Smith	JSA - Q3 - Manual Handling	SeaChange	Pending	null
John	Smith	Food Safety	Virtual Academy	Pending	null
John	Smith	General Deli	Virtual Academy	Pending	null
John	Smith	Fire Security	Virtual Academy	Pending	null
Arthur	Morgan	Chemical Safety	HSE	Pending	null
Arthur	Morgan	JSA - Q3 - Manual Handling	SeaChange	Pending	null
Arthur	Morgan	General Deli	Virtual Academy	Pending	null
Arthur	Morgan	Fire Security	Virtual Academy	Pending	null

Figure 114 - Report generated

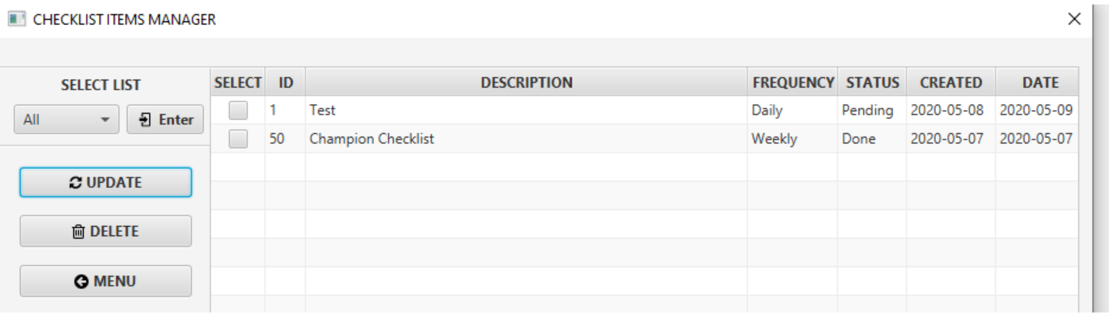
2nd Test Script:

The second test script will test the Checklist Tab and Reports Tab.

- Add 2 new checklists.
- Update two of the checklists in the system as done.
- Alter one of the checklists to an earlier date so it resets in the "Update Status" table.
- Show a report of all pending weekly reports and one of late checklists.

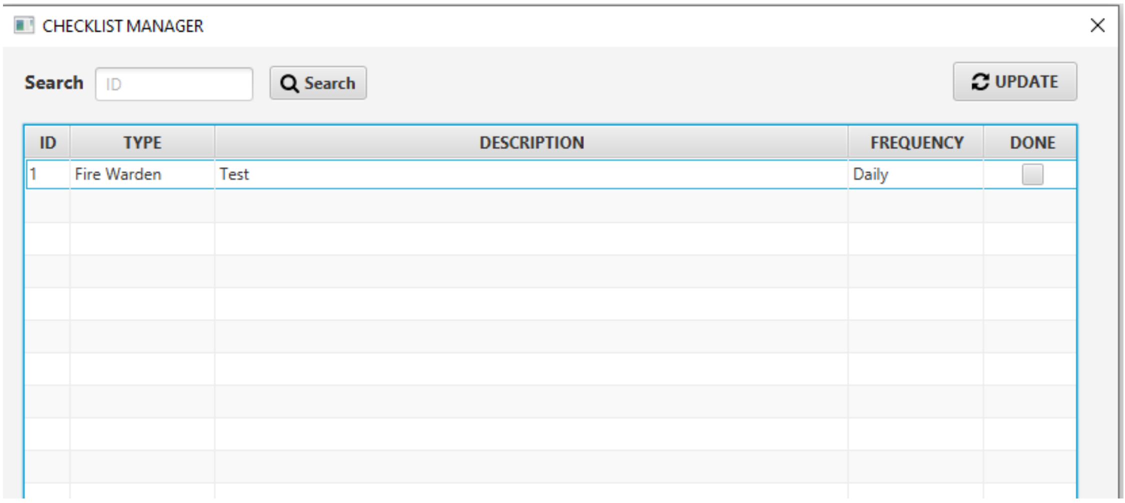
Results:

Checklist Manager and Update Status before entering our 2 new checklists:



SELECT	ID	DESCRIPTION	FREQUENCY	STATUS	CREATED	DATE
□	1	Test	Daily	Pending	2020-05-08	2020-05-09
□	50	Champion Checklist	Weekly	Done	2020-05-07	2020-05-07

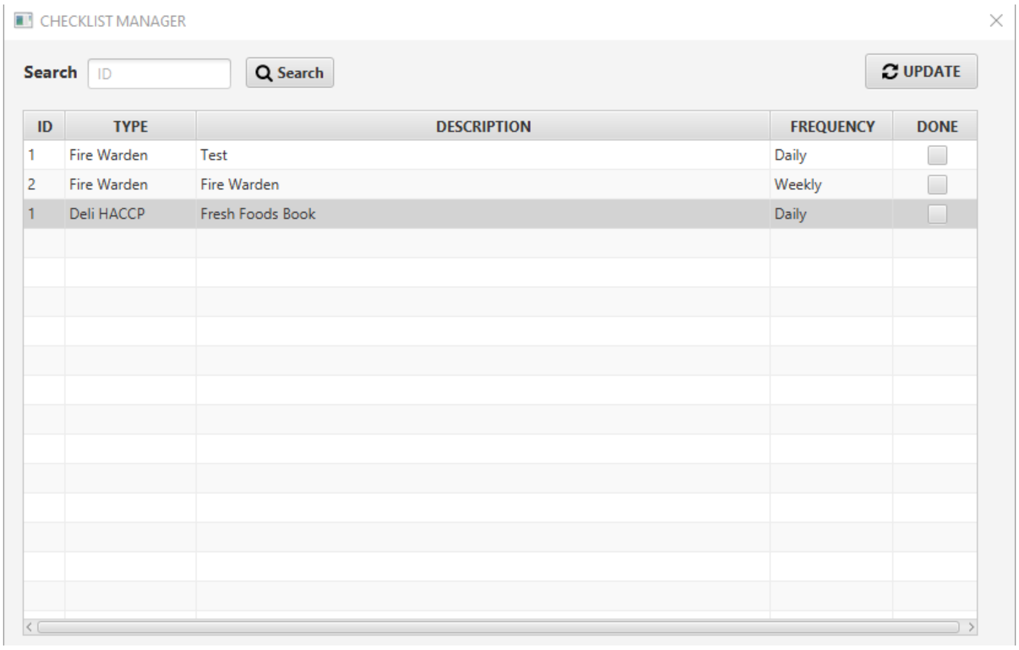
Figure 115 - Checklist manager



ID	TYPE	DESCRIPTION	FREQUENCY	DONE
1	Fire Warden	Test	Daily	□

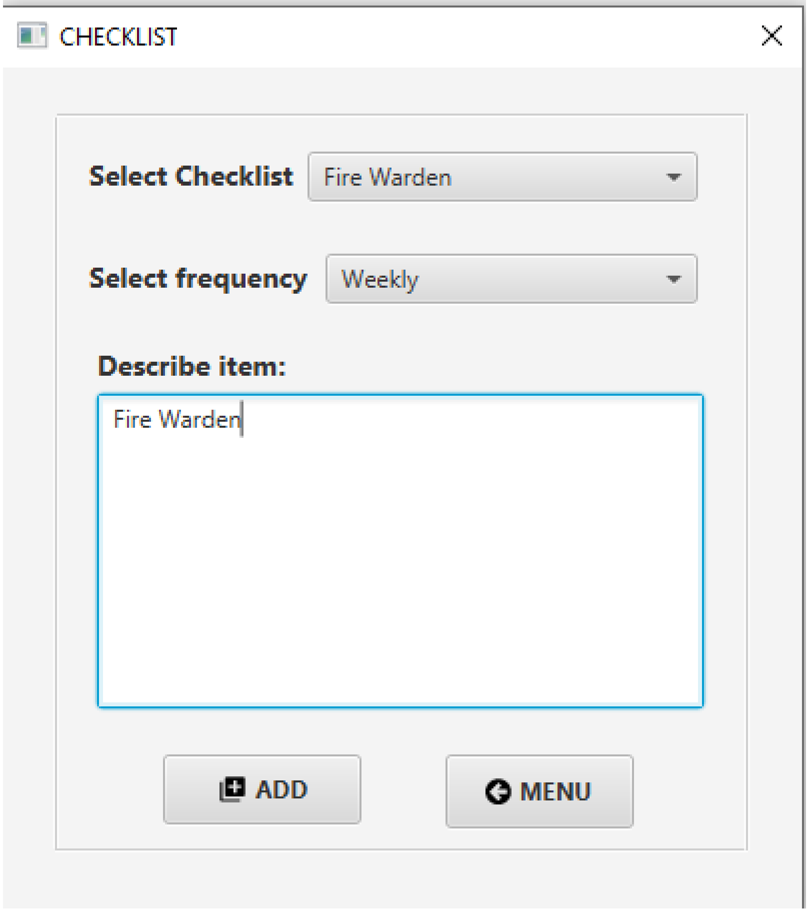
Figure 116 - Checklist update item window

Creating two new checklists (only one screenshot):



ID	TYPE	DESCRIPTION	FREQUENCY	DONE
1	Fire Warden	Test	Daily	□
2	Fire Warden	Fire Warden	Weekly	□
1	Deli HACCP	Fresh Foods Book	Daily	□

Figure 117 - Checklist manager



Select Checklist: Fire Warden

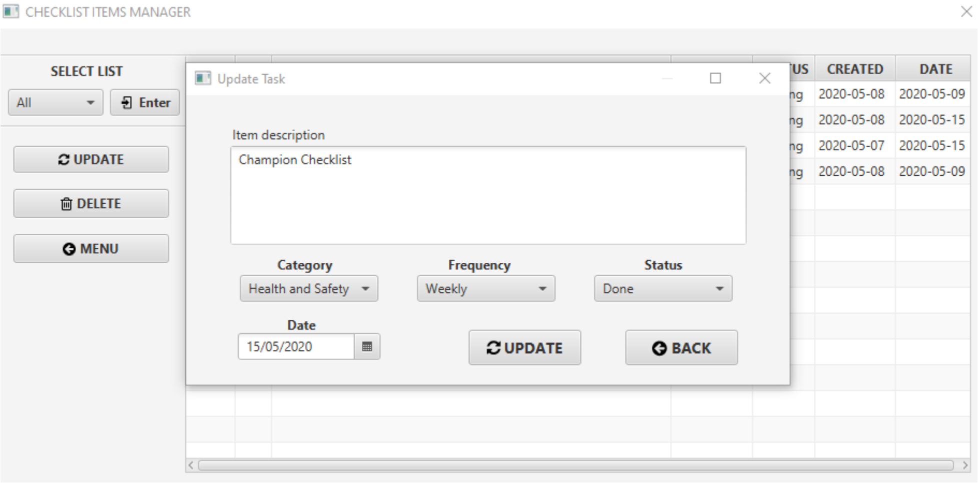
Select frequency: Weekly

Describe item:

Fire Warden

Figure 118 - Add checklist item window

The champion checklist was dated back as it was as done in the system, so it does not show in the Update Status. Now it is supposed to show again, and because I changed the date to a week before the system's date it will show as a checklist that is "late"(it passed its due date):



US	CREATED	DATE
ng	2020-05-08	2020-05-09
ng	2020-05-08	2020-05-15
ng	2020-05-07	2020-05-15
ng	2020-05-08	2020-05-09

Item description: Champion Checklist

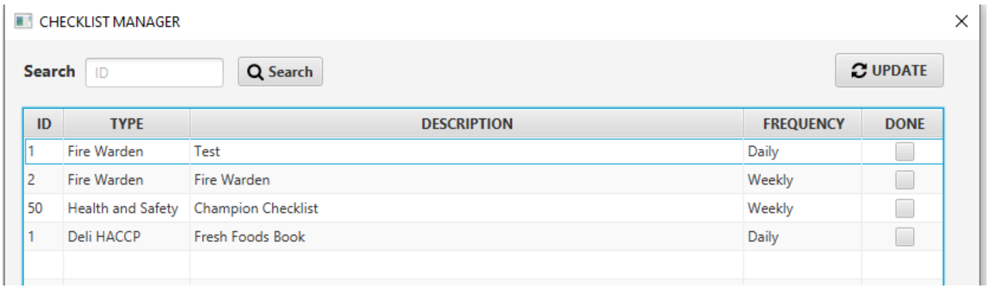
Category: Health and Safety

Frequency: Weekly

Status: Done

Date: 15/05/2020

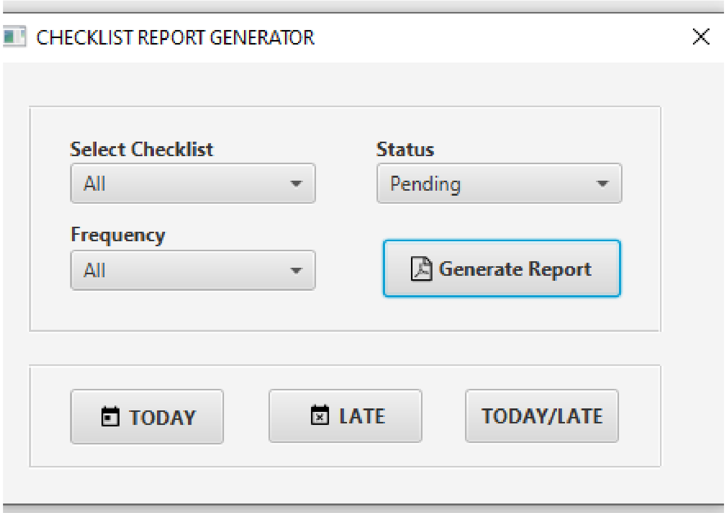
Figure 119 - Update task window



ID	TYPE	DESCRIPTION	FREQUENCY	DONE
1	Fire Warden	Test	Daily	□
2	Fire Warden	Fire Warden	Weekly	□
50	Health and Safety	Champion Checklist	Weekly	□
1	Deli HACCP	Fresh Foods Book	Daily	□

Figure 120 - Checklist manager window

This is the Checklist Report Generator, you besides checking what is pending you can also check the late checklists and the ones that still need to be done for the day:



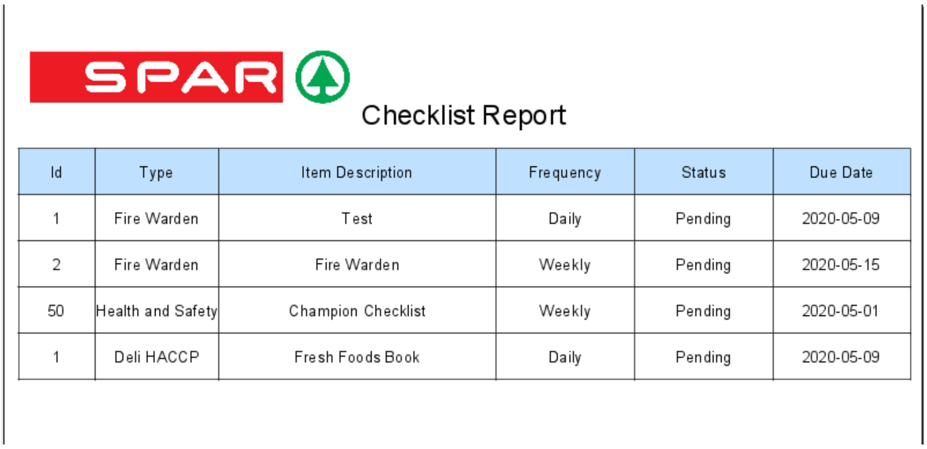
Select Checklist: All

Status: Pending

Frequency: All

Figure 121 - Checklist report generator

Report for all pending checklists:

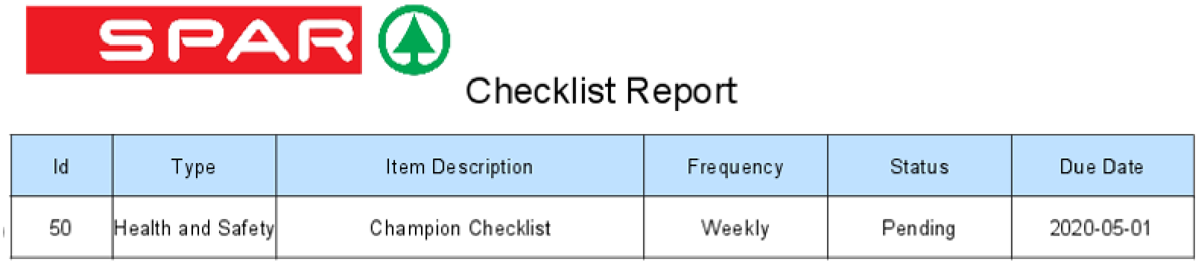


Checklist Report

Id	Type	Item Description	Frequency	Status	Due Date
1	Fire Warden	Test	Daily	Pending	2020-05-09
2	Fire Warden	Fire Warden	Weekly	Pending	2020-05-15
50	Health and Safety	Champion Checklist	Weekly	Pending	2020-05-01
1	Deli HACCP	Fresh Foods Book	Daily	Pending	2020-05-09

Figure 122 - Checklist report generated - pending

Report for all late checklists:



Checklist Report

Id	Type	Item Description	Frequency	Status	Due Date
50	Health and Safety	Champion Checklist	Weekly	Pending	2020-05-01

Figure 123 - Late checklist items

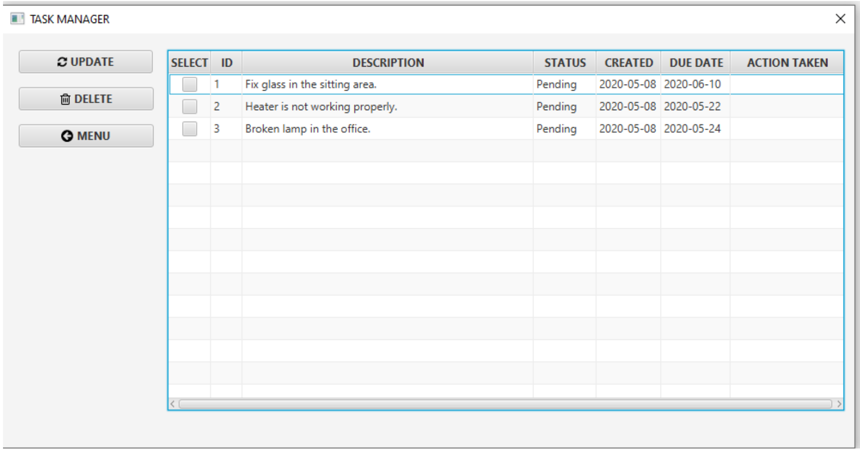
3rd Test Script:

The test script will test the Task Tab and Reports Tab.

- Add 4 new tasks.
- Update two of the tasks as done.
- Generate three reports of all tasks: done, pending and all.

Results:

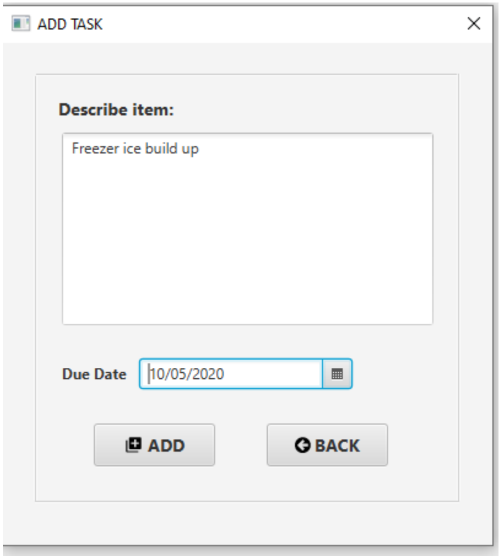
Task manager before adding the new tasks:



SELECT	ID	DESCRIPTION	STATUS	CREATED	DUE DATE	ACTION TAKEN
□	1	Fix glass in the sitting area.	Pending	2020-05-08	2020-06-10	
□	2	Heater is not working properly.	Pending	2020-05-08	2020-05-22	
□	3	Broken lamp in the office.	Pending	2020-05-08	2020-05-24	

Figure 124 - Task manager window

Adding tasks:

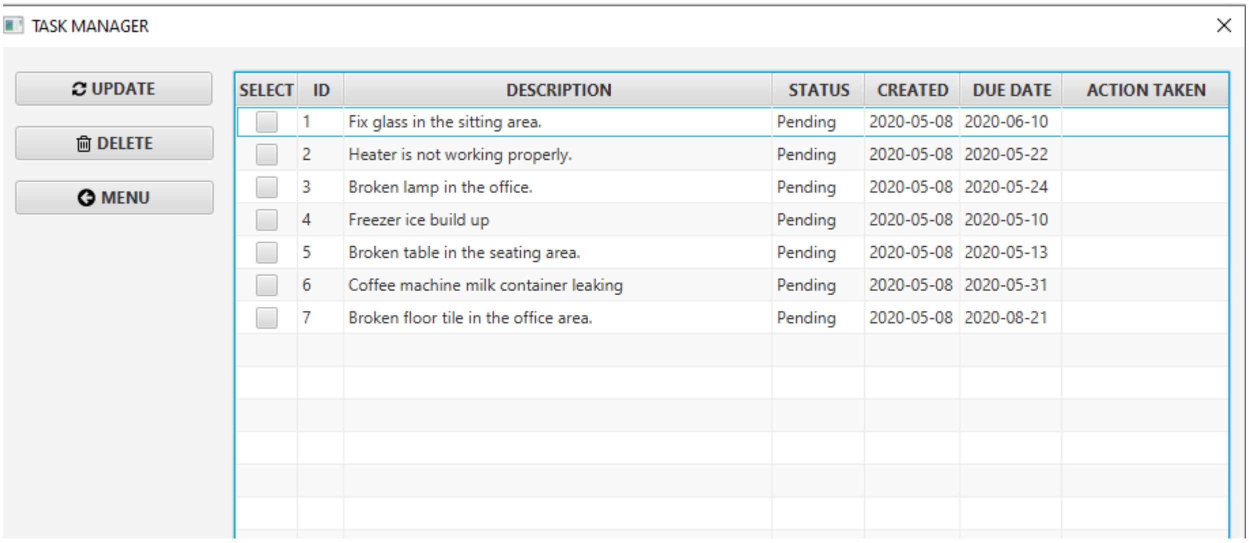


Describe item: Freezer ice build up

Due Date 10/05/2020

Figure 125 - Add task window

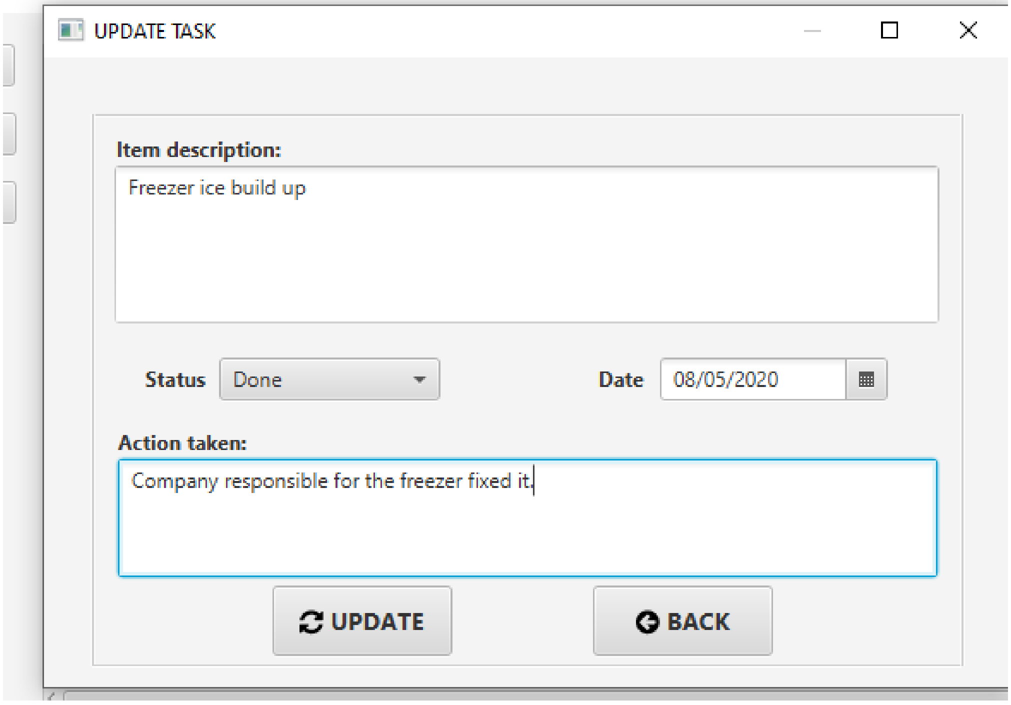
Task manager with 4 new tasks:



SELECT	ID	DESCRIPTION	STATUS	CREATED	DUE DATE	ACTION TAKEN
□	1	Fix glass in the sitting area.	Pending	2020-05-08	2020-06-10	
□	2	Heater is not working properly.	Pending	2020-05-08	2020-05-22	
□	3	Broken lamp in the office.	Pending	2020-05-08	2020-05-24	
□	4	Freezer ice build up	Pending	2020-05-08	2020-05-10	
□	5	Broken table in the seating area.	Pending	2020-05-08	2020-05-13	
□	6	Coffee machine milk container leaking	Pending	2020-05-08	2020-05-31	
□	7	Broken floor tile in the office area.	Pending	2020-05-08	2020-08-21	

Figure 126 - Task manager window

Updating tasks:



Item description: Freezer ice build up

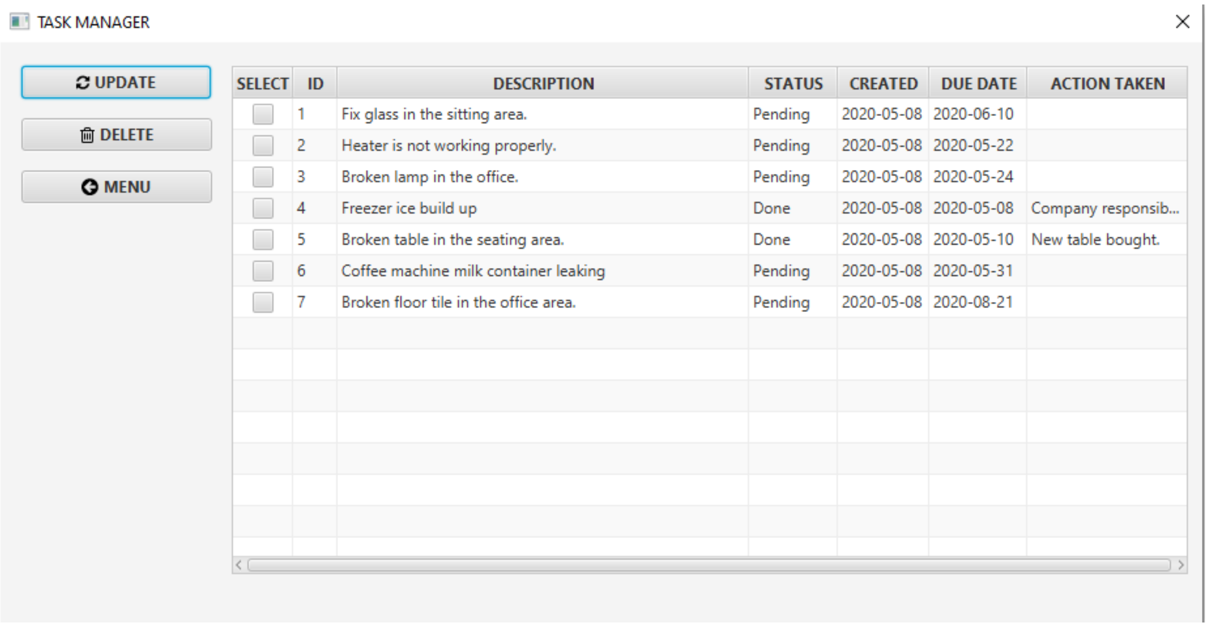
Status: Done

Date: 08/05/2020

Action taken: Company responsible for the freezer fixed it.

Figure 127 - Update task window

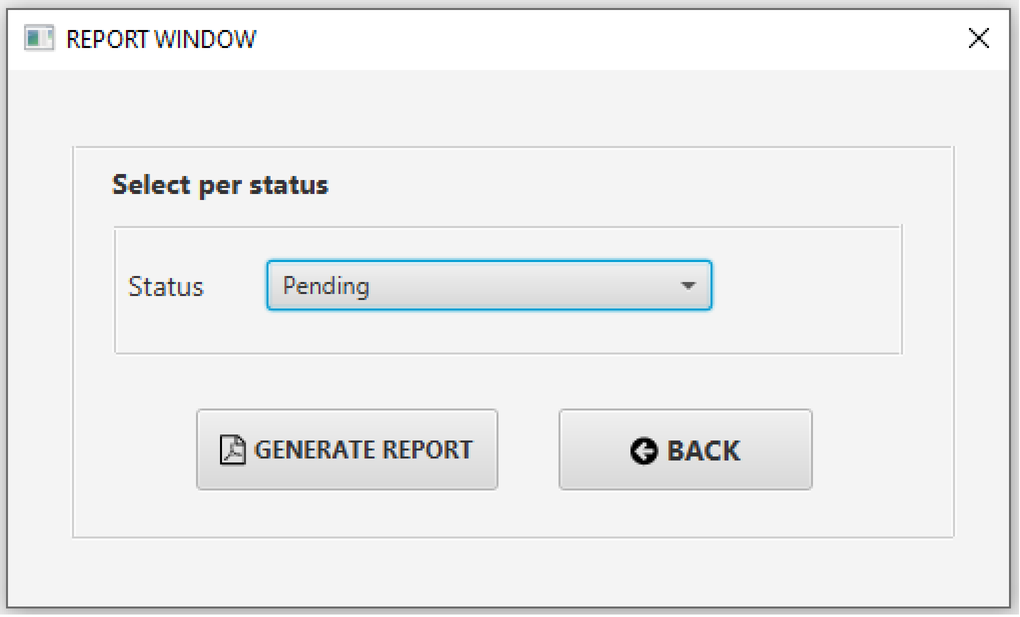
Updated tasks table:



SELECT	ID	DESCRIPTION	STATUS	CREATED	DUE DATE	ACTION TAKEN
□	1	Fix glass in the sitting area.	Pending	2020-05-08	2020-06-10	
□	2	Heater is not working properly.	Pending	2020-05-08	2020-05-22	
□	3	Broken lamp in the office.	Pending	2020-05-08	2020-05-24	
□	4	Freezer ice build up	Done	2020-05-08	2020-05-08	Company responsib...
□	5	Broken table in the seating area.	Done	2020-05-08	2020-05-10	New table bought.
□	6	Coffee machine milk container leaking	Pending	2020-05-08	2020-05-31	
□	7	Broken floor tile in the office area.	Pending	2020-05-08	2020-08-21	

Figure 128 - Update task window

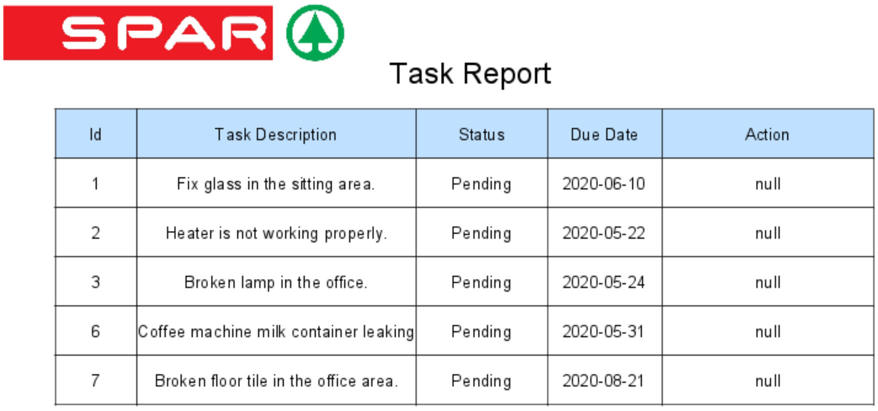
Report Generator:



Status: Pending

Figure 129 - Task report generator

Pending Report:

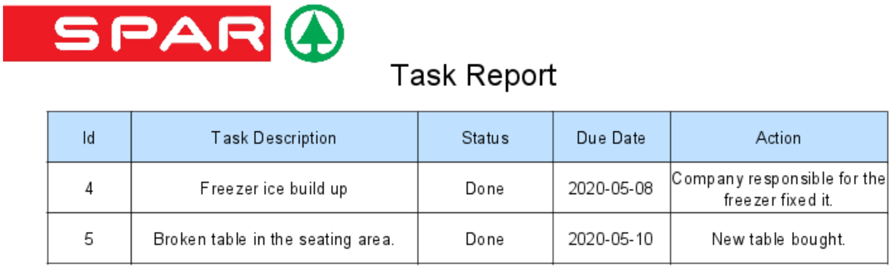


Task Report

Id	Task Description	Status	Due Date	Action
1	Fix glass in the sitting area.	Pending	2020-06-10	null
2	Heater is not working properly.	Pending	2020-05-22	null
3	Broken lamp in the office.	Pending	2020-05-24	null
6	Coffee machine milk container leaking	Pending	2020-05-31	null
7	Broken floor tile in the office area.	Pending	2020-08-21	null

Figure 130 - Task report - pending

Done Report:

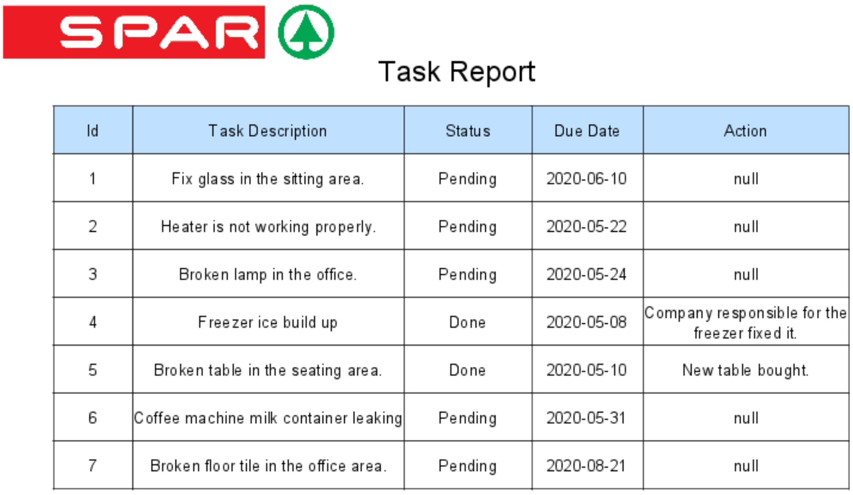


Task Report

Id	Task Description	Status	Due Date	Action
4	Freezer ice build up	Done	2020-05-08	Company responsible for the freezer fixed it.
5	Broken table in the seating area.	Done	2020-05-10	New table bought.

Figure 131 - Task report - done tasks

All Report:



Task Report

Id	Task Description	Status	Due Date	Action
1	Fix glass in the sitting area.	Pending	2020-06-10	null
2	Heater is not working properly.	Pending	2020-05-22	null
3	Broken lamp in the office.	Pending	2020-05-24	null
4	Freezer ice build up	Done	2020-05-08	Company responsible for the freezer fixed it.
5	Broken table in the seating area.	Done	2020-05-10	New table bought.
6	Coffee machine milk container leaking	Pending	2020-05-31	null
7	Broken floor tile in the office area.	Pending	2020-08-21	null

Figure 132 - Task report - Pending and done

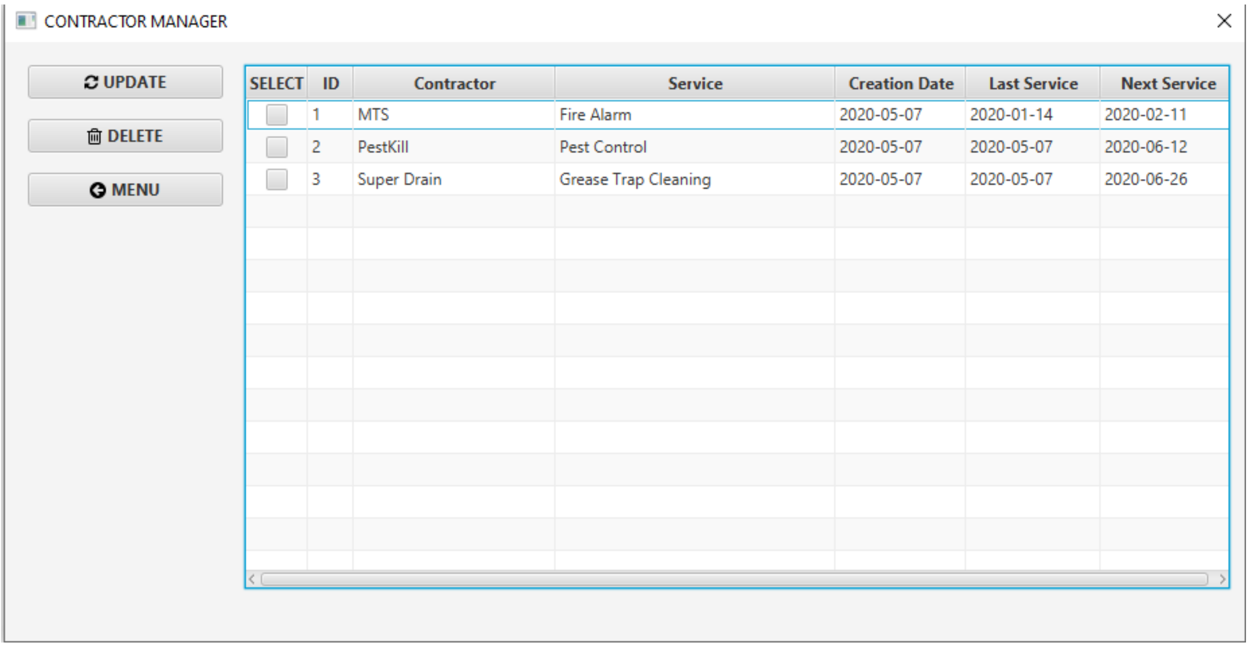
4th Test Script:

The last test script will test the Maintenance Tab:

- Add 2 new contractors to the system one with an expired date and one with regular date.
- Close the program and check if it gives any warning when you restart it.
- Update an expired contractor service to a newer date.
- Print 3 reports: Expired, Done and All.

Results:

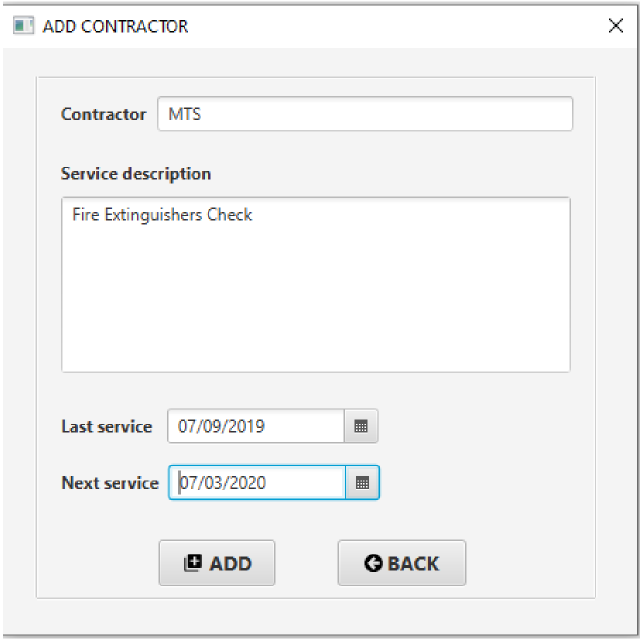
Contractor's Manager without new entries:



SELECT	ID	Contractor	Service	Creation Date	Last Service	Next Service
☐	1	MTS	Fire Alarm	2020-05-07	2020-01-14	2020-02-11
☐	2	PestKill	Pest Control	2020-05-07	2020-05-07	2020-06-12
☐	3	Super Drain	Grease Trap Cleaning	2020-05-07	2020-05-07	2020-06-26

Figure 133 - Maintenance manager

Adding an expired maintenance:



Contractor: MTS

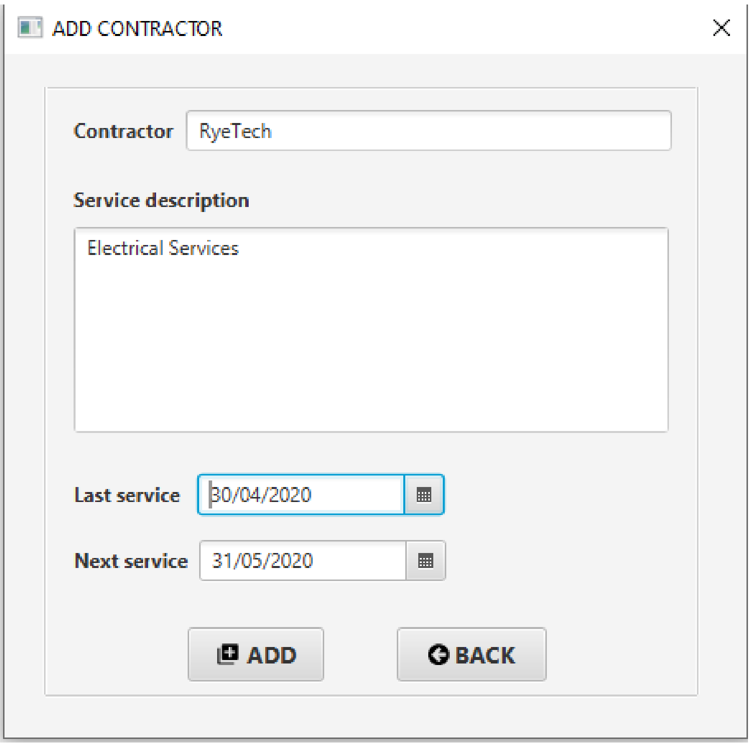
Service description: Fire Extinguishers Check

Last service: 07/09/2019

Next service: 07/03/2020

Figure 134 - Add contractor window

Adding a regular maintenance:



Contractor: RyeTech

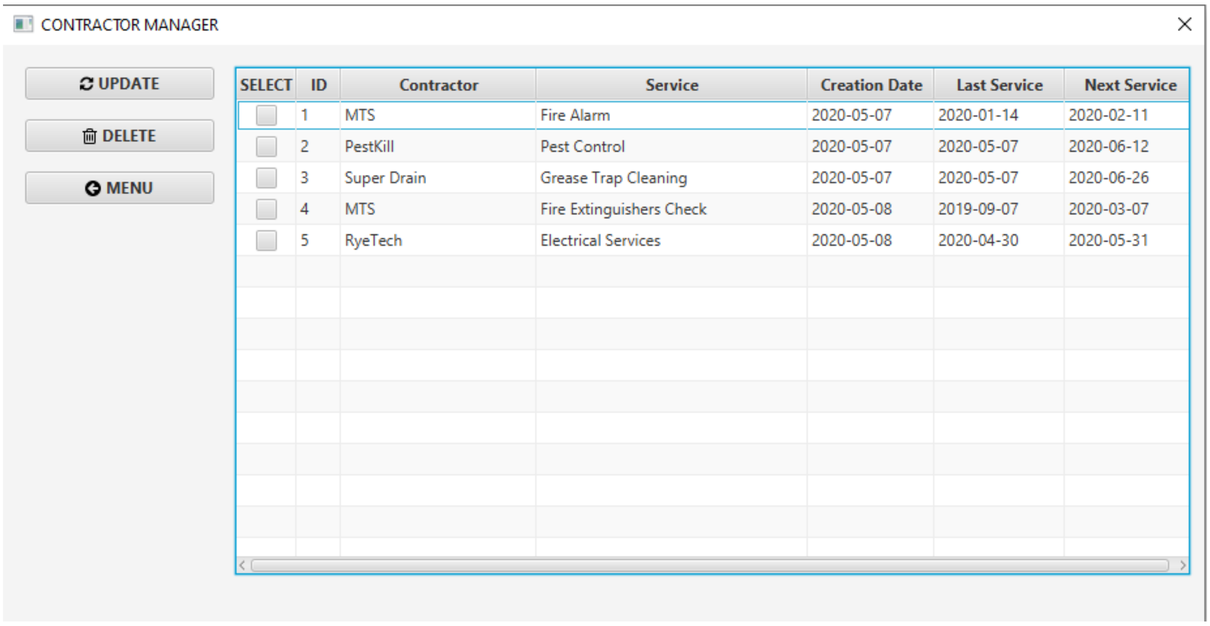
Service description: Electrical Services

Last service: 30/04/2020

Next service: 31/05/2020

Figure 135 - Adding contractor

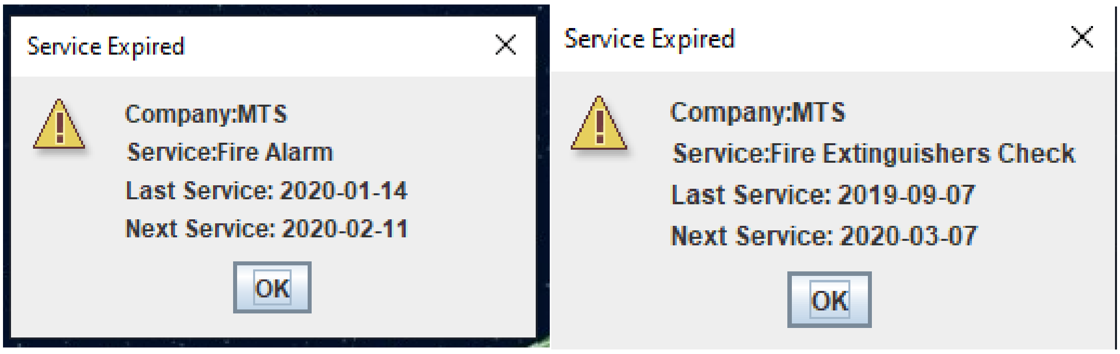
Checking the table with the new info:



SELECT	ID	Contractor	Service	Creation Date	Last Service	Next Service
□	1	MTS	Fire Alarm	2020-05-07	2020-01-14	2020-02-11
□	2	PestKill	Pest Control	2020-05-07	2020-05-07	2020-06-12
□	3	Super Drain	Grease Trap Cleaning	2020-05-07	2020-05-07	2020-06-26
□	4	MTS	Fire Extinguishers Check	2020-05-08	2019-09-07	2020-03-07
□	5	RyeTech	Electrical Services	2020-05-08	2020-04-30	2020-05-31

Figure 136 - Contractor manager window

Closed the program and restarted it, we receive a warning that those companies services are expired:


Service Expired

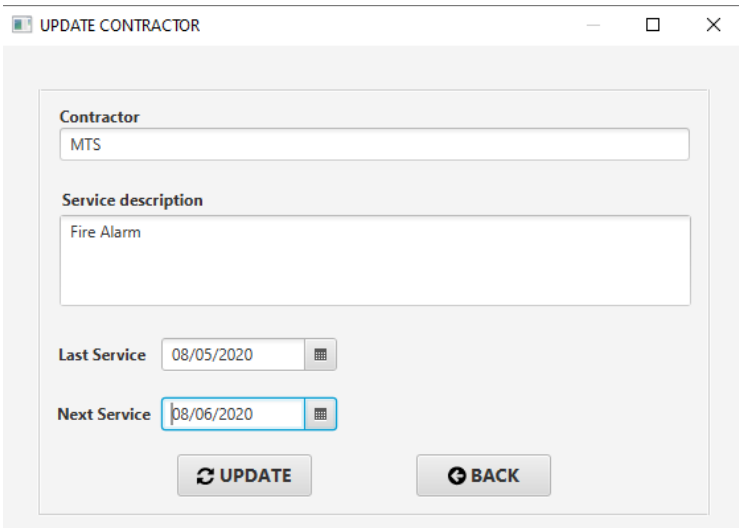
Company:MTS
Service:Fire Alarm
Last Service: 2020-01-14
Next Service: 2020-02-11

Service Expired

Company:MTS
Service:Fire Extinguishers Check
Last Service: 2019-09-07
Next Service: 2020-03-07

Figure 137 - Service expired warning

Updating one of the expired services:



Contractor: MTS

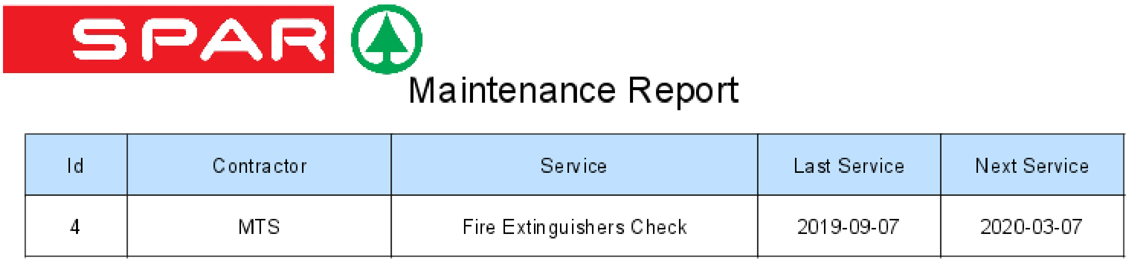
Service description: Fire Alarm

Last Service: 08/05/2020

Next Service: 08/06/2020

Figure 138 - Update contractors window

Expired Report:

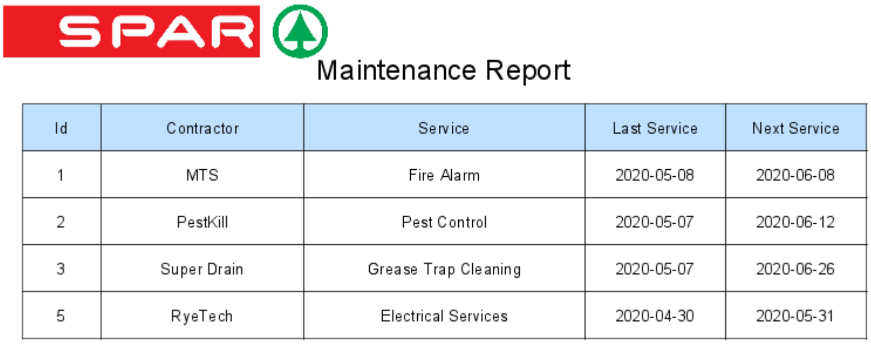


Maintenance Report

Id	Contractor	Service	Last Service	Next Service
4	MTS	Fire Extinguishers Check	2019-09-07	2020-03-07

Figure 139 - Expired services

Done Report:

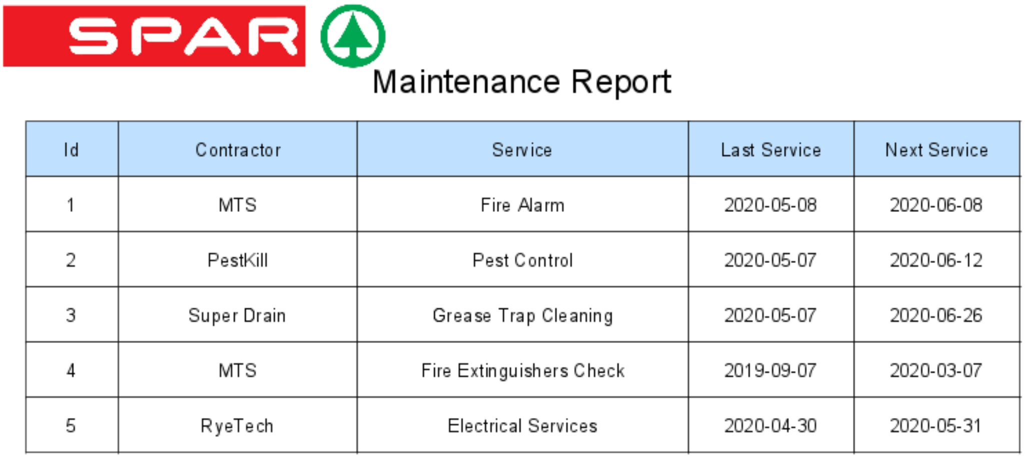


Maintenance Report

Id	Contractor	Service	Last Service	Next Service
1	MTS	Fire Alarm	2020-05-08	2020-06-08
2	PestKill	Pest Control	2020-05-07	2020-06-12
3	Super Drain	Grease Trap Cleaning	2020-05-07	2020-06-26
5	RyeTech	Electrical Services	2020-04-30	2020-05-31

Figure 140 - Services done

All Report:



Maintenance Report

Id	Contractor	Service	Last Service	Next Service
1	MTS	Fire Alarm	2020-05-08	2020-06-08
2	PestKill	Pest Control	2020-05-07	2020-06-12
3	Super Drain	Grease Trap Cleaning	2020-05-07	2020-06-26
4	MTS	Fire Extinguishers Check	2019-09-07	2020-03-07
5	RyeTech	Electrical Services	2020-04-30	2020-05-31

Figure 141 - All contractors report

5.1 User Testing

After our testing was done, we took our program to the shop so it could be tested by the management team, which constitutes from 2 supervisors, the shop manager and the owner, as they are the main users of the program. Our test subjects were:

- Camila Oliveira – Supervisor.
- Siti Bakri – Manager.
- Alan Kealy – Shop Owner.

Three test scripts were created for them to follow through, we also measured the time they took to perform the tasks and how successful it was. Before the tests began, we gave them a brief explanation on how the program works and its functionalities. After that we performed the testing.

5.1.0 Test Scripts

1st Test Script:

The first test script is to add a new employee to the system, create a new training, check if the training is added to the employee and print a report of which training needs to be done.

- Enter the login: admin@admin.com password: admin.
- Go to the administrator tab.
- Click Add Employee Button.
- Enter employee info.
- Click register.
- Go to the training tab.
- Click Add training.
- Select a training company and add new training.
- Click Employee/Training.
- Select the new employee, click enter, check if the training is there. Click Exit.
- Go to the reports tab and select training report.
- Click generate report.

2nd Test Script:

The second test script is to add a new contractor, add the last date that the maintenance was done and add the next date. After that print the report for the all contractors/maintenance.

- Click in the Maintenance tab.
- Add Contractor.
- Enter the contractor info, when the last service was done and when the next one is due.
- Click add, and go to the Contractor Manager tab.
- Update the dates for the other Contractor in the list and close the tab.
- Go to the Reports tab, click in Maintenance record and Generate Report.

3rd Test Script:

The third script is to add 4 new tasks, check the manager, mark a couple as done and then generate the report.

- Click in the tasks tab.
- Add 4 new task, insert info of the new task and when is the due date for those tasks.
- Go to the Task Manager and update 2 of the pending tasks to done and change the date to when it was done.
- Exit and go to the reports tab.
- Click task reports and generate report.

5.1.1 Questionnaire:

1 – What did you think of the user interface?

2 – What did you think of the functionalities?

3 – Did you feel any difficulties to navigate the program?

4 – Do you think this program would help in the management routine of the shop?

5 – What do you think could be changed and/or improved?

5.1.2 Results:

Camila Oliveira:

1st Script: 4 min.

2nd Script: 2 min 30sec.

3rd Script: 3 min 20 sec.

A1: It was very objective and easy to navigate. It is very organised and I didn't get lost when looking for the tabs and tasks.

A2: It works well as everything you need for the managing of H&S tasks is there and it is easy to use.

A3: No, not really.

A4: Yes, it would make things easier and save a lot of time as you have all the information in one place instead of a bunch of many folders.

A5: As I am a supervisor on the shop and I don't deal with health and safety procedures management I don't know if there's anything else that needs to be added or made better, the only thing that I thought it should be different is how you select the date of birth of the employee. It only let you select it, not write it. Besides that, I thought everything was good.

Observation: A much quicker explanation was given to her as she does not deal with the H&S management, she does know how it works in general but not how it is organized.

Alan Kealy:

1st Script: 3 min.

2nd Script: 1 min 10 sec.

3rd Script: 2 min 45 sec.

A1: A: Easy to navigate.

A2: Detailed and efficient.

A3: No.

A4: Yes.

A5: Some of the icons and tabs could be colour coded. Images could be uploaded to the tasks too.

Observations: The tasks were done very quickly, but in the 3rd script as I checked later the two tasks that were supposed to be changed from pending to done were not still as pending even though the action was taken, that was a user mistake as the user needs to change it manually. That is something that could be changed later in the prototype.

Siti Bakri:

1st: 4 min 40 sec. //Hiccup on the employee adding.

2nd: 1 min 15 sec.

3rd: 2 min 45 sec.

A1: Easy to see.

A2: Applicable to our environment/job.

A3: No, just needed a few minutes to understand how to use it.

A4: Yes.

A5: Color coded on the interface and alerts for a few of the functionalities.

Observations: She had a bit of a delay on the first script as something happened during the adding of the employee, when she checked for the employee with the training it was not there. I looked at the system console and nothing was sent to the database, so it was possibly that the register but was not clicked and both of us didn't see it. This issue has not happened again in other tests.

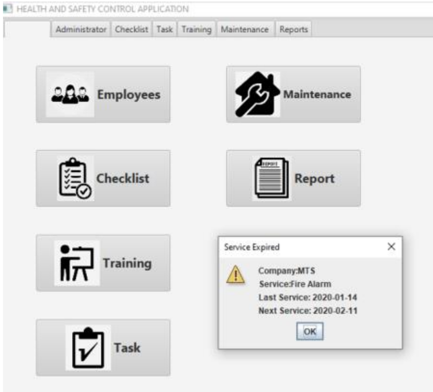
5.2 Conclusion of Usability

In general, we had good feedback from the managing team of the shop and some great observations about the program. The time for the users to realise the tasks were very close to each other, none of them had any considerable difference if we take in consideration that two of the subjects (Alan and Camila) are not responsible for the frequent management of the documentation.

None of them had difficulties when navigating the program and they all thought it was easy to understand how to use it. Camila pointed out to us that the date of birth selection when registering an employee cannot be written, you have to use the little selection menu, it was also something that bothered the other two testers, we took it into consideration and will apply changes to it.

The lack of colour coding of the program was also an issue to the testers as all icons look very similar and there are only a few colours being used in the system. Alan also asked about inserting images to the tasks, for example if a pipe is broken, we could upload a picture of the pipe to that respective task in the system. We thought about it in the designing phase of the program but we did not apply it as we were not sure where this could be used at the time, but it is a great idea and could be used in the future.

Alerts were missing in the functionalities was something one of the testers observed. We were using an older version of the system for the testing where we had not implemented a few extra functionalities and corrected a few bugs. Siti asked us about adding alerts to the maintenance if they are expired, we already had implemented it in a newer version which does work correctly as it can be seen in the picture below.



Company:MTS
Service:Fire Alarm
Last Service: 2020-01-14
Next Service: 2020-02-11

Figure 142 - Main menu with a service expired warning

5.3 Commercialisation/Marketing

As the project is already being done for a specific company, we would first try to test the usability of the prototype and how it can be enhanced for a better experience. After that try to scale to the owner’s other shop locations which have different environments and try to adapt the program there. If the project proves successful, we will then expand and commercialise the software to franchises across Dublin and Ireland.

If everything runs smoothly, we will direct our marketing to small retail companies in Ireland. A lot of the owners of those companies know each other and could recommend us by word of mouth. During this time, we would build a website to expand our name and put our idea out there and try to register it for precaution, as it is a simple to create but useful concept.

Every shop could have it is own customised program where they can manage their H&S documentation in different ways.

Chapter 6 - Conclusion

Knowing how complicated and hard to manage various types of documentation on Health and Safety can be for a small company, our project showed that our idea of a management software for that problem can be the solution. Creating a one stop shop for all that documentation proves to be much easier than having to run every day through an absurd amount of paper documents spread throughout the business. Even though it does not remove completely the paper documentation as they are a necessity for legal reasons, it did show how it can make the management of all those papers much more bearable and less complicated.

Our prototype showed how this could be done through a series of menus where you can manage trainings for the employees, check if all maintenances routines are in order, control what needs to be fixed around the shop and see which checklist runs need to be done during your week. All of this information is translated into easy to read and customisable reports for the management team to take decisions on what needs priority, what needs to be fixed, issues that need to be changed and even remember things that could have been forgotten in midst of the daily chaos that is a small retail business.

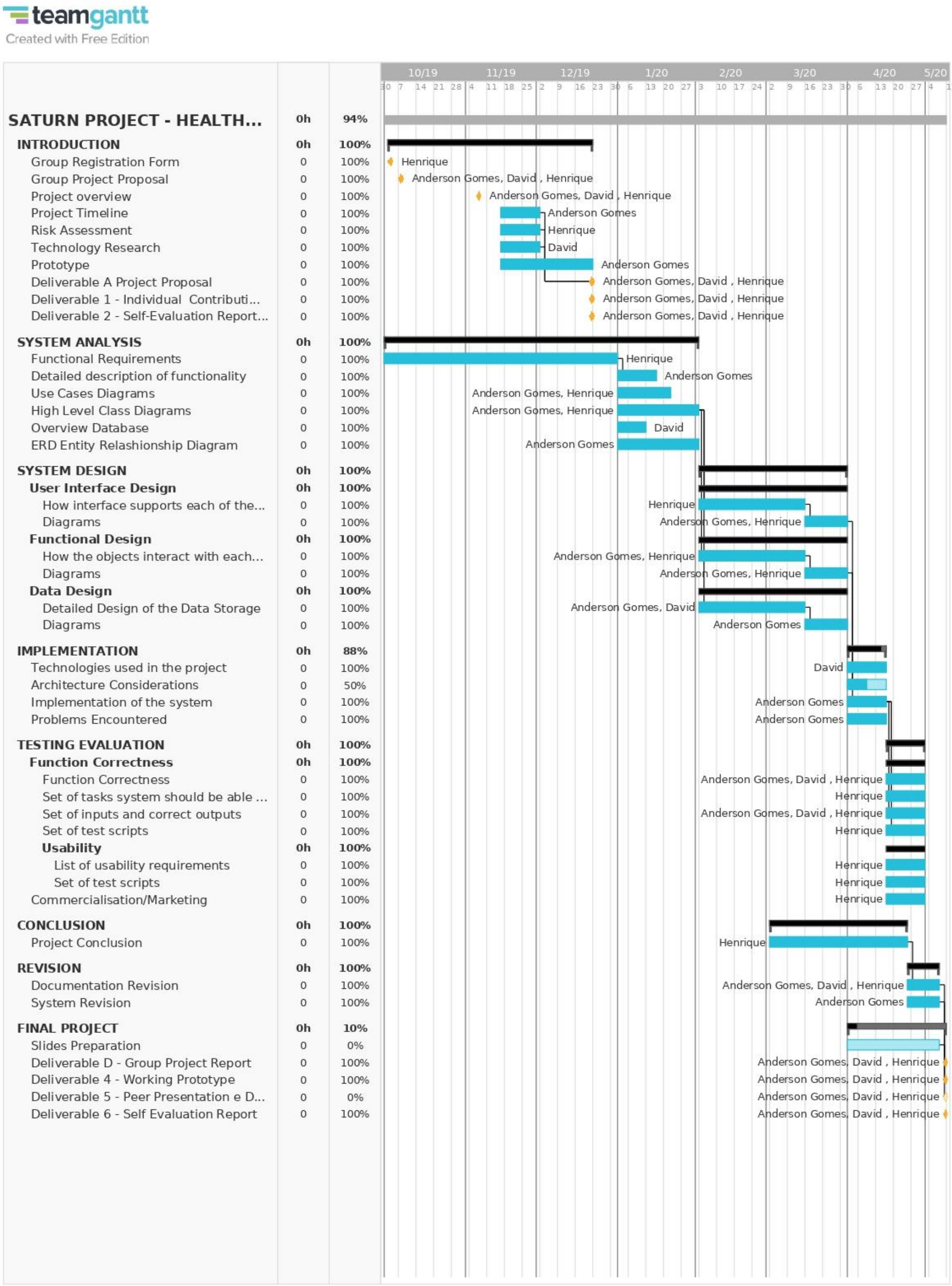
Testing our prototype with the management staff proved to be a success as we showed them how our software would work and how it could help. We were received with great feedback from the staff and the owner of the shop which proved that the project can really be used by the management team to improve the performance of the shop and help with these tasks.

A few things could not be implemented during the project as we focused more on the system main functionalities which involved mostly the management of the documentation and a friendly user interface that does not make the user work harder. These functionalities could be implemented later as we develop a full working software. Some of these were ideas that came up during the development of the project, feedback from our supervisor and even ideas from the shop management team:

- We would like to implement backup of the data straight to a cloud service. This was in the plans from the beginning, but we changed between being sent via email or hosted by a cloud service. In the end it was scratched off completely.
- Upload of pictures for a few of the functionalities.

- Transform it into a web service so it is not only locally accessible and could also make it easier to scale up.
- Create a better user interface, with colour codes and more customisable functionalities.

Appendix A – Project Planning



teamgantt
Created with Free Edition

SATURN PROJECT - HEALTH...	**0h**	**94%**
INTRODUCTION	**0h**	**100%**
Group Registration Form	0	100%
Group Project Proposal	0	100%
Project overview	0	100%
Project Timeline	0	100%
Risk Assessment	0	100%
Technology Research	0	100%
Prototype	0	100%
Deliverable A Project Proposal	0	100%
Deliverable 1 - Individual Contributi...	0	100%
Deliverable 2 - Self-Evaluation Report...	0	100%
SYSTEM ANALYSIS	**0h**	**100%**
Functional Requirements	0	100%
Detailed description of functionality	0	100%
Use Cases Diagrams	0	100%
High Level Class Diagrams	0	100%
Overview Database	0	100%
ERD Entity Relashionship Diagram	0	100%
SYSTEM DESIGN	**0h**	**100%**
User Interface Design	**0h**	**100%**
How interface supports each of the...	0	100%
Diagrams	0	100%
Functional Design	**0h**	**100%**
How the objects interact with each...	0	100%
Diagrams	0	100%
Data Design	**0h**	**100%**
Detailed Design of the Data Storage	0	100%
Diagrams	0	100%
IMPLEMENTATION	**0h**	**88%**
Technologies used in the project	0	100%
Architecture Considerations	0	50%
Implementation of the system	0	100%
Problems Encountered	0	100%
TESTING EVALUATION	**0h**	**100%**
Function Correctness	**0h**	**100%**
Function Correctness	0	100%
Set of tasks system should be able ...	0	100%
Set of inputs and correct outputs	0	100%
Set of test scripts	0	100%
Usability	**0h**	**100%**
List of usability requirements	0	100%
Set of test scripts	0	100%
Commercialisation/Marketing	0	100%
CONCLUSION	**0h**	**100%**
Project Conclusion	0	100%
REVISION	**0h**	**100%**
Documentation Revision	0	100%
System Revision	0	100%
FINAL PROJECT	**0h**	**10%**
Slides Preparation	0	0%
Deliverable D - Group Project Report	0	100%
Deliverable 4 - Working Prototype	0	100%
Deliverable 5 - Peer Presentation e D...	0	0%
Deliverable 6 - Self Evaluation Report	0	100%

For most of our project we used the project planning above to monitor how was our progress and if we were in schedule to finish all of our deliverables. We did have a few hiccups along the way, especially with the COVID-19 situation where everyone was affected as we could not meet in person to discuss about the project, work and college schedules changed making it even harder. It was a bit hard to even talk online as our schedules were all changed, and we were overloaded with assignments to compensate for the tests.

Most of these problems were assessed in our risk assessment table which we managed to solve all of them, but a pandemic was not one of them. Even with all of these we were still on schedule and did not have to re-plan our project or change any timelines. Most, if not all, of our deadlines were met in time.

Appendix B – Group Journal

We first started our project idea in the beginning of 2019 with an IT Deliveroo/Uber kind of application in which people could call for individual people to provide IT support in their own homes. We were not very happy with that idea as we started the second semester of 2019 and we needed a good project idea that we could all agree on.

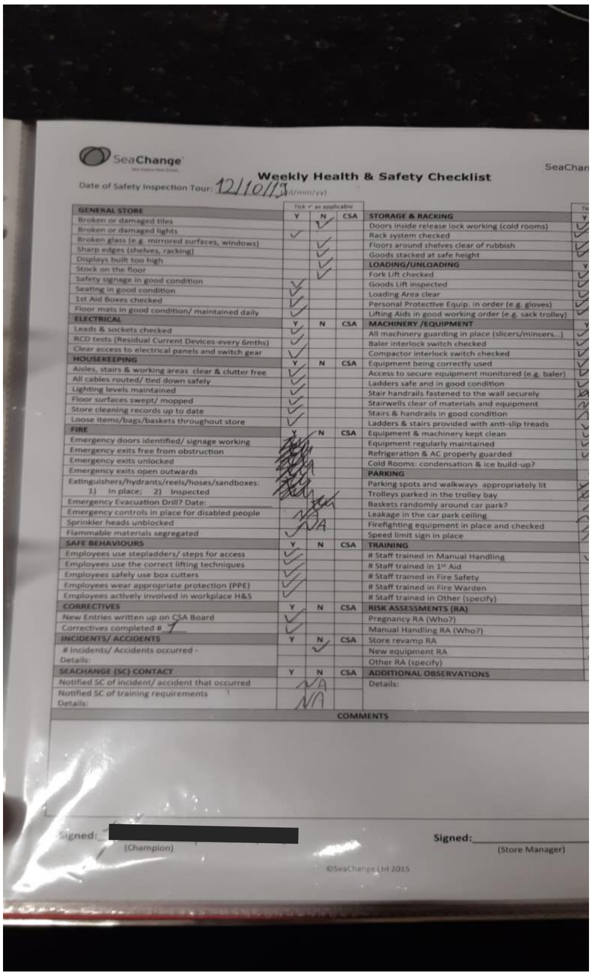
Henrique came back with the idea after having a meeting at work with a representant from a health and safety advisor company, in which was brought up all types of documentation and tasks that need to be followed by the company to be compliant with the law. As the days passed, he realized that managing all those documents and tasks was much harder than it looked like, it had many redundancies and a lot of paperwork. The idea was brought to the group to create a type of task manager for the all those H&S procedures in which it would help the efficiency of the management team in the shop.

We had the idea but did not know where to start, so after our project idea was reviewed and approved and we were assigned our supervisor (Mark Morrissey), he helped us get on track on what to do first, what we needed to prioritize and how our project would be different from any other calendar, task managers and similar ideas in the market.

Researching about H&S procedures in retail and it's difficulties was our first step to understand how our application could help the company be more efficient and what the functionalities of our app would be. When we talked to our supervisor, he advised us on choosing the technologies we would build the app on and how to divide responsibilities within our group.

The responsibilities were divided according to our skills and what we were more comfortable with. Anderson became our lead programmer, David was responsible for designing the database and Henrique was responsible for the design of the program and its functionalities and the contact with the company. All of us started contributing for the documentation and research so we could finish the first semester with everything ready on time.

As the second semester started, documents were brought from the company so we could base our design and functionalities on, they can be seen down below:



SeaChange

Weekly Health & Safety Checklist

SeaChan

Date of Safety Inspection Tour: 12/10/19 (dd/mm/yy)

Tick ✓ as applicable

GENERAL STORE	Y	N	CSA
Broken or damaged tiles		✓	
Broken or damaged lights	✓		
Broken glass (e.g. mirrored surfaces, windows)		✓	
Sharp edges (shelves, racking)		✓	
Displays built too high		✓	
Stock on the floor		✓	
Safety signage in good condition	✓		
Seating in good condition	✓		
1st Aid Boxes checked	✓		
Floor mats in good condition/ maintained daily	✓		
ELECTRICAL	**Y**	**N**	**CSA**
Leads & sockets checked	✓		
RCD tests (Residual Current Devices-every 6mths)	✓		
Clear access to electrical panels and switch gear	✓		
HOUSEKEEPING	**Y**	**N**	**CSA**
Aisles, stairs & working areas clear & clutter free	✓		
All cables routed/ tied down safely	✓		
Lighting levels maintained	✓		
Floor surfaces swept/ mopped	✓		
Store cleaning records up to date	✓		
Loose items/bags/baskets throughout store	✓		
FIRE	**Y**	**N**	**CSA**
Emergency doors identified/ signage working	[illegible]		
Emergency exits free from obstruction	[illegible]		
Emergency exits unlocked	[illegible]		
Emergency exits open outwards	[illegible]		
Extinguishers/hydrants/reels/hoses/sandboxes: 1) in place; 2) Inspected	[illegible]		
Emergency Evacuation Drill? Date:		[illegible]	
Emergency controls in place for disabled people	NA		
Sprinkler heads unblocked	NA		
Flammable materials segregated	✓		
SAFE BEHAVIOURS	**Y**	**N**	**CSA**
Employees use stepladders/ steps for access	✓		
Employees use the correct lifting techniques	✓		
Employees safely use box cutters	✓		
Employees wear appropriate protection (PPE)	✓		
Employees actively involved in workplace H&S	✓		
CORRECTIVES	**Y**	**N**	**CSA**
New Entries written up on CSA Board	✓		
Correctives completed # 1	✓		
INCIDENTS/ ACCIDENTS	**Y**	**N**	**CSA**
# Incidents/ Accidents occurred - Details:		✓	
SEACHANGE (SC) CONTACT	**Y**	**N**	**CSA**
Notified SC of incident/ accident that occurred	NA		
Notified SC of training requirements Details:	NA		

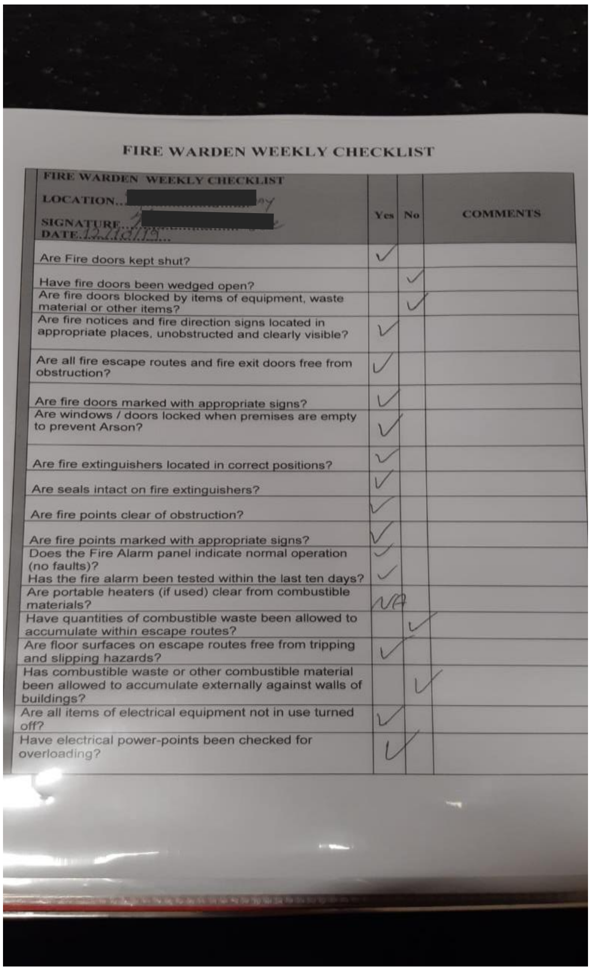
STORAGE & RACKING	Y
Doors inside release lock working (cold rooms)	✓
Rack system checked	✓
Floors around shelves clear of rubbish	✓
Goods stacked at safe height	✓
LOADING/UNLOADING	**Y**
Fork Lift checked	✓
Goods Lift inspected	✓
Loading Area clear	✓
Personal Protective Equip. in order (e.g. gloves)	✓
Lifting Aids in good working order (e.g. sack trolley)	✓
MACHINERY /EQUIPMENT	**Y**
All machinery guarding in place (slicers/mincers...)	✓
Baler interlock switch checked	✓
Compactor interlock switch checked	✓
Equipment being correctly used	✓
Access to secure equipment monitored (e.g. baler)	✓
Ladders safe and in good condition	✓
Stair handrails fastened to the wall securely	[illegible]
Stairwells clear of materials and equipment	[illegible]
Stairs & handrails in good condition	[illegible]
Ladders & stairs provided with anti-slip treads	✓
Equipment & machinery kept clean	✓
Equipment regularly maintained	✓
Refrigeration & AC properly guarded	✓
Cold Rooms: condensation & ice build-up?	
PARKING	
Parking spots and walkways appropriately lit	✓
Trolleys parked in the trolley bay	[illegible]
Baskets randomly around car park?	[illegible]
Leakage in the car park ceiling	[illegible]
Firefighting equipment in place and checked	[illegible]
Speed limit sign in place	[illegible]
TRAINING	
# Staff trained in Manual Handling	✓
# Staff trained in 1st Aid	
# Staff trained in Fire Safety	
# Staff trained in Fire Warden	
# Staff trained in Other (specify)	
RISK ASSESSMENTS (RA)	
Pregnancy RA (Who?)	
Manual Handling RA (Who?)	
Store revamp RA	
New equipment RA	
Other RA (specify)	
ADDITIONAL OBSERVATIONS	
Details:	

COMMENTS

Signed: [redacted]
(Champion)

Signed:__________
(Store Manager)

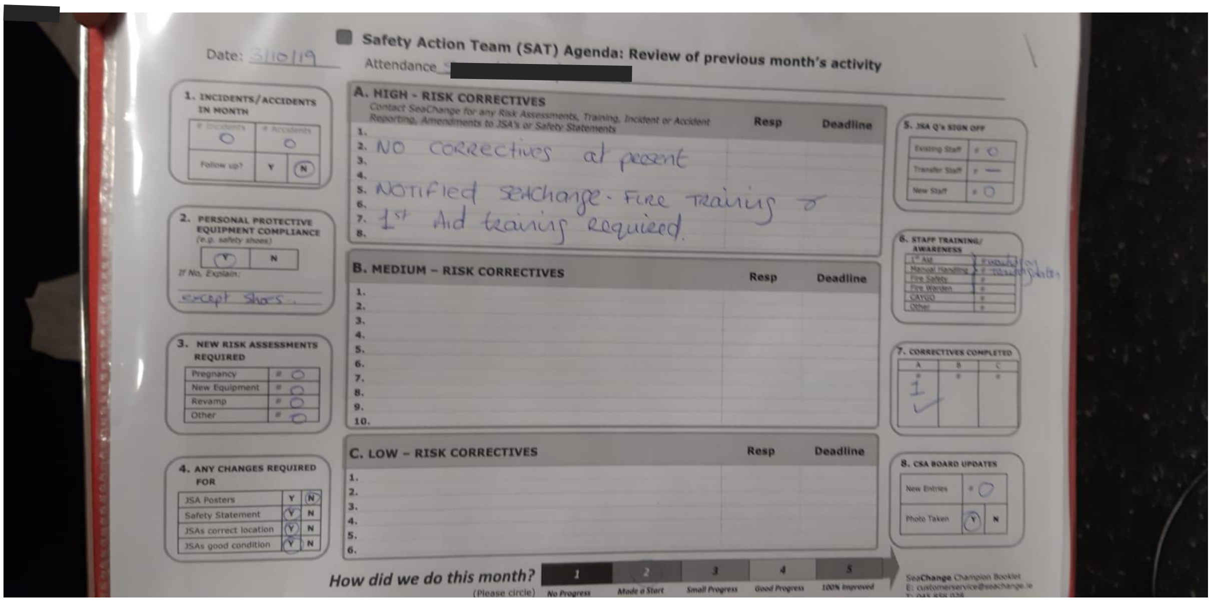
©SeaChange Ltd 2015



FIRE WARDEN WEEKLY CHECKLIST

FIRE WARDEN WEEKLY CHECKLIST LOCATION..[redacted] SIGNATURE..[redacted] DATE..12/10/19....	Yes	No	COMMENTS
Are Fire doors kept shut?	✓		
Have fire doors been wedged open?		✓	
Are fire doors blocked by items of equipment, waste material or other items?		✓	
Are fire notices and fire direction signs located in appropriate places, unobstructed and clearly visible?	✓		
Are all fire escape routes and fire exit doors free from obstruction?	✓		
Are fire doors marked with appropriate signs?	✓		
Are windows / doors locked when premises are empty to prevent Arson?	✓		
Are fire extinguishers located in correct positions?	✓		
Are seals intact on fire extinguishers?	✓		
Are fire points clear of obstruction?	✓		
Are fire points marked with appropriate signs?	✓		
Does the Fire Alarm panel indicate normal operation (no faults)?	✓		
Has the fire alarm been tested within the last ten days?	✓		
Are portable heaters (if used) clear from combustible materials?	NA		
Have quantities of combustible waste been allowed to accumulate within escape routes?		✓	
Are floor surfaces on escape routes free from tripping and slipping hazards?	✓		
Has combustible waste or other combustible material been allowed to accumulate externally against walls of buildings?		✓	
Are all items of electrical equipment not in use turned off?	✓		
Have electrical power-points been checked for overloading?	✓		

Figure 144 and 145 – Weekly Health and Safety Checklist and Weekly Fire Warden Checklist



Safety Action Team (SAT) Agenda: Review of previous month's activity

Date: 3/10/19

Attendance [redacted]

1. INCIDENTS/ACCIDENTS IN MONTH

# Incidents	# Accidents	
0	0	
Follow up?	Y	[✓ N]

2. PERSONAL PROTECTIVE EQUIPMENT COMPLIANCE (e.g. safety shoes)

[✓ Y] N

If No, Explain:

except shoes.

3. NEW RISK ASSESSMENTS REQUIRED

Pregnancy	# 0
New Equipment	# 0
Revamp	# 0
Other	# 0

4. ANY CHANGES REQUIRED FOR

JSA Posters	Y	[✓ N]
Safety Statement	[✓ Y]	N
JSAs correct location	[✓ Y]	N
JSAs good condition	[✓ Y]	N

A. HIGH - RISK CORRECTIVES

Contact SeaChange for any Risk Assessments, Training, Incident or Accident Reporting, Amendments to JSA's or Safety Statements

	Resp	Deadline
1.		
2. NO CORRECTIVES at present		
3.		
4.		
5. NOTIFIED SEAChange - FIRE TRAINING &		
6.		
7. 1st Aid training Required.		
8.		

B. MEDIUM – RISK CORRECTIVES

	Resp	Deadline
1.		
2.		
3.		
4.		
5.		
6.		
7.		
8.		
9.		
10.		

C. LOW – RISK CORRECTIVES

	Resp	Deadline
1.		
2.		
3.		
4.		
5.		
6.		

5. JSA Q's SIGN OFF

Existing Staff	# 0
Transfer Staff	# —
New Staff	# 0

6. STAFF TRAINING/ AWARENESS

1st Aid	#
Manual Handling	#
Fire Safety	#
Fire Warden	#
CAYGO	#
Other	#

[illegible]

7. CORRECTIVES COMPLETED

A	B	C
#	#	#
1 ✓		

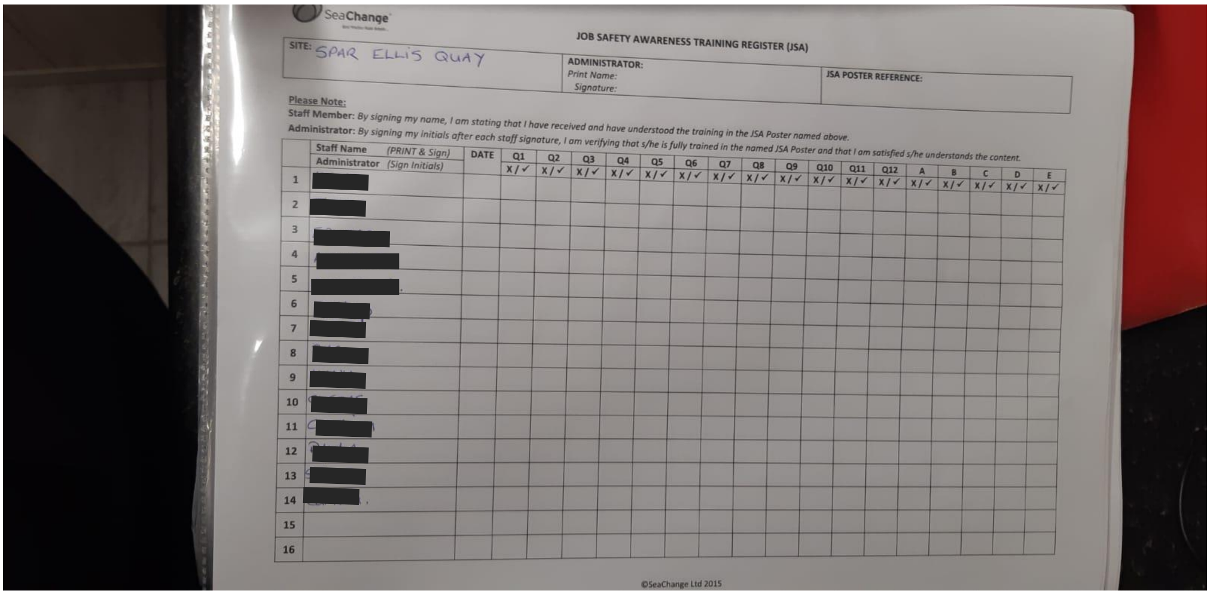
8. CSA BOARD UPDATES

New Entries	# 0	
Photo Taken	[✓ Y]	N

How did we do this month? (Please circle) 1 No Progress 2 Made a Start 3 Small Progress 4 Good Progress 5 100% Improved

SeaChange Champion Booklet
E: customerservice@seachange.ie
T: 045 858 028

Figure 146 – SAT Agenda



SeaChange

JOB SAFETY AWARENESS TRAINING REGISTER (JSA)

SITE: SPAR ELLIS QUAY

ADMINISTRATOR:
Print Name:
Signature:

JSA POSTER REFERENCE:

Please Note:
Staff Member: *By signing my name, I am stating that I have received and have understood the training in the JSA Poster named above.*
Administrator: *By signing my initials after each staff signature, I am verifying that s/he is fully trained in the named JSA Poster and that I am satisfied s/he understands the content.*

	Staff Name (PRINT & Sign) Administrator (Sign Initials)	DATE	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	A	B	C	D	E
			X / ✓	X / ✓	X / ✓	X / ✓	X / ✓	X / ✓	X / ✓	X / ✓	X / ✓	X / ✓	X / ✓	X / ✓	X / ✓	X / ✓	X / ✓	X / ✓	X / ✓
1	[redacted]																		
2	[redacted]																		
3	[redacted]																		
4	[redacted]																		
5	[redacted]																		
6	[redacted]																		
7	[redacted]																		
8	[redacted]																		
9	[redacted]																		
10	[redacted]																		
11	[redacted]																		
12	[redacted]																		
13	[redacted]																		
14	[redacted]																		
15																			
16																			

©SeaChange Ltd 2015

Figure 147 – Staff Training List

From these documents we started building our designs which can be seen on chapter 3. We already had the most functionalities ideas ready but it was hard for Anderson and David to understand them as they do not work in the company, so that helped understand how those procedures and documentations work to be able to implement them in a concise manner to the application.

We started working in our parts, from diagrams, designing classes, how the GUI would look like, trying different things to se if they worked until the first prototype skeleton of the project was finished which we all agreed on that was a good start to start implementing the functions and modify it to our taste.

Unfortunately that was around the time where the COVID-19 pandemic hit the entire world and made things a little bit more complicated for all of us, as we all are essential workers and we already had different schedules which were changed because of the situation. Also, with that we had a wave of assignments as all our exams were cancelled and we could not meet with each other. Even though the situation was not great it was a good learning experience on how to work remotely and still manage to finish our project. As this situation had already been assessed in our risk assessment part, we were prepared for these situations.

A few changes had to be done on some functionalities during the whole process, but the main design was intact, and we had to tweak very few things. Testing went well and we had good feedback from the Spar team, which made us feel very positive about our project.

We feel we accomplished all our main objectives and we could still improve the project and ourselves if we continue working on it. Creating a project in a team is a great experience on team building and a different experience than working only as an individual, it also taught us on how to adapt to the many roadblocks that show up during a project like that.

INDIVIDUAL CONTRIBUTION REPORT

Henrique Atanasio Wegner – ID: 2017403

I came with the project idea at work when I was in a meeting with the representant of a company (SeaChange) that advises our franchise on H&S procedures. They made me responsible for most of all the control and procedures of that shop. I then realised it was a LOT of paperwork and things to keep track of and it was very hard to make all staff compliant to all these procedures if even I had trouble doing it by myself.

The project idea came from this necessity of managing all of this and the shop at the same time. I gave a look on all documents that needed to be organises and came with a sketch for how the program would look/work like. I brought the idea to the team and we all agreed on it and thought it was a good idea as it could be used commercially.

I researched on how Health and Safety procedures effect a business in Ireland and discovered that it impacts a lot in many areas. It can influence on how much a company will pay in insurance, cause high risk in claims, accidents at work, increase workload and of course it can tarnish the reputation of a business if a company is not compliant to the rigorous procedures. Our application could reduce all of these in the business if done right.

I also did the risk management which is an essential part of any project in development. I came with most of the risks, but I also asked for the opinion of my teammates and they came back to me with a few other ones. Use case scenarios and diagrams were also my responsibility together with sequence diagrams and checklist diagrams.

When designing the program, I did the initial design on how it would look like and how the functionalities would work in it and during the implementation where things start to change, I helped with the redesign and changes that should be made to the program to make it work better and be more user friendly.

The testing of the program (input/output) was made by me, to check if everything was working properly and if anything needed to be changed. I also did the testing (including questionnaires, timing, etc) with my co-workers and employer to get some outside feedback from them and to check how the good the usability was. Some chapters and extra pieces were also done by me closer to the end of the project (abstract, conclusion, acknowledgments and some extra information added to the documentation).

The formatting of the final document was divided between me and Anderson, I organised where everything needed to be and he organised the formatting (font, size, references, etc).

INDIVIDUAL CONTRIBUTION REPORT

Anderson Oliveira Gomes – ID: 2017330

I researched the benefits of health and safety management and the disadvantages of not having it. I think it is very important to highlight how important it is for the company and why a system is needed. In all the sources I went through I only found benefits and based on my research I wrote the introduction.

Henrique had already written the ‘PROBLEM AREA’, but I thought it was a good to add a small introduction about the problem and how hard would be for a SPAR owner to keep H&S management. So, I added some information to it.

I did the timeline of the project. I used TeamGantt which is a really good tool for Gantt chart. All the tasks and milestones I got from Applied Technology Group Project Handbook, Moodle and our supervisor.

I drew all the class diagrams, flowcharts, and most of the state diagrams. They show a graphical representation of the many functions of the application. I researched on the database design and I drew the conceptual and logical models for our database.

I did all the front-end of the application. I started with a simple Java project adding a few external libraries, but later on I realized that we would need a lot of external libraries, so I changed the project to Maven which is project management tool that allows the use of dependencies. I added all the dependencies necessary for this project. I set up the GitHub repository linking it with my Eclipse IDE, and I used it as a storage for all the code related to the application. In total, we had 80 commits that show the growth of the application since the very beginning. I did not have the knowledge to work with JavaFX and Jasper Reports, so I researched and watched tutorials of how to implement and work with them.

I wrote the implementation of the system. I detailed how the system was implemented based on the system analysis and design. The frameworks, libraries, and tools that were necessary to build this application. I described the design patterns that we used to make the code more efficient. Also, the problems that we encountered building the application.

I worked with David who did the back-end of the application. I helped him to set up the database. In the process of developing the application, I realized that a few extra tables would be necessary to accommodate the application needs, then I added these tables in the database schema that David had already created.

I did part of the formatting. I researched on fonts, fonts sizes, indention, spacing, etc. I found some discrepancies among some authors and I used what I found common among them. Henrique organized the sections of the document where they were supposed to be.

INDIVIDUAL CONTRIBUTION REPORT

David Carvalho– ID: 2017368

I started researching about the technologies we are going to need to approach the application such as helpful platforms, IDE and java tools and features. I researched about what type of database to use which at first was thought to be a good idea using PostgreSQL before Anderson come to the great idea of using Hibernate tool so we could stick with MySQL which we are already familiar to.

I also created a user story mapping in an online collaborative platform where our group members could add new user stories organized by different sections and mark them as done as each new function was implemented.

I did some tests on the application pointing errors and bugs for trouble shooting and did the Software user manual containing all the steps to get start with the application, buttons and sections description and a walk through showing how to use all the functions the application provides.

Software User Manual

Retail Health & Safety Digital Management System

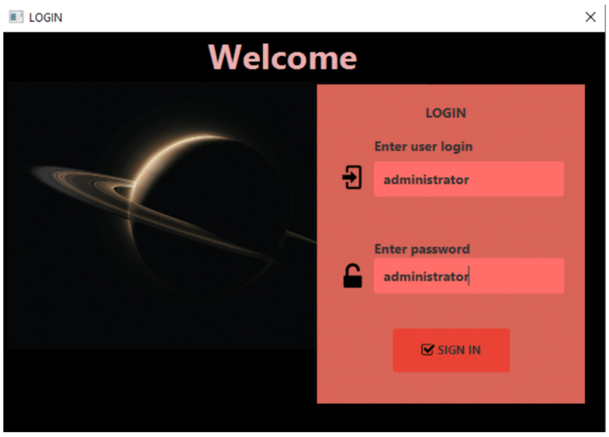
May 2020

TABLE OF CONTENTS

Software Overview

RH&S Management System is a software created to simplify and organize health and safety tasks and trainings procedures done by a retail shop, reducing paperwork considerably and mistakes made over lost documentation. RH&S projects main goal is to assist your business to conduct all Healthy and Safety procedures in an easier and more efficient manner.

Login

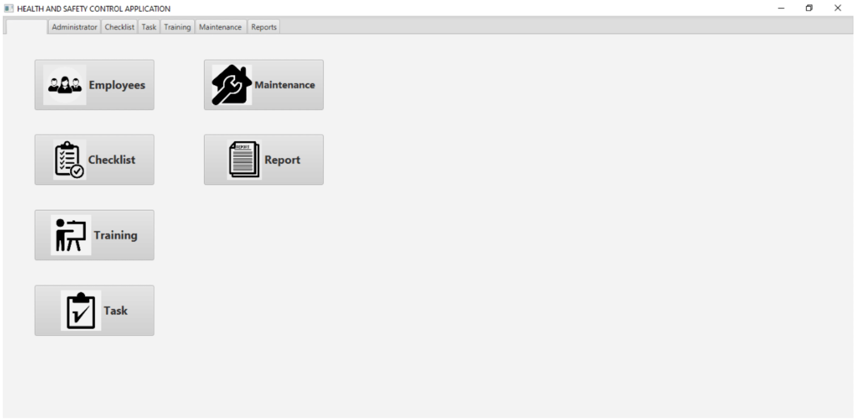


administrator

administrator

For your first access use the default credentials "administrator" as login ID and password (password can be changed later)

MAIN MENU



- **ADMINISTRATOR**
  ADD NEW EMPLOYEES, MANAGE EMPLOYEES AND CHANGE ADMINISTRATOR PASSWORD.

- **CHECKLIST**
  ADD NEW CHECKLIST, UPDATE CHECKLIST STATUS AND MANAGE CHECKLISTS.

- **TASK**
  ADD NEW TASKS AND MANAGE TASKS

- **TRAINING**
  ADD NEW TRAININGS, MANAGE TRAININGS AND EMPLOYEES PENDING/DONE TRAININGS VIEW

- **MAINTENANCE**
  ADD NEW CONTRACTORS AND MANAGE CONTRACTORS

- **REPORT**
  GENERATE REPORTS

MENU SHORTCUTS

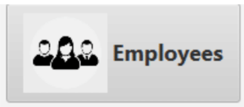


Figure 1 Employee manager window

Figure 2 Checklist manager window

Figure 3 View employee pending/done trainings window

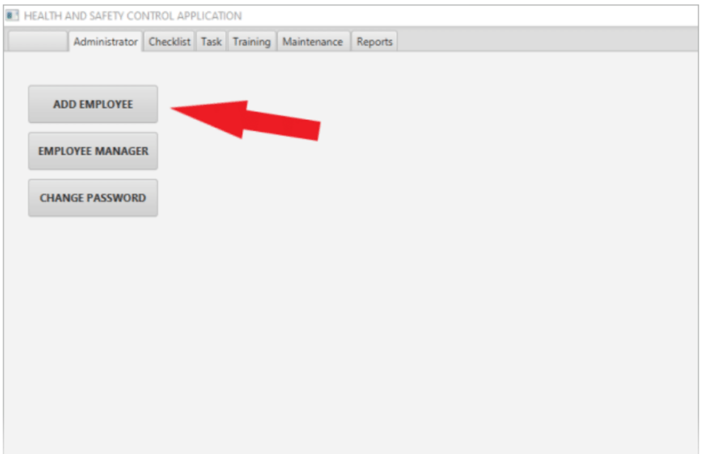
Figure 4 Task Manager window

Figure 5 Contractor manager window

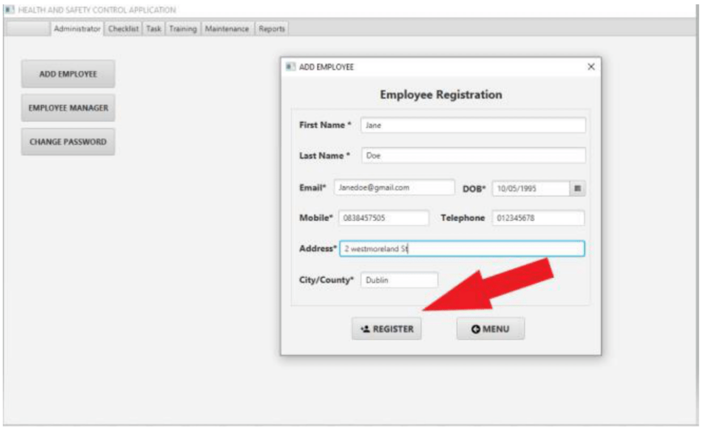
Figure 6 Fast report window

ADDING NEW EMPLOYEES

TO ADD A NEW EMPLOYEE, CLICK ON THE ADD EMPLOYEE BUTTON WHICH IS IN THE ADMINISTRATOR SECTION.



A NEW WINDOW WILL OPEN, FILL UP THE FORM WITH THE EMPLOYEE DETAILS AND CLICK ON THE REGISTER BUTTON OR CLICK ON THE MENU BUTTON TO GO BACK TO THE MENU.

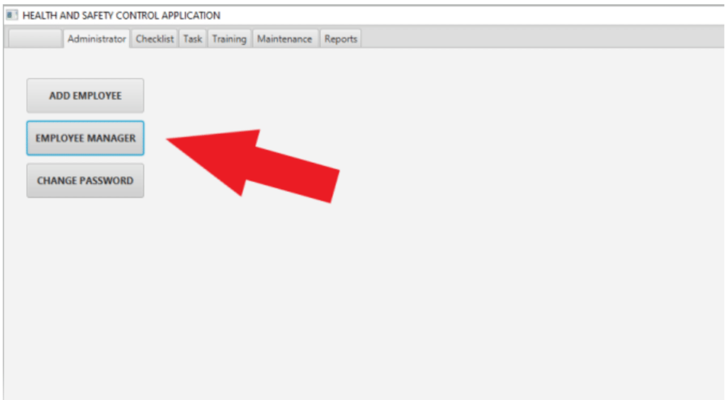


Employee Registration

First Name *	Jane
Last Name *	Doe
Email*	Janedoe@gmail.com
DOB*	10/05/1995
Mobile*	0838457505
Telephone	012345678
Address*	2 westmoreland St
City/County*	Dublin

MANAGING EMPLOYEES

TO MANAGE EMPLOYEES, CLICK ON THE EMPLOYEE MANAGER BUTTON IN THE ADMINISTRATOR SECTION.

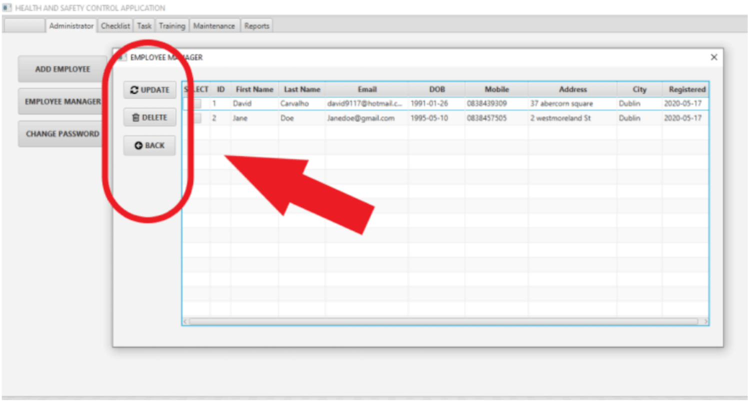


A NEW WINDOW WILL OPEN, IN THIS WINDOW YOU WILL SEE A LIST CONTAINING ALL EMPLOYEES REGISTERED IN THE SYSTEM AND THREE BUTTONS, SELECT AN EMPLOYEE AND USE:

UPDATE – TO UPDATE EMPLOYEE DETAILS.

DELETE – TO DELETE EMPLOYEE.

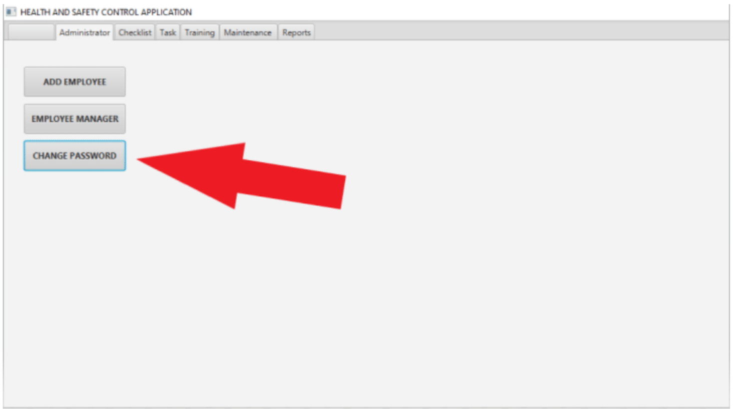
BACK – TO GO BACK TO THE MENU.



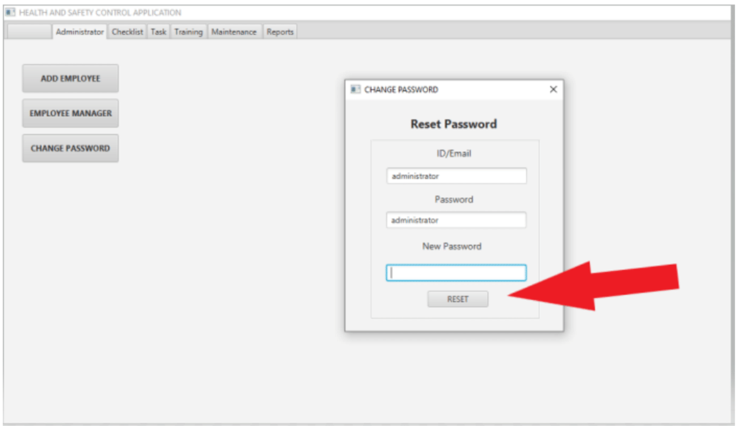
SELECT	ID	First Name	Last Name	Email	DOB	Mobile	Address	City	Registered
□	1	David	Carvalho	david9117@hotmail.c...	1991-01-26	0838439309	37 abercorn square	Dublin	2020-05-17
□	2	Jane	Doe	Janedoe@gmail.com	1995-05-10	0838457505	2 westmoreland St	Dublin	2020-05-17

CHANGING ADMINISTRATOR PASSWORD

CLICK ON THE CHANGE PASSWORD BUTTON IN THE ADMINISTRATOR SECTION.



A NEW WINDOW WILL OPEN, INSERT THE DEFAULT LOGIN AND PASSWORD "ADMINISTRATOR" FOLLOWED BY THE NEW PASSWORD AND CLICK ON THE RESET BUTTON.



Reset Password

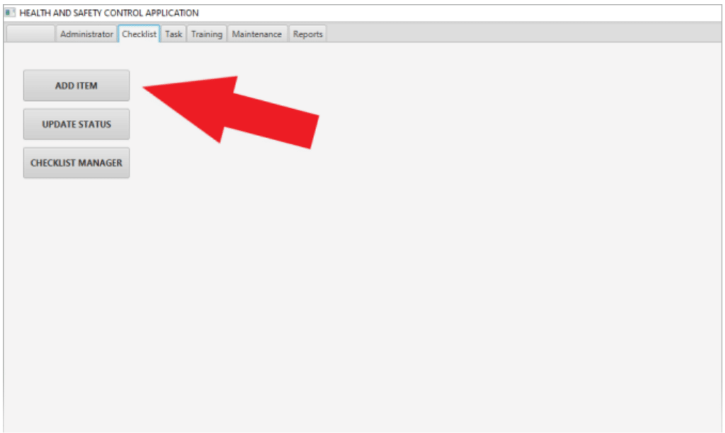
ID/Email: administrator

Password: administrator

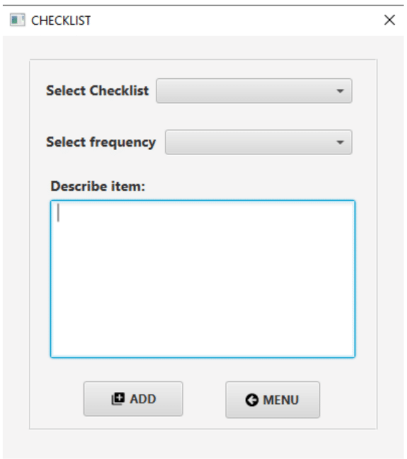
New Password:

ADDING NEW CHECKLIST

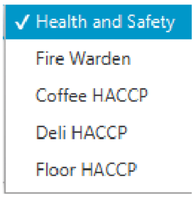
CLICK ON THE ADD ITEM BUTTON IN THE CHECKLIST SECTION.



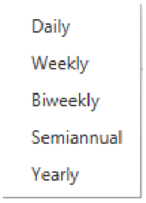
THIS NEW WINDOW WILL OPEN.



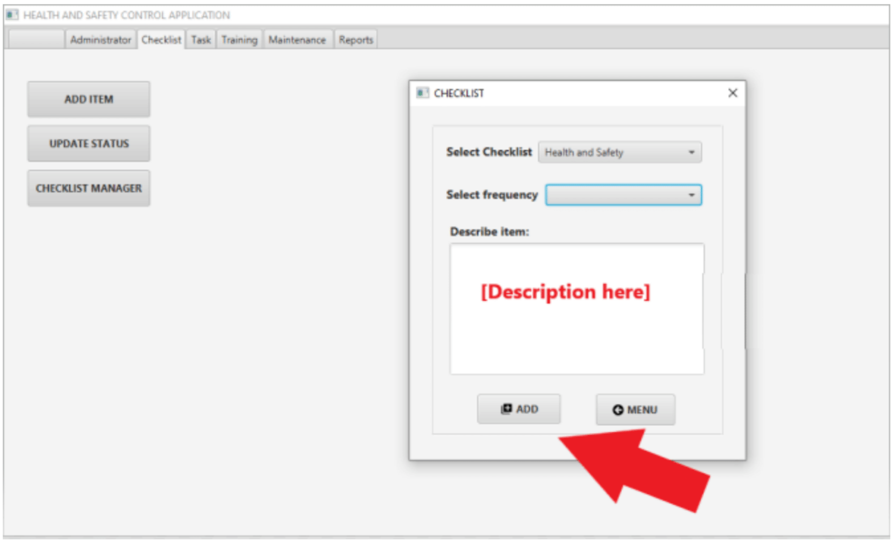
Select the correct checklist type.


✓ Health and Safety
Fire Warden
Coffee HACCP
Deli HACCP
Floor HACCP

Select a period the checklist will need to be done by.


Daily
Weekly
Biweekly
Semiannual
Yearly

Describe the checklist on the text box below and click on the add button or menu to go back to the menu.



Select Checklist: Health and Safety

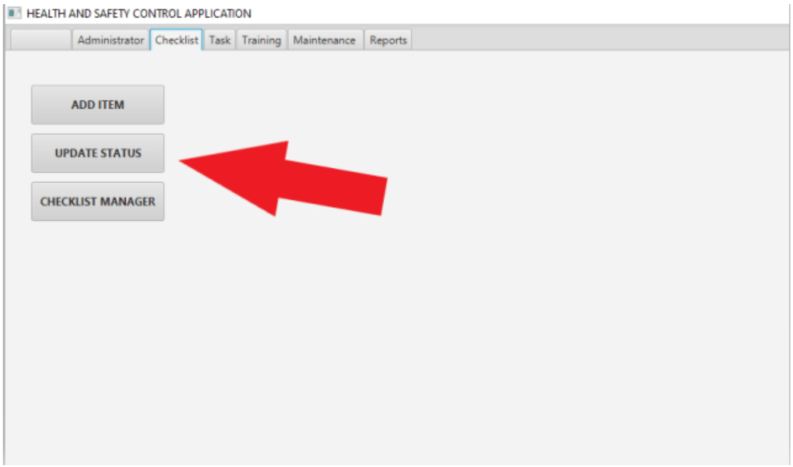
Select frequency:

Describe item:

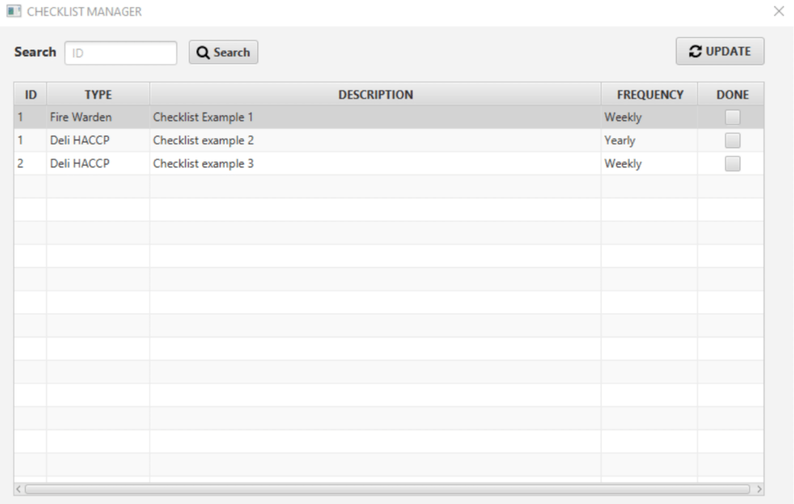
[Description here]

CHECKLIST UPDATE

CLICK ON THE UPDATE STATUS BUTTON IN THE CHECKLIST SECTION.

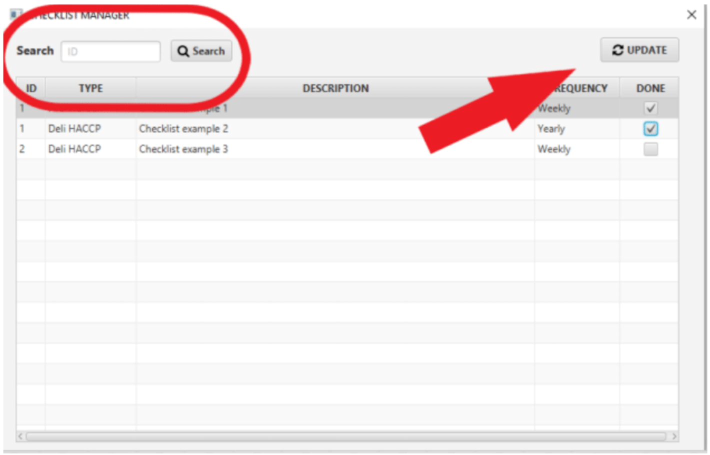


ON THIS NEW WINDOW YOU WILL SEE THE LIST CONTAINING ALL CHECKLISTS REGISTERED ON THE SYSTEM.



ID	TYPE	DESCRIPTION	FREQUENCY	DONE
1	Fire Warden	Checklist Example 1	Weekly	□
1	Deli HACCP	Checklist example 2	Yearly	□
2	Deli HACCP	Checklist example 3	Weekly	□

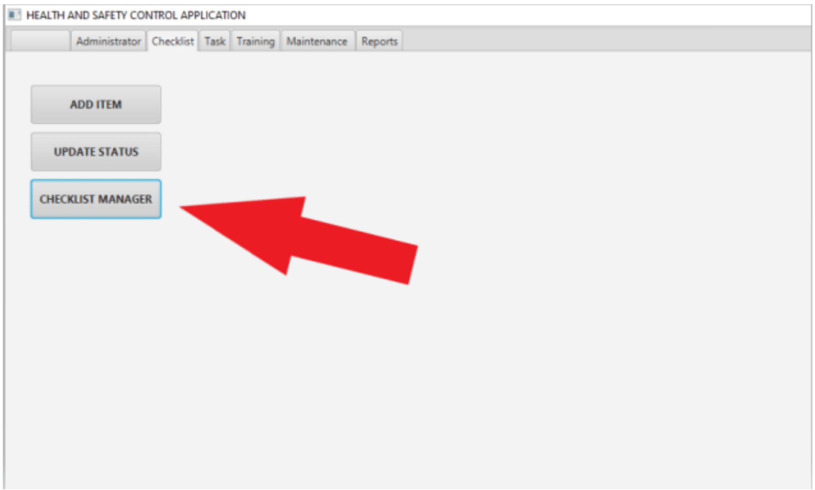
CLICK ON THE CHECK BOX ON THE RIGHT SIDE OF A CHECKLIST AND THEN CLICK ON THE UPDATE BUTTON TO REGISTER A CHECKLIST AS DONE. YOU CAN ALSO SEARCH A CHECKLIST BY ID ON TOP LEFT SIDE.



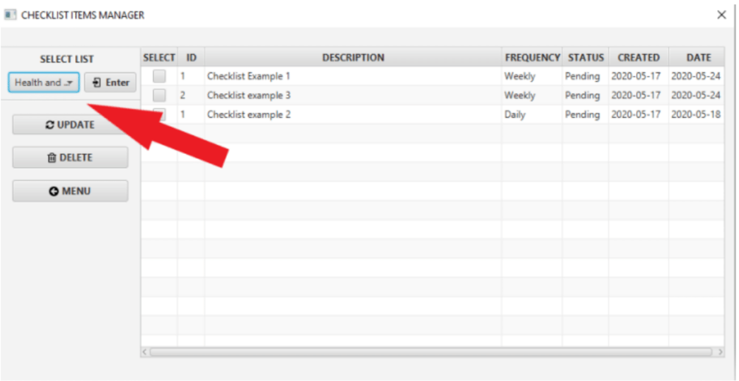
ID	TYPE	DESCRIPTION	REQUENCY	DONE
1		ple 1	Weekly	✓
1	Deli HACCP	Checklist example 2	Yearly	☑
2	Deli HACCP	Checklist example 3	Weekly	□

CHECKLIST MANAGER

CLICK ON THE CHECKLIST MANAGER BUTTON IN THE CHECKLIST SECTION.



ON THIS WINDOW YOU WILL SEE ALL THE CHECKLISTS REGISTERED ON THE SYSTEM, YOU CAN FILTER THE LIST BY TYPE SELECTING THE WANTED TYPE AND CLICK ON ENTER AS YOU CAN SEE BELOW.



CHECKLIST ITEMS MANAGER

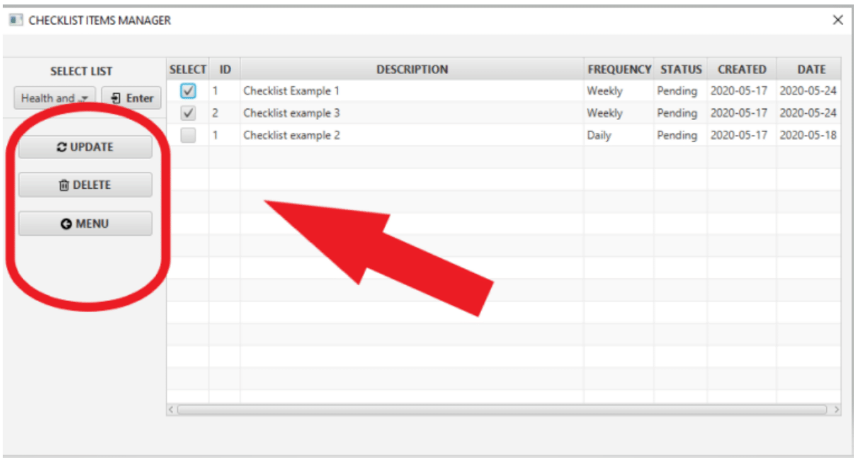
SELECT	ID	DESCRIPTION	FREQUENCY	STATUS	CREATED	DATE
□	1	Checklist Example 1	Weekly	Pending	2020-05-17	2020-05-24
□	2	Checklist example 3	Weekly	Pending	2020-05-17	2020-05-24
□	1	Checklist example 2	Daily	Pending	2020-05-17	2020-05-18

YOU CAN MANAGE THE CHECKLISTS BY USING THE BUTTONS ON THE LEFT SIDE:

UPDATE – SELECT THE CHECKLIST BY CHECKING THE CHECKBOX AND THEN CLICK ON UPDATE BUTTON.

DELETE - SELECT THE CHECKLIST BY CHECKING THE CHECKBOX AND THEN CLICK ON DELETE BUTTON.

MENU – GO BACK TO THE MENU

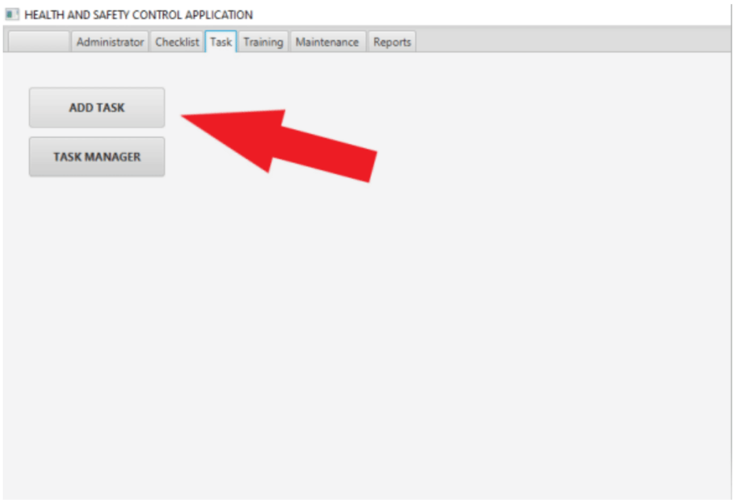


CHECKLIST ITEMS MANAGER

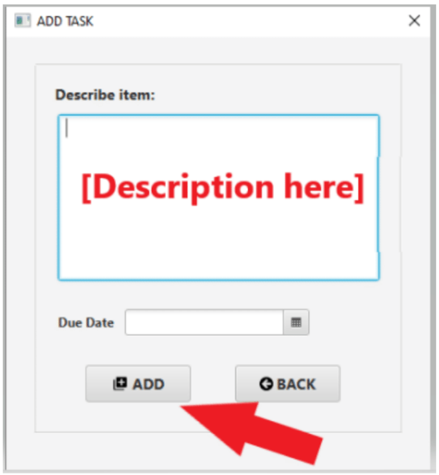
SELECT	ID	DESCRIPTION	FREQUENCY	STATUS	CREATED	DATE
☑	1	Checklist Example 1	Weekly	Pending	2020-05-17	2020-05-24
☑	2	Checklist example 3	Weekly	Pending	2020-05-17	2020-05-24
□	1	Checklist example 2	Daily	Pending	2020-05-17	2020-05-18

ADDING NEW TASKS

CLICK ON THE ADD TASK BUTTON IN THE TASK SECTION.

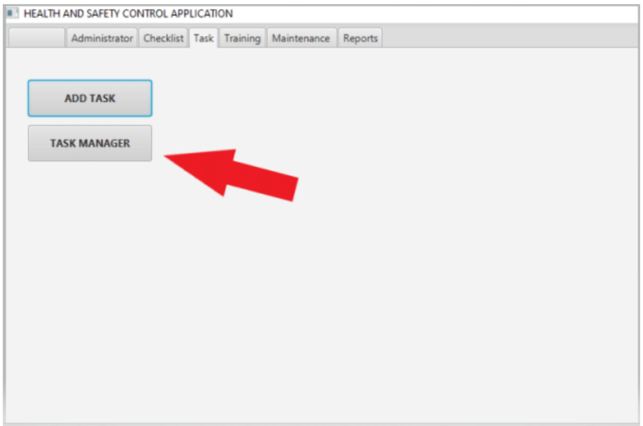


ON THE NEW WINDOW DESCRIBE THE TASK ON THE TEXT BOX, REGISTER A DATE THE TASK IS DUE TO AND CLICK ON THE ADD BUTTON AS YOU CAN SEE BELOW.

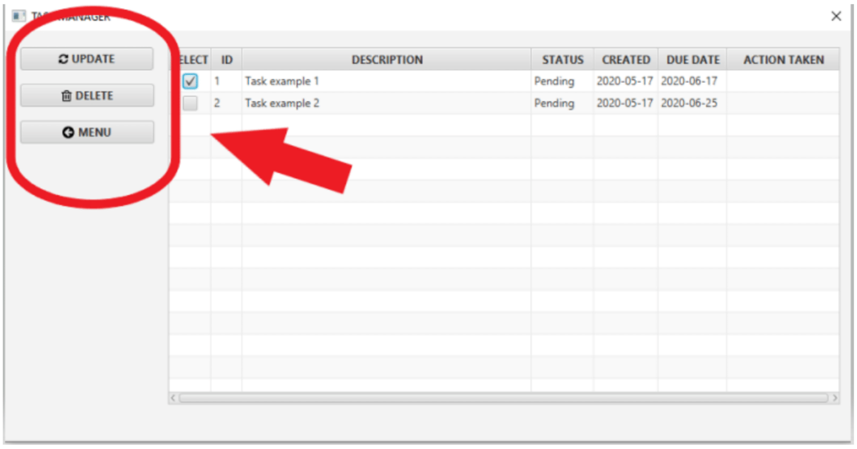


MANAGING TASKS

CLICK ON THE TASK MANAGER BUTTON IN THE TASK SECTION.



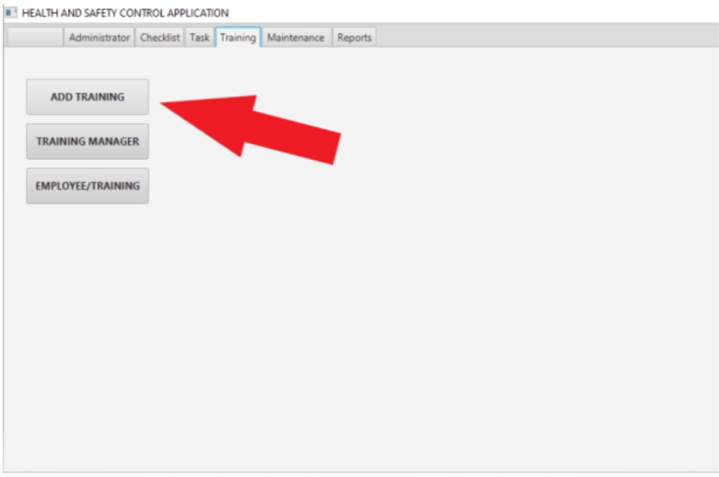
ON THE NEW WINDOW YOU CAN VIEW ALL TASKS FOLLOWED BY DATE AND DUE DATE REGISTERED ON THE SYSTEM, YOU CAN MANAGE USING THE BUTTONS ON THE LEFT-HAND SIDE AS YOU CAN SEE BELOW.



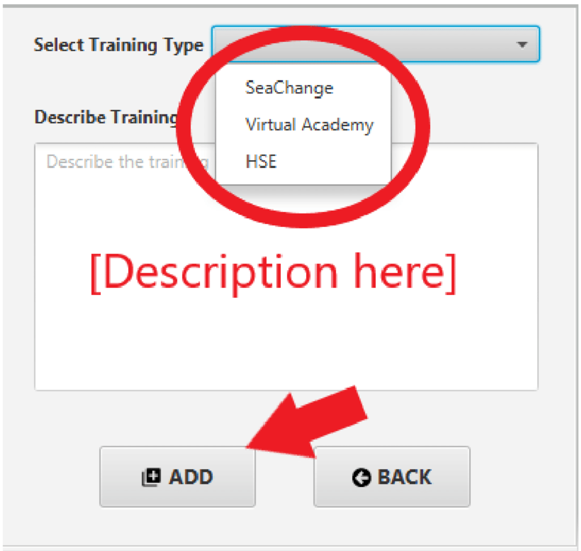
ELECT	ID	DESCRIPTION	STATUS	CREATED	DUE DATE	ACTION TAKEN
☑	1	Task example 1	Pending	2020-05-17	2020-06-17	
□	2	Task example 2	Pending	2020-05-17	2020-06-25	

ADDING NEW TRAININGS

CLICK ON ADD TRAINING BUTTON IN THE TRAINING SECTION.

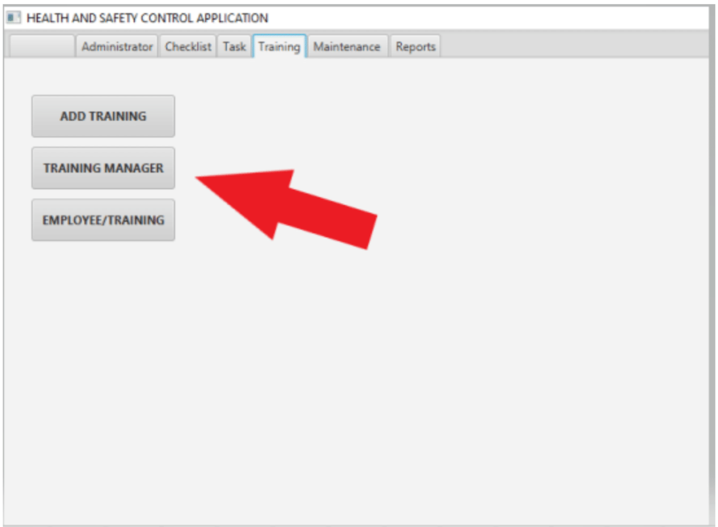


SELECT THE TRAINING TYPE, ADD SOME DESCRIPTION IF YOU NEED AND THEN CLICK ON THE ADD BUTTON AS YOU CAN SEE BELOW



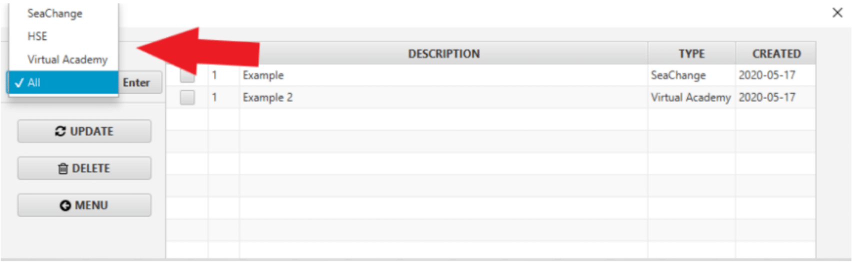
MANAGING TRAININGS

CLICK ON TRAINING MANAGER BUTTON IN THE TRAINING SECTION.



ON THE NEW WINDOW YOU CAN SEE ALL TRAININGS REGISTERED ON THE SYSTEM

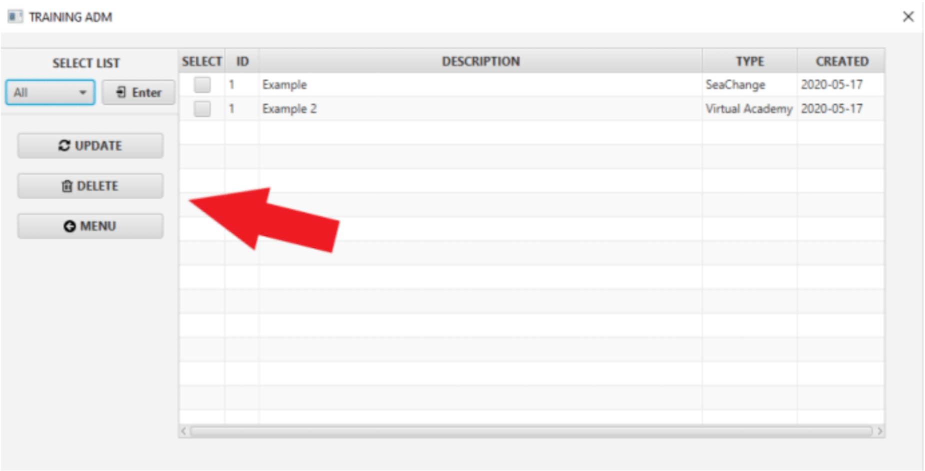
YOU CAN ALSO FILTER THE TRAININGS LIST BY TYPE BY CLICKING ON THE SELECT BOX AND ENTER BUTTON ON THE LEFT SIDE.



		DESCRIPTION	TYPE	CREATED
□	1	Example	SeaChange	2020-05-17
□	1	Example 2	Virtual Academy	2020-05-17

SELECT THE TRAINING YOU WANT TO MANAGE BY CLICKING ON THE CHECKBOX AND THEN USE THE BUTTONS ON THE LEFT SIDE AS YOU CAN SEE BELOW.

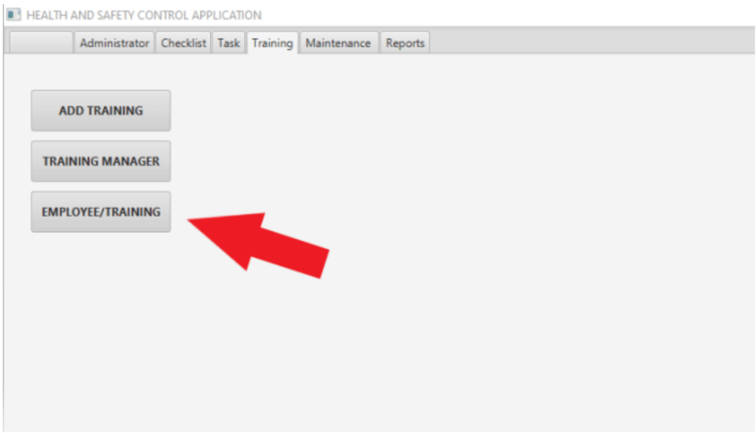
USE:
UPDATE- TO UPDATE A TRAINING (TYPE, DESCRIPTION).
DELETE- TO DELETE A TRAINING.
MENU- TO GO BACK TO THE MENU.



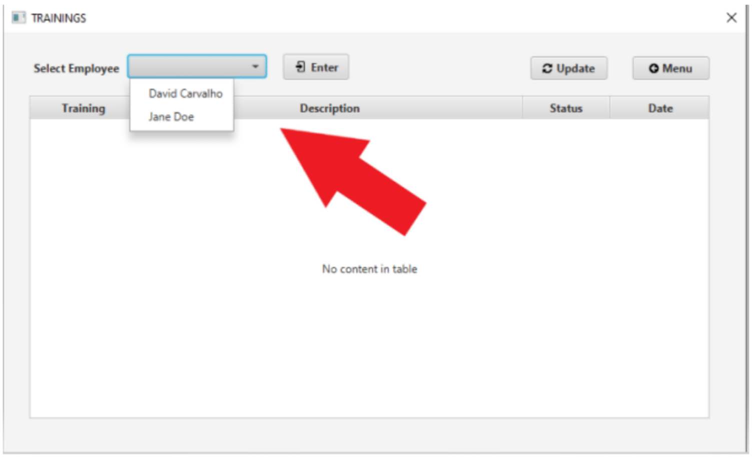
SELECT	ID	DESCRIPTION	TYPE	CREATED
☐	1	Example	SeaChange	2020-05-17
☐	1	Example 2	Virtual Academy	2020-05-17

EMPLOYEE/TRAINING

CLICK ON THE EMPLOYEE/TRAINING BUTTON IN THE TRAINING SECTION.



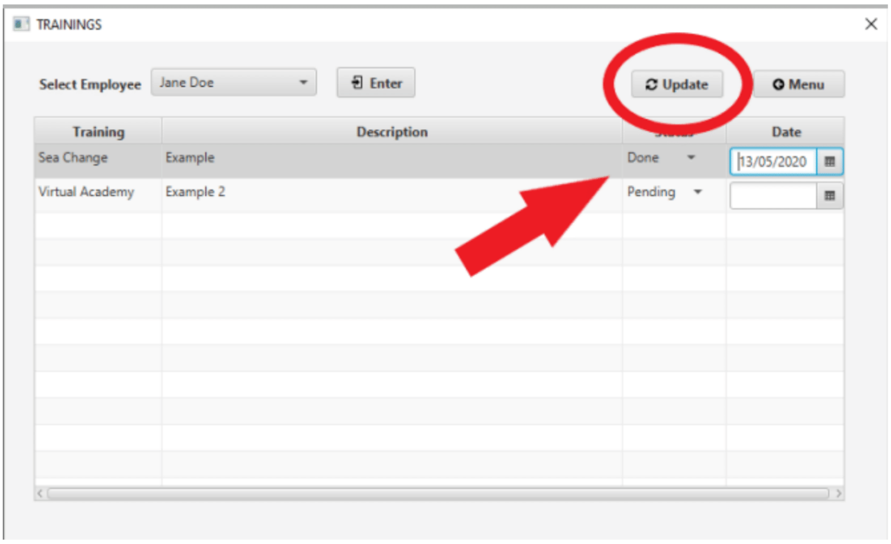
ON THE NEW WINDOW SELECT AN EMPLOYEE AND CLICK ENTER.



Training	Description	Status	Date

No content in table

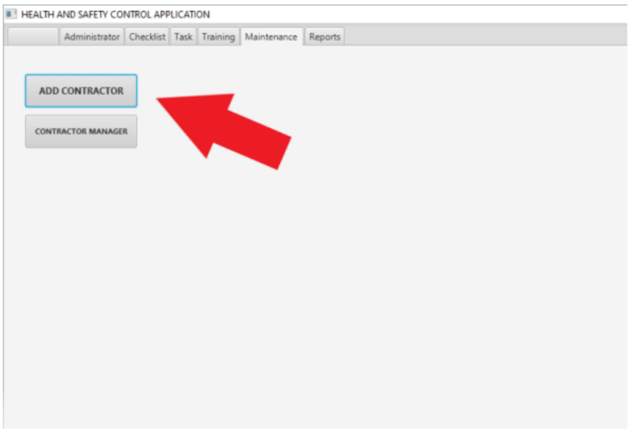
Click on the status selection box, you can change it from pending to done and register the date the employee selected had finished the training and then click on the update button as you can see below.



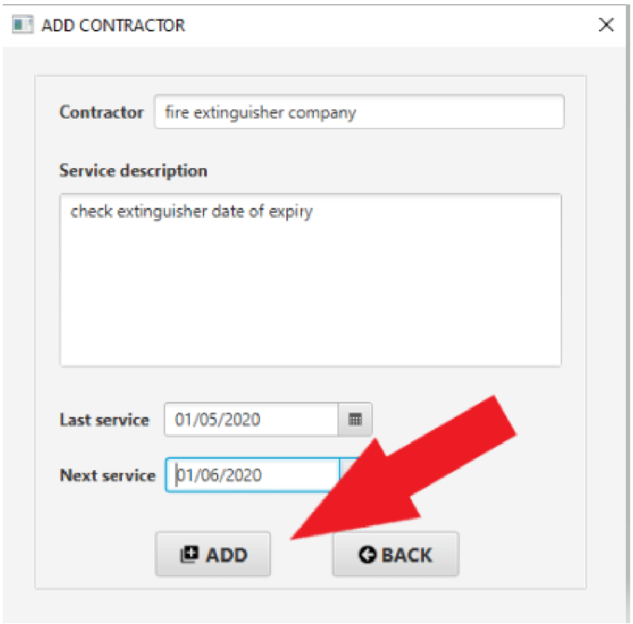
Training	Description		Date
Sea Change	Example	Done	13/05/2020
Virtual Academy	Example 2	Pending	

Adding new contractors

Click on the add contractor button in the maintenance section.



INSERT THE CONTRACTOR NAME A DESCRIPTION FOR THE SERVICE THEY WILL DO AND THE DATES OF LAST AND NEXT SERVICES FOR THE SHOP AND CLICK ON THE BUTTON ADD AS YOU CAN SEE BELOW.



ADD CONTRACTOR

Contractor: fire extinguisher company

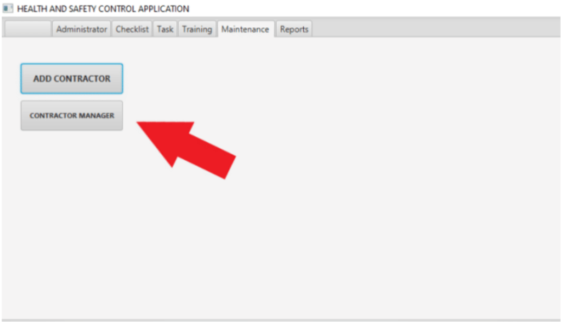
Service description: check extinguisher date of expiry

Last service: 01/05/2020

Next service: 01/06/2020

Contractor manager

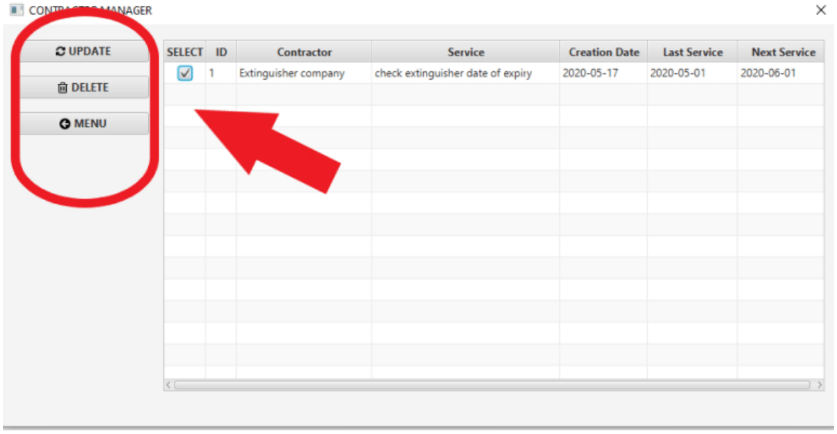
Click on the contractor manager button in the maintenance section.



HEALTH AND SAFETY CONTROL APPLICATION

On the new window you can view all contractors and dates of last and next service for the shop. Select a contractor and use the buttons on the left hand-side as you can see below.

Use:
Update- change contractor detail.
Delete- delete contractor.
Menu- go back to menu.

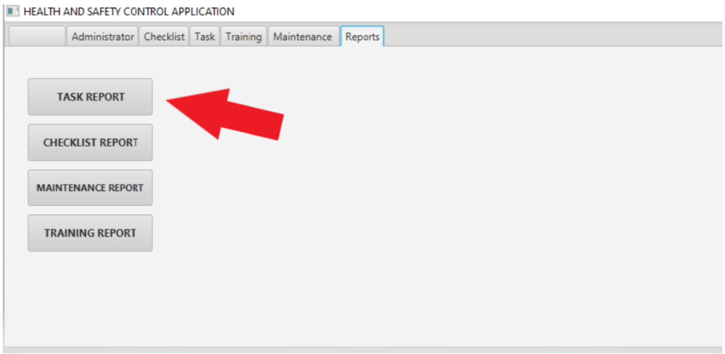


CONTRACTOR MANAGER

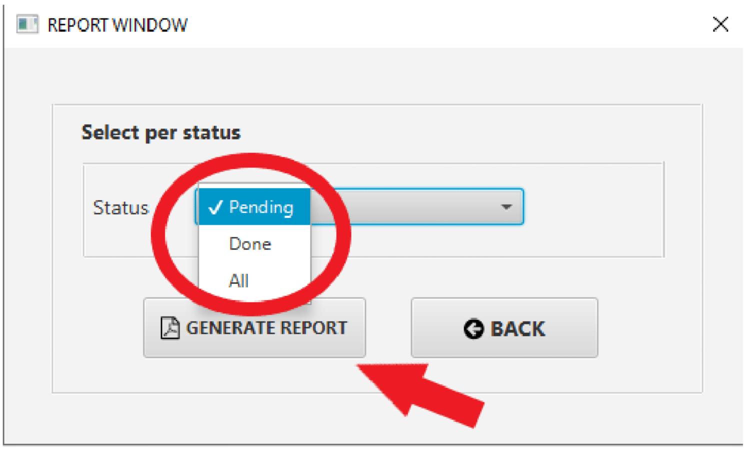
SELECT	ID	Contractor	Service	Creation Date	Last Service	Next Service
☑	1	Extinguisher company	check extinguisher date of expiry	2020-05-17	2020-05-01	2020-06-01

TASK REPORT

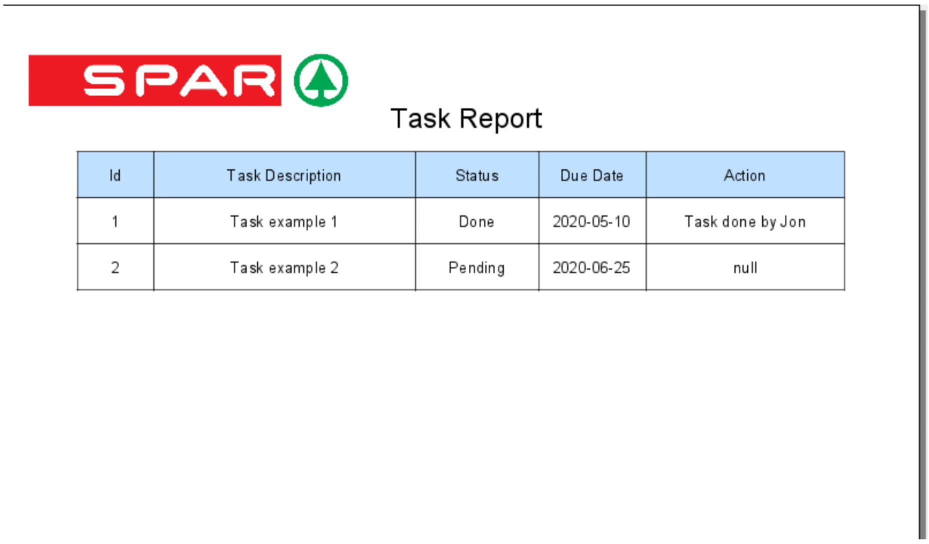
CLICK ON THE TASK REPORT BUTTON IN THE REPORTS SECTION.



ON THE NEW WINDOW YOU CAN CHOOSE THE STATUS OF TASK REPORT TO BE GENERATED. SELECT PENDING, DONE OR ALL AND CLICK ON THE GENERATE BUTTON AS YOU CAN SEE BELOW.



ON THE SCREENSHOT BELOW YOU CAN SEE AN EXAMPLE OF REPORT OF ALL PENDING AND DONE TASKS REGISTERED ON THE SYSTEM.

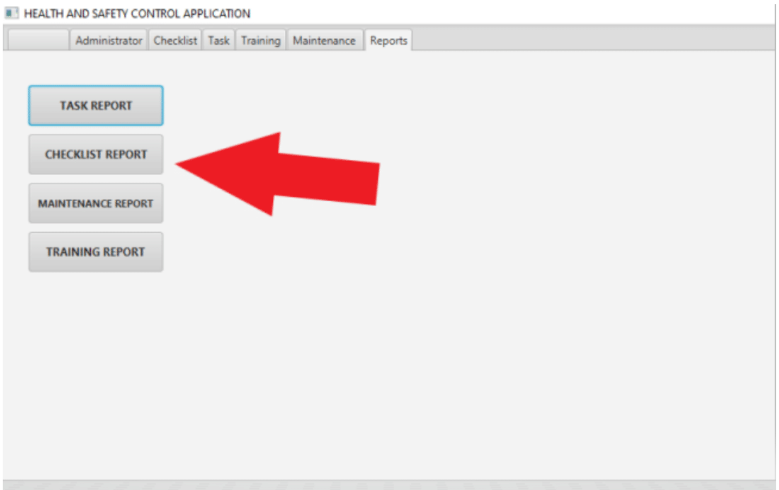


Task Report

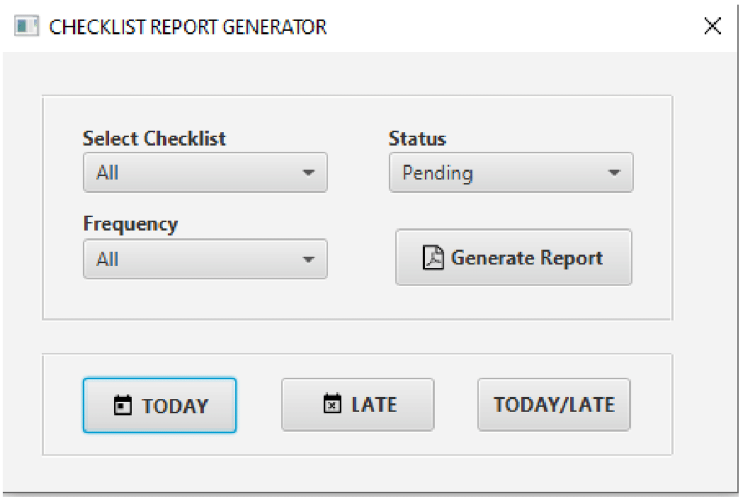
Id	Task Description	Status	Due Date	Action
1	Task example 1	Done	2020-05-10	Task done by Jon
2	Task example 2	Pending	2020-06-25	null

CHECKLIST REPORT

CLICK ON THE CHECKLIST REPORT BUTTON IN THE REPORTS SECTION.



A NEW WINDOW WILL OPEN.

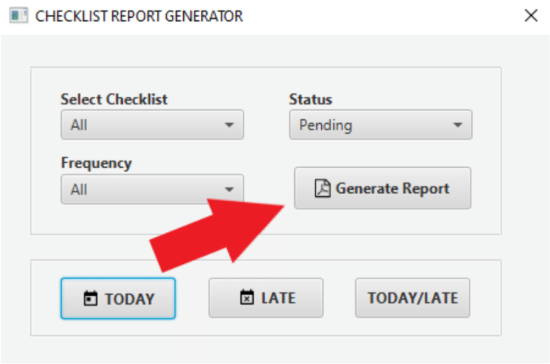


Select Checklist: All

Status: Pending

Frequency: All

On this new window you can filter the checklists by type, status and frequency to generate the report from, select the options you want or leave them on "All" and click on generate button.



CHECKLIST REPORT GENERATOR

Select Checklist: All

Status: Pending

Frequency: All

You can also see three buttons on the button of the window Today, late and Today/late. Use:

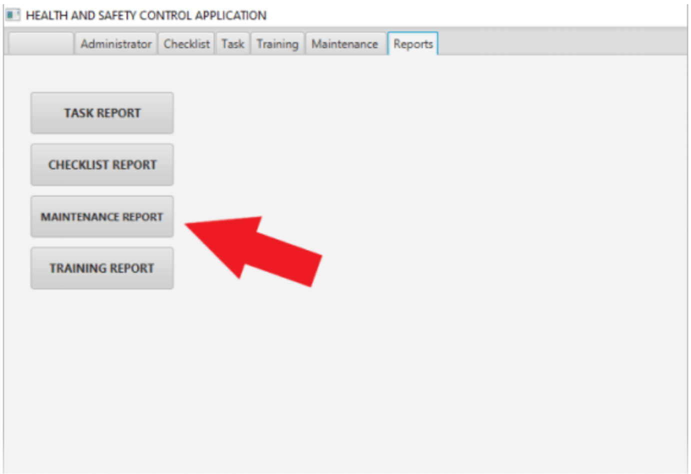
Today – generate report of duty that need to be done on the day.

Late – generate all the late duties.

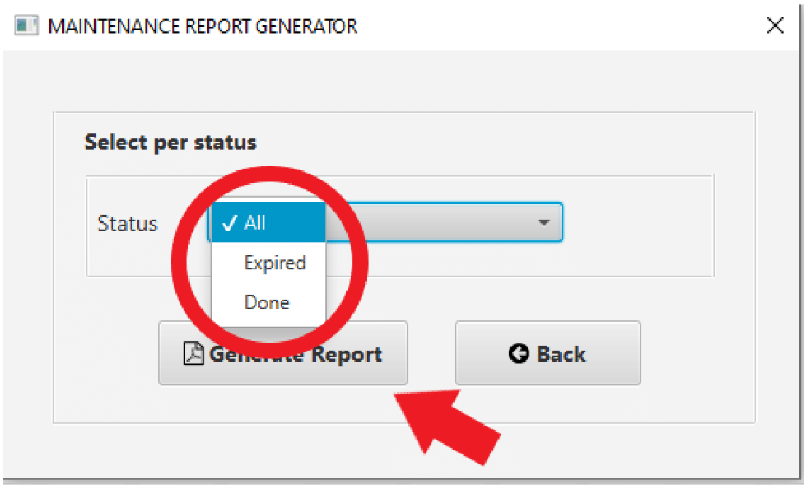
Today/Late – generate both late and duties that need to be done on the day.

Maintenance Report

Click on the maintenance report button in the reports section.

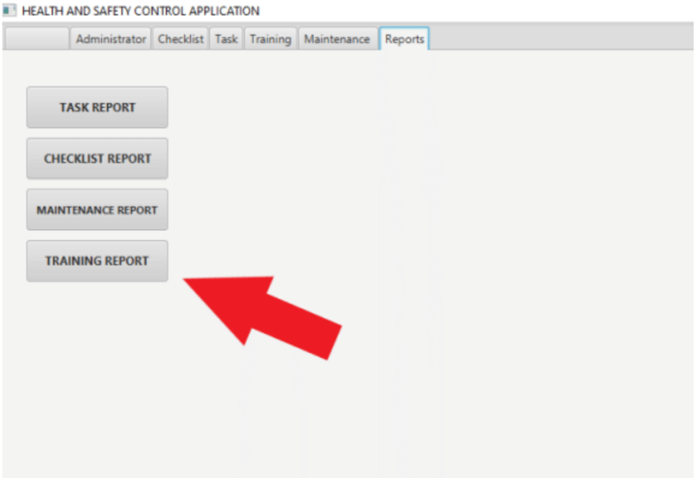


Select the status you want the report to be generated and click on the generate report button as you can see below.

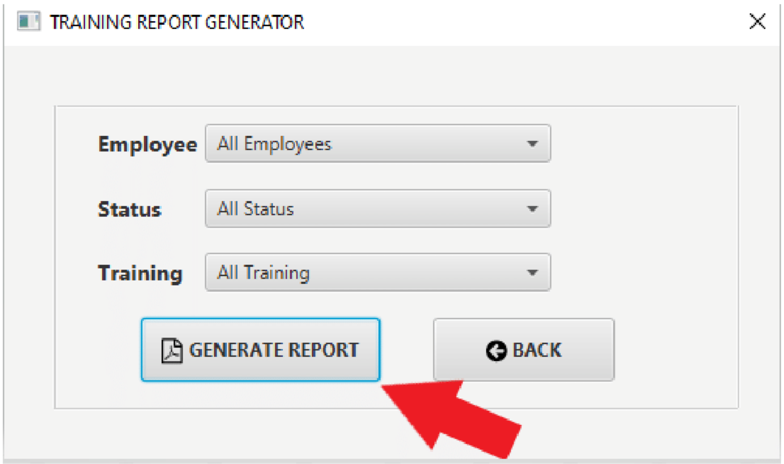


Training Report

Click on the training report button in the reports section.



To generate the report you want you can filter by employee, status and training or simply select "all" on the options to generate a complete report and click on generate report button as you can see below.


TRAINING REPORT GENERATOR

Employee: All Employees

Status: All Status

Training: All Training

Academia (2019). *Comparison of RDBMS, OODBMS and ORDBMS*. Available at:https://www.academia.edu/36323938/Comparison_of_RDBMS_OODBMS_and_ORDBMS. [Accessed 10th of December 2019].

Barki, H., Rivard, S. and Talbot J. (1993). *Toward an Assessment of Software Development Risk*. Journal of Management Information Systems, vol. 10, pp. 203-225.

Citizens Information (2018). *Health and safety at work*. Available at: https://www.citizensinformation.ie/en/employment/employment_rights_and_conditions/health_and_safety/health_safety_work.html. [Accessed 20th of October 2019].

Coleman, A. (2018). *'Health and safety – it's about the bottom line, not just bad backs'*, The Guardian, 2 February. Available at: https://www.theguardian.com/careers/2018/feb/02/health-and-safety-its-about-the-bottom-line-not-just-bad-backs. [Accessed 20th of October 2019].

Cool It Help, 2017. *JavaFX TableView | Delete Checkbox Selected Rows on Button Click*. Available at: https://www.youtube.com/watch?v=dJdKPjZEeuo&t=74s. [Accessed 13 03 2020].

Corporate OHS (2018). *Financial consequences of poor health and safety procedures*. Available at: https://corporateohs.com/financial-consequences-poor-health-safety-procedures/. [Accessed 11th of November 2019].

Debnath, M., 2019. *Developer.* [Online]
Available at: https://www.developer.com/java/data/implementing-singleton-patterns-in-java.html [Accessed 10 05 2020].

Deters, J., 2017. *MVNREPOSITORY.* [Online]
Available at: https://mvnrepository.com/artifact/de.jensd/fontawesomefx-fontawesome/4.7.0-9.1.2 [Accessed 13 05 2020].

Deters, J., 2017. *MVNREPOSITORY.* [Online]
Available at: https://mvnrepository.com/artifact/de.jensd/fontawesomefx-materialicons [Accessed 13 03 2020].

Deters, J., 2017. *MVNREPOSITORY.* [Online]
Available at: https://mvnrepository.com/artifact/de.jensd/fontawesomefx-materialdesignfont/2.0.26-9.1.2 [Accessed 13 03 2020].

Fernandes,B (2018). *Eclipse — Still the Best IDE!* Available at: https://www.genuitec.com/eclipse-still-best-ide/. [Accessed 10th of December 2019].

Fitzgerald, K. (2015). *'How safety in the workplace has evolved'*, Irish Examiner, 6 May. Available at: https://www.irishexaminer.com/business/how-safety-in-the-workplace-has-evolved-328895.html. [Accessed 20th of October 2019].

GlassFish Comunity, 2018. *MVN REPOSITORY.* [Online]
Available at: https://mvnrepository.com/artifact/javax.annotation/javax.annotation-api/1.3.2 [Accessed 13 03 2020].

Health And Safety Authority (2019). *Retail*. Available at: https://www.hsa.ie/eng/Your_Industry/Retail/. [Accessed 23rd of October 2019].

Hibernate (2020). Hibernate ORM. Your Relational Data. Objectively. Available at: https://hibernate.org/orm/. [Accessed 14th of March of 2020].

Hibernate.org, 2020. *MVNREPOSITORY.* [Online] Available at: https://mvnrepository.com/artifact/org.hibernate/hibernate-entitymanager/5.4.11.Final [Accessed 13 03 2020].

JavaQuery, 2015. *How to fill Jasper Report table using Collection of data in Java*. Available at: https://www.youtube.com/watch?v=AfC6MlWrXqY. [Accessed 13 03 2020].

Lavanya, N. & Malarvizhi, T. (2008). *Risk analysis and management: a vital key to effective project management*. Paper presented at PMI ® Global Congress 2008 – Asia Pacific, Sydney, New South Wales, Australia. Newtown Square, PA: Project Management Institute.
Lynch,W (2019). *Why Visual Paradigm is a better choice than Microsoft Visio*. Available at: https://medium.com/@warren2lynch/why-visual-paradigm-is-a-better-choice-than-microsoft-visio-eeba72aa050a . [Accessed 5th of December 2019].

OpenJFX, 2019. *MVNREPOSITORY.* [Online]
Available at: https://mvnrepository.com/artifact/org.openjfx/javafx-base/13 [Accessed 13 03 2020].

OpenJFX, 2019. *MVNREPOSITORY.* [Online]
Available at: https://mvnrepository.com/artifact/org.openjfx/javafx-controls/13 [Accessed 13 03 2020].

OpenJFX, 2019. *MVNREPOSITORY.* [Online]
Available at: https://mvnrepository.com/artifact/org.controlsfx/controlsfx/11.0.1 [Accessed 13 03 2020].

OpenJFX, 2019. *MVNREPOSITORY.* [Online]
Available at: https://mvnrepository.com/artifact/org.openjfx/javafx-graphics/13 [Accessed 13 03 2020].

OpenJFX, 2019. *MVNREPOSITORY.* [Online]
Available at: https://mvnrepository.com/artifact/org.openjfx/javafx-fxml/13 [Accessed 13 03 2020].

OpenJFX, 2020. *MVNREPOSITORY.* [Online]
Available at: https://mvnrepository.com/artifact/org.openjfx/javafx-swing/15-ea+5 [Accessed 15 03 2020].

Oracle (2013) *What Is JavaFX?* Available at https://docs.oracle.com/javafx/2/overview/jfxpub-overview.htm.[Accessed 10th of December 2019].

Oracle Corporation, 2019. *MVNREPOSITORY.* [Online]
Available at: https://mvnrepository.com/artifact/mysql/mysql-connector-java/8.0.19 [Accessed 15 03 2020].

Pankaj, n.d. *JournalDev.* [Online]
Available at: https://www.journaldev.com/1392/factory-design-pattern-in-java [Accessed 10 05 2020].

Pronschins, M. (2010). *Versant ODB as Easy as ORM.* Available https://dzone.com/articles/versant-odb-easy-orm. [Accessed 20th of December 2019].

Reinier Zwitserloot, R. S., 2017. *MVNREPOSITORY.* [Online]
Available at: https://mvnrepository.com/artifact/org.projectlombok/lombok/1.18.10 [Accessed 30 03 2020].

SeaChange (2019). *Evidence shows that a Proactive Safety Culture reduces both Frequency and Cost of Liability Claims across All Sectors*. Available at: http://www.seachange.ie/evidence-shows-that-a-proactive-safety-culture-reduces-both-the-frequency-and-cost-of-liability-claims-across-all-sectors-3/. [Accessed 18th of December 2019].

Shubham, n.d. *JournalDev.* [Online]
Available at: https://www.journaldev.com/16813/dao-design-pattern [Accessed 10 05 2020].

Tutorial Point (2019). *Spring framework - Overview*. Available https://www.tutorialspoint.com/spring/spring_overview.htm. [Accessed 15th of December 2019].

Usability.gov (2020). User Interface Design Basics. Available at: https://www.usability.gov/what-and-why/user-interface-design.html. [Accessed 14th of March 2020].

Visual Paradigm (2020). *What is Use Case Diagram*. Available at: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/. [Accessed 14th of March 2020].

Zolfaghari, S. and Mousavi, S. M. (2018). *Construction-project risk assessment by a new decision model based on De-Novo multi-approaches analysis and hesitant fuzzy sets under uncertainty*. Journal of Intelligent and Fuzzy Systems, vol.35, pp. 639-649.