

# **HOSPITALITY MATERIALS CONTROL**

Submitted in part fulfilment for the degree of  
**HIGHER DIPLOMA IN SCIENCE IN COMPUTING**  
Institute of Technology Blanchardstown  
Dublin, Ireland

by  
**Florica Coste**

May 2017



# Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Higher Diploma in Science in Computing in the Institute of Technology Blanchardstown, is entirely my own work except where otherwise stated, and has not been submitted for assessment for an academic purpose at this or any other academic institution other than in partial fulfilment of the requirements of that stated above.

Signed: \_\_\_\_\_ Dated: \_\_\_\_/\_\_\_\_/\_\_\_\_



# Abstract

Everyone is concerned with planning, organizing and controlling the flow of materials from their initial purchase and the impact of internal operations. This project make determined efforts to deal with the problem of creating a WEB application to model and manipulate a Hospitality Materials Control. The problem to be solved was how to create a WEB application that is easy to use and also allows any employ which has access to a computer to obtain information about different products used in the hotel. This report contain the specification and analysis of the problem, the life cycle of the system that was developed and the problem solution.

Key features of the developed WEB application, called “Hospitality Materials Control”, including:

- to operate efficiently;
- to have adequate materials on hand when needed;
- to pay the lowest possible prices, consistent with quality and value requirement for purchases materials;
- to minimize the inventory investment;

The result is a simple, but powerful tool for hotel industry, that will be very useful tools in getting the right quality and right quantity of supplies at right time, having good inventory control and will improve the efficiency of the hotel.



# Acknowledgements

I am very thankful to my group coordinator Maria Brennan and supervisor Matt Smith who guided me in a best professional way.

I would like to dedicate this work to my husband Florin and my little boys: Alex and Bogdi.

I am indebted to my classmates as they made it a wonderful place to learn.





# Table of Contents

<b>Declaration</b>	<b>i</b>
<b>Abstract</b>	<b>i</b>
<b>Acknowledgements</b>	<b>iii</b>
<b>I Introduction</b>	<b>1</b>
<b>1 Introduction</b>	<b>3</b>
1.1 Project Goal . . . . .	3
1.2 Project Objectives . . . . .	3
1.3 Context of Project . . . . .	4
1.4 Motivation of Choice of Project . . . . .	5
1.5 Deliberation . . . . .	6
1.6 Report Structure Summary . . . . .	6
<b>II Overview of Hospitality Materials Control</b>	<b>9</b>
<b>2 Overview of Hospitality Materials Control</b>	<b>11</b>
<b>III Literature review</b>	<b>13</b>
<b>3 Literature review</b>	<b>15</b>
3.1 Overview of Technical Resources Consulted . . . . .	15
<b>IV System analysis</b>	<b>19</b>
<b>4 System analysis</b>	<b>21</b>
4.1 User login: - Use case . . . . .	22
4.2 Manager login: - Use case . . . . .	31

<b>V Database design</b>	<b>41</b>
<b>5 Database design</b>	<b>43</b>
5.1 Product details . . . . .	44
5.2 PO - purchase order . . . . .	45
5.3 Quotation . . . . .	47
5.4 Stock . . . . .	49
5.5 Supplier . . . . .	49
5.6 User details . . . . .	52
5.7 EED Diagram . . . . .	54
 <b>VI Implementation of System</b>	 <b>57</b>
<b>6 Implementation of System</b>	<b>59</b>
6.1 Entity classes . . . . .	59
<b>7 System design</b>	<b>67</b>
7.1 User Interface . . . . .	67
 <b>VII Testing</b>	 <b>77</b>
<b>8 Testing and work review</b>	<b>79</b>
 <b>VIII Conclusion</b>	 <b>83</b>
<b>9 Conclusion</b>	<b>85</b>
 <b>IX Future work</b>	 <b>87</b>
<b>10 Future work</b>	<b>89</b>
<b>A Appendix</b>	<b>91</b>
A.1 Creating Applications with Composer in Symfony . . . . .	91
<b>B Create a Symfony entity</b>	<b>93</b>
<b>C Generating database with Symfony</b>	<b>97</b>
C.1 Creating tables in the database . . . . .	98
<b>List of References</b>	<b>99</b>

---

TABLE OF CONTENTS

---

<b>X</b>	<b>Generating a CRUD Controller Based on a Doctrine Entity</b>	<b>101</b>
<b>XI</b>	<b>Database</b>	<b>103</b>
<b>XII</b>	<b>Entities</b>	<b>105</b>
<b>XIII</b>	<b>User Interface</b>	<b>107</b>



## **Part I**

# **Introduction**



# 1

## Introduction

This chapter gives a brief description of “Hospitality Materials Control”. Include the purpose, scope, definitions, references, and overview of the WEB application.

“Material control refers to the management function concerned with acquisition and use of materials so as to minimise wastage and losses, derive maximum economy and establish responsibility for various operations through physical checks, record keeping, accounting and other devices.”

### 1.1 Project Goal

The purpose of this project is to create Create a WEB application for the hotel which will give permission to user to access suppliers details and products information. Material control for Hotels refers to the various measures adopted to reduce the amount of loss time of receiving and issuing the raw materials. Material control in practice is exercised through periodical records and reports relating to purchase, receipt, storage and issuing direct and indirect materials. Proper control over material can contribute substantially to the efficiency of a business.

### 1.2 Project Objectives

In a Hotel Industry the need of materials control arises on account of the following reasons:

1. For keeping the stock of materials within limits in the stores , to avoid overstocking and under stocking of materials, materials control is significant.

2. It ensures proper information about different items used in the Hotel materials. For the proper preservation and safety of materials, adequate storage facilities are to be provided.
3. Certain techniques and methods are developed under the system of materials control thereby ensuring optimum utilisation of materials.
4. In order to undertake continuous checking of materials, the necessity of a proper system of materials control cannot be ignored.
5. A well managed system of materials control ensures the availability of different kinds of materials without delay.

### 1.3 Context of Project

This is a database driven application requiring knowledge of:

1. MVC - Model-View-Controller

Models represent knowledge. A model could be a single object, or it could be some structure of objects. There should be a one-to-one correspondence between the model and its parts on the one hand, and the represented world as perceived by the owner of the model on the other hand.

A view is a representation of its model. It would ordinarily highlight certain attributes of the model and suppress others. It is thus acting as a presentation filter.

ON the same time a view is attached to its model and gets the data necessary for the presentation from the model by asking questions. It may also update the model by sending appropriate messages. All these questions and messages have to be in the terminology of the model, the view will therefore have to know the semantics of the attributes of the model it represents.

A controller is the link between a user and the system. It provides the user with input by arranging for relevant views to present themselves in appropriate places on the screen. It provides means for user output by presenting the user with menus or other means of giving commands and data. The controller receives such user output, translates it into the appropriate messages and pass these messages on to one or more of the views.

2. Twig which is a modern template engine for PHP

Twig compiles templates down to plain optimized PHP code. The overhead compared to regular PHP code was reduced to the very minimum.

Twig has a sandbox mode to evaluate untrusted template code. This allows Twig to be used as a template language for applications where users may modify the template design.

Twig is powered by a flexible lexer and parser. This allows the developer to define its own custom tags and filters, and create its own DSL.

3. JavaScript



JavaScript is a programming language that adds interactivity to the website (for example: responses when buttons are pressed or data entered in forms in a dynamic style)

#### 4. AJAX

AJAX is a set of Web development techniques using many Web technologies on the client side to create asynchronous WEB applications.

#### 5. Symfony

“Symfony is a set of PHP Components, a Web Application framework, a Philosophy, and a Community — all working together in harmony.”

- Symfony Framework

A framework is one of the tools that is available to help you develop better and faster! Better, because a framework provides you with the certainty that you are developing an application that is in full compliance with the business rules, that is structured, and that is both maintainable and upgradable. Faster, because it allows developers to save time by re-using generic modules in order to focus on other areas.

- Symfony Components

Symfony Components are a set of reusable PHP libraries. They are becoming the standard foundation on which the best PHP applications are built on. These components are reused in applications independently from the Symfony Framework.

- Symfony Community

A huge community of Symfony fans committed to take PHP to the next level.

- Symfony Philosophy

Behind any embracing and promoting professionalism, best practices, standardization and interoperability of applications.

More about Symfony at:

- [Symfony, High Performance PHP Framework for Web Development](#)
- [GITHUB Symfony](#)

## 1.4 Motivation of Choice of Project

In reality more than 60 percent of the information regards to suppliers and items provided in a hotel are store on the paper.

The main reason of doing this project is that is a potential tool that could be useful in many place of work, specially now that hotel industry is now well placed to deliver significant employment and foreign earnings growth to 2020. The hotel is a home away from home where all the modern

amenities are available. To be able to offer to the guest a unique experience you need to deliver te service on time at the highest standard.

To be a ble to achieve the guest expectation everyone which work on the same place need to know information about the products which they work with and the suppliers which provide them.

It is well know that the human resources movement across the departament in the hotel and between the hotels is high. Most of the information get lost on the moment when one of the key person which use to please a order leave. So it is take time to find the detail and reconnect with supplier.

## 1.5 Deliberation

- Fully working website.
- User documentation - details of how use the WEB application.
- System evaluation and evidence of testing

## 1.6 Report Structure Summary

The report will contain the following section:

- Chapter 1 - Introduction
- Chapter 2 - Overview of Hospitality Materials Control

This chapter gives a brief description of Hospitality Materials Control

- Chapter 3 - Literature review

This chapter will give an examination of the structure of Hospitality Materials Control

- Chapter 4 - System Analysis: Function Requirements
- Chapter 5 - Design

This section provides an overview of the entire software design. It references and complies with design interface contracts, requirements and module decomposition approach.

- Chapter 6 - Implementation

This chapter outlines the construction of the actual project result. It will describe the steps taken to to construct the application. The project becomes visible to outsiders, so will be presented from the user perspective

- Chapter 7 - Testing and evaluation

This chapter describe the functionality of the system were met.

- Chapter 8 - Conclusion

This chapter outlines concise statements about the main findings.

- Chapter 9 - Future work

This chapter describe the recognition of necessary work need it to improve the system:

- Appendix - Program Listings

This section give a list of scripts that have been used in the program

- List of reference

This chapter lists and details the source of information used for this project



## Part II

# Overview of Hospitality Materials Control



# 2

## Overview of Hospitality Materials Control

Hospitality Materials Control is a collection of information for items inventory and cost control customized to meet the requirements of the hospitality industry. The solution plane to manage all property inventory, providing essential real-time information on cost, stock on hand, order proposals, and stocking requirements, and bringing efficiencies to daily workflow.

The ordering processes in the hotel industry it a link between suppliers and products. Hotel managers are constantly placing purchase requests. Most of the hotels due to the economic situation eliminate the “purchasing department”. So the manager will send a request to the General Manager of the hotel which will consult the Financial Controller. They will meet to review and approve a single purchase request or combine multiple requests from several departments into a single order. Entire process is long and take lots of time. This WEB application plan to eliminate the gap between the time when you place an order and the time when the order rich the suppliers.

In a real time data provided by Hospitality Materials Control enables a variety of information as: stock on hand, orders database, and par lists. Information can be reviewed or added. Data validity is increased for all departments with an interface to log in and create a ROLE user interface. Hospitality Materials Control in addition get month-end reports.

Hospitality Materials Control provide a daily processes related to purchasing, supplier and price management and internal requisitions. To the side of ordering and receiving process, you can transmit purchase orders directly to suppliers from within the application e-mail.

We look to the material from more perspective:

- Purchasing side: (price list of purchasing items, receiving with or without purchasing orders

and invoice management. This supports control of purchasing process by defining authorization levels for each inventory item. Reporting includes purchase per item or per supplier, quantity and amount deviation between ordered and received and many other useful reports.

- Store side (requisition, store transfers on the basis of requisition, inventory, stock control, defining minimal and optimal stock, creating inventories.) Reporting in this module includes article history, requests and stock on hand.
- Master data which will contain information about (suppliers, purchasing items - raw materials and supplies, tax rates, units).
- There is a possibility of interfacing with the accounting software to be able to automatically precede the payments.

The benefits of Materials Control would be: managing a large number of Suppliers, managing groups of items. On the same time you can apply for price quotes. and managing inventory. Managing POs system through a interface the will give more transparency to the environment.



## **Part III**

# **Literature review**



# 3

## Literature review

Materials control is a definition of a systematic control over purchasing and dealing with suppliers. Materials control helps to maintain a regular and timely supply of materials by avoiding over and under-stocking. Materials control ensures that the right quality and quantity of materials is available to the company at the right time. Materials control helps to reduce the losses and wastage of materials by maintaining their efficient purchase.

On the market they are some application that provide this facilities, but they are expensive due to the fact that they are build by the big Company.

### 3.1 Overview of Technical Resources Consulted

This chapter will present the technical resources used to achieve the goal of project:

- MVC - Model-View-Controller;
- Object Oriented in PHP extend to Symfony;
- Twig;
- Java Script;
- AJAX
- Symfony;

#### 1. MVC - Model-View-Controller

Model View Controller (MVC) is used to implement user interfaces: it is therefore a popular choice for architecting web page. In general, it separates out the application logic into three separate

parts, promoting modularity and ease of collaboration and reuse. It also makes applications more flexible and welcoming to iterations.

For a better understanding we can look to the MVC apply for a Basket list - a concept used in this application. The list of items will store: names, quantities and price of each items which we need to have in stock. Below we'll describe how we could implement some of this functionality using MVC.

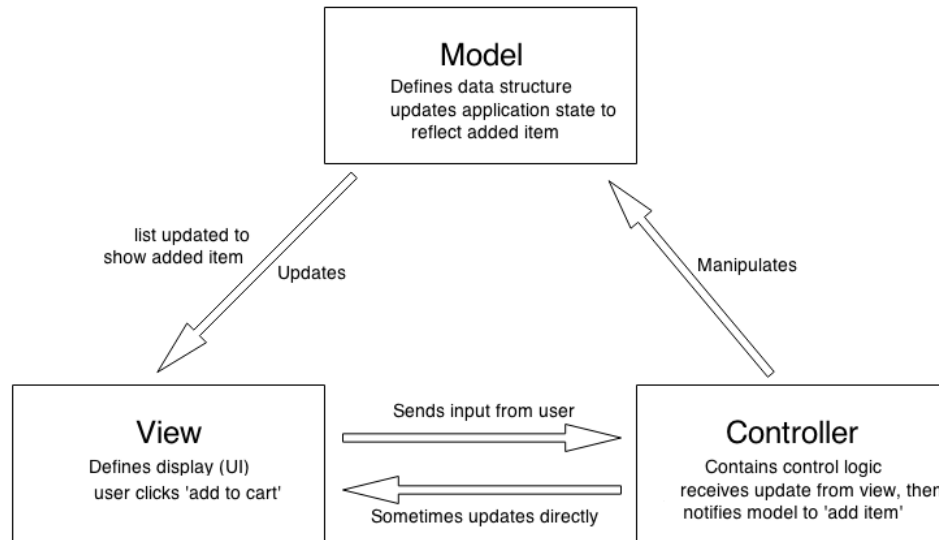


Figure 3.1: Model-View-Controller.

- Model

The model defines what data should be contain. If the state of this data changes, then the model will usually notify the view (so the display can change as needed) and sometimes the controller (if different logic is needed to control the updated view).

In our case the model would specify what data the list items should contain — items, price, name.

- View

The view defines how data should be displayed.

In our Basket list, the view would define how the list is presented to the user, and receive the data to display from the model.

- Controller

The controller contains logic that updates the model and/or view in response to input from the users of the interface.

Basket list could have input forms and buttons that allow to add or delete items. These actions require the model to be updated, so the input is sent to the controller, which then manipulates the model as appropriate, which then sends updated data to the view.

It may be need it just the update of view display for data in a different format, as an example change the item order to alphabetical or lowest to highest price. In this case the controller could handle this directly without needing to update the model.

## 2. Object Oriented;

Object can be anything including physical and conceptual. Like suppliers, products are of physical type of objects and conceptual like user, software.

Class is a collection of similar types of objects and also includes attributes (properties of objects) and operations (methods) those are used to define actions that can be performed with the help of objects.

In object oriented approach, following main features are included:

- Encapsulation

Encapsulation is a feature of object oriented approach, which indicates data hiding. We can hide data that we don't want to make accessible. Objects can be grouped into classes. Classes represent a set of objects that must have a certain common attributes and behaviours. All these attributes and behaviours are bounded in a class.

- Abstraction

Abstraction is a feature of object oriented approach, in which we can provide accessibility of some specific variables and method for users according to their relevancy. If we want to provide accessibility to user to fetch personal details of theirs only, we can do it by using the concept of abstraction.

- Inheritance

Inheritance allows inheriting a class in other subclasses. A derived or subclass can inherit the attributes and methods of parent class. This makes our code reusable, which is the most important advantage of an object oriented approach. This concept creates relationship between parent and child classes.

In PHP when we inherit one class in another, we have to use the keyword extends.

- Polymorphism

In object oriented approach polymorphism represents different behaviour for the same operation. Polymorphism can be static and dynamic.

In static polymorphism more than one function having same name but different number of arguments, type of arguments and sequence of arguments

Polymorphism can be dynamic also, in which functions with same function signature present in both parent and child class. In this process it will decided at the run time

## 3. Twig

Twig is a template engine for the PHP programming language. Its syntax originates from Jinja and Django templates. The initial version was created by Armin Ronacher. It's an open source product licensed under a BSD License and maintained by Fabien Potencier. Symfony PHP framework comes with a bundled support for Twig as its default template engine.

#### 4. JavaScript

JavaScript is a programming language that adds interactivity to the website (for example: responses when buttons are pressed or data entered in forms in a dynamic style).

#### 5. AJAX

AJAX is a set of Web development techniques using many Web technologies on the client side to create asynchronous Web applications.

#### 6. Symfony

Symfony is a Model-View-Controller (MVC) framework written in PHP that's aimed at building web applications.

Symfony an MVC implementation in PHP, it also integrates a lot of objects that facilitate the development of web applications — and integrates them all with a coherent syntax. It is a code generation, easy templating, internationalization, caching, automated form validation, and Ajax, are among the most appreciated symfony features. Developing an application with symfony is slightly different than building it with any other framework, or without any framework at all. It's faster and more productive.

## Part IV

# System analysis





# 4

## System analysis

The systems analysis is a stage of process developing information. This serves to set the stage and bound the project. Working on concepts such as data implementation, interfaces, and roles, you will describe in details whom and what would the WEB application will do.

Object-oriented analysis is the process of analyzing a task, to develop a conceptual model that can then be used to complete the task. The models describe computer software that could be used to satisfy a set of customer-defined requirements

Use case are use in this project to describe each operation. Each use case provides one or more sequence that describe how the system should interact with the users called actors to achieve a specific function. Actors are end users: user, managers, account and super user (General Manager).

### **Functional Requirements**

This chapter presents what the system is expecting to do and also how it in different scenarios, in the Use Cases

System requirement:

In case user login the WEB application will provide the following function: \* the user can see supplier's details; \* the user can see list of products from a supplier; \* the user can see if any other supplier provides the same product at different price; \* the user can send a request for a quotation; \* the user can add a comment to a product; \* the user can add a review to a supplier;

In case manager user login the WEB application will provide the same function as a regular user and in addition will have the following function:

- do what a regular user does;
- add new supplier;
- update details of a supplier;
- to be able to add/update/remove a product from a supplier;
- to update the stock on hand;
- to create order proposals based on the stock requirements;
- transfer an order to a related user (General Manager and Account Manager);
- follow up with a supplier on orders;
- to update the price for the items;
- to add, remove items from the list;
- to modified the contact details;
- once an order it is place to update the delivery date and the due date for the payment;
- see the stock level;

In case manager user login the WEB application will provide the same function as a regular user and in addition will have another function that will give permission to approve/deny/place on hold a PO;

Account Manager can do what the general manager. In addition the general manager and the account manager – they both must agree before an order is placed. When an order is approved then automatically get an PO number, send to the supplier a copy of the POs, and confirm to the manager that the order has been place

## 4.1 User login: - Use case

**Title:** This use case describes how a user logs into the WEB application.

**Actor :** any user;

**Scenario :**

This use case starts when an actor wishes to log into the WEB application.

- The system requests that the actor enter username and password.
- The actor enters username and password.
- The system validates the entered username and password and logs the actor into the system.

**Alternative Scenario**

- Invalid username or password;

If the actor enters an invalid username or password, the system displays an error message. The actor can choose to either return to the beginning of the scenario or cancel the login.

Pre-Conditions \* None

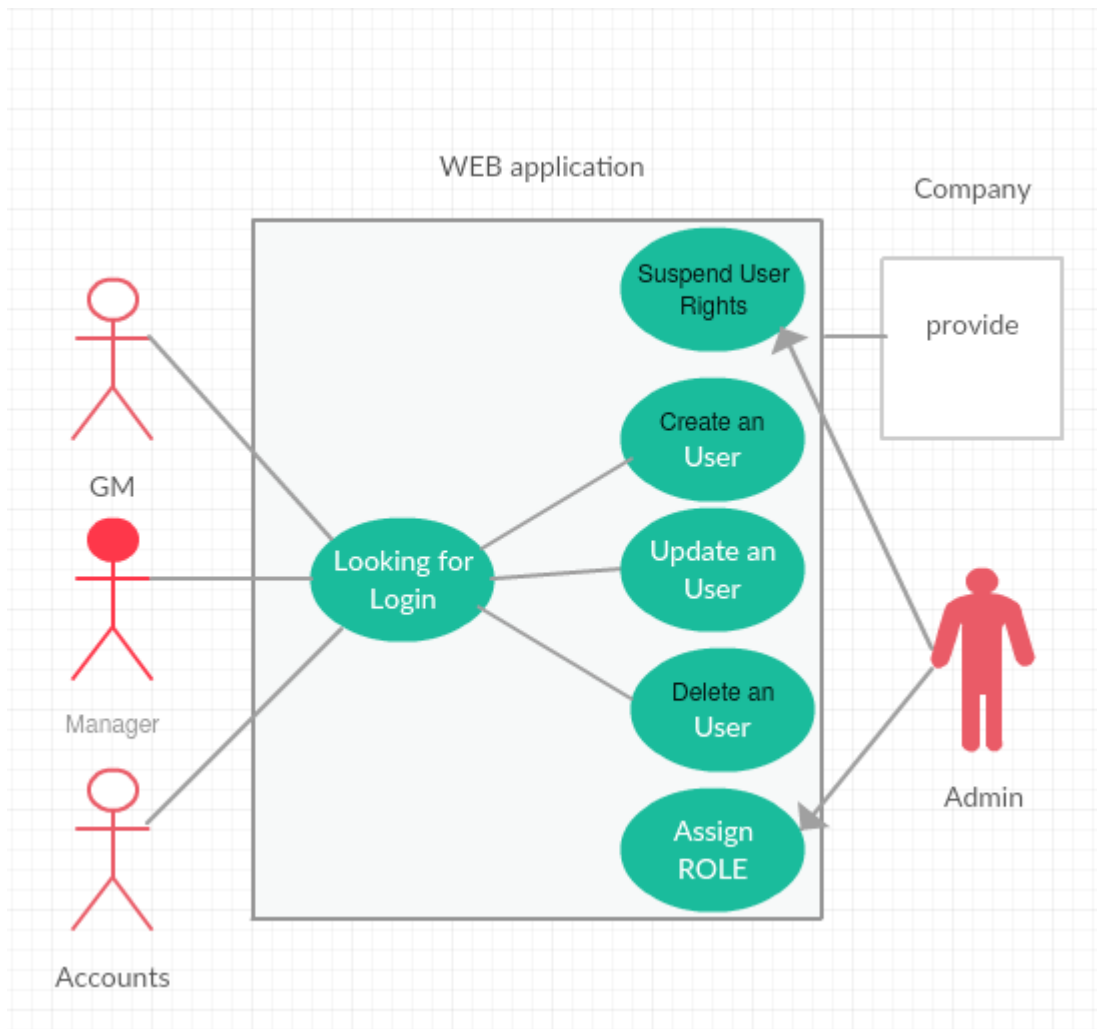


Figure 4.1: Before user login.



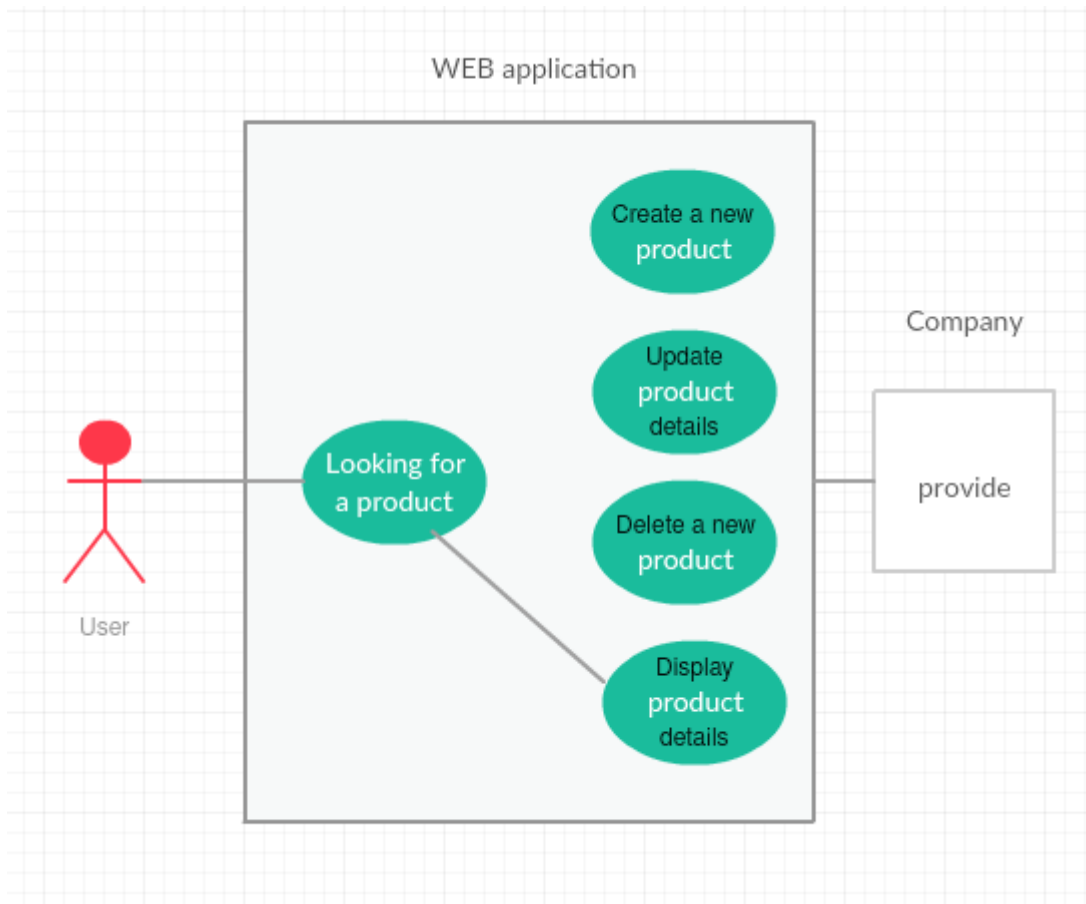


Figure 4.3: User login can see the products details.

**Title:** User login is looking for information to see product details;

**Actor:** User login;

**Scenario**

- user select from the menu the option product;
- the system displays a list of products;
- user can select a specific product;
- user will see the product description, some products details are not accessible;
- user logout;

**Alternative Scenario**

At any time before selecting a product, the user can cancel the request and the use case ends.

**3 User login can see the stock level**

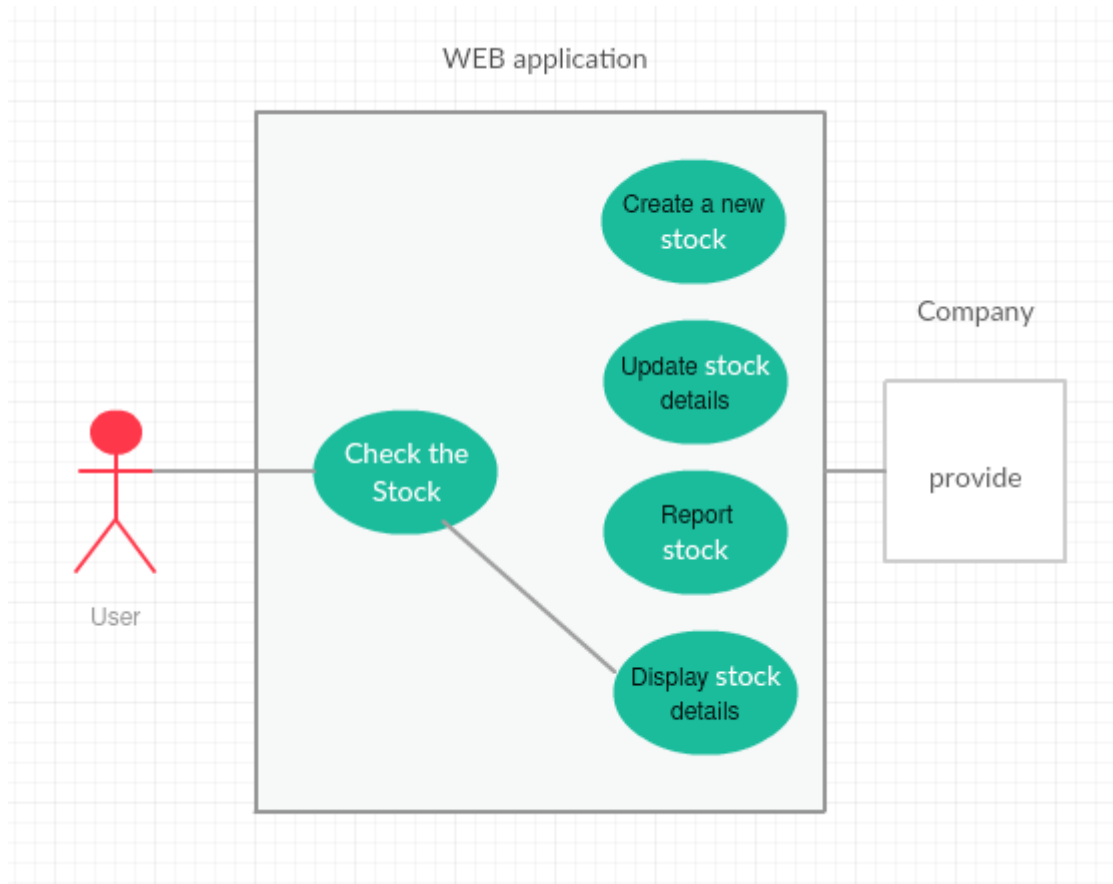


Figure 4.4: User login can see stock level.

**Title:** User login is looking to see stock level;

**Actor:** User login;

### Scenario

- user select from the menu the option stock level;
- the system displays a list of products stock level;
- user can select a specific product to see the stock level;
- user will see the product stock level;
- user logout;

### Alternative Scenario

At any time before selecting a product, the user can cancel the request and the use case ends.

#### 4. User login can request a quotation

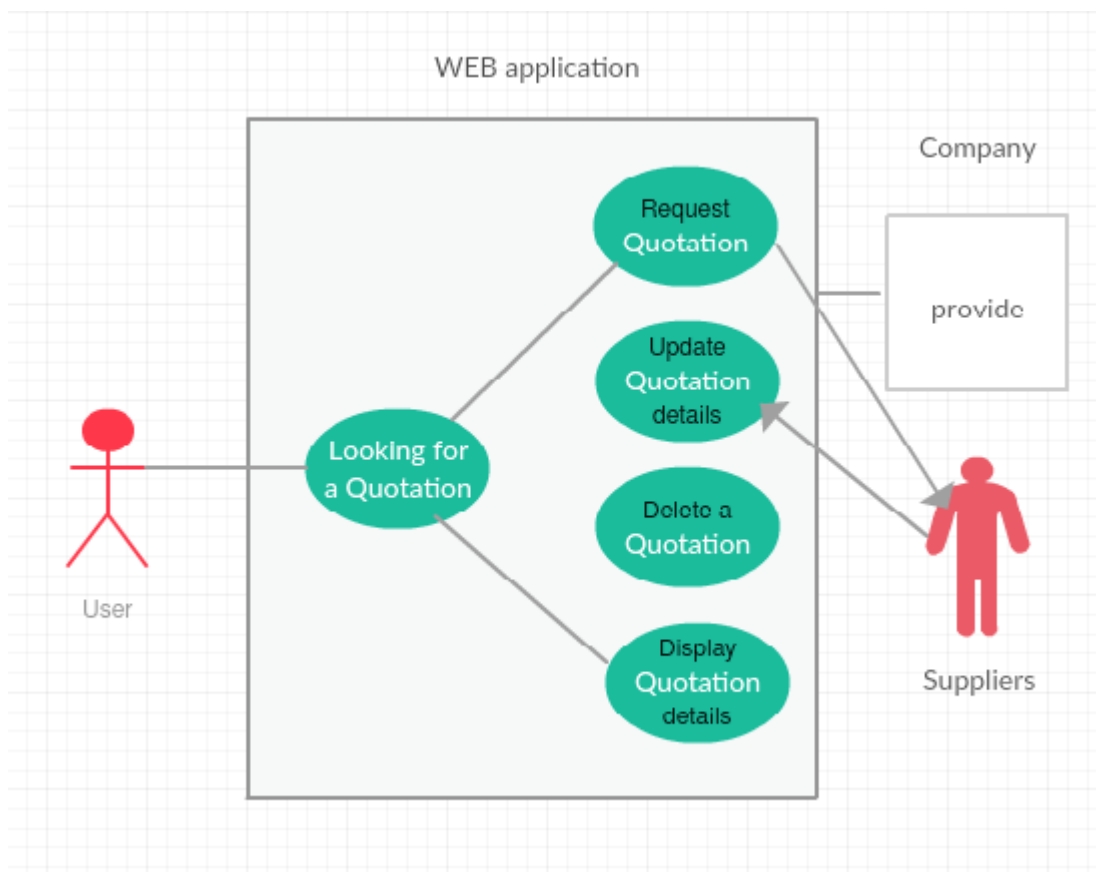


Figure 4.5: User login can send request for a quotation.

**Title:** User login is looking to get a quotation;

**Actor:** User login;

### Scenario

- user select from the menu the option quotation;
- the system displays a list of suppliers;

- user can select a specific supplier to be able to send a quote request;
- user will send a request via e-mail to the supplier to get a quote for a specific product;
- user logout;

#### Alternative Scenario

- At any time before selecting a supplier, the user can cancel the request and the use case ends.
- At any time before sending the request, the user can cancel the request and the use case ends.

#### 5. User is login to create an user

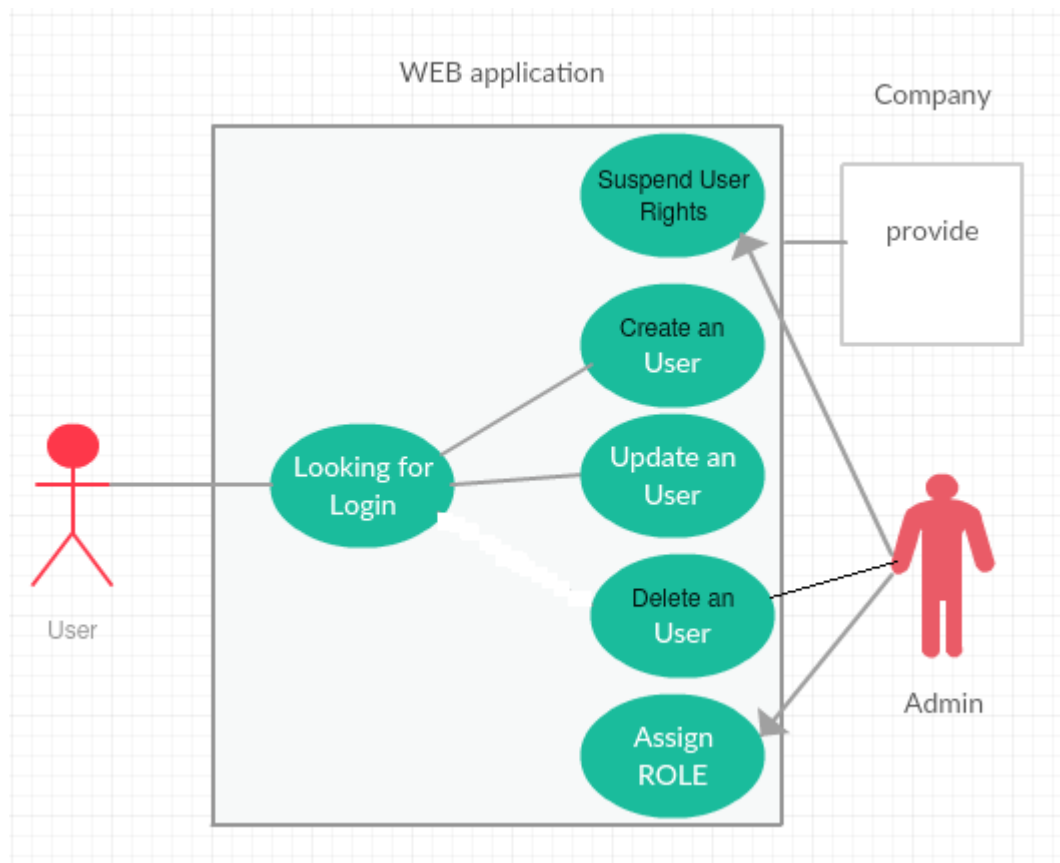


Figure 4.6: User login request for a login.

**Title:** User is looking to get user account;

**Actor:** Any user;

#### Scenario

- user select from the menu the option “Create user”;
- the system displays a list of user form;
- user can complete the form in order to create am user;
- user submit the request;



- user has created an user account;
- user logout;

#### Alternative Scenario

- At any time before sending the request, the user can cancel the request and the use case ends.

#### 6. User login can add a comment to a product

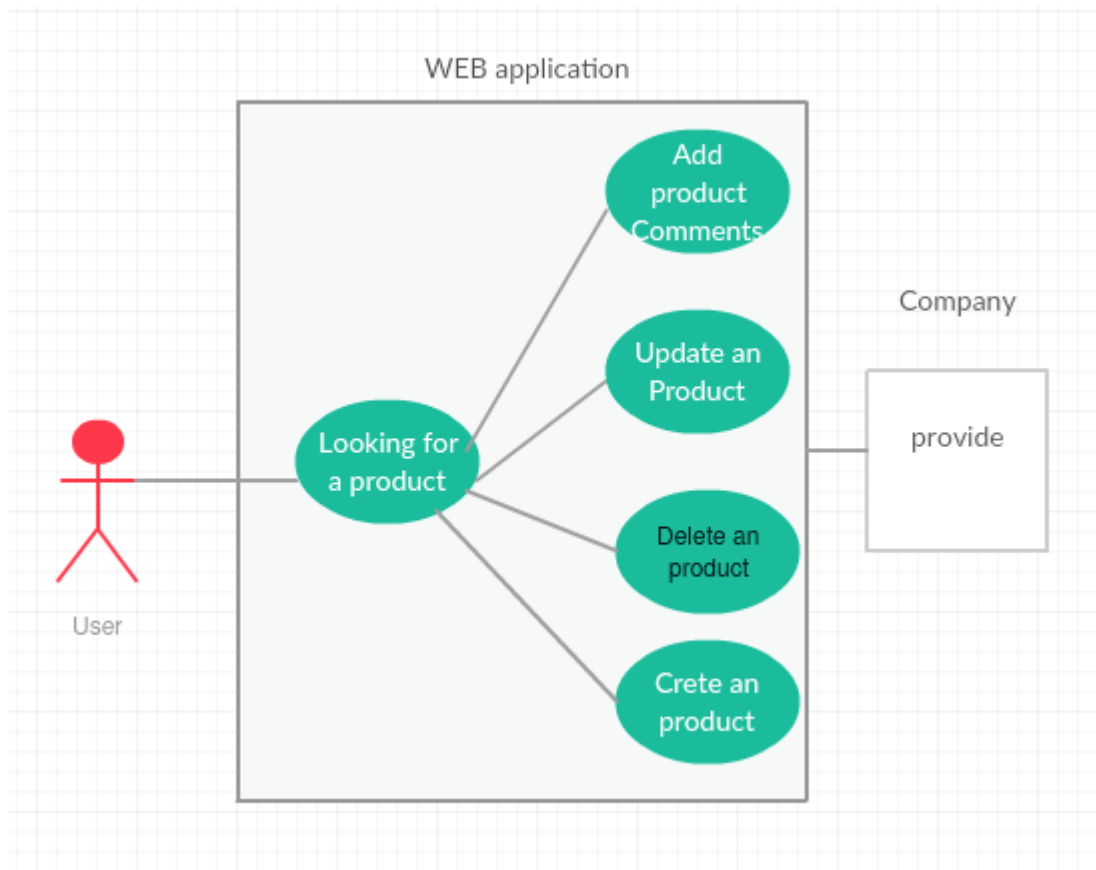


Figure 4.7: User login can add a comment to a product.

**Title:** User is looking to add a comment to a product;

**Actor:** User login;

#### Scenario

- user select from the menu the option product;
- the system displays a list of products;
- user select update comment for a specific product;
- user write the comment;
- user submit the comment;

- user logout;

#### Alternative Scenario

- At any time before completing the comment, the user can cancel the request and the use case ends.

#### 7. User login can add a comment to a product

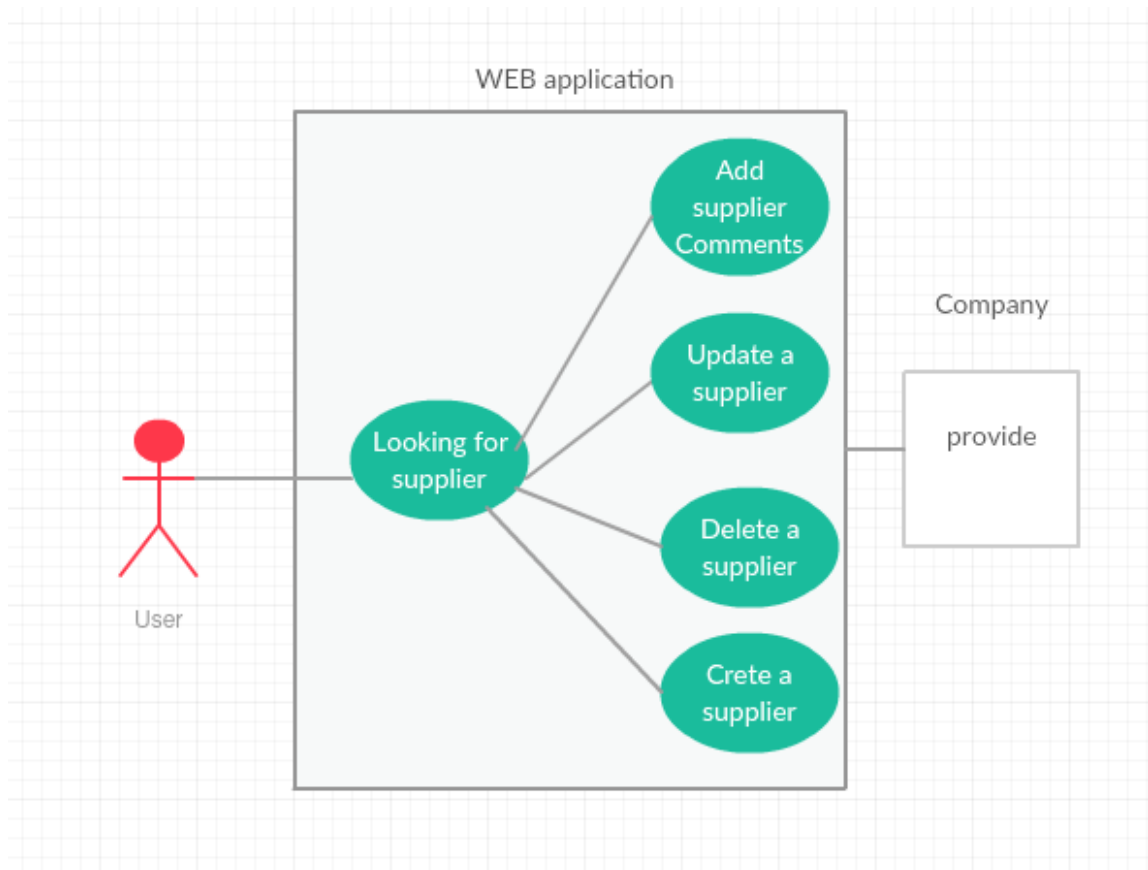


Figure 4.8: User login can add a comment to a supplier.

**Title:** User is looking to add a comment to a supplier;

**Actor:** User login;

#### Scenario

- user select from the menu the option supplier;
- the system displays a list of supplier;
- user select update comment for a specific supplier;
- user write the comment;
- user submit the comment;

- user logout;

#### Alternative Scenario

- At any time before completing the comment, the user can cancel the request and the use case ends.

## 4.2 Manager login: - Use case

### 1. Supplier interface

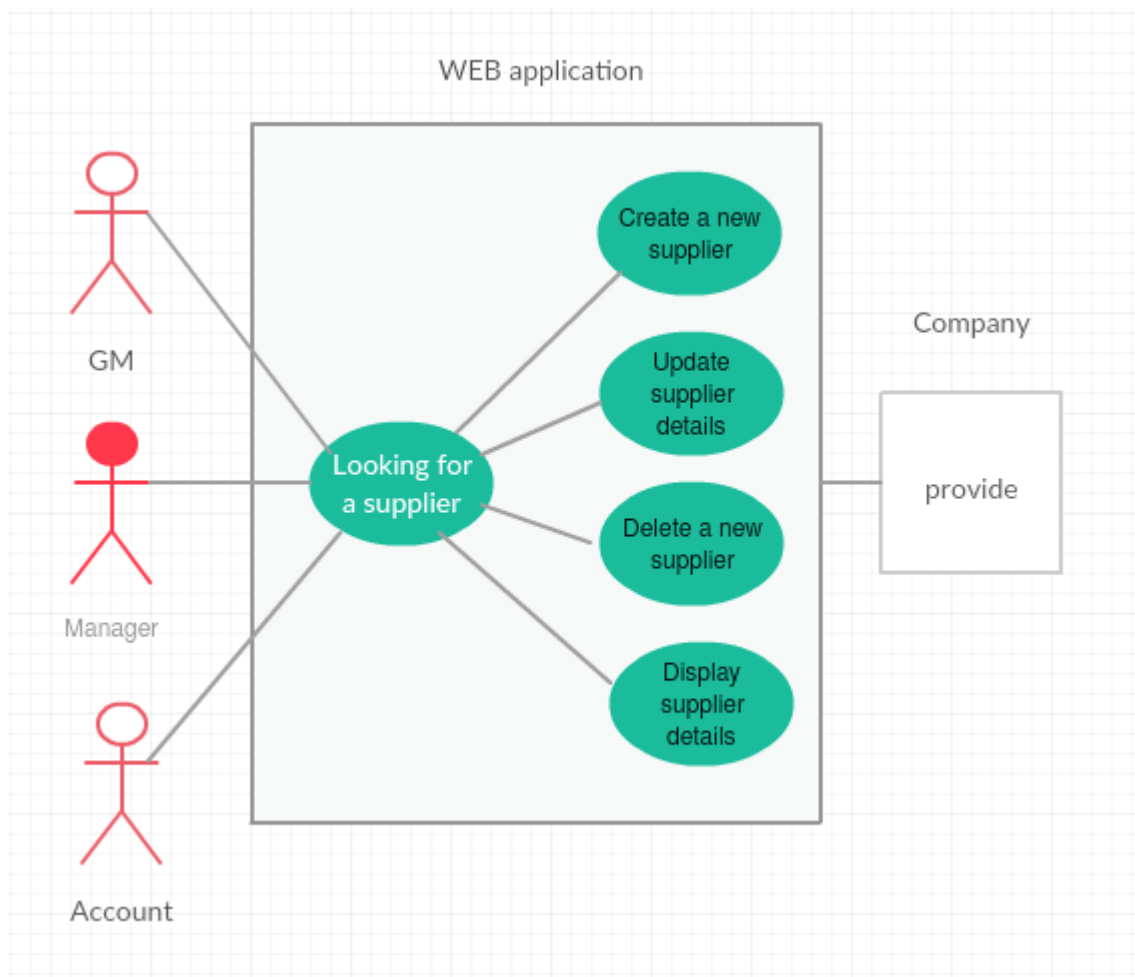


Figure 4.9: Manager login can add/update/delete/display a supplier.

**Title:** Manager user is looking to add/update/delete/display a comment to a supplier;

**Actor:** Manager login;

**Scenario**

add:

- Manager login;
- Manager login select from the menu button a supplier;
- Manager login select from the list of supplier the option New;
- Manager login complete the supplier details form;
- Manager login “Save” the details of the supplier;
- Manager login can see in the list of suppliers that a new supplier has been added;
- Manager logout;

update:

- Manager login
- Manager login select from the menu button a supplier;
- Manager login select from the list of supplier a supplier
- Manager select the option Update for the supplier selected;
- Manager login updates the supplier details form;
- Manager login "Save" the details of the supplier;
- Manager login can see in the list of suppliers the changes of supplier details;
- Manager logout;

delete: \* Manager login \* Manager login select from the menu button a supplier; \* Manager login select from the list of supplier a supplier \* Manager login select the option Delete; \* Manager login “Delete” the details of the supplier; \* Manager login can see in the list of suppliers that the supplier has been change; \* Manager logout;

display:

- Manager login;
- Manager login select from the menu button a supplier;
- Manager login select from the list of suppliers select the option display;
- Manager login can see the details of the supplier;
- Manager login can return to the list of the supplier;
- Manager logout;

## 2. Product interface

**Title:** Manager user is looking to add/update/delete/display details of a product;

**Actor:** Manager login;

**Scenario**

add:

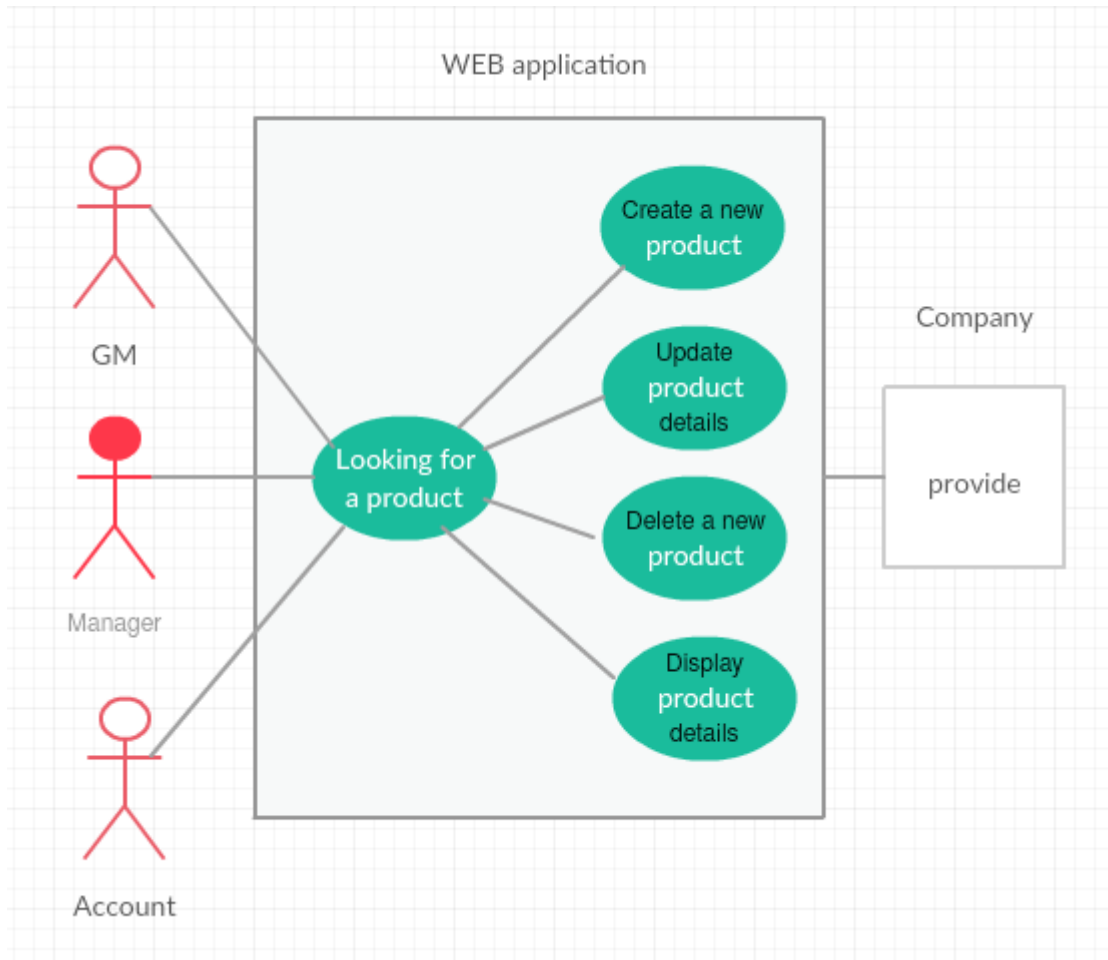


Figure 4.10: Manager login can add/update/delete/display details of a product.

- Manager login;
- Manager login select from the menu button a product;
- Manager login select from the list of product select the option “New”;
- Manager login complete the product details form;
- Manager login “Save” the details of the product;
- Manager login can see in the list of products that a new product has been added;
- Manager logout;

update:

- Manager login
- Manager login select from the menu button a product;
- Manager login select from the list of products a product;
- Manager select the option Update for the product selected;
- Manager login updates the product details form;
- Manager login “Save” the details of the product;
- Manager login can see in the list of product the changes of product details;
- Manager logout;

delete:

- Manager login;
- Manager login select from the menu button a product;
- Manager login select from the list of products a product;
- Manager login select the option Delete;
- Manager login “Delete” the details of the product;
- Manager login can see in the list of products that the product has been remove;
- Manager logout;

display:

- Manager login;
- Manager login select from the menu button a product;
- Manager login select from the list of products select the option display;
- Manager login can see the details of the product;
- Manager login can return to the list of the product;
- Manager logout;

### 3. Stock interface

**Title:** Manager user is looking to create/update/report/display stock;

**Actor:** Manager login;

**Scenario**

crete:

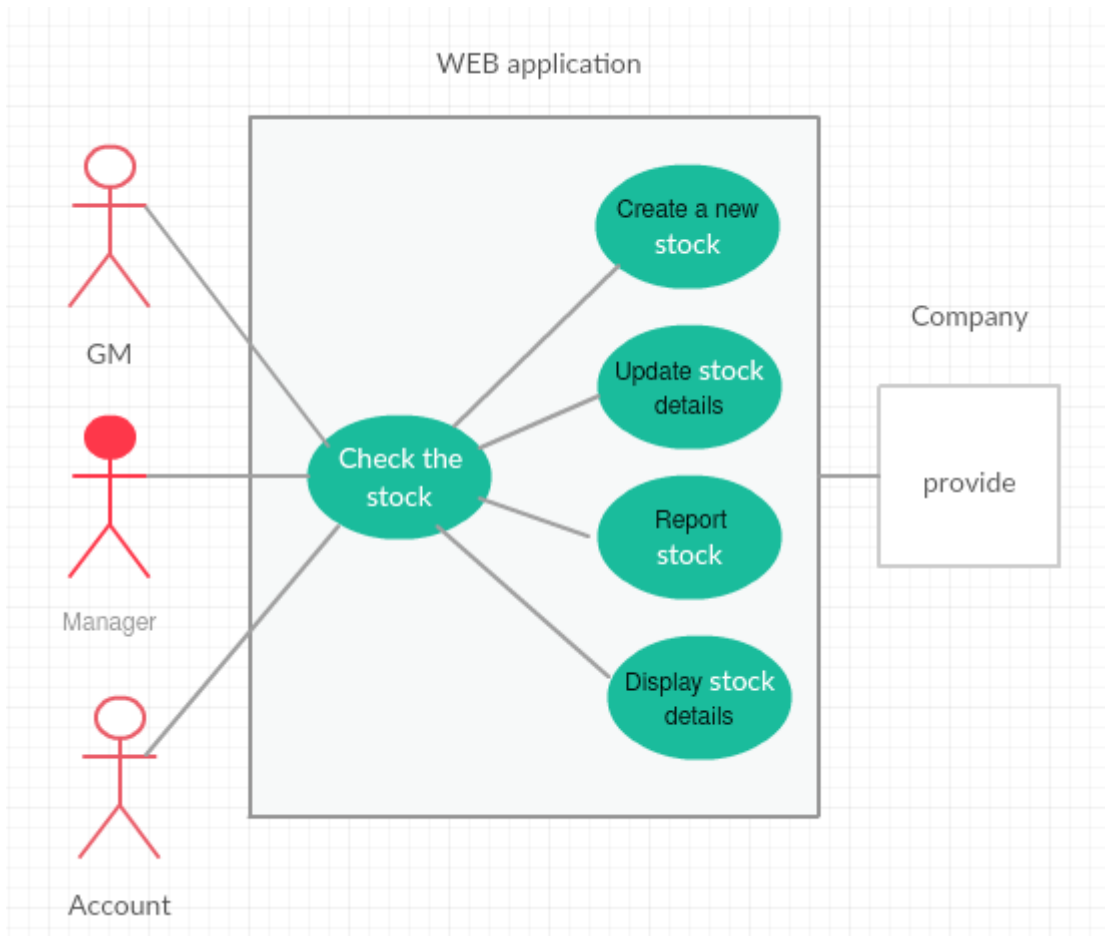


Figure 4.11: Manager login can create/update/report/display stock.

- Manager login;
- Manager login select from the menu button stock;
- Manager login select the option create stock;
- Manager login complete the stock using a form;
- Manager login “Save” the details of the stock;
- Manager login can see in the stock list that the stock level has been created;
- Manager logout;

update:

- Manager login;
- Manager login select from the menu button stock;
- Manager login select from the list of product the products that have been updated;
- Manager select the option Update for the stock;
- Manager login “Save” the details of the stock;
- Manager login can see in the stock list that stock has been updated;
- Manager logout;

report:

- Manager login;
- Manager login select from the menu button a stock;
- Manager login select from the form a period of time;
- Manager login select create the report;
- Manager login can see the report;
- Manager login can return to the stock level list;
- Manager logout;

display:

- Manager login;
- Manager login select from the menu button a stock;
- Manager login select from the form a period of time;
- Manager login select display;
- Manager login can see the stock display on the screen;
- Manager login can return to the stock level list;
- Manager logout;

#### 4. Quotation interface

**Title:** Manager user is looking to request/update/report/display Quotation;

**Actor:** Manager login;

**Scenario**

request:



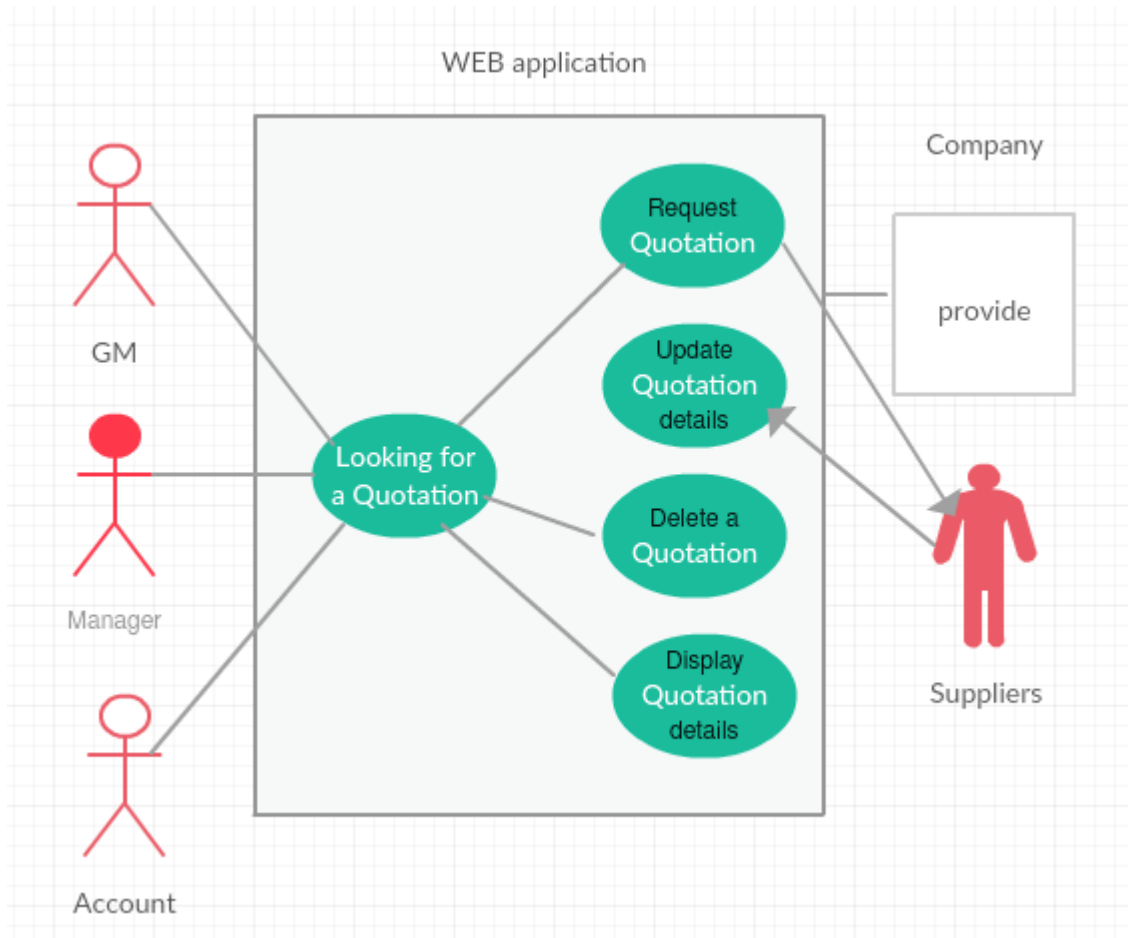


Figure 4.12: Manager login can request/update/report/display Quotation.

- Manager login;
- Manager login select from the menu button quotation;
- Manager login select the option request quotation;
- Manager login complete the request using a form;
- Manager login a supplier from a list of suppliers;
- Manager login “Send” the details of the supplier; \* Suppliers can: \* Select from the menu button quotation replay; \* Select the option request quotation replay; \* Supplier complete the request using a form; \* Supplier “Send” the details of the quote to the user (Manager...);
- Manager login can see in the request list that the quotation has been add to the list of quotation;
- Manager logout;

update:

- Manager login;
- Manager login select from the list of quotation the quotation that you want to be updated;
- Manager login select from the menu button quotation;
- Manager select the option Update for the quotation;
- Manager login “Save” the details of the quotation;
- Manager login can see in the quotation list that quotations has been updated;
- Manager logout;

report:

- Manager login;
- Manager login select from the menu button a quotation;
- Manager login select from the list a quotation;
- Manager login select create the report;
- Manager login can see the report;
- Manager login can return to the quotation level list;
- Manager logout;

display:

- Manager login;
- Manager login select from the menu button a quotation;
- Manager login select from the list a quotations;
- Manager login select quotation;
- Manager login can see the quotation display on the screen;
- Manager login can return to the quotations list;
- Manager logout;

## 5. Orders interface

**Title:** Manager user is looking to add/update/delete/display details of a Orders;

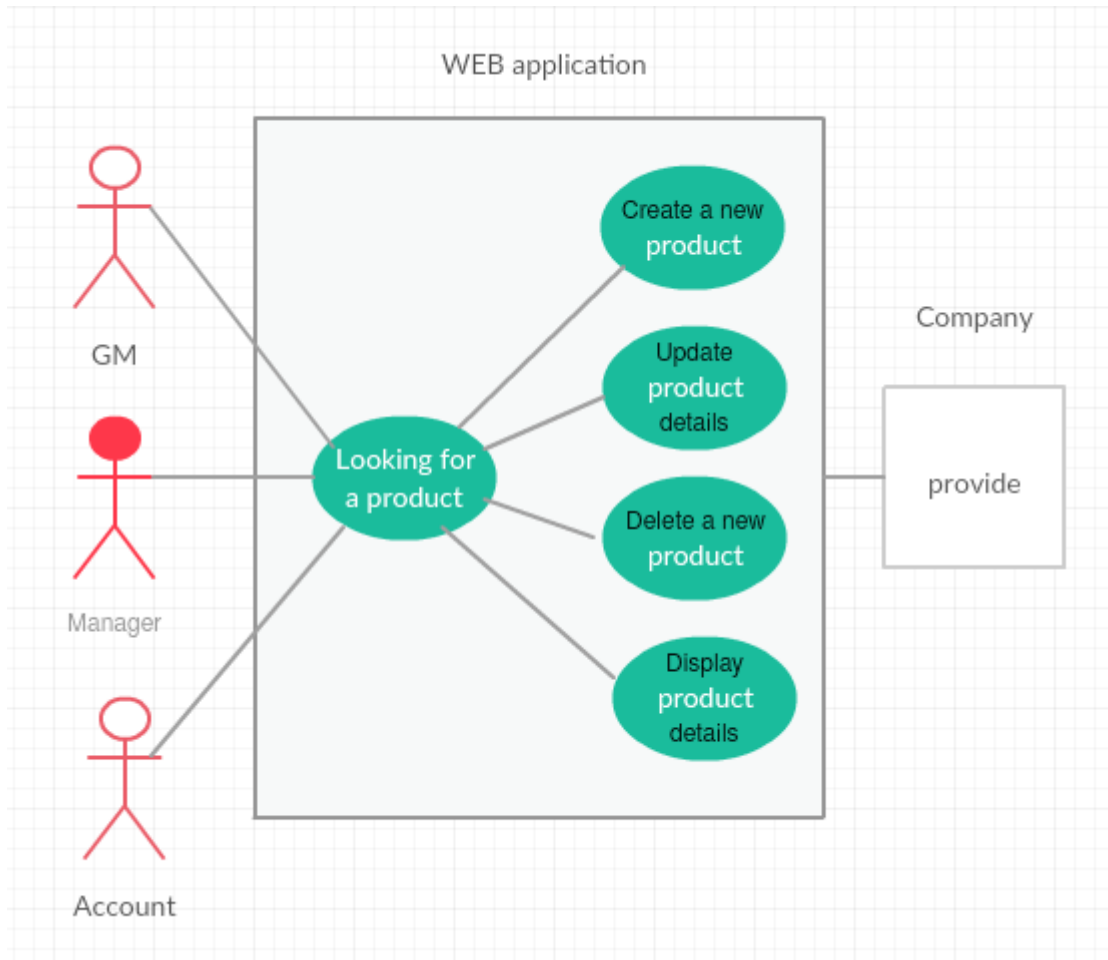


Figure 4.13: Manager login can add/update/delete/display details of a Orders.

**Actor:** Manager login;

**Scenario**

add:

- Manager login;
- Manager login select from the menu button a orders;
- Manager login select from the list of product select the option New;
- Manager login complete the order details form;
- Manager login “Save” the details of the order;
- Manager login can see in the list of orders that a new order has been added;
- Manager logout;

update:

- Manager login;
- Manager login select from the menu button an orders;
- Manager login select from the list of orders an order;
- Manager select the option Update for the order selected;
- Manager login updates the Orders details form;
- Manager login “Save” the details of the Orders;
- Manager login can see in the list of Orders the changes of Orders details;
- Manager logout;

delete:

- Manager login;
- Manager login select from the menu button an orders;
- Manager login select from the list of orders an order;
- Manager login select the option “Delete”;
- Manager login delete the details of the order;
- Manager login can see in the list of Orders that the Order has been remove;
- Manager logout;

display:

- Manager login
- Manager login select from the menu button an orders;
- Manager login select from the list of orders select the option display;
- Manager login can see the details of the orders;
- Manager login can return to the list of the orders;
- Manager logout;

## Part V

# Database design



# 5

## Database design

In today's fast-moving environment with short lead times, getting the right goods, delivered on time, at the agreed price is important. Inevitably without a formal system in place you will always need the goods in a hurry in order to complete a job. You can also record your supplier details against your purchase orders. Any discrepancies will be visually so you can see if you have been overcharged.

The objective of this assessment is to design the full data requirements required by the material control for hotels system.

This must include:

- Product details including details and inventory.
- Orders which will include the details in regards list of PO, orders, orders details.
- Suppliers with all details, including address, contact details and bank details.
- Stock details
- Quotation request details and suppliers responses details.
- User details including name, address, phone number, e-mail, and specialisation and log in details.

## 5.1 Product details

The table will contain details about a product used in the hotel including its name, price, description, and ordering options. It also includes a link to the offer listing page, which includes offers from sellers as well and allows users to quickly make price comparisons.

Product details	
<code>`id` int(11) NOT NULL,</code>	Product ID
<code>`product_Name` varchar(255) COLLATE utf8_unicode_ci NOT NULL,</code>	Product Name
<code>`product_Type` varchar(255) COLLATE utf8_unicode_ci NOT NULL,</code>	Product Type
<code>`suppliers_ID` int(11) NOT NULL,</code>	Suppliers ID
<code>`product_Description` varchar(255) COLLATE utf8_unicode_ci NOT NULL</code>	Product Description

Figure 5.1: Product details.



Product Inventory	
<code>`id` int(11) NOT NULL,</code>	id
<code>`product_ID` int(11) NOT NULL,</code>	Product ID
<code>`product_Counting_Group` varchar(255) COLLATE utf8_unicode_ci NOT NULL,</code>	Product Counting Group
<code>`product_Unit` varchar(255) COLLATE utf8_unicode_ci NOT NULL,</code>	Product Unit
<code>`quality_Unit` int(11) NOT NULL,</code>	Quality per Unit
<code>`price_Unit` double NOT NULL,</code>	Price per Unit
<code>`product_Description` varchar(255) COLLATE utf8_unicode_ci NOT NULL,</code>	Product Description
<code>`URL` varchar(255) COLLATE utf8_unicode_ci NOT NULL,</code>	URL
<code>`estimated_Delivery_Time` varchar(255) COLLATE utf8_unicode_ci NOT NULL</code>	Estimated Delivery

Figure 5.2: Product Inventory .

## 5.2 PO - purchase order

A purchase order, or PO, is an document issued by the Hotel to pay the seller for the sale of specific products or services to be delivered in the future. A PO specifies: quantity purchased, product or service purchased, price per unit, delivery date, delivery location, payment terms, such as on delivery or in 30 days

Each PO has a unique number associated with it that helps both buyer and seller track delivery and payment.

Orders will contain general indormation about whom and when en PO have been place.

Where the Order will hold general information about the product order.

PO	
`id` int(11) NOT NULL,	ID
`hotel_ID` varchar(255) COLLATE utf8_unicode_ci NOT NULL,	Hotel ID
`suppliers_ID` int(11) NOT NULL,	Suppliers ID
`po_no` varchar(12) COLLATE utf8_unicode_ci NOT NULL,	Po No
`po_date` datetime NOT NULL,	PO date
`ship_to_address` varchar(255) COLLATE utf8_unicode_ci NOT NULL,	Ship to address
`total` double NOT NULL,	Total
`VAT` double NOT NULL,	VAT
`discount` varchar(255) COLLATE utf8_unicode_ci NOT NULL,	Discount
`discount_Value` double NOT NULL,	Discount Value
`shipment_Costs` double NOT NULL,	Shipment Costs
`TAX_rate` double NOT NULL,	TAX
`aprovedapproved` tinyint(1) NOT NULL	Approved

Figure 5.3: PO details.

Orders	
`id` int(11) NOT NULL,	Id
`orders_id` int(11) NOT NULL,	Orders ID
`suppliers_id` int(11) NOT NULL,	Suppliers ID
`user_id` int(11) NOT NULL,	User ID
`orders_date` datetime NOT NULL,	Orders Date
`delivery_date` datetime NOT NULL,	Delivery Date
`ship_address` varchar(255) COLLATE utf8_unicode_ci NOT NULL,	Shipping address
`orders_cost` double NOT NULL	Orders Cost

Figure 5.4: Orders.

Orders Detail	
<code>`id` int(11) NOT NULL,</code>	ID
<code>`orders_id` int(11) NOT NULL,</code>	Orders ID
<code>`product_id` int(11) NOT NULL,</code>	Product ID
<code>`product_quantity` int(11) NOT NULL,</code>	Product Quantity
<code>`quality_Unit` int(11) NOT NULL,</code>	Quality Unit
<code>`price_Unit` double NOT NULL,</code>	Price Unit
<code>`suppliers_id` int(11) NOT NULL</code>	Suppliers ID

Figure 5.5: Orders details.

### 5.3 Quotation

Quotation refers to stock quote. A stock quote is an estimate of price or a price at which one party is willing to buy a certain number of shares of stock from the other. A quotation consists of a bid price and an ask price. A Quotation specified who request the price, the product name, and description, date of request.

To be able to get a quote in real time the suppliers can access the WEB application and respond to the request. In this case the information stored are: supplier ID, product group, quantity, price, description and a link to the product if exist online.

Quotation	
<code>`id` int(11) NOT NULL,</code>	Id
<code>`quotation no` int(11) NOT NULL,</code>	Quotation number
<code>`user id` int(11) NOT NULL,</code>	User id
<code>`suppliers id` int(11) NOT NULL,</code>	Suppliers id
<code>`product Name` varchar(255) COLLATE utf8_unicode_ci NOT NULL,</code>	Product name
<code>`product Description` varchar(255)</code>	Product Description
<code>`request on` datetime NOT NULL</code>	Request on the date

Figure 5.6: Quotation.

Supplier Quotation	
<code>`id` int(11) NOT NULL,</code>	Id
<code>`quotation no` int(11) NOT NULL,</code>	Quotation no
<code>`supplier id` int(11) NOT NULL,</code>	Supplier Id
<code>`product Counting Group` varchar(255) COLLATE utf8_unicode_ci NOT NULL,</code>	Product Counting Group
<code>`product Unit` varchar(255) COLLATE utf8_unicode_ci NOT NULL,</code>	Product Unit
<code>`quality Unit` varchar(255) COLLATE utf8_unicode_ci NOT NULL,</code>	Quality Unit`
<code>`price Unit` double NOT NULL,</code>	Price Unit
<code>`product Description` varchar(255) COLLATE utf8_unicode_ci NOT NULL,</code>	Product Description
<code>`URL` varchar(255) COLLATE utf8_unicode_ci NOT NULL,</code>	`URL`
<code>`estimated Delivery Time` varchar(255) COLLATE utf8_unicode_ci NOT NULL</code>	Estimated Delivery Time

Figure 5.7: Supplier quotation.

## 5.4 Stock

The stock of the hotel is the amount of goods, such as parts, materials, and finished products, that a company has available at a particular time. They are depleting their stock on hand before ordering new inventory. So the information kept on the table are:

Stock	
<code>`id` int(11) NOT NULL,</code>	Id
<code>`stock_date` datetime NOT NULL,</code>	Stock take on the date
<code>`product_ID` int(11) NOT NULL,</code>	Product ID
<code>`quantity` double NOT NULL,</code>	Quantity
<code>`suppliers_ID` int(11) NOT NULL</code>	Suppliers ID

Figure 5.8: Stock.

## 5.5 Supplier

A supplier is a person or entity that is the source for goods or services. The information store when you come to this are huge. You don't want that all information store in regards to your supplier to be seen. In this case the information will be distributed in different table, each one having specific information: supplier details, supplier address, contact details for the supplier and the bank details as follow:

Supplier details:

Supplier Details	
<code>`id` int(11) NOT NULL,</code>	Id
<code>`suppliers_id` int(11) NOT NULL,</code>	Suppliers Id
<code>`susuppliers_Name` varchar(255) COLLATE utf8_unicode_ci NOT NULL,</code>	Susuppliers Name
<code>`suppliers_business_type` varchar(255) COLLATE utf8_unicode_ci NOT NULL,</code>	Suppliers Business Type
<code>`suppliers_web` varchar(150) COLLATE utf8_unicode_ci NOT NULL</code>	Suppliers web

Figure 5.9: Supplier details.

Supplier address:

Supplier contact person details:

Supplier contact:

Suppliers address	
<code>`id` int(11) NOT NULL,</code>	Id
<code>`suppliers_ID` int(11) NOT NULL,</code>	Suppliers ID
<code>`suppliers Address` varchar(255) COLLATE utf8_unicode_ci NOT NULL,</code>	Suppliers Address
<code>`suppliers Region` varchar(255) COLLATE utf8_unicode_ci NOT NULL</code>	Suppliers Region

Figure 5.10: Supplier address.

Supplier Contact	
<code>`id` int(11) NOT NULL,</code>	Id
<code>`suppliers_id` int(11) NOT NULL,</code>	Suppliers id
<code>`suppliers contact Name` varchar(255) COLLATE utf8_unicode_ci NOT NULL,</code>	Suppliers Contact Name
<code>`suppliers_phone` varchar(255) COLLATE utf8_unicode_ci NOT NULL,</code>	Suppliers phone
<code>`suppliers_mobile` varchar(20) COLLATE utf8_unicode_ci NOT NULL,</code>	Suppliers mobile
<code>`suppliers e_mail` varchar(50) COLLATE utf8_unicode_ci NOT NULL,</code>	Suppliers e_mail
<code>`suppliers_fax` varchar(20) COLLATE utf8_unicode_ci NOT NULL,</code>	Suppliers fax
<code>`suppliers_web` varchar(150) COLLATE utf8_unicode_ci NOT NULL</code>	Suppliers Web

Figure 5.11: Supplier contact person details.

Supplier bank details	
<code>`id` int(11) NOT NULL,</code>	Id
<code>`suppliers_id` int(11) NOT NULL,</code>	Suppliers Id
<code>`suppliers_Bank_Name` varchar(255) COLLATE utf8_unicode_ci NOT NULL,</code>	Suppliers Bank Name
<code>`suppliers_Acc_no` varchar(255) COLLATE utf8_unicode_ci NOT NULL</code>	Suppliers Account No
<code>`suppliers_IBAN` varchar(255) COLLATE utf8_unicode_ci NOT NULL</code>	Suppliers IBAN
<code>`suppliers_BIC` varchar(255) COLLATE utf8_unicode_ci NOT NULL,</code>	Suppliers BIC
<code>`suppliers_Country` varchar(255) COLLATE utf8_unicode_ci NOT NULL,</code>	Suppliers Country
<code>`suppliers_Bank_Address` varchar(255) COLLATE utf8_unicode_ci NOT NULL,</code>	Suppliers Bank Address
<code>`suppliers_Swift` varchar(255) COLLATE utf8_unicode_ci NOT NULL,</code>	suppliers Swift
<code>`suppliers_Routing_Cod` varchar(255) COLLATE utf8_unicode_ci NOT NULL</code>	Suppliers Routing Cod

Figure 5.12: Supplier bank details.

## 5.6 User details

The User Details side of WEB application comes with two blocks: “User LOGIN” which will hold basic information about login and “User Details”: which will hold general information about each person which access the information.

UserLOGIN	
<code>`id` int(11) NOT NULL,</code>	ID
<code>`username` varchar(255) COLLATE utf8_unicode_ci NOT NULL,</code>	Username
<code>`password` varchar(255) COLLATE utf8_unicode_ci NOT NULL,</code>	Password
<code>`e_mail` varchar(255) COLLATE utf8_unicode_ci NOT NULL,</code>	<u>e_mail</u>
<code>`isActive` tinyint(1) NOT NULL,</code>	<u>isActive</u>
<code>`roles` longtext COLLATE utf8_unicode_ci NOT NULL COMMENT '(DC2Type:json_array)'</code>	Roles

Figure 5.13: User LOGIN.

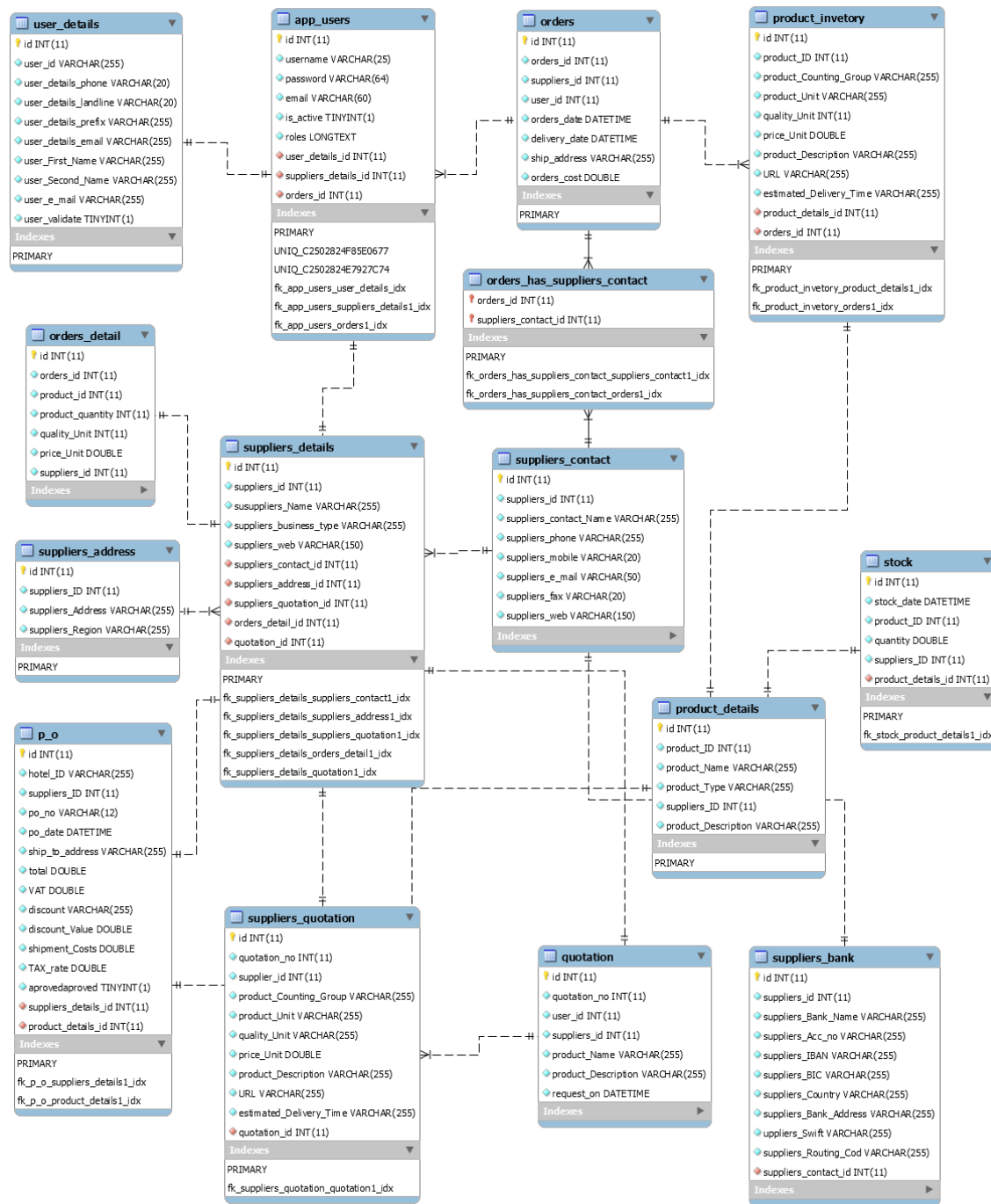


UserDetails	
<code>`id` int(11) NOT NULL,</code>	Id
<code>`user_id` varchar(255) COLLATE utf8_unicode_ci NOT NULL,</code>	User Id
<code>`user_details_phone` varchar(20) COLLATE utf8_unicode_ci NOT NULL,</code>	User details phone
<code>`user_details_landline` varchar(20) COLLATE utf8_unicode_ci NOT NULL,</code>	User details landline
<code>`user_details_prefix` varchar(255) COLLATE utf8_unicode_ci NOT NULL,</code>	User details prefix
<code>`user_details_email` varchar(255) COLLATE utf8_unicode_ci NOT NULL,</code>	User details email
<code>`user_First Name` varchar(255) COLLATE utf8_unicode_ci NOT NULL,</code>	User First Name
<code>`user_Second Name` varchar(255) COLLATE utf8_unicode_ci NOT NULL,</code>	User Second Name
<code>`user_e_mail` varchar(255) COLLATE utf8_unicode_ci NOT NULL,</code>	User e_mail
<code>`user_validate` tinyint(1) NOT NULL</code>	User_validate

Figure 5.14: User details.

## 5.7 EED Diagram

An entity-relationship model (ERM) is a theoretical and conceptual way of showing data relationships in software development. ERM is a database modeling technique that generates an abstract diagram or visual representation of a system's data that can be helpful in designing a relational database. For a better understanding between the table in the database, was used a graphical representation of entities and their relationships to each other.





## Part VI

# Implementation of System



# 6

## Implementation of System

The implementation phase involves putting the project plan into action. This is the time to meet the objectives of the project proposal. In this phase of the project life cycle: you follow the plan you've put together and handle any problems that come up.

The implementation phase is where your project work is done to produce the deliverable. On the same time you have to put the documentation together.

### 6.1 Entity classes

As described before, each entity classes are combined key/value classes that are managed by entity. In this project Supplier with the supplier details, Product with product details, Orders with orders details, user with user details classes are entity classes. These classes contain fields that are a union of the fields of the key and value classes that were defined earlier for each store.

For a better understanding it is use the entity “product\_details”. All other entities are following the same rules. The difference between the entity classes defined are just the key and value classes defined earlier is that the entity classes are not serializable

All the entity classes are saved in directory named **Entity** under **AppBundle**.

The Doctrine has to know what table name this entity should map to. On the same time the data types of each field it is store using annotation comments. The **use** statement and the definition of the namespace alias are define in each class.

```
namespace AppBundle\Entity;
```

```
use Doctrine\ORM\Mapping as ORM;
```

The first comment is for the class, stating that it is an ORM entity and mapping it to database table `product_details`:

```
/**
 * product_details
 *
 * @ORM\Table(name="product_details")
 * @ORM\Entity(repositoryClass="AppBundle\Repository\product_detailsRepository")
 */
class product_details
```

The data/variables inside a class (ex: `var $ product_Name ;`) are called ‘properties’. Add functions/methods to your class. In the same way that variables get a different name when created inside a class (they are called: properties) functions also referred to (by nerds) by a different name when created inside a class – they are called ‘methods’. A class’s methods are used to manipulate its own data / properties.

Annotations are used to declare the types and the lengths of each field.

```
/**
 * @var string
 *
 * @ORM\Column(name="product_Name", type="string", length=255)
 */
private $productName;
```

Each class created will contain an ‘id’ which is `AUTO_INCREMENT`.

```
/**
 * @ORM\Column(type="string", length=100)
 */
private $name;
```

Getter and setter functions

To be able to operate with a class we have to define - functions/methods: `get__product__name ()` and `set__product__name ()`. It is a convention that getter and setter names should match the property names. The getter and setter `product_names`, match the associated property name. This way, when other programmers want to use your objects, they will know that if you have a method/function called ‘`set__product__name ()`’, there will be a property/variable called ‘`product__name`’.

```
## Repository Classes ##
```



Once we create class and table which match the entity is easy to create, read, update and delete entries in the projects. Setting up the skeleton of our class is fairly simple once we figure out exactly what we need. In order to do this, we need the following functions:

- newAction - this is the function which will insert a new record to the table;
- delete - in the function which will delete the items on the table;
- editAction - in the function which will update the items on the table;
- showAction - this is the function which will show the information from the table;

Those seem pretty basic, but going through, we'll notice that a lot of them utilize some similar aspects, so we may have to create more classes. Here is what your class definition should look like. Notice that I made sure that the methods were created with the public keyword.

- public function newAction()

This function allows us to insert a new information into the database. As such we will require an additional argument to the name of the table. We will require a variable that corresponds to the values we wish to input. We can simply separate each value with a comma. Then, all we need to do is quickly check to see if our table exists, and then build the insert statement by manipulating our arguments to form an insert statement. Then we just run our query.

```
/**
 * Creates a new suppliers_detail entity.
 *
 * @Route("/new", name="suppliers_details_new")
 * @Method({"GET", "POST"})
 */
public function newAction(Request $request)
{
    $suppliers_detail = new Suppliers_details();
    $form = $this->createForm('AppBundle\Form\suppliers_detailsType', $suppliers_detail);
    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) {
        $em = $this->getDoctrine()->getManager();
        $em->persist($suppliers_detail);
        $em->flush($suppliers_detail);

        return $this->redirectToRoute('suppliers_details_show', array('id' => $suppliers_detail
    )

    return $this->render('suppliers_details/new.html.twig', array(
        'suppliers_detail' => $suppliers_detail,
        'form' => $form->createView(),
```

```
    ));  
}
```

As you can see, this function is a lot simpler than our rather complex select statement. Our delete function will actually be even simpler.

- public function delete()

This function simply deletes a row from our database. As such we must pass the table name and an optional where clause. The where clause will let us know if we need to delete a row or the whole table. If the where clause is passed, that means that entries that match will need to be deleted. After we figure all that out, it's just a matter of compiling our delete statement and running the query.

```
/**  
 * Deletes a suppliers_detail entity.  
 *  
 * @Route("/{id}", name="suppliers_details_delete")  
 * @Method("DELETE")  
 */  
public function deleteAction(Request $request, suppliers_details $suppliers_detail)  
{  
    $form = $this->createDeleteForm($suppliers_detail);  
    $form->handleRequest($request);  
  
    if ($form->isSubmitted() && $form->isValid()) {  
        $em = $this->getDoctrine()->getManager();  
        $em->remove($suppliers_detail);  
        $em->flush();  
    }  
  
    return $this->redirectToRoute('suppliers_details_index');  
}
```

On the same time it is created a for to delete the entity;

```
/**  
 * Creates a form to delete a suppliers_detail entity.  
 *  
 * @param suppliers_details $suppliers_detail The suppliers_detail entity  
 *  
 * @return \Symfony\Component\Form\Form The form  
 */  
private function createDeleteForm(suppliers_details $suppliers_detail)
```

```
{
    return $this->createFormBuilder()
        ->setAction($this->generateUrl('suppliers_details_delete', array('id' => $suppliers_detail
        ->setMethod('DELETE')
        ->getForm()
    ;
}
```

- public function editAction()

This function simply serves to update a row in the database with some new information. Because of the slightly more complex nature of it, it will come off as a bit larger and infinitely more confusing. Never fear, it follows much of the same pattern of our previous function. First it will use our arguments to create an update statement. It will then proceed to check the database to make sure that the tableExists. If it exists, it will simply update the appropriate row. The hard part, of course, comes when we try and create the update statement. Since the update statement has rules for multiple entry updating (IE – different columns in the same row via the cunning use of comma's), we will need to take that into account and create a way to deal with it. I have opted to pass the where clause as a single array. The first element in the array will be the name of the column being updated, and the next will be the value of the column. In this way, every even number (including 0) will be the column name, and every odd number will be the new value. The code for performing this is very simple, and is presented below outside the function:

```
/**
 * Displays a form to edit an existing suppliers_detail entity.
 *
 * @Route("/{id}/edit", name="suppliers_details_edit")
 * @Method({"GET", "POST"})
 */
public function editAction(Request $request, suppliers_details $suppliers_detail)
{
    $deleteForm = $this->createDeleteForm($suppliers_detail);
    $editForm = $this->createForm('AppBundle\Form\suppliers_detailsType', $suppliers_detail);
    $editForm->handleRequest($request);

    if ($editForm->isSubmitted() && $editForm->isValid()) {
        $this->getDoctrine()->getManager()->flush();

        return $this->redirectToRoute('suppliers_details_edit', array('id' => $suppliers_detail->getId
    }

    return $this->render('suppliers_details/edit.html.twig', array(
```

```
'suppliers_detail' => $suppliers_detail,
'edit_form' => $editForm->createView(),
'delete_form' => $deleteForm->createView(),
));
}
```

Now that we have that we've finished our last function, our simple CRUD interface for MySQL is complete. You can now create new entries, read specific entries from the database, update entries and delete things. Also, by creating and reusing this class you'll find that you are saving yourself a lot of time and coding. Ah, the beauty of object oriented programming.

Doctrine repositories offer us lots of useful methods, including:

- query for a single record by its primary key (usually "id"), used in this application to identify the product from the same suppliers.

```
$suppliers_detail = $repository->find(id);
```

- dynamic method names to find a single record based on a column value, used in this application to identify the supplier which provides a product

```
$suppliers_detail = $repository->findOneById(id); $suppliers_detail = $repository-
>findOneByName('Alex');
```

- find all suppliers\_detail, used in this application to identify the supplier which provides the same product;

```
$suppliers_detail = $repository->findAll();
```

- dynamic method names to find a group of products based on a value, as an example if you are looking to get all the products that cost 1 euro, you can use the following command:

```
$products = $repository->findByPrice(19.99);
```

- showAction - this is the function which will display the information from the table;

```
/**
 * Finds and displays a suppliers_detail entity.
 *
 * @Route("/{id}", name="suppliers_details_show")
 * @Method("GET")
 */
public function showAction(suppliers_details $suppliers_detail)
{
    $deleteForm = $this->createDeleteForm($suppliers_detail);

    return $this->render('suppliers_details/show.html.twig', array(
        'suppliers_detail' => $suppliers_detail,
```

```
        'delete_form' => $deleteForm->createView(),  
    ));  
}
```



# 7

## System design

### 7.1 User Interface

This chapter details how the WEB application laid out. The layout it is logical and easy to use, as it determine an interaction with the users, an it determine how an inetraction it is perform by the user. If the instruction are not clear, the the user won't be able to to perform the tasks given. So that may reflect to the pur quality of result from the user perspective of view.

#### 1. Home page

The home page will describe the purpose of this site to the user.

#### 2. User login:

The user Login support the Use case “**User Login**” This use case starts when an actor wishes to log into the WEB application. The system requests that the actor enter username and password, after the actor enters username and password. The system validates the entered username and password and logs the actor into the system.

#### 3. Access product details

Before the user is able to access the product details it is ask to login. If the login succeed the manager can select from the menu the option product

If the Manager select the option product, He can see the list of product from the database.From the list of product select the option “New” and complete the product details form.

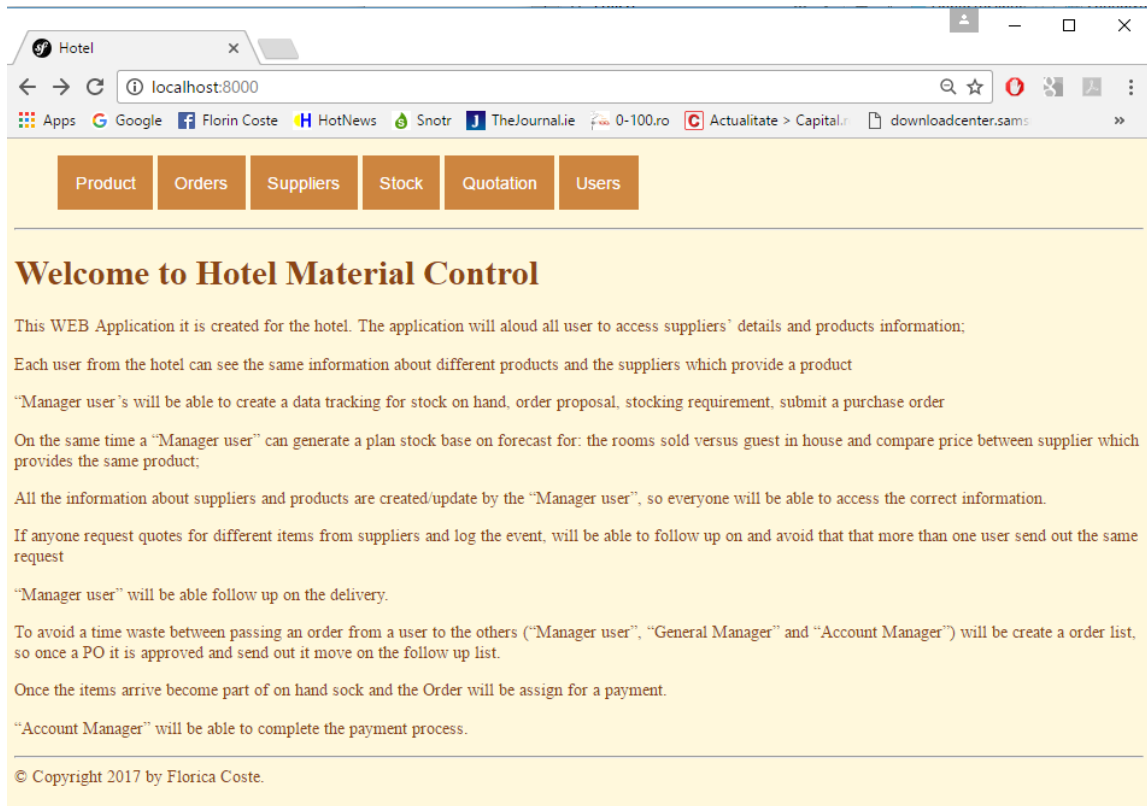


Figure 7.1: Home page.

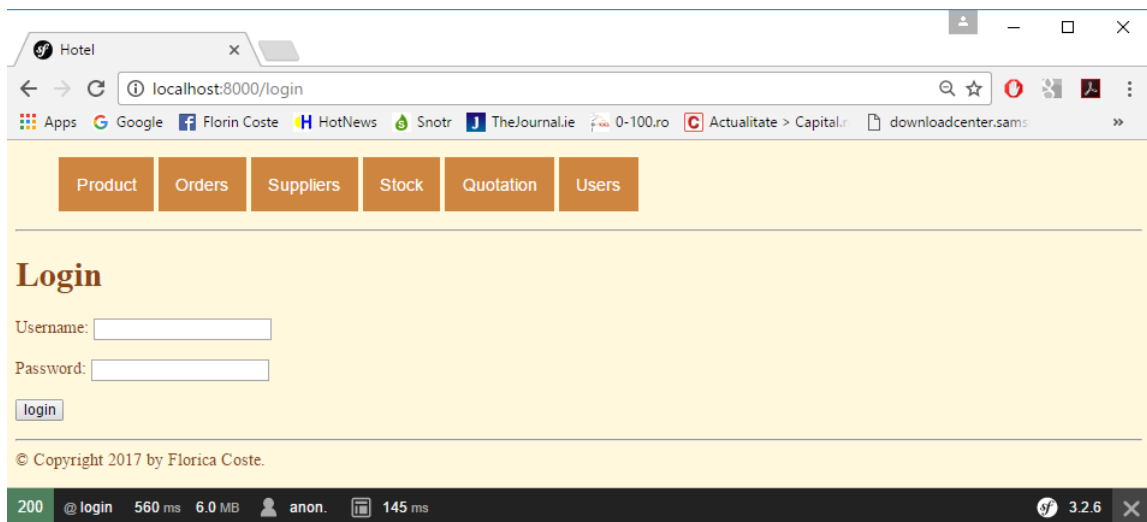


Figure 7.2: User login.



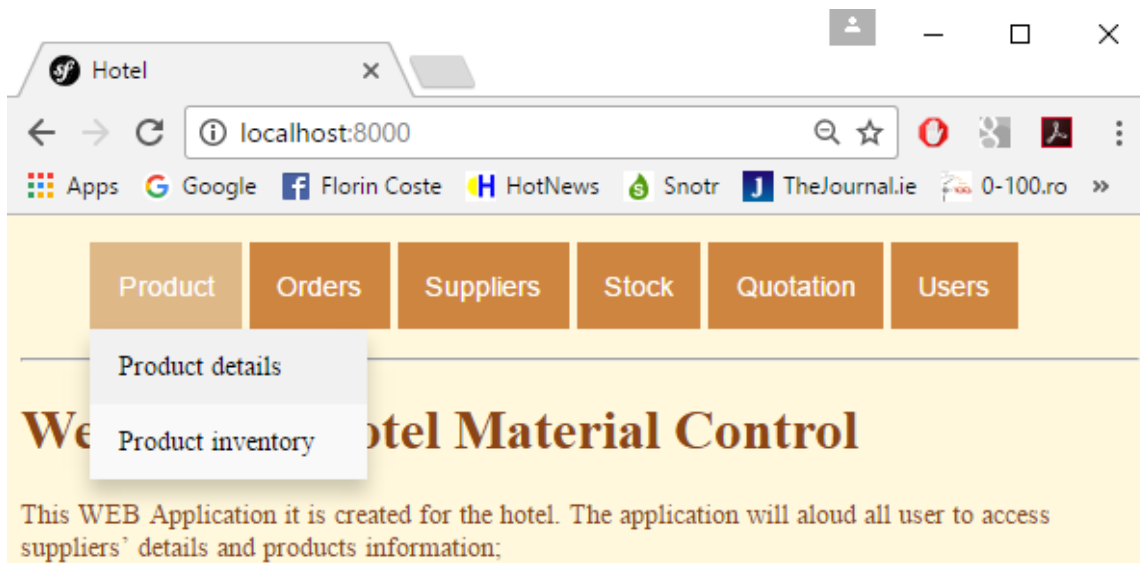
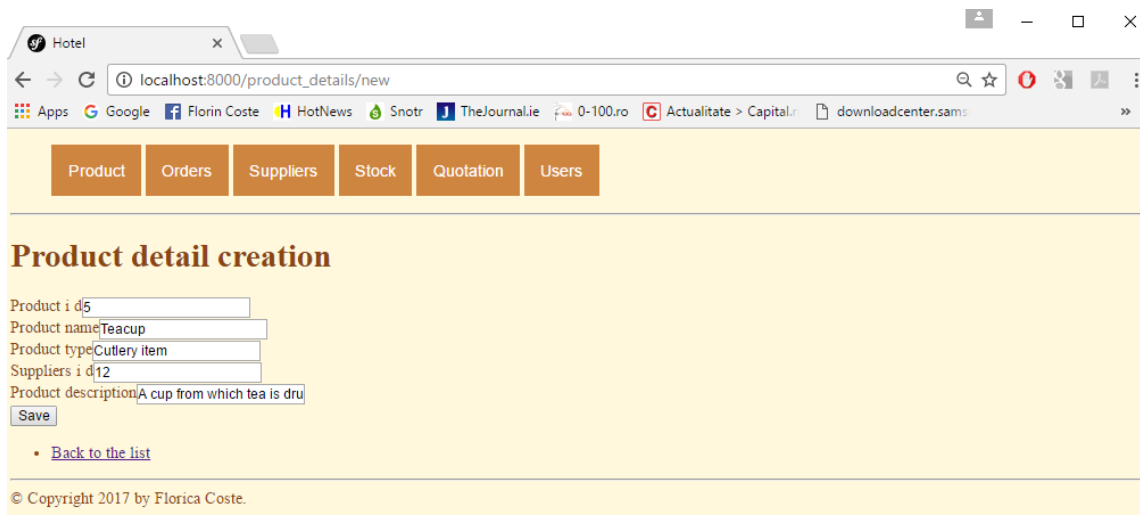


Figure 7.3: Manager login can add/update/delete/display details of a product.



If you “Save” the details of the product then you can see in the list of products that a new product has been added.

If the user would like to edit a Product can select from the list product a product and select the option Update:

Once the user updates the product details can “Save” the details of the product. Managers can see in the list of product the changes of product details;

If a manager wants to delete a product has to select from the list of products a product, Edit the product before he can select the option Delete.

If the user “Delete” the details of the product, then the changes can be seen on the list of products, so the product has been remove;



Figure 7.4: See a product.

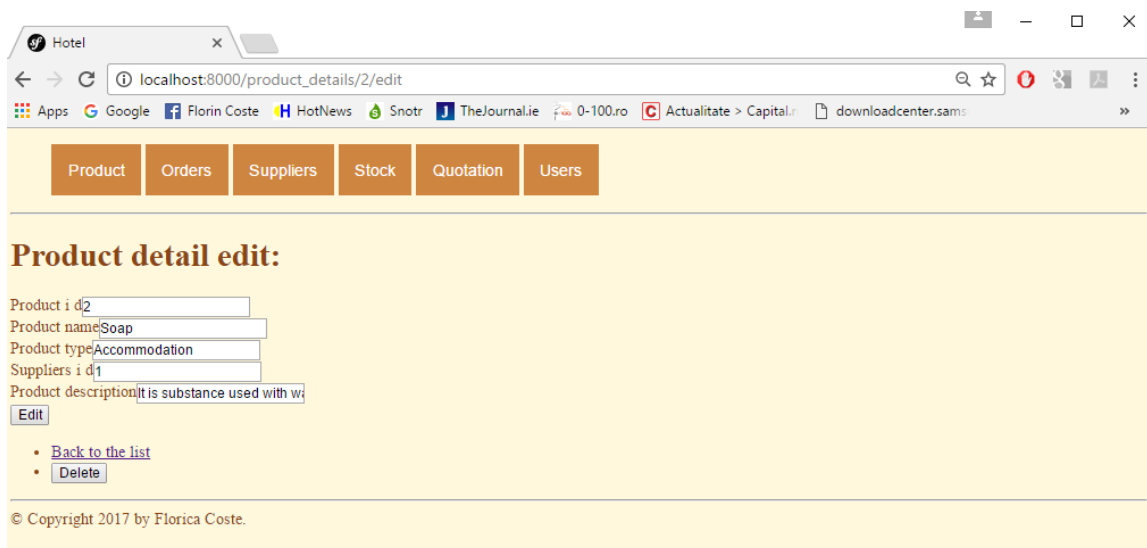


Figure 7.5: See a product.

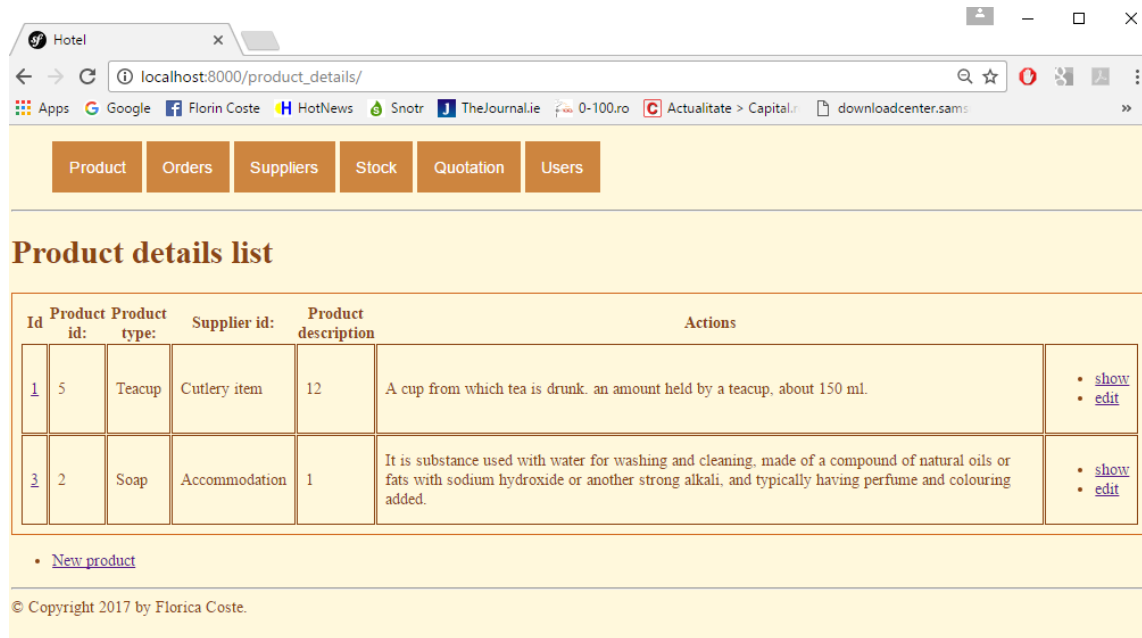


Figure 7.6: See the list of products after delete.

#### 4. Access suppliers details

Before the user is able to access the suppliers details it is ask to login. If the login succeed the manager can select from the menu the option supplier.

From the dropped menu the User Manager can select one of the flowing option:

- Supplier details;
- Supplier address;
- Supplier contact;
- Bank details;

If the user will select from option list the “Supplier details” option then the user can see a list of product from the database.

If the user would lice to access the supplier web page direct from hire can just select from the list of supplier “Suppliers web” In this case the user will be aromatically deserted to the supplier WEB page:

From the list of Supplier select the option “Create” and complete the Supplier details form.

If you “Save” the details of the Supplier then you can see in the list of Supplier that a new Supplier has been added.

If the user would like to edit a Supplier can select from the list product a product and select the option Update: Once the user updates the Supplier details can “Save” the details of the Supplier. Managers can see in the list of suppliers the changes of supplier details.

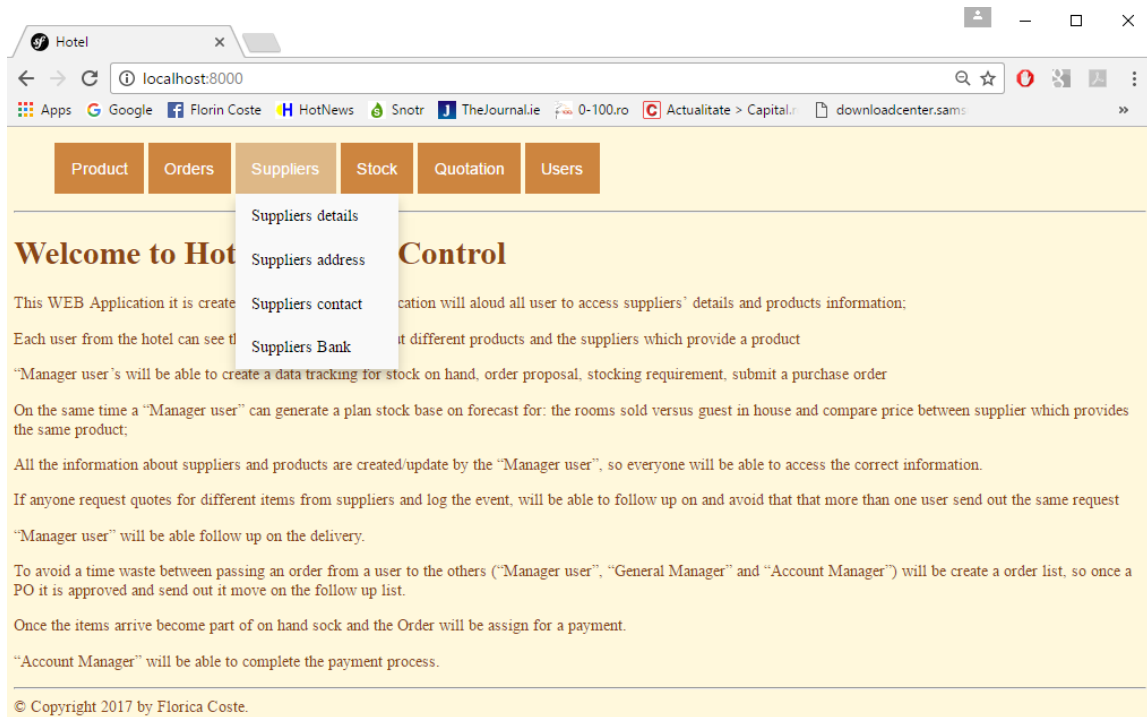


Figure 7.7: Manager login can add/update/delete/display details of a Orders.



Figure 7.8: List of Suppliers.

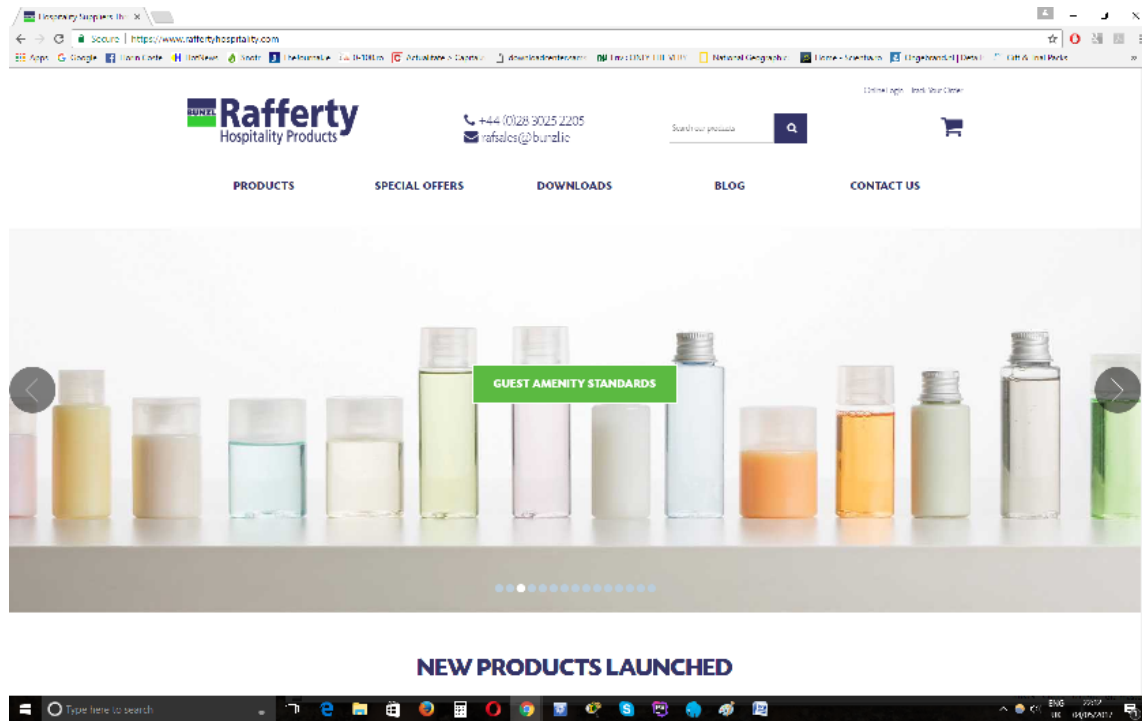


Figure 7.9: Access Supplier WEB page.

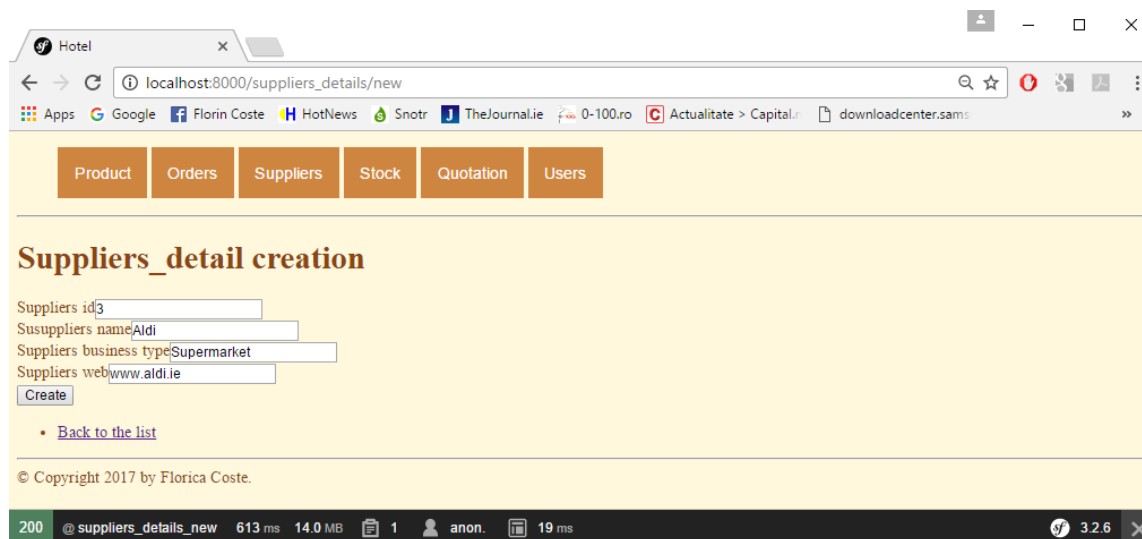


Figure 7.10: Add a new product.



Figure 7.11: See a product.

If a manager wants to delete a supplier has to select from the list of suppliers a supplier, edit the supplier before he can select the option Delete.

If the user “Delete” the details of the supplier, then the changes can be seen on the list of supplier, so the supplier has been remove;

For all other Supplier details it can be follow the same cycle.

#### 4. Access orders details

Before the user is able to access the stock details it is ask to login. If the login succeed the manager can select from the menu the option stock

From the dropped menu the User Manager can select one of the flowing option:

- Create a PO;
- Orders;
- Orders details;

#### 4. Access orders stock

Before the user is able to access the stock details it is ask to login. If the login succeed the manager can select from the menu the option stock

When the user Manager will log in and access the stock option for the first time the stock level has to be created. In this case you have 2 option:

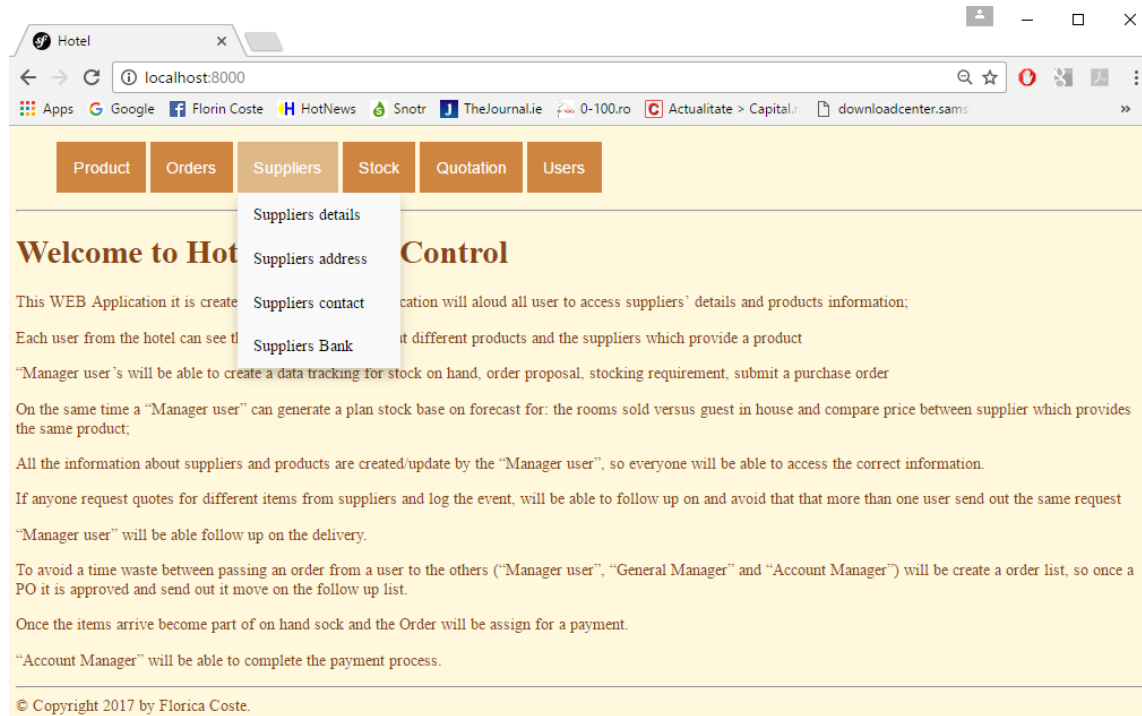


Figure 7.12: Manager login can add/update/delete/display details of a Orders.

You can add manually each item to the inventory or you can import the items from the list of products.





## Part VII

# Testing



# 8

## Testing and work review

### Testing

To build better and more reliable applications, whenever you write a new line of code, you should test your code using both functional and unit tests. Symfony integrates with an independent library - called PHPUnit - to give you a rich testing framework.

All test are automatically saved in the tests/

A unit test is a test against a single PHP class, when functional tests check the integration of the different layers of an application

A function test have a very specific workflow:

- Make a request;
- Test the response;
- Click on a link or submit a form;
- Test the response;
- Rinse and repeat.

Each application has its own PHPUnit configuration, stored in the phpunit.xml.dist file. only the tests stored in /tests are run via the phpunit command. You can add more directory to be tested in phpunit.xml.dist

The tests run for this application used PHAR (PHp ARchive). To run those tests was need it to be downland from Codeception page the file and the root of the project.

Once we run the bootstrap codeception we will have install, you need to create your

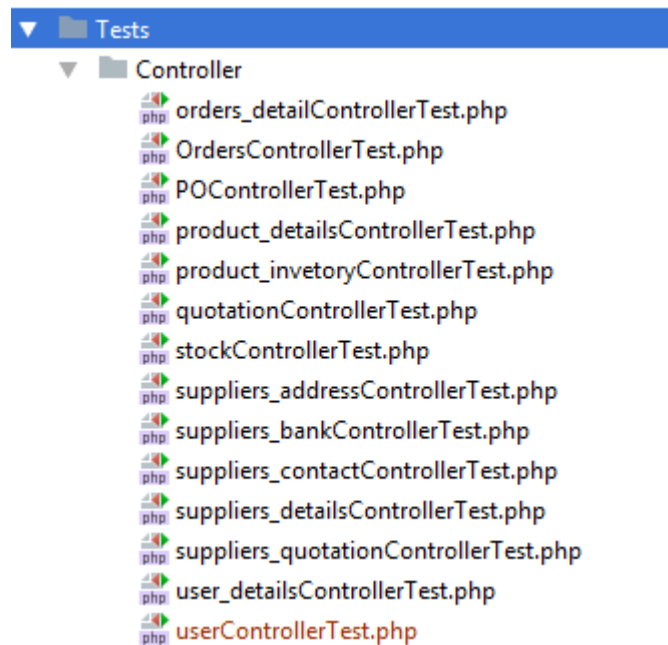


Figure 8.1: Test details.

```

λ php codecept.phar bootstrap
Initializing Codeception in C:\laragon\www\Hotel\pro

File codeception.yml created      <- global configuration
tests/unit created               <- unit tests
tests/unit.suite.yml written     <- unit tests suite configuration
tests/functional created         <- functional tests
tests/functional.suite.yml written <- functional tests suite configuration
tests/acceptance created        <- acceptance tests
tests/acceptance.suite.yml written <- acceptance tests suite configuration
tests/_output was added to .gitignore
---
tests/_bootstrap.php written <- global bootstrap file
Building initial Tester classes
Building Actor classes for suites: acceptance, functional, unit
-> AcceptanceTesterActions.php generated successfully. 0 methods added
\AcceptanceTester includes modules: PhpBrowser, \Helper\Acceptance
AcceptanceTester.php created.
-> FunctionalTesterActions.php generated successfully. 0 methods added
\FunctionalTester includes modules: \Helper\Functional
FunctionalTester.php created.
-> UnitTesterActions.php generated successfully. 0 methods added
\UnitTester includes modules: Asserts, \Helper\Unit
UnitTester.php created.

Bootstrap is done. Check out C:\laragon\www\Hotel\pro/tests directory

```

Figure 8.2: Codeception setup into the project.

```
C:\laragon\www\Hotel\pro (entities) 07/05/2017 6:43:13.83
λ php codecept.phar run
Codeception PHP Testing Framework v2.2.10
Powered by PHPUnit 5.7.17 by Sebastian Bergmann and contributors.

Acceptance Tests (3) -----
+ SigninCept: Login to website (0.00s)
- TestCept: Login to website
$I = new AcceptanceTester($scenario);
$I->am('ROLE_ADMIN');
$I->wantTo('login to website');
$I->lookForwardTo('access all website features');
$I->amOnPage('/login');
$I->fillField('username','admin');
$I->fillField('Password','$2y$13$xGIf/AAUTEaS8uYp.ofNPOUtCgmnAaahSIMEuICpA5228aXcMpFq');
$I->click('login');
+ TestCept: Login to website (0.00s)
- WelcomeCept: Ensure that front page works$I = new AcceptanceTester($scenario);
$I->wantTo('ensure that front page works');
$I->amOnPage('/');
$I->see('Welcome to the front page ');
+ WelcomeCept: Ensure that front page works (0.00s)/ result to achieve
-----

Functional Tests (0) -----
-----

Unit Tests (0) -----
-----

Time: 301 ms, Memory: 12.00MB
OK (3 tests, 0 assertions)
```

Figure 8.3: Codeception setup into the project.

**Review**

Our main focus was to identify a method to save information about suppliers and products details, which will give to all employ from the company a easy way to access the same information in real time.

The first step to complete the minion was to spend time with Accounts Team to identified which kind of information would need to be stored in database.

Once the information was collected, the next logical step was to create data model for the database and create a Normalised Relational Database Structure (Convert to Third Normal Form ).

Using our ER Diagram as a guide, was created the database and the tables, and populate with data.

The purpose of creating generic query result class is to be able to convert SQL queries. Objects are far easier to work with and allow us to use properties instead of array keys.

Even in the beginning I start to create the classes in PHP I found difficult to make sure that classes correspond to each table.

This was the time when I decide to move from PHP to Symfony. This thesis illustrates on the development chapter the different between.

PHP (PHP: Hypertext Preprocessor) is a computer scripting language, originally designed for producing dynamic web pages.

Symfony is a web application framework written in PHP which follows the model-view-controller (MVC) paradigm.

Symfony is written in PHP and builds on it with a set of classes specially designed for the web. So you need to know PHP before you can use Symfony.

Symfony adds custom extensions on top of Twig to integrate some components into the Twig templates.

To be able to complete this project I spend some time learning twig and Symfony.

## Part VIII

# Conclusion





# 9

## Conclusion

Having developed a WEB application for “Hospitality Materials Control” and analyse the impact of using it, we conclude that using Symfony as a software for implementation drove the result in rapid development.

Material control should be used in all the hotels and by all employ. A large, medium or small hotels would be able to use efficient the WEB application. Will be very useful tools in getting the right quality and right quantity of supplies at right time, having good inventory control and will improve the efficiency of the hotel. This would turn in an efficient lead to the growth of these firms.

WEB application, called “Hospitality Materials Control”, help the users to operate efficiently having adequate materials on hand when needed;



## Part IX

# Future work



# 10

## Future work

I would like to complete the Hospitality Materials Control and test for the hotel which I'm working.

I would like as well to add the following components:

- Real-time data tracking for cost stock on hand, order proposals, and stocking requirements;
- Transmission of purchase orders from within the application via e-mail to suppliers to expedite ordering/receiving processes;
- Supplier to be able to control the stock on the site;
- Comparison between actual with planned stock usage;
- Ability to compare prices and plan purchases;
- Mobile solutions for orders, receipts and inventory;
- From financial point of view support for multiple tax schemas and direct payments;



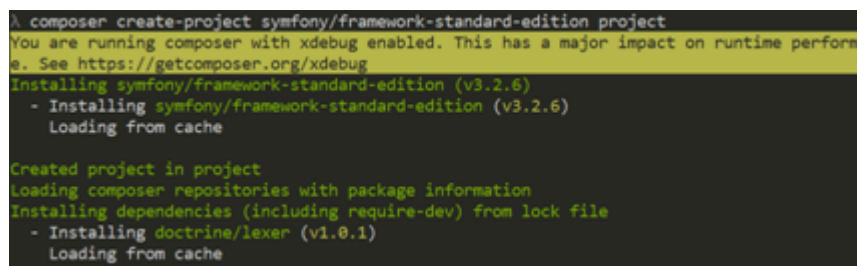


## Appendix

### A.1 Creating Applications with Composer in Symfony

in this project was used the command line to create a new project:

```
$ composer create-project symfony/framework-standard-edition pro
```



```
A composer create-project symfony/framework-standard-edition project
You are running composer with xdebug enabled. This has a major impact on runtime perform
a. See https://getcomposer.org/xdebug
Installing symfony/framework-standard-edition (v3.2.6)
- Installing symfony/framework-standard-edition (v3.2.6)
  Loading from cache

Created project in project
Loading composer repositories with package information
Installing dependencies (including require-dev) from lock file
- Installing doctrine/lexer (v1.0.1)
  Loading from cache
```

Figure A.1: Initialize a symphony project.

When creating the project, the DB connections are to be setup. On the same time this can be done later, so you need to change it, got to { parameters.yml } file and modify them

To be able to work on the project and tested on the same time you will need to move into the directory which hold all the tiles

Before starting to work on the it is recommended to run the local host to make sure that the configuration went well.

```

Creating the "app/config/parameters.yml" file
Some parameters are missing. Please provide them.
database_host (127.0.0.1):
database_port (null):
database_name (symfony):
database_user (root):
database_password (null):
mailer_transport (smtp):
mailer_host (127.0.0.1):
mailer_user (null):
mailer_password (null):
secret (ThisTokenIsNotSoSecretChangeIt):
> Sensio\Bundle\DistributionBundle\Composer\ScriptHandler::buildBootstrap
> Sensio\Bundle\DistributionBundle\Composer\ScriptHandler::clearCache

// Clearing the cache for the dev environment with debug
// true

```

Figure A.2: Initialize DB connections for project.

```
$ php bin/console server:run
```

As a result, if you open the browser and access the <http://localhost:8000/> you can see the Welcome Page of Symfony:

## Welcome to Symfony 2.8.0



Your application is ready to start working on it at: `/var/www/symfony-app/`

### What's next?



Read Symfony documentation to learn  
[How to create your first page in Symfony](#)

Figure A.3: Welcome Page of Symfony.

If any error occurs then you will need to fix file permissions



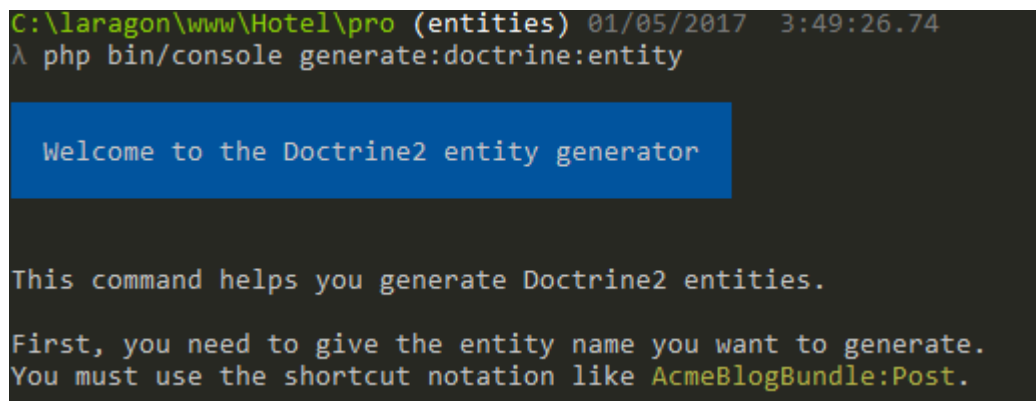
# B

## Create a Symfony entity

The advantage of using Symfony is that you can use the `generate:doctrine:entity` command on the command line.

```
$ php bin/console generate:doctrine:entity
```

This command generates a new Doctrine entity stub including the mapping definition and the class properties, getters and setters.

A screenshot of a terminal window with a dark background. The prompt is 'C:\laragon\www\Hotel\pro (entities) 01/05/2017 3:49:26.74'. The command entered is 'php bin/console generate:doctrine:entity'. The output shows a blue banner with the text 'Welcome to the Doctrine2 entity generator'. Below this, it says 'This command helps you generate Doctrine2 entities.' and 'First, you need to give the entity name you want to generate. You must use the shortcut notation like AcmeBlogBundle:Post.'

```
C:\laragon\www\Hotel\pro (entities) 01/05/2017 3:49:26.74
λ php bin/console generate:doctrine:entity

Welcome to the Doctrine2 entity generator

This command helps you generate Doctrine2 entities.

First, you need to give the entity name you want to generate.
You must use the shortcut notation like AcmeBlogBundle:Post.
```

Figure B.1: Create a new class.

By default, the command is run in the interactive mode and asks questions to determine the bundle name, location, configuration format and default structure:

The command can be run in a non-interactive mode by using the `--no-interaction` option, but you need to pass all needed options:

```
The Entity shortcut name: AppBundle:product
AppBundle:product

Determine the format to use for the mapping information.

Configuration format (yaml, xml, php, or annotation) [annotation]:

Instead of starting with a blank entity, you can add some fields now.
Note that the primary key will be added automatically (named id).

Available types: array, simple_array, json_array, object,
boolean, integer, smallint, bigint, string, text, datetime, datetimetz,
date, time, decimal, float, binary, blob, guid.

New field name (press <return> to stop adding fields): product_name
Field type [string]:

Field length [255]: 30
Is nullable [false]:

Unique [false]:

New field name (press <return> to stop adding fields):

Entity generation

created .\src\AppBundle\Entity\product.php
> Generating entity class C:\laragon\www\Hotel\pro\src\AppBundle\Entity\product.php: OK!
> Generating repository class C:\laragon\www\Hotel\pro\src\AppBundle\Repository\productRepository.php:
OK!

Everything is OK! Now get to work :).

C:\laragon\www\Hotel\pro (entities) 01/05/2017 3:52:26.18
```

Figure B.2: Create a new class in interactive mode.

```
$ php bin/console generate:doctrine:entity --no-interaction --entity=AcmeBlogBundle:Post --fields="prod"
```





## Generating database with Symfony

The easiest way to understand how Doctrine works is to see how it is working.

To be able to generate a table in a database you'll configure the database, use the Product object created before, persist it to the database and fetch it back out.

### Configuring the Database

Before you really begin, you'll need to configure your database connection information. By convention, this information is usually configured in an `app/config/parameters.yml` file:

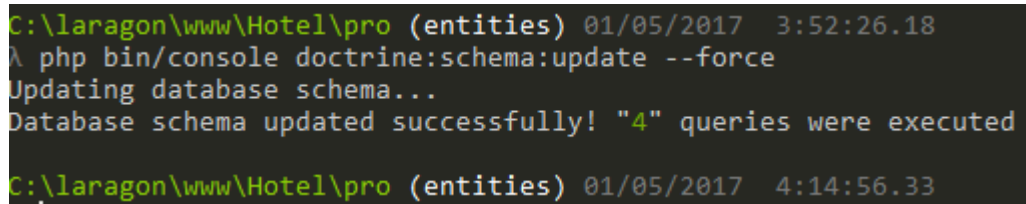
```
parameters:
    database_host: localhost
    database_port: null
    database_name: hotel
    database_user: root
    database_password: Florica22
    mailer_transport: smtp
    mailer_host: 127.0.0.1
    mailer_user: null
    mailer_password: null
    secret: ThisTokenIsNotSoSecretChangeIt
```

Figure C.1: Connect to database.

## C.1 Creating tables in the database

Before you begin, you'll need to configure your database connection information. By convention, this information is usually configured in an 'app/config/parameters.yml' file. Once the connection with the database it is setup we can tell Doctrine to create a corresponding table.

```
$ php bin/console doctrine:schema:update --force
```



```
C:\laragon\www\Hotel\pro (entities) 01/05/2017 3:52:26.18
A php bin/console doctrine:schema:update --force
Updating database schema...
Database schema updated successfully! "4" queries were executed
C:\laragon\www\Hotel\pro (entities) 01/05/2017 4:14:56.33
```

Figure C.2: Create table.

Once the command is lunch on the terminal a couple of confirmation messages will confirm that table has been created: If you update the structure of an entity and run the same command line you can se that the table it is updated.

Updating database schema...

Database schema updated successfully! "3" query was executed

If you access your database you can see that see a new table in the database it is created.

Figure C.3 shows our new `product_details` table created for us.

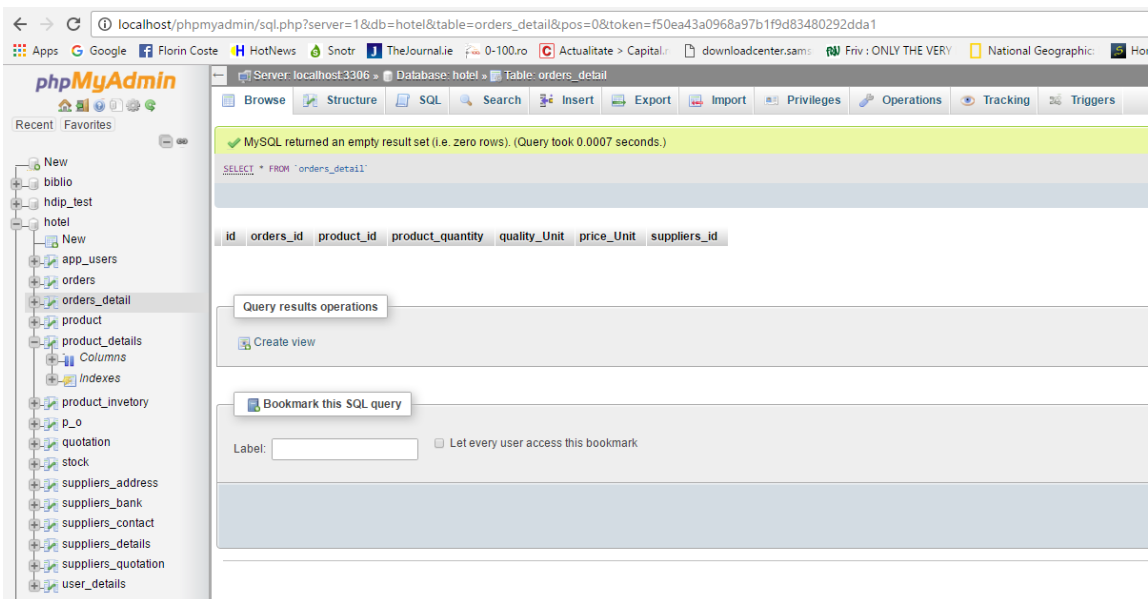


Figure C.3: CLI created table in PHPMyAdmin.

## List of References

1. Dr. Matt Smith, 2017, **An Introduction To Symfony 3**, Dublin, book, pages: 207;
2. Francois Zaninotto and Fabien Potencier, 2007, **The Definitive Guide to symfony**, book, New York, Apress, pages: 213;
3. Tim Bowler and Wojciech Bancer, 2009, **Symfony 1.3 Web Application Development**, book, Birmingham, Packt Publishing Ltd., pages: 514;
4. Wojciech Bancer, September 2015, **Symfony2 Essentials**, book, Birmingham, Packt Publishing Ltd., pages: 133;
5. Sohail Salehi, April 2016, **Mastering Symfony**, book, Birmingham, Packt Publishing Ltd., pages: 263;
6. Symfony DebugBundle Retrieved from [.http://api.symfony.com/2.8/Symfony/Bundle/DebugBundle.html](http://api.symfony.com/2.8/Symfony/Bundle/DebugBundle.html)
7. Symfony DebugBundle Retrieved from [.http://symfony.com/](http://symfony.com/)





## Part X

# Generating a CRUD Controller Based on a Doctrine Entity



## Part XI

# Database



## **Part XII**

# **Entities**



## **Part XIII**

# **User Interface**

