Table of contents

1 Restaurant Manager Requirements 4

1.1 Problem statement 4

1.2 Glossary 5

1.3 Supplementary Specification 7

1.4 Use-Case Model 9

1.4.1 Login 13

1.4.2 Change Profile 14

1.4.3 View Menu Info 15

1.4.4 View Reports 16

1.4.5 Maintain Menu Info 17

1.4.6 Maintain User Info 19

1.4.7 Manage Reports 20

1.4.8 Manage Active Dishes 22

1.4.9 Create New Order 24

1.4.10 Billing 25

1.4.11 Get Statistical Data 26

1.4.12 Chat 27

1.4.13 Register 28

2 Restaurant Manager Analysis 29

2.1 Architectural Analysis 29

2.1.1 Key Abstractions 29

2.1.1.1 Key Abstraction Definitions 30

2.1.2 Upper-Level Components and Their Dependencies 31

2.1.2.1 Component Definitions 31

2.2 Use Case Analysis 32

2.2.1 Use-Case Realization Interaction Diagrams 32

2.2.1.1 Login 32

2.2.1.2 Change Profile 33

2.2.1.3 View Menu Info 33
2.2.1.4 View Reports 34
2.2.1.5 Maintain Menu Info 34
2.2.1.6 Maintain User Info 36
2.2.1.7 Manage Reports 37
2.2.1.8 Manage Active Dishes 39
2.2.1.9 Create New Order 39
2.2.1.10 Billing 40
2.2.1.11 Get Statistical Data 40
2.2.1.12 Chat 41
2.2.1.13 Register 42
2.2.2 Use-Case Realization View of Participating Class (VOPCs) 42
2.2.2.1 Login 42
2.2.2.2 Change Profile 42
2.2.2.3 View Menu Info 43
2.2.2.4 View Reports 43
2.2.2.5 Maintain Menu Info 44
2.2.2.6 Maintain User Info 45
2.2.2.7 Manage Reports 46
2.2.2.8 Manage Active Dishes 46
2.2.2.9 Create New Order 47
2.2.2.10 Billing 47
2.2.2.11 Get Statistical Data 47
2.2.2.12 Chat 48
2.2.2.13 Register 48
2.2.3 Analysis-Class-To-Analysis-Mechanism Map 49
3 Restaurant Manager Design 50
3.1 Identify Design Elements 50
3.1.1 Subsystem Context Diagram 50
3.1.1.1 Billing subsystem 50

3.1.1.2 Maintain Catalog Subsystem 51
3.1.2 Analysis-Class-to-Design-Element Map 51
3.1.3 Design-Element-to-Owning-Package Map 52
3.1.4 Architectural Components and Their Dependencies 54
3.1.5 Packages and Their Dependency 55
3.2 Describe the Run-time Architecture 56
3.3 Describe Distribution 56
3.4 Use Case Design 56
3.5 Subsystem Design 57
3.6 Class Design 57
3.6.1 Describe each class or interface 57
3.6.1.1 LoginForm 57
3.6.1.2 ChangeProfileForm 58
3.6.1.3 ICatalog (interface) 58
3.6.1.4 UserCatalog 58
3.6.1.5 MenuViewer 59
3.6.1.6 DishCatalog 59
3.6.1.7 ReportViwer 60
3.6.1.8 ReportCatalog 60
3.6.1.9 MenuForm 60
3.6.1.10 AddNewDishForm 60
3.6.1.11 EditDishForm 61
3.6.1.12 DeleteDishForm 61
3.6.1.13 Item 61
3.6.1.14 Dish 61
3.6.1.15 UserForm 61
3.6.1.16 AddNewUserForm 62
3.6.1.17 DeleteUserForm 62
3.6.1.18 User 62
3.6.1.19 ReportForm 62
3.6.1.20 WriteReportForm 62

3.6.1.21 EditReportForm 63
3.6.1.22 DeleteReportForm 63
3.6.1.23 Report 63
3.6.1.24 ActiveDish 63
3.6.1.25 ActiveDishViewer 64
3.6.1.26 CreateNewOrderForm 64
3.6.1.27 Bill 64
3.6.1.28 BillingForm 64
3.6.1.29 IBilling 65
3.6.1.30 BillingSystem 65
3.6.1.31 Statistic Form 65
3.6.1.32 StatisticEngine 65
3.6.1.33 ChatWindow 66
3.6.1.34 RegisterForm 66
3.6.2 Class diagram in total 66

1 Restaurant Manager Requirements

1.1 Problem statement

Today, the information technology permeates all areas of our lives and of course the restaurant is no exception. We are tasked with developing a new restaurant manager system to manage almost activities of a restaurant. Such systems have been built, but overseas and in accordance with restaurants in which. So we need to develop a system suitable for restaurants in Vietnam, not only that the system will be built in both desktop and mobile platform.

The system will have the form of a WEB application that allows users to log in and perform management tasks. All types of user of the system are: Manager, Chef, Waiter, Cashier and Customer. They can log into the system concurrently to carry out their work. The condition needed for interacting with the system is they have a laptop, PC, tablet or smart phone that connects to the Internet.

After logged into the system, the managers, chefs, waiters, cashiers and customers can create a new order of a meal that customer want. The information of an order includes a list of dishes with their quantity and price and also the total price of the order. After the users save the order, a corresponding bill is created and store into database, then all dishes contained in which are sent to chefs for cooking. This order may be modified any time before payment.

The chefs will be notified about required dishes on his device. He/she then can refuse to cook, accept to cook these dishes. If a dish was refused, a message about this problem is sent to the actor that sent the message. When a chef completes a dish, he/she notifies that this dish is ready to be served.

When a customer finished her/his meal and want to pay money. All managers, waiters and cashiers can do billing.

The system also allows manager to manage all dishes information (menu) of the restaurant as well as manage all employee information. He/she can add new dishes to the menu, edit an available dish or delete an available dish and the same with the employee.

Another interesting of the system is to provide some useful statistic such as the most favorite dishes, the least favorite dishes, common type of customer of the restaurant. Based on these benefit information, the manager can make a good business strategy.

The manager can post reports to inform the employees and customers some important activities of the restaurant.

An anonymous customers can register to the system for viewing. the activities of the restaurant and ordering.

All users of the system (also all employee – managers are a special employee) can log in to the system to chat with each other, change their profile and view report that was post by the manager.

1.2 Glossary

Introduction

This document is used to define terminology specific to the problem domain, explain terms, which may be unfamiliar to the reader of the use-

case description or other project documents. Often, this document can be used as an informal data dictionary, capturing data definitions so that use-case description and other project documents can focus on what the system must do with the information.

Definition

The glossary contains the working definitions for the key concepts in the Restaurant Manager System.

Manager

A person, who leads all employees of the restaurant, manages all business activities of the restaurant.

Chef

A person, who works in the kitchen, cooks dishes served by the restaurant.

Waiter

Also stands for Waitress, is person who do greeting with customers for ordering and deploys meals to customers.

Cashier

The person stands beside the desk for billing.

Customer

A person pays the restaurant money for a meal.

Order

The work of request meal or may be referred as the bill that may be modified.

Bill

A record contains a list of dishes and the price of the meal ordered by customers.

Employee

A person works in the restaurant.

User

An account belongs to the system. It may be employee, customer, waiter, cashier or manager.

Statistic data

Benefit data which was gotten by querying the database.

Dish

Item that is served by the restaurant.

Report

Is an announcement of the manager to all users.

Billing System

The component that can access, query and process on database of bills.

Catalog

A table of a type of object

Active Dish

The dish that are ordered by the customers.

Statistic Engine

The component processes on database to get the statistical data.

1.3 Supplementary Specification

Objectives

The purpose of this document is to define requirements of the Restaurant Manager System. This Supplementary Specification lists the requirements that are not readily captured in the use case of the use-case model. The

Supplementary Specification and the use-case model together capture a complete set of requirements on the system.

Scope

This Supplementary Specification applies to the Restaurant Manager System, which will be developed by OOAD students.

This specification defines non-functional requirements of the system; such as reliability, usability, performance and supportability, as well as functional requirements that are common across a number of use cases. (The functional requirements are defined in the Use Case Specification).

References

IBM Rational Software Documentation (Version 2004)

Functionality

Multiple user must be able to perform their works concurrently.

Usability

The software must be easy to use so that a new user can learn how to use the system within 30 minutes. This is very important requirement.

The user interface has to be nice and clear.

Reliability

The system shall be available 24 hours a day 7 days a week, with no more than 10% down time.

Performance

The latency of get statistic data must be less than 10 seconds and that one of other operations are less than 2 seconds.

The GUI transitions must be smooth.

No error of billing.

Supportability

None

Security

The system must prevent people are not manager to modify bill after store to the database.

Almost changes of the system databases can only be done by the manager. Require confirm password before submit the changes.

Design Constraints

The system shall provide both Window-based Desktop interface and Mobile Application Interface

1.4 Use-Case Model

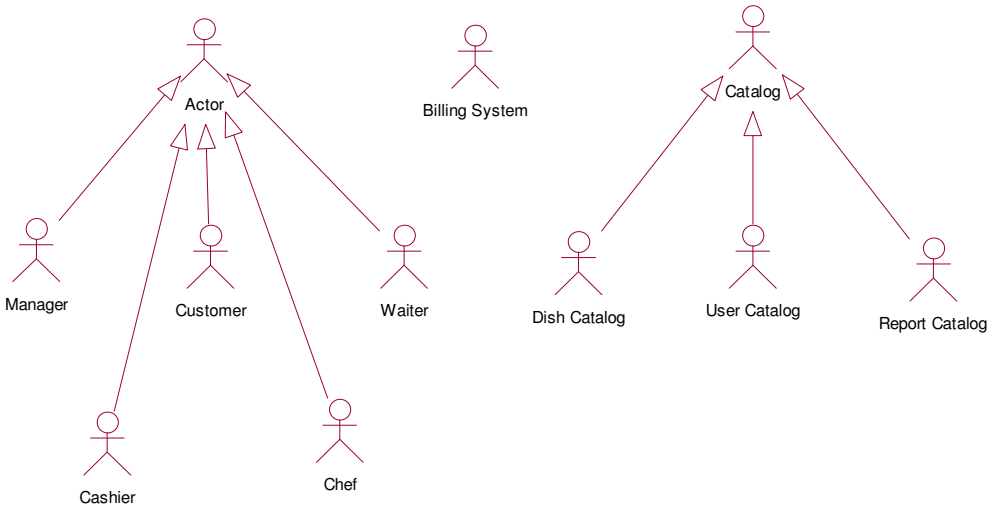

Figure 1.4.1: Use-Case Model - Actors and their dependency

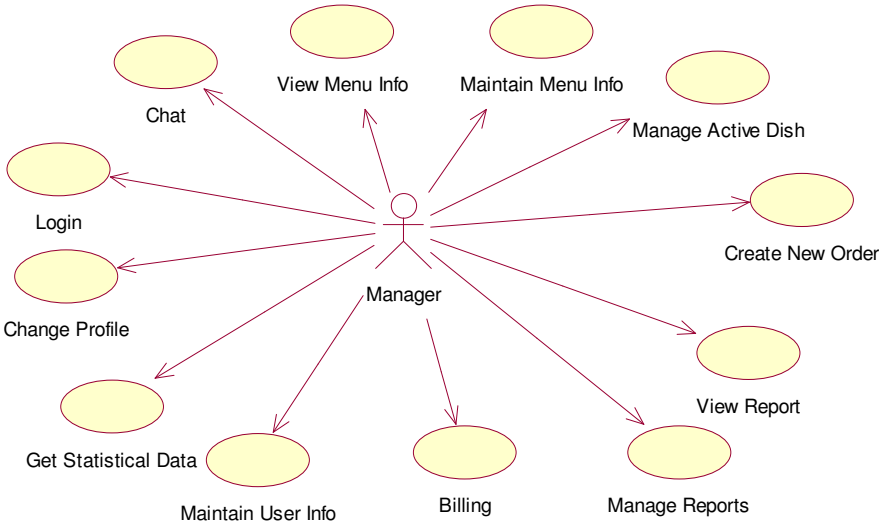

Figure 1.4.2: The use-case model in the Manager view

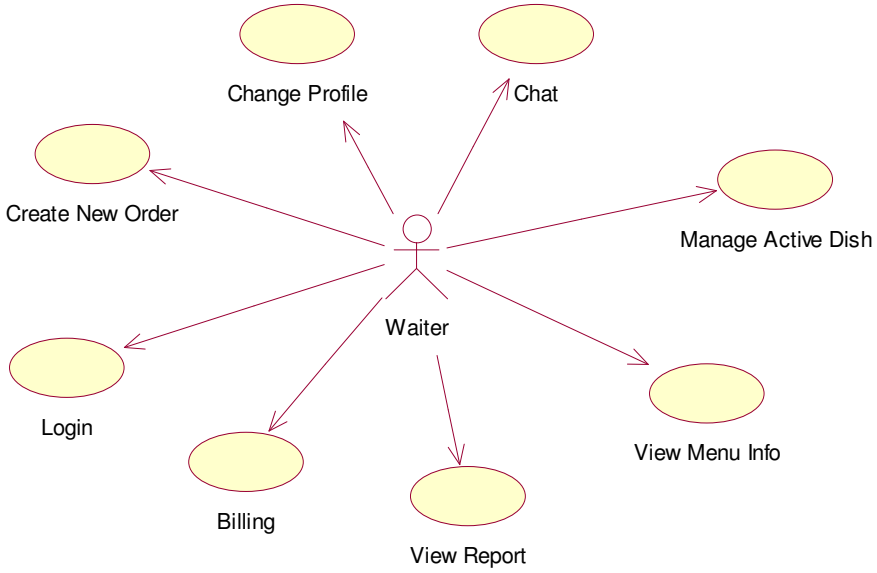

Figure 1.4.3: The use-case model in Waiter view

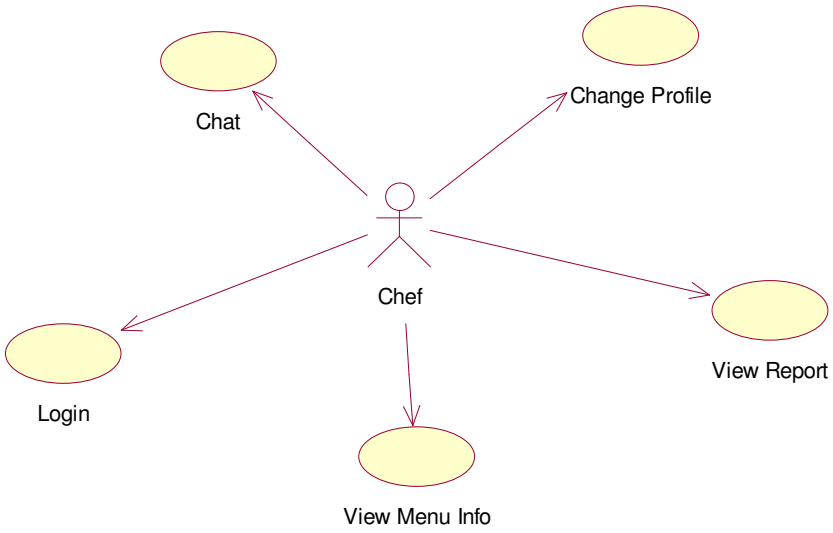

Figure 1.4.4: The use-case model in the Chef view

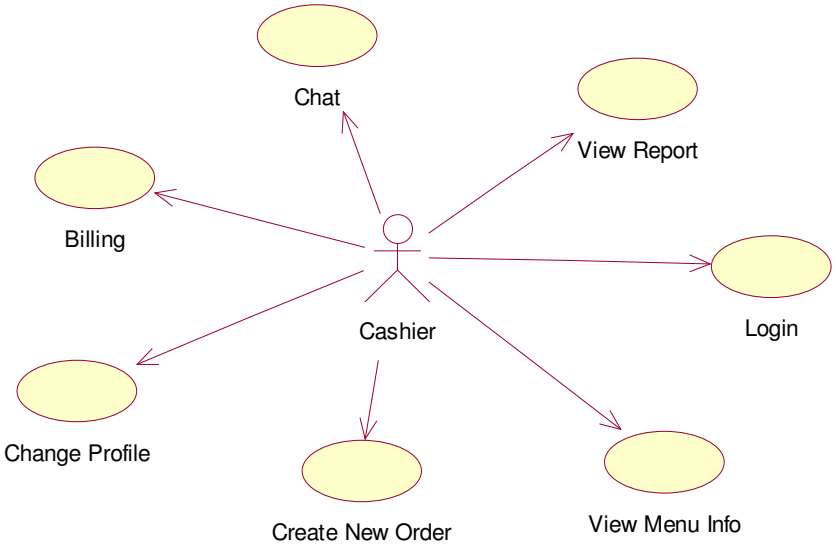

Figure 1.4.5: The use-case model in Cashier view

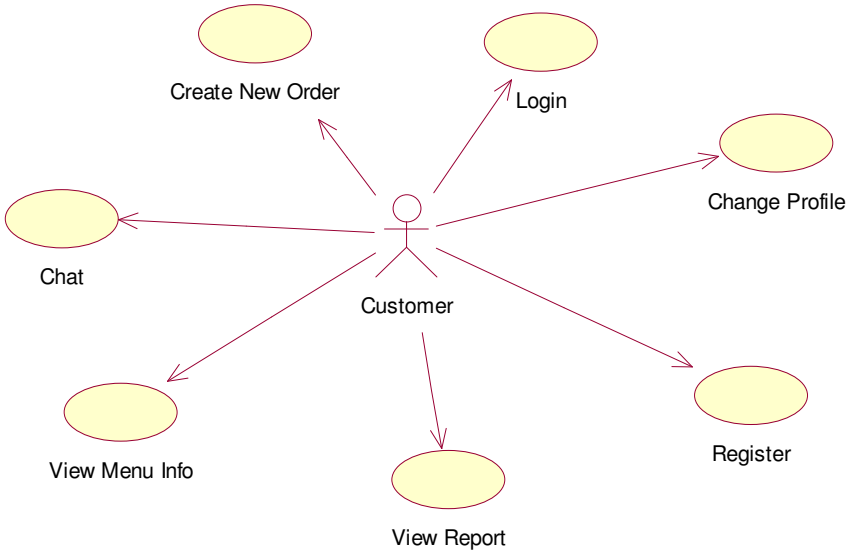

Figure 1.4.6: Use case model in the Customer view

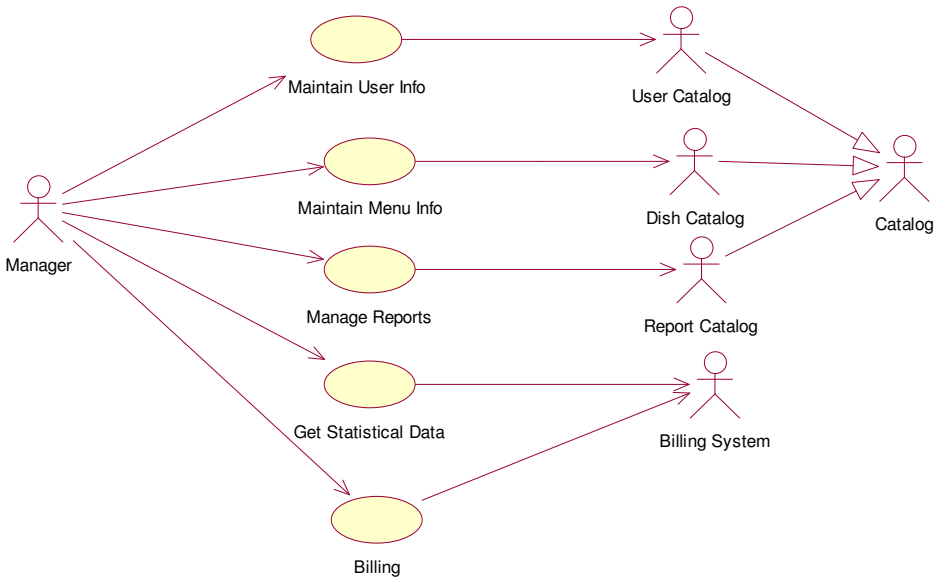

Figure 1.4.7: Use-Case model in the view of integrated with subsystem

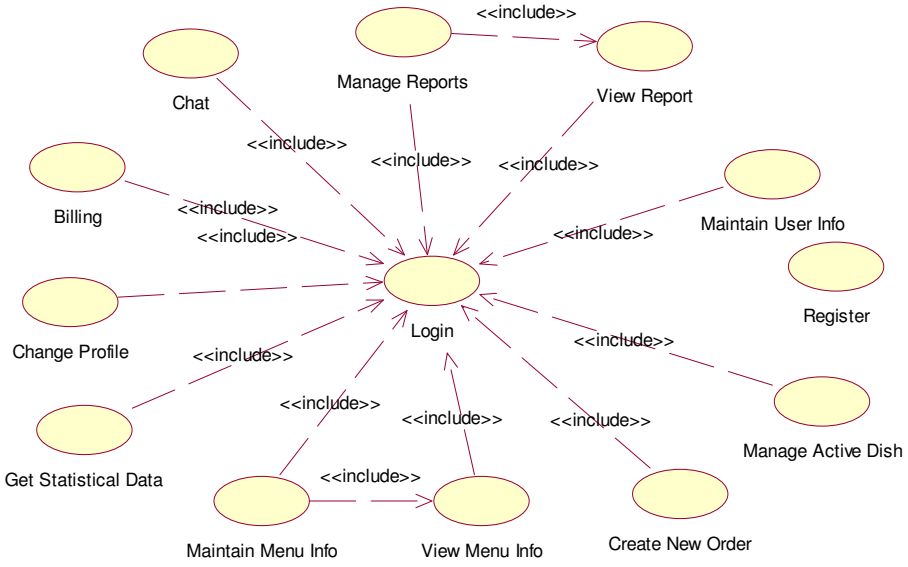

Figure 1.7: The use case dependencies

1.4.1 Login

Brief Description

This use case describes how a user logs into the Restaurant Manager System.

Flow of events

Basic Flow

This use case starts when the actor wishes to Login to the Restaurant Manager System.

1. Actor enters his/her name and password
2. The system validates the entered name and password and logs the actor to into the system.

Alternative Flows

Invalid Name/Password

If, in the **Basic Flow**, the actor entered an invalid name and/or password, the system displays an error message. The actor can choose to either return

to the beginning of the **Basic Flow** or cancel the login, at which point the use case ends.

Special Requirements

None

Pre-Conditions

The system has the login screen displayed.

Post-Conditions

If the use case was successful, the actor is now logged into the system. If not, the system state is unchanged.

Extension Points

None

1.4.2 Change Profile

Brief Description

This use case describes how a user changes his/her information

Flow of events

Basic Flow

This use case starts when an actor want to change something in his/her profile.

1. The system retrieves and displays the detail information of the actor.
2. The actor makes the desired changes.
3. The system retrieves the changes then validates inputs and then updates the actor's information in database.

Alternative Flows

Invalid Inputs

If in the **Basic Flow**, the actor entered invalid type of inputs, the system displays an error message and highlights the position of error. The actor can

choose to either back to the beginning of the **Basic Flow** or cancel the use case.

Special Requirements

If the actor is employee but not manager, the system only displays to the actor some type of information. Some other such as name, working days, number of hours work per day and salary are not displayed to the actor.

Pre-Conditions

The system is in the login state and the actor has clicked to request change profile.

Post-Conditions

If the use case was successful, the actor's information is updated. If not, the system is unchanged.

Extension Points

None

1.4.3 View Menu Info

Brief Description

This use case describes how a user views all the dishes that the restaurant serves.

Flow of events

Basic Flow

This use case starts when an actor want to view dishes' information for order or other special purpose.

1. The system display the menu based on categories.
2. The actor navigates through some sub-categories go to specific dish.
3. The system displays the dish information in detail for viewing of the actor.

Alternative Flows

None

Special Requirements

None

Pre-Conditions

System is in the login state and the actor has requested to view menu.

Post-Conditions

None

Extension Points

None

1.4.4 View Reports

Brief Description

This use case describes how a user read a report posted by the manager.

Flow of events

Basic Flow

This use case start begin when the actor want to read reports (may be at the beginning of a working day for get new report).

1. System display the list of reports with short description arranged from newest to older.
2. The actor clicks on a specific report for reading.
3. The actor comments at the end of the report for feedback something.
4. The system appends the comment to the end of report as a part of it and then updates the report.

Alternative Flows

View other report

If the actor wants to read some other report, he/she can back to the step 1 in the **Basic Flow** again and again.

Special Requirements

None

Pre-Conditions

The system is in login state and in the home page of the restaurant (The home page of the restaurant displays reports as the main look).

Post-Conditions

If actor commented, then the report is updated.

Extension Points

None

1.4.5 Maintain Menu Info

Brief Description

This use case allows the manager to maintain the menu of the restaurant. It includes adding, editing and deleting dish informaiton from the menu.

Flow of events

Basic Flow

This use case start when the manager want to add a new dish to menu, edit an available dish information or/and deleting a dish from the menu

1. The manager does the use case "View Menu Info"
2. The system always displays the prompt to add new dish, edit this dish or delete this dish for the actor choose. Based on the action of the manager, one of the follow sub-flows is executed.
   If the actor clicked "Add new dish", the **Add New Dish** sub-flow is executed.
   If the actor clicked "Edit this dish", the **Edit Dish** sub-flow is executed.
   If the actor clicked "Delete this dish", the **Delete Dish** sub-flow is executed.
   Add New Dish

1. The system requests that the manager enter new dish information. This includes: name, image, short description, price and category.
2. Once the manager provides the requested information, the system generates and assigns a unique dish id number to the new dish. The new dish is added to the system.
3. System display to the manager the new dish information.

Edit Dish

1. The system displayed the dish information in detail that in the editable state.
2. The actor makes the desired changes. This includes the information described in **Add New Dish** sub-flow
3. System display to the manager the new dish with updated information.

Delete Dish

1. The system prompts the manager to confirm the deletion of dish.
2. The manager verifies the deletion.
3. The system remove that dish from the menu and back to the step 2 in the **Basic Flow**.

Alternative Flows

Cancel Deletion

If in the **Delete Dish** sub-flow, the actor decides not to delete the dish, the deletion is cancelled and the **Basic Flow** is re-stared at the beginning.

Special Requirements

None

Pre-Conditions

The manager is logged in to the system.

Post-Conditions

The menu is updated if use case is successful. Otherwise, the system is unchanged.

Extension Points

None

1.4.6 Maintain User Info

Brief Description

This use case allows the manager to maintain the user information of the system. It includes adding, editing and deleting a user's information from the system.

Flow of events

Basic Flow

This use case start when the manager want to add a new user to the system, edit an available user information or/and deleting a user from the system

1. The system displays all users as a lists with short description.
2. The manager navigates to view a specific user information
3. The system always displays the prompt to add new user, edit this user or delete this user for the actor choose. Based on the action of the manager, one of the follow sub-flows is executed.
   If the manager clicked "Add new user", the **Add New User** sub-flow is executed.
   If the manager clicked "Edit this user", the **Edit User** sub-flow is executed.
   If the manager clicked "Delete this user" the **Delete User** sub-flow is executed.
   Add New User
   1. The system requests that the manager enter new user information. This includes: name, working days, number of hours work per day salary.
   2. Once the manager provides the requested information, the system generates and assigns a unique user id number to the new user. The new user is added to the system.
   3. System displays to the manager the new user information.

   Edit User
   1. The system displayed the user information in detail that in the editable state.
   2. The manager makes the desired changes. This includes the information described in **Add New User** sub-flow

3. System display to the manager the new user with updated information.

Delete User

1. The system prompts the manager to confirm the deletion of user.
2. The manager verifies the deletion.
3. The system remove that user from the system and back to the step 2 in the **Basic Flow**.

Alternative Flows

Cancel Deletion

If in the **Delete User** sub-flow, the manager decides not to delete the user, the deletion is cancelled and the **Basic Flow** is re-stared at the beginning.

Special Requirements

The manager can edit any information of customers and add customers also but can delete them.

The manager doesn't have privilege to change the follow information of employees: image, email, introduce, phone number, birth date. The information of employee that the manager can changes or edits are name, working days, hour work per day, salary.

Pre-Conditions

The manager must be logged in the system.

Post-Conditions

If the use-case was successful, the user information is added, updated or deleted from the system. Otherwise, the system is unchanged.

Extension Points

None

1.4.7 Manage Reports

Brief Description

This use case allows the manager manage all the reports displayed on the homepage of the restaurant. It includes the use case "View Report" as their first step in the flow of events.

Flow of events

Basic Flow

This use case starts when the manager want to post new report or delete/edit an available reports.

1. The manager "View Report"
2. The manager chooses the action that he/she wants to do. Based on the action, one of the follow sub-flows is executed.
   If the manager clicked "Create New Report", the **Create New Report** sub-flow is executed.
   If the manager clicked "Edit", the **Edit Report** sub-flow is executed.
   If the manager clicked "Delete", the **Delete Report** sub-flow is executed.
   Create New Report
   1. The system displays the form for the manager write new report.
   2. The system stores the report into the database.

   Edit Report
   1. The system display the report in detail that in the editable state.
   2. The manager makes the desired changes such as modify contents or delete comment or comment.
   3. The system update report's content, modify the "last modified filed" to current time.

   Delete Report
   1. The system prompts the manager to confirm the deletion of report
   2. The manager verifies the deletion.
   3. The system removes this report from database.

Alternative Flows

Cancel Deletion

If in the **Delete Report** sub-flow, the manger decides not to delete the report. The deletion is cancelled and the Basic Flow is re-started at the beginning.

Special Requirements

None

Pre-Conditions

The manager must be logged into the system and are in the homepage.

Post-Conditions

If the use case was successful, the report is created, updated or deleted from the system. Otherwise the system is unchanged.

Extension Points

None

1.4.8 Manage Active Dishes

Brief Description

This use case allows managers, waiters, chefs change the state of a dishes in active. A dish can have some flow states:

- **Requested**: When customers order and the order is sent to the chefs.
- **Refused**: The chefs can't cook this dish because of some reasons.
- **Waiting**: A chef accepts to cook the dish but not complete.
- **Ready**: The chef completes the dish.
- **Served**: The dish is deployed to the customers.

Flow of events

Basic Flow

This use case usually appears in the restaurant. It starts when customers order. Based on the actor, this use case has different behavioral with different actors.

On the chef view

1. System displays to the chefs all dishes are in the **Requested** or **Waiting** state.
2. The chef then can meet 3 situations corresponding to 3 follow sub-flows

Refuse a requested dish

1. If he/she knows that the kitchen doesn't have enough material for cook this dish, he/she refuses that dish.
2. The system change the state of the dish from **Requested** to **Refused**

Accept a requested dish

1. If the chef can cook the dish, then he/she clicks "Accept" to accept to cook the dish.
2. The system changes the state of the dish from **Requested** to **Waiting**.

Notifies for completed dish

1. If the chef completes a dish, he/she click "Complete" that dish.
2. The system changes the state of that dish from **Waiting** to **Ready**.

On the waiter view

1. The system displays to the waiter all dishes are in the state Refused, **Waiting** or **Ready.**
2. After deployed the **Ready** dish to the customers, the waiter mark that dish as Served.
3. The system changes the state of that dish from **Ready** to **Served**.

On the manager view

1. The system displays to the manager the list of all active dishes.
2. The manager removes a dish from the list.

Alternative Flows

Change the Refused dish

In the case of the materials for the **Refused** dish was bought, the manager can changes the state of the dish from **Refused** to **Requested.**

Special Requirements

None

Pre-Conditions

The actors participate to the use case must be logged in the system.

Post-Conditions

The state of the dish is updated on the bill.

Extension Points

None

1.4.9 Create New Order

Brief Description

This use-case is done by mangers, waiter, cashier or customers to order a meal.

Flow of events

Basic Flow

This use-case starts when a customer order a meal

1. The actor chooses “Create new order”, the system then display an empty order for the actor edit.
2. The actor click “Add dish” to add a dish to the order
3. The actor then does the use-case “View Menu Info” and chooses a specific dish.
4. The actor enters the number of dish.
5. The actor clicks “save” then the system stores this order as a bill and create all dishes contained in the bill and makes them as **Requested.**

Alternative Flows

Add other dish to the order

After a dish was added, the actor may want to add other dish to the order, he/she clicks “add” again and back to the step 3.

Edit the order.

After save the order, the actor can modifies the order any time before the payment. He/she can edits the number of specific dish, delete a dish from the order or add a new dish then save again.

Special Requirements

The actor can't delete or decrease the number of a dish that sare in the **Ready** or **Waiting** state.

Pre-Conditions

The actor must be logged into the system.

Post-Conditions

At the first saving, the system creates a new bill of the order. At the other save the available bill will be updated. The system also makes the state of this bill as **Have Not Paid.**

Extension Points

None

1.4.10 Billing

Brief Description

This use case allows manager, cashier and waiter mainly for billing.

Flow of events

Basic Flow

This use case starts when customer wants a billing.

1. The system displays to the actor the list of bill are in active.
2. The actor click "billing" then the systems store the bill into database.
3. The system delete the bill object.

Alternative Flows

None

Special Requirements

None

Pre-Conditions

Actor must be logged in to the system.

Post-Conditions

The bill record is created in database.

Extension Points

None

1.4.11 Get Statistical Data

Brief Description

This use case allows the manager get some benefit information from the system based on statistic.Some types of information are the most favorite dishes, the least favorite dishes, the common customers of the restaurant...

Flow of events

Basic Flow

This use case starts when the manager want to get some benefit information for make a good business strategy.

1. The system displays the form input to the manager for input the Type of information and the Time interval.
2. The manager fills the form and submits.
3. The system filter the data in the Time interval then processes on it to get the result.

Alternative Flows

Error Time Interval

In the Basic Flow, if the manager entered an invalid Time Interval, the system displays an error message. The manager can either back to step 2 or terminate the use case.

Data not found

If there is no data in the Time Interval, the system display the message about “Empty database”. The manager can either choose back to step 2 ore terminate the use case.

Special Requirements

The progress of retrieving the statistic data must be within 10 seconds.

Pre-Conditions

The manager must be logged in to the system.

Post-Conditions

None

Extension Points

None

1.4.12 Chat

Brief Description

This use case allows all users of the system can communicate with each other at the working time to increase the work efficiency.

Flow of events

Basic Flow

This use case starts when a user wants to talk with each other at working time without say out.

1. The actor chooses “Start new chat” to begin the use case.
2. The system creates a window chat to the actor that only contain the actor as the participant of the chat session.
3. The actor can add other actors to the chat session (can add all logged in actor at once).
4. The actor write message on the window chat, all other participants can see that message.
5. All actors switch out the window chat, the chat session is terminated.

Alternative Flows

None

Special Requirements

Only the logged in actors can participate to the chat system.

The latency of the chat message must be less than 1 second.

Pre-Conditions

The actor must be logged in the system.

Post-Conditions

None

Extension Points

None

1.4.13 Register

Brief Description

This use case allow the customer create an account on the system..

Flow of events

Basic Flow

This use case starts when a customer wants to become a user of the system.

1. The customer fills the form registration.
2. The system validates the inputs
3. The system creates a new user and add this new user to the system.
4. The system displays the register success message.

Alternative Flows

Invalid Input

In the **Basic Flow**, if the customer entered an invalid input, the system then display the error message and highlight the position the error. The customer can either back to step 1 in the **Basic Flo**w or terminate the registration.

Special Requirements

None

Pre-Conditions

The actor must be logged in the system.

Post-Conditions

New customer was added to the system and logs him/her to the system.

Extension Points

None

2 Restaurant Manager Analysis

2.1 Architectural Analysis

2.1.1 Key Abstractions

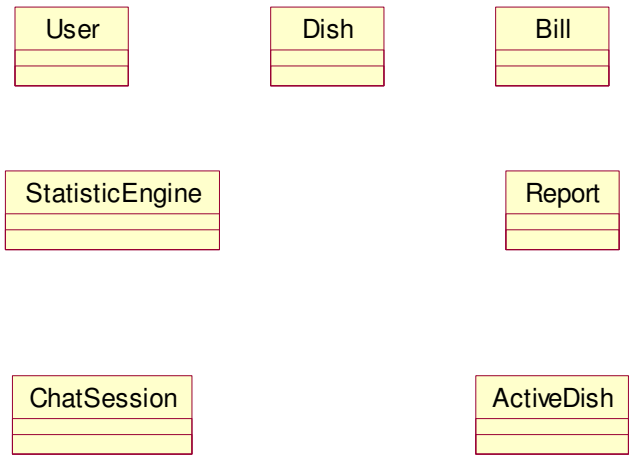

Figure 2.1.1: Key abstractions

2.1.1.1 Key Abstraction Definitions

User: Is an account of the system, may be a manager, chef, waiter, cashier or customer.

Analysis Mechanism: Persistency, Security

Dish: A record that contains all information about a dish

Analysis Mechanism: Persistency, Security

Bill: A record of dishes and the price of a meal ordered by customers.

Analysis Mechanism: Distribution, Process control and synchronization, Security, Communication.

StatisticEngine: The component that is responsible for retrieving statistical data.

Analysis Mechanism: Security, Error detection/handling/reporting

Report: The article was written by the manager, contains contents and list of followed comments.

Analysis Mechanism: Persistency

ChatSession: A session of chat between users.

Analysis Mechanism: Message routing, Distribution, Process control and synchronization.

ActiveDish: A dish ordered by customer, it can move through some states are Requested, Waiting, Refused, Ready, Serve.

Analysis Mechanism: Distribution, Communication.

2.1.2 Upper-Level Components and Their Dependencies

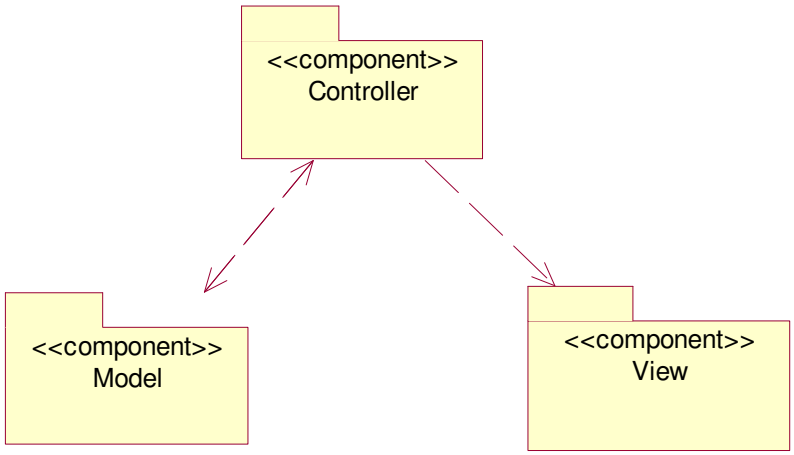

Figure 2.1.2: The upper-level layer architecture

2.1.2.1 Component Definitions

Controller: Keep navigation tasks received requests from the user and call the accordance methods to handle them. For example, this component will receive the request from url and form to manipulate directly with database.

Model: This is the component that contains the static data (database), method to access, query and process with it.

View: Assume the information display, interaction with users, which contains all the GUI objects such as text boxes, images ... Understand a simple way, it is a set of form or HTML files.

2.2 Use Case Analysis

2.2.1 Use-Case Realization Interaction Diagrams

2.2.1.1 Login

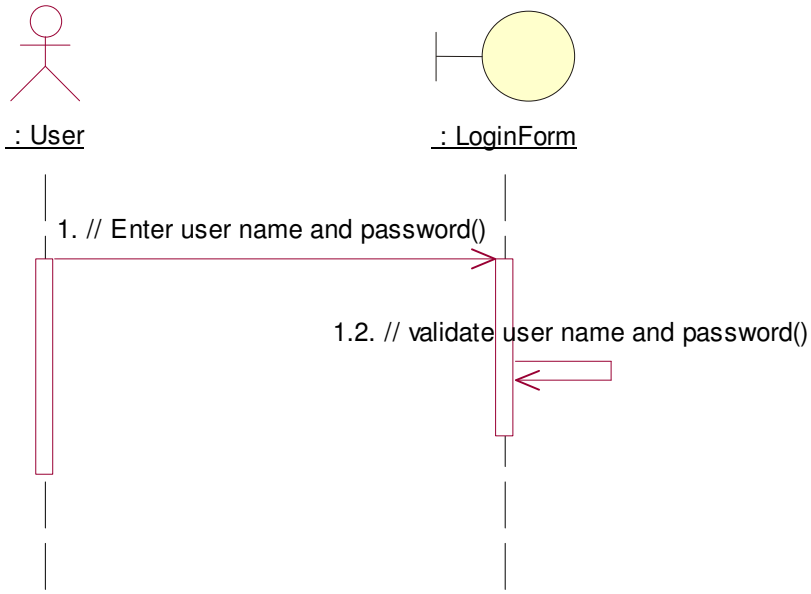

Figure 2.2.1: Login Basic - Flow

2.2.1.2 Change Profile

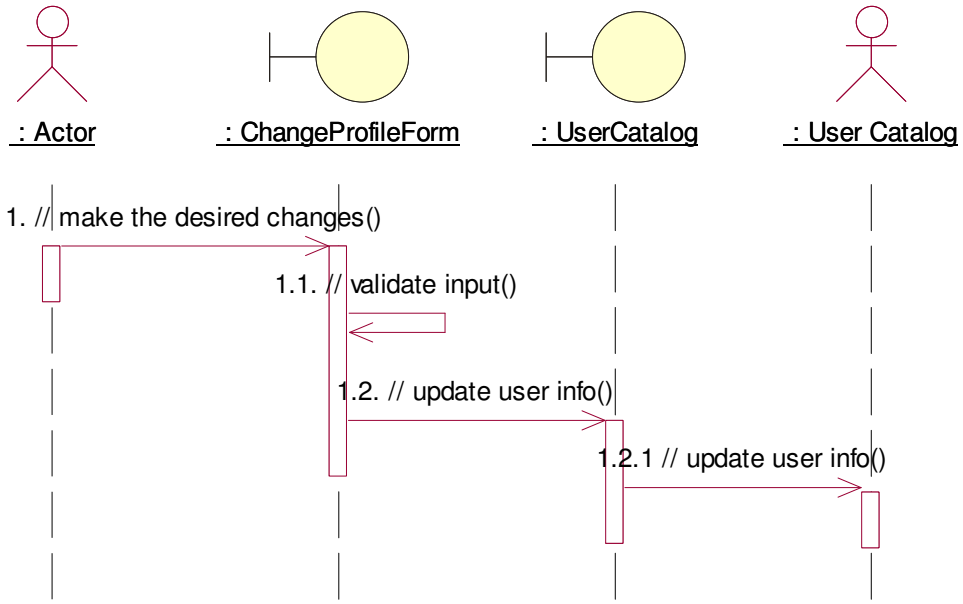

Figure 2.2.2: Change Profile - Basic Flow

2.2.1.3 View Menu Info

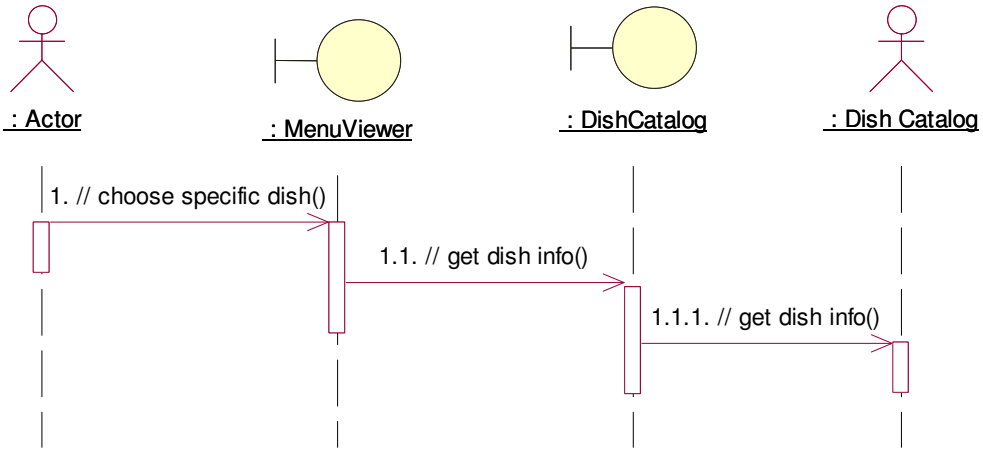

Figure 2.2.3: View Menu Info – Basic Flow

2.2.1.4 View Reports

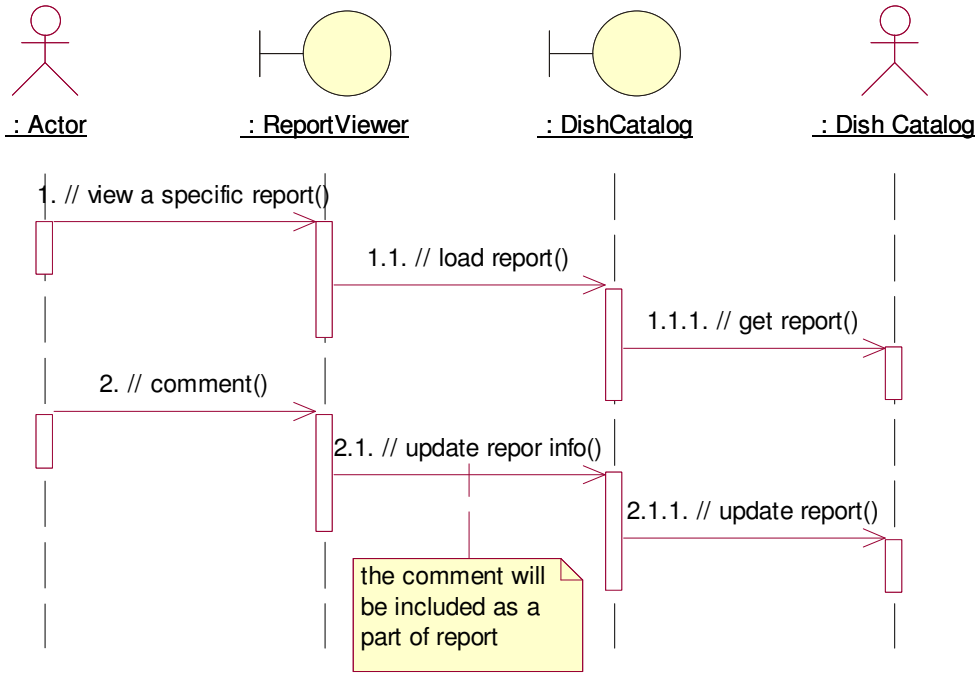

Figure 2.2.4: View Reports – Basic Flow

2.2.1.5 Maintain Menu Info

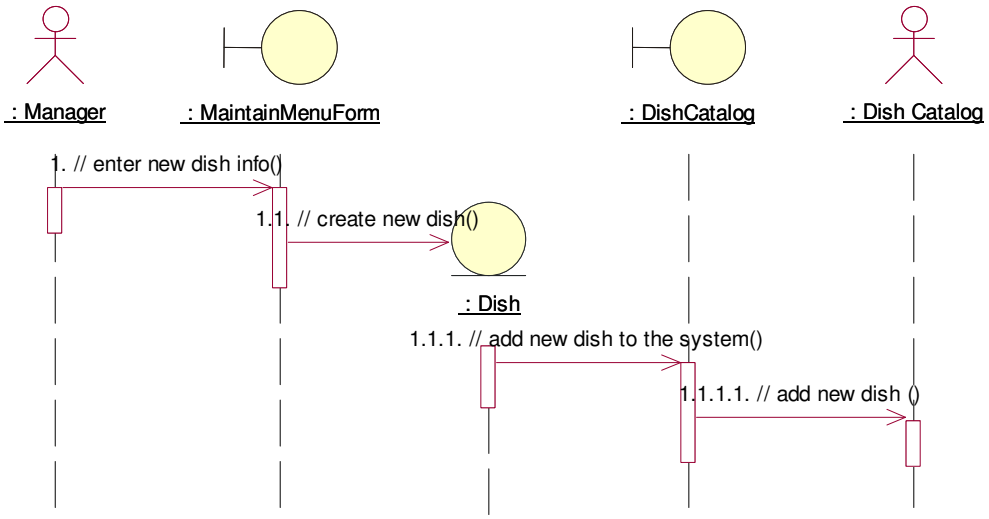

Figure 2.2.5: Maintain Menu Info – Basic Flow – Add option

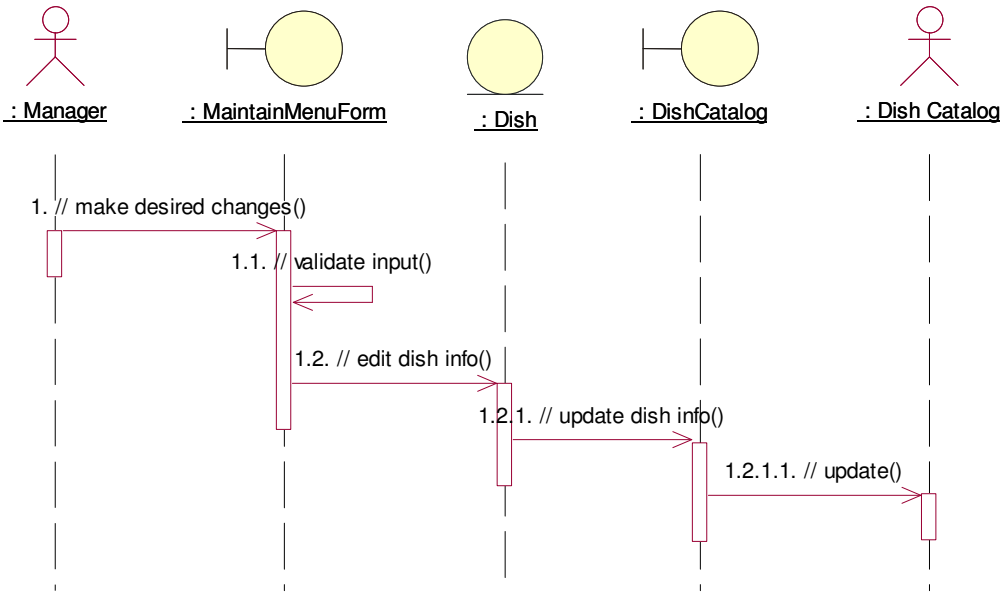

Figure 2.2.6: Maintain Menu Info – Basic Flow – Edit option

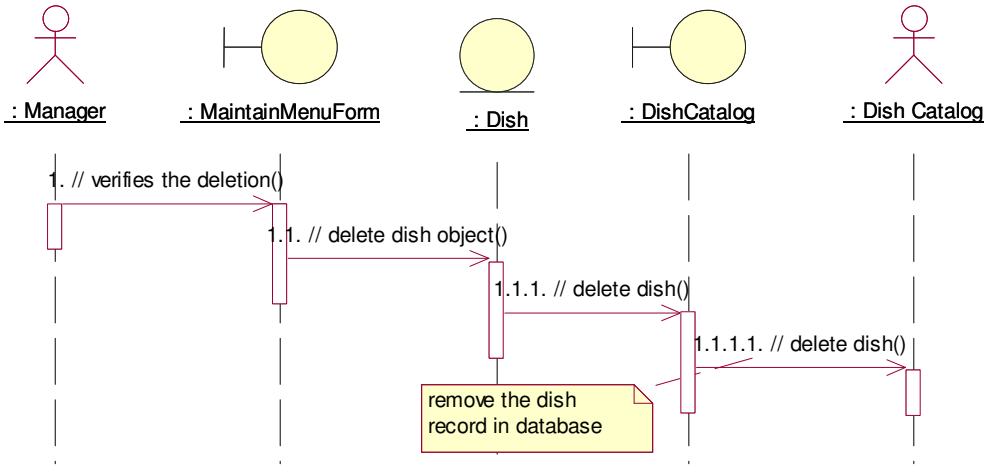

Figure 2.2.7: Maintain Menu Info – Basic Flow – Delete option

2.2.1.6 Maintain User Info

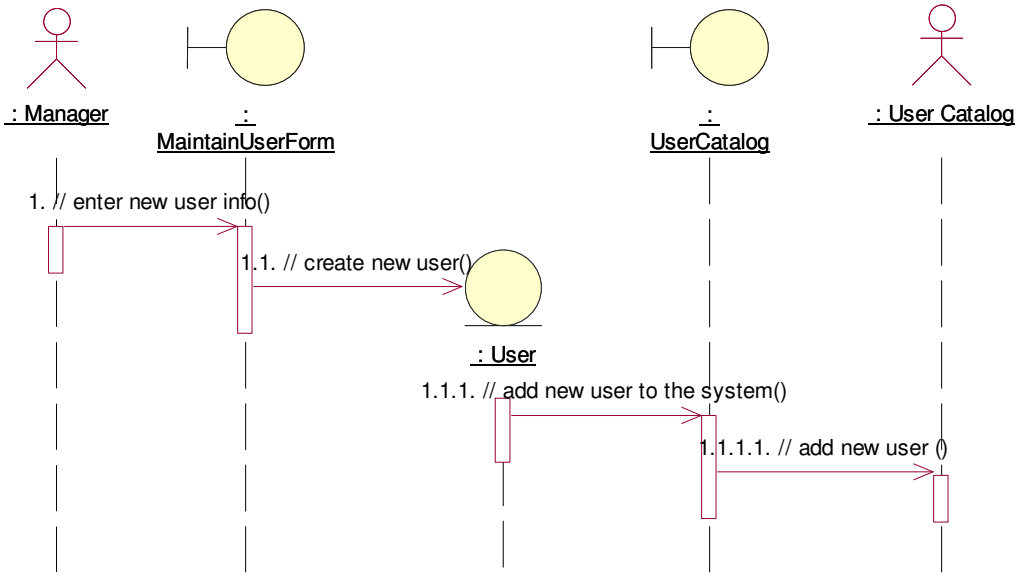

Figure 2.2.8: Maintain user info – Basic Flow – Add option

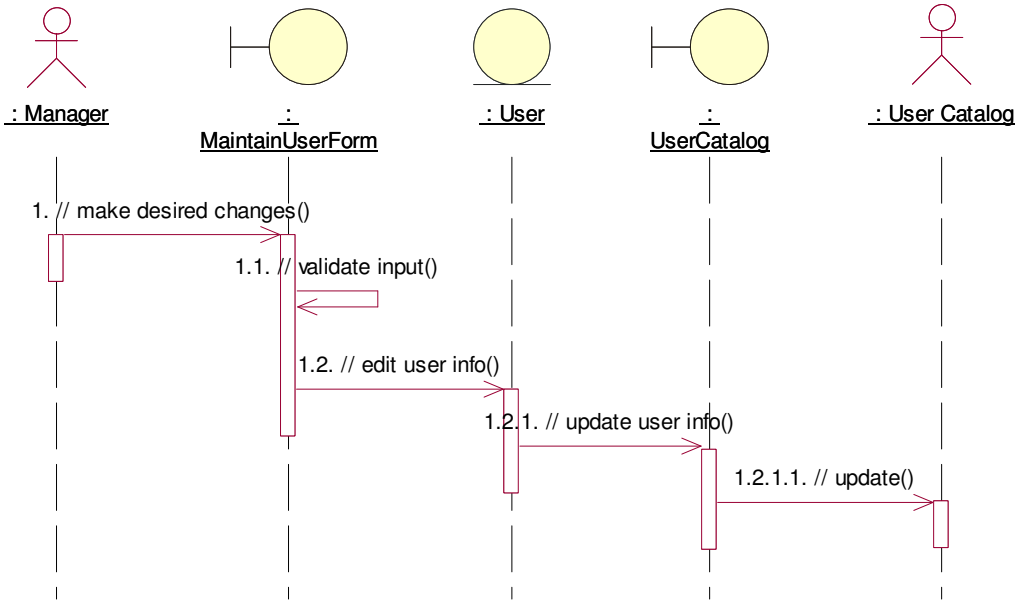

Figure 2.2.9: Maintain User Info – Basic Flow – Edit option

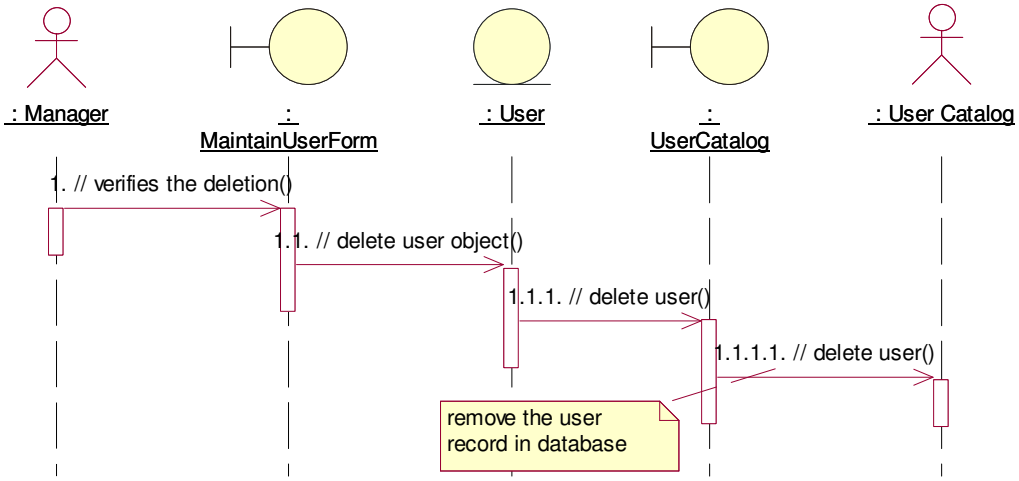

Figure 2.2.10: Maintain User Info – Basic Flow – Delete option

2.2.1.7 Manage Reports

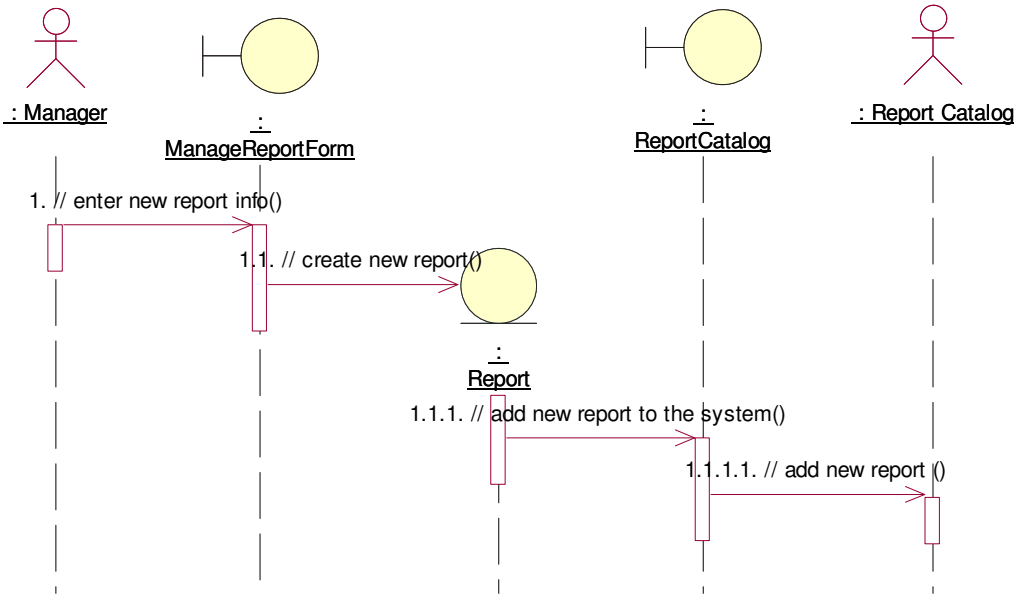

Figure 2.2.11: Manage Reports – Basic Flow – Add option

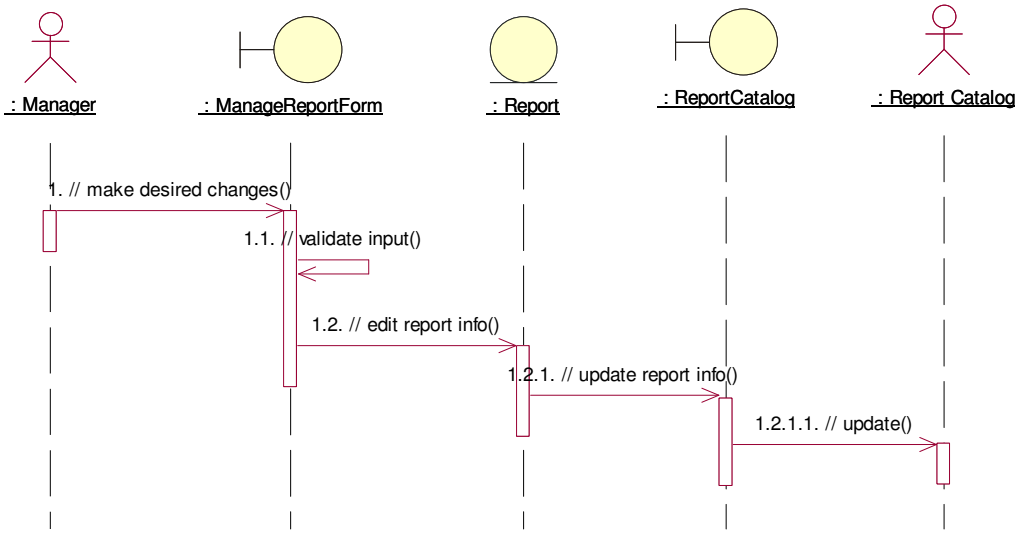

Figure 2.2.12: Manage Reports – Basic Flow – Edit option

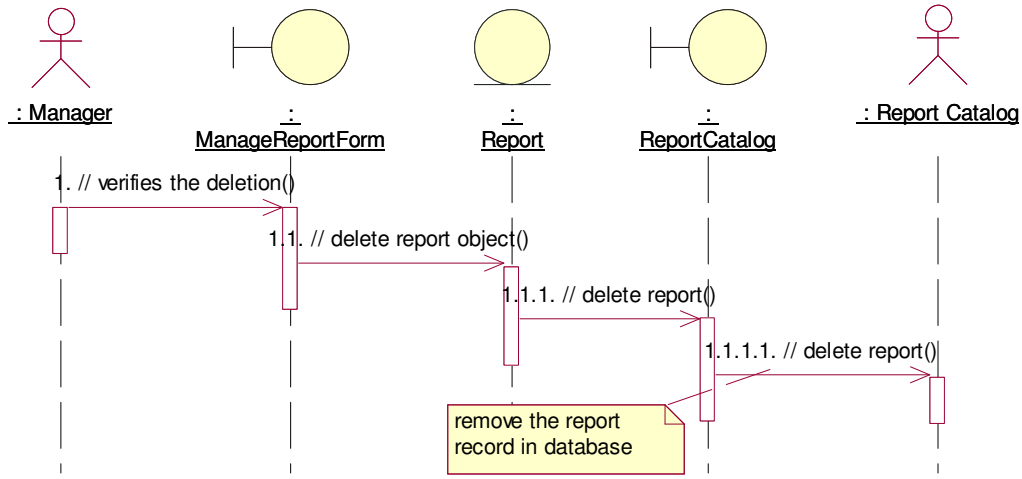

Figure 2.2.13: Manage Reports – Basic Flow – Delete Option

2.2.1.8 Manage Active Dishes

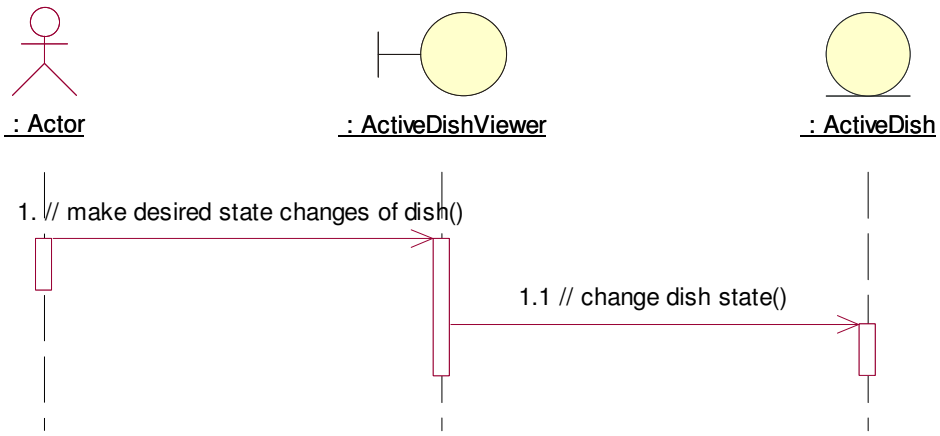

Figure 2.2.14: Manage Active Dishes – Basic Flow

2.2.1.9 Create New Order

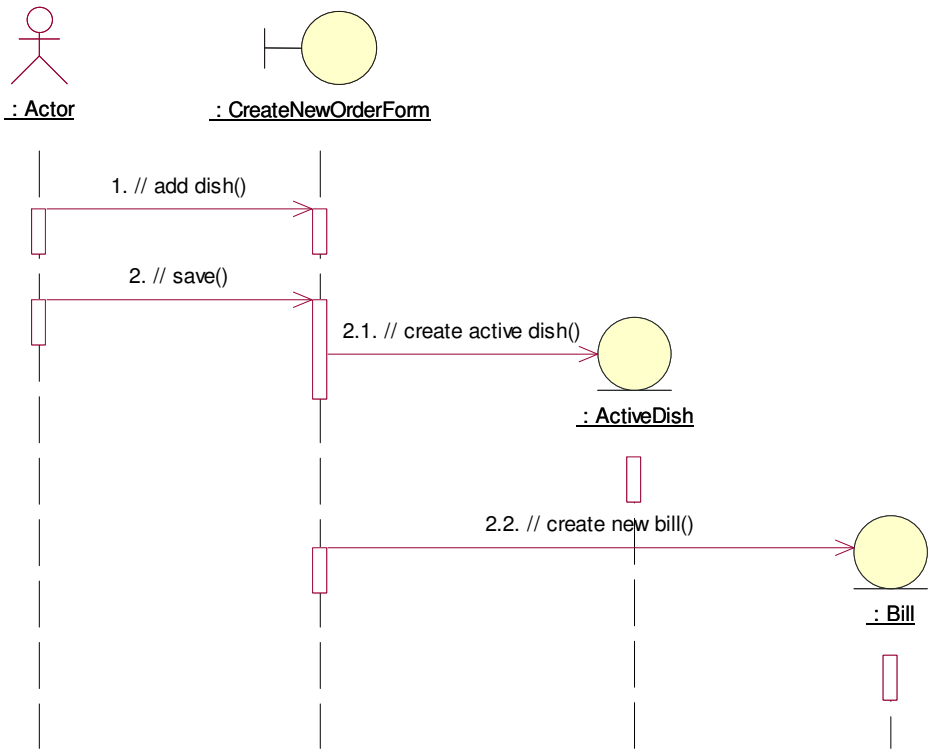

Figure 2.2.15: Create New Order – Basic Flow

2.2.1.10 Billing

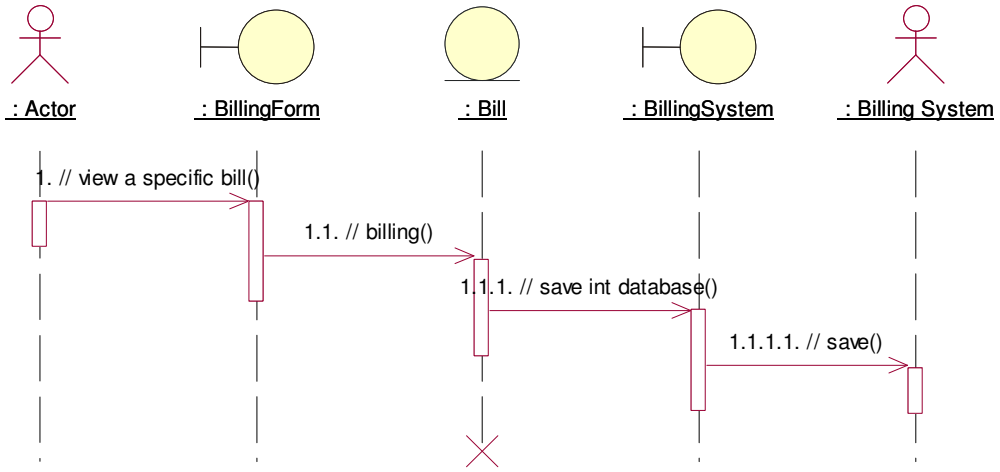

Figure 2.2.16: Billing – Basic Flow

2.2.1.11 Get Statistical Data

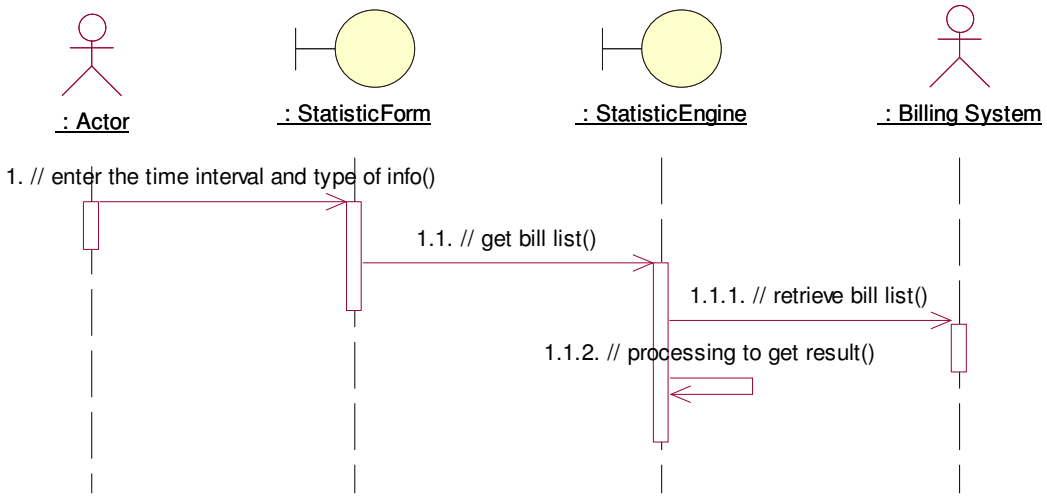

Figure 2.2.17: Get Statistical Data – Basic Flow

2.2.1.12 Chat

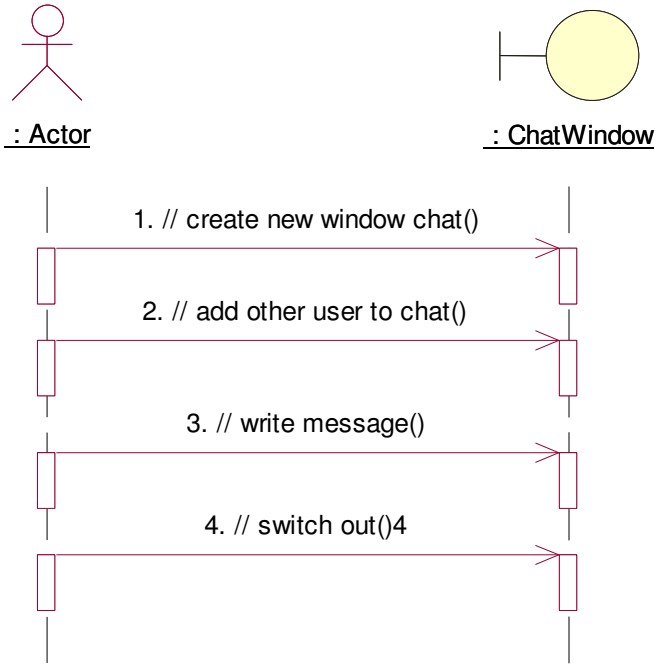

Figure 2.2.18: Chat – Basic Flow

2.2.1.13 Register

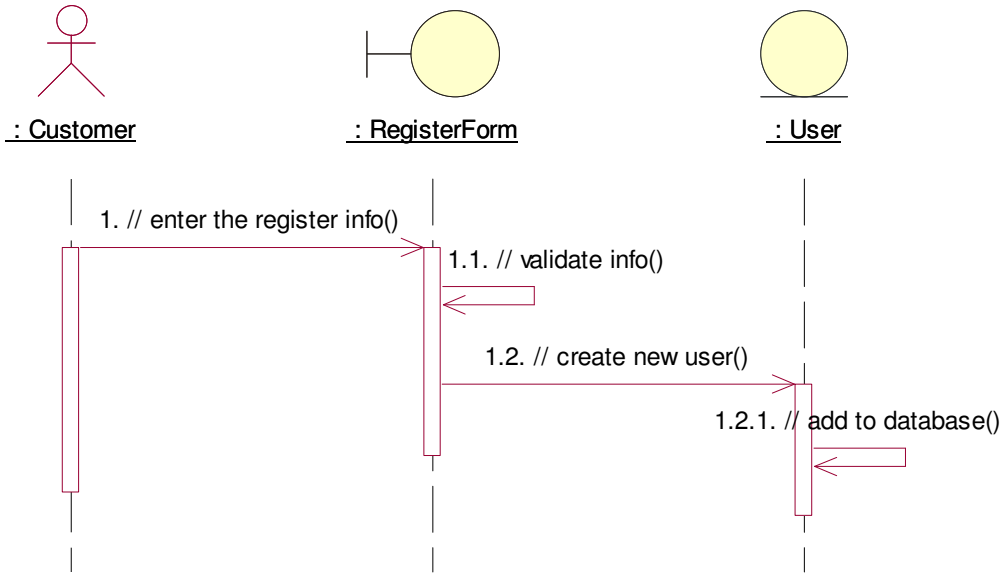

Figure 2.2.19: Register – Basic Flow

2.2.2 Use-Case Realization View of Participating Class (VOPCs)

2.2.2.1 Login

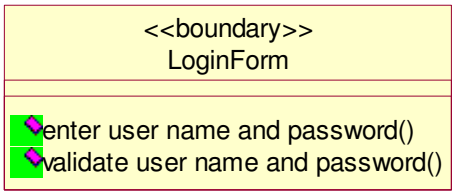

Figure 2.2.20: Login - VOPC

2.2.2.2 Change Profile

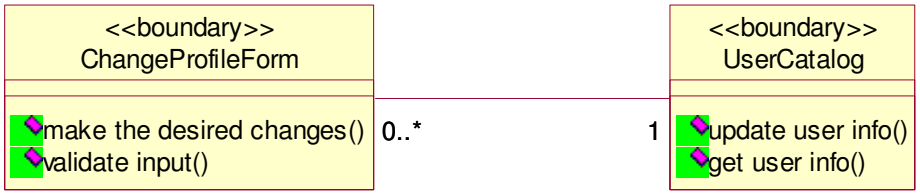

Figure 2.2.21: Change Profile -VOPC

2.2.2.3 View Menu Info

Figure 2.2.22: View Menu Info - VOPC

2.2.2.4 View Reports

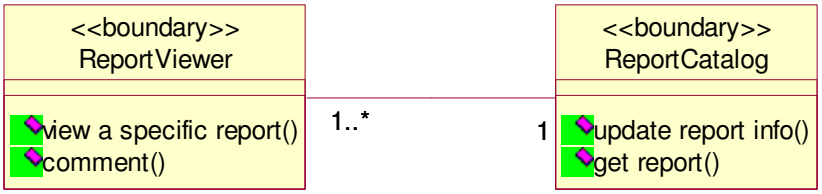

Figure 2.2.23: View Reports - VOPC

2.2.2.5 Maintain Menu Info

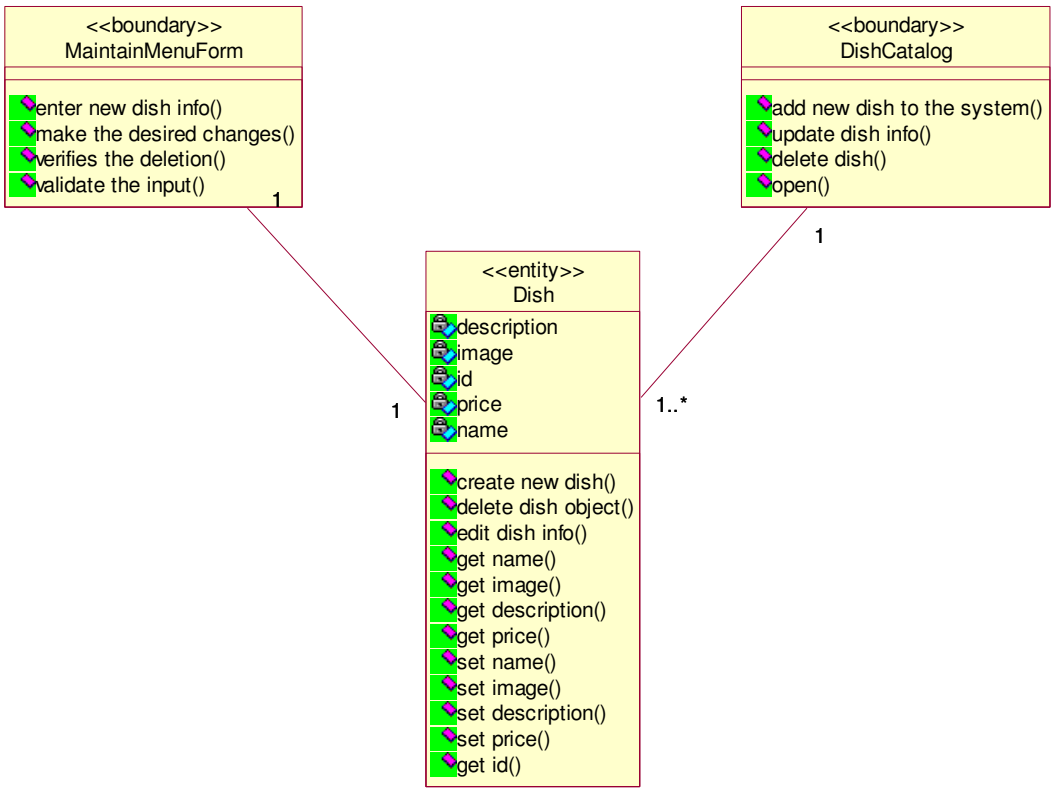

Figure 2.2.24: Maintain Menu Info – Add option -

2.2.2.6 Maintain User Info

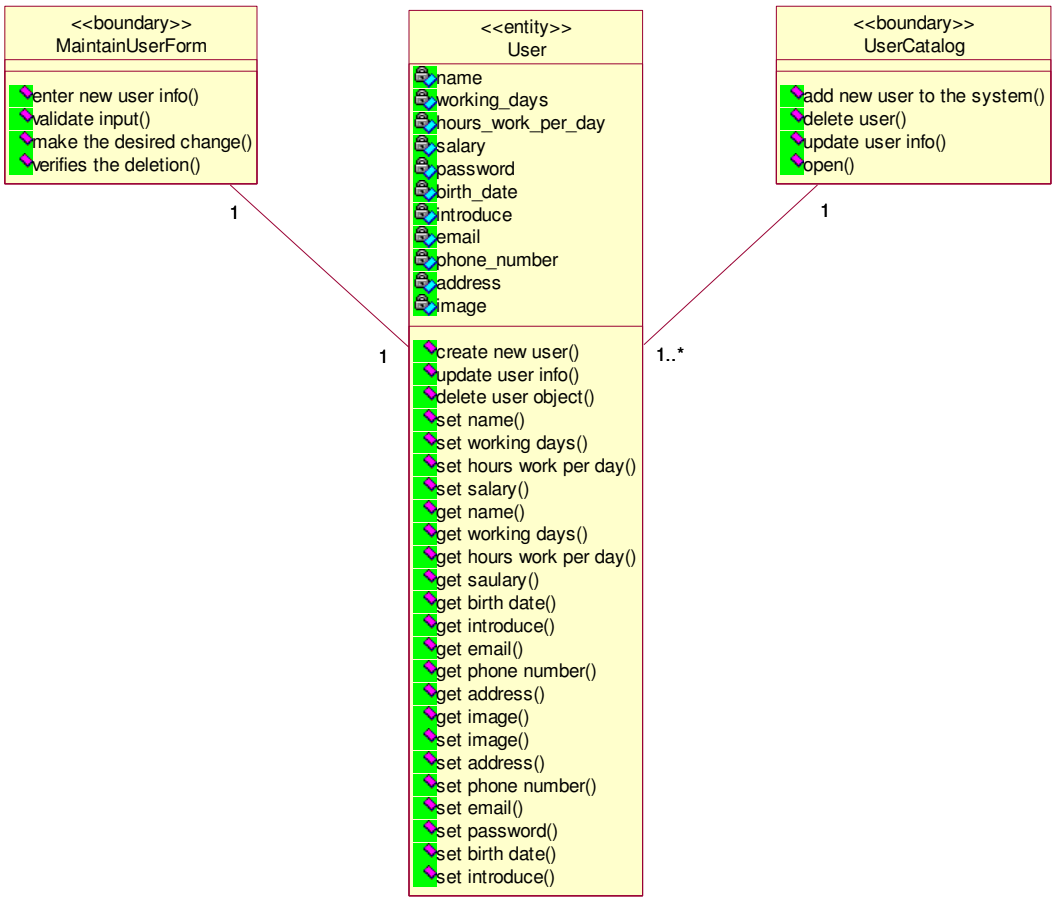

Figure 2.2.25: Maintain User Info - VOPC

2.2.2.7 Manage Reports

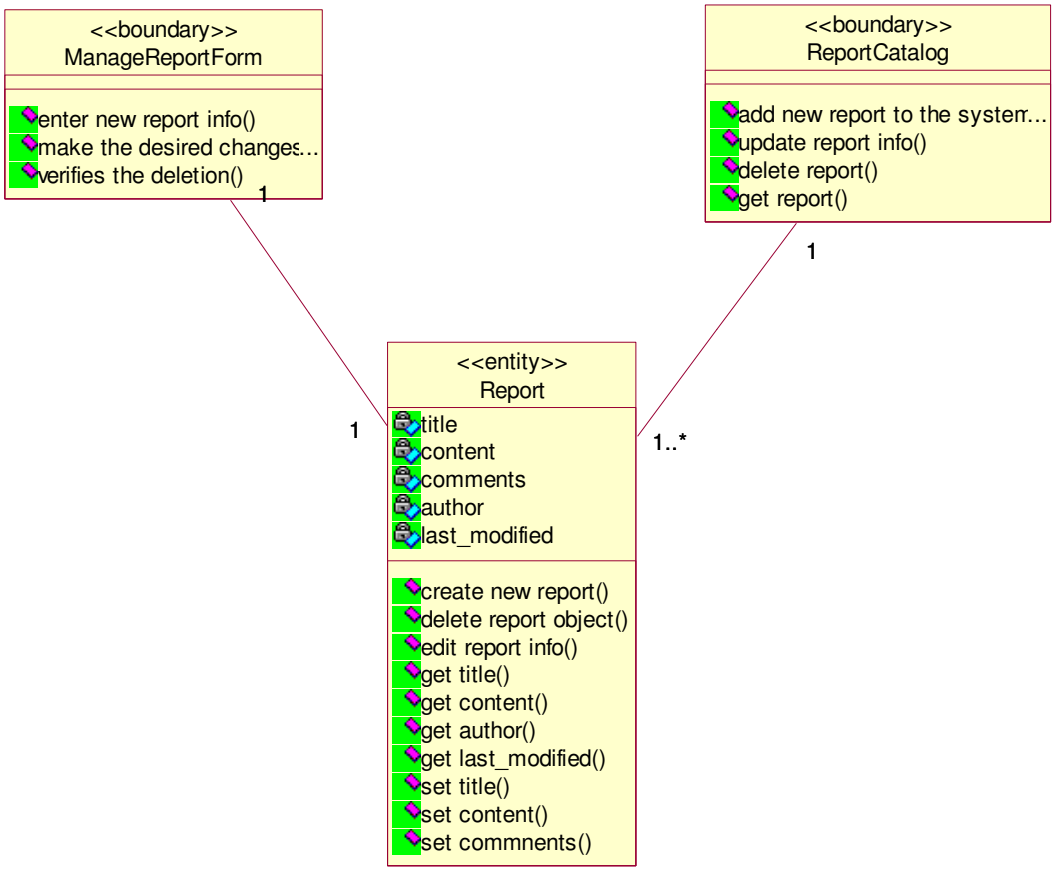

Figure 2.2.26: Manage Reports - VOPC

2.2.2.8 Manage Active Dishes

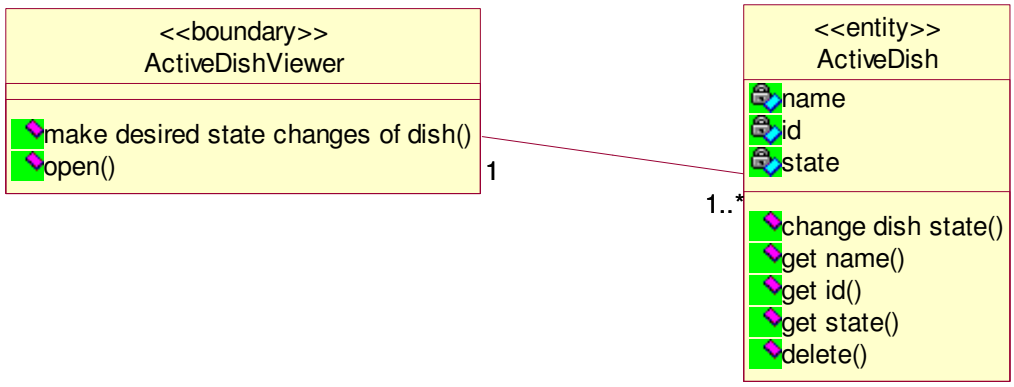

Figure 2.2.27: Manage Active Dishes - VOPC

2.2.2.9 Create New Order

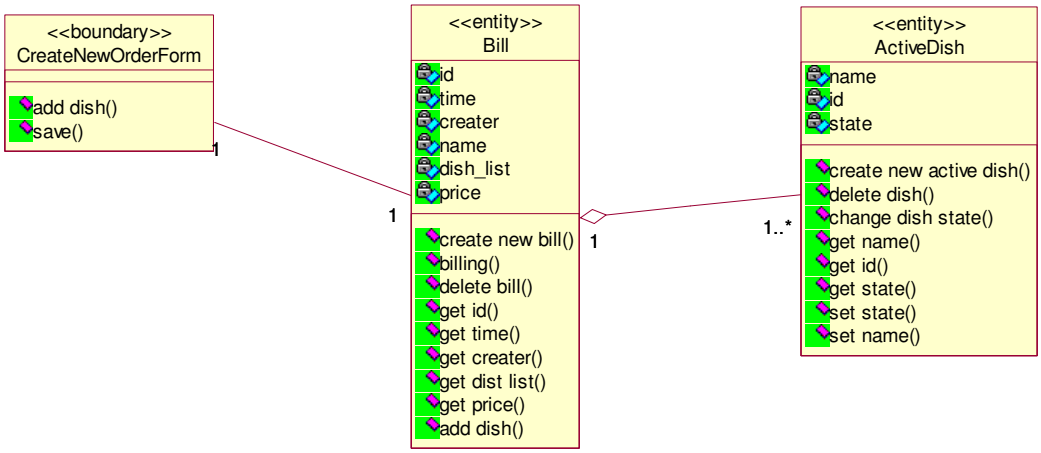

Figure 2.2.28: Create New Order - VOPC

2.2.2.10 Billing

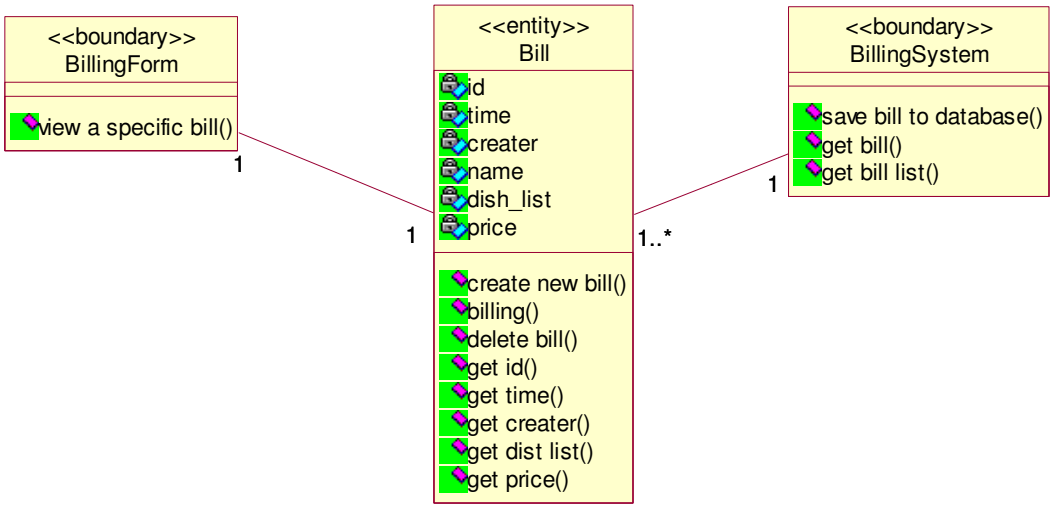

Figure 2.2.29: View Bill List - VOPC

2.2.2.11 Get Statistical Data

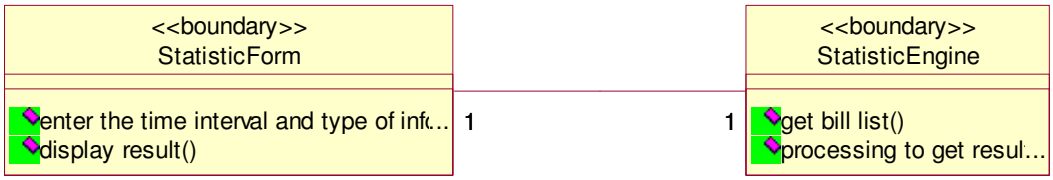

Figure 2.30: Get Statistic Data - VOPC

2.2.2.12 Chat

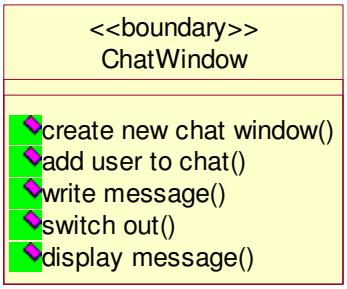

Figure 2.2.31: Chat - VOPC

2.2.2.13 Register

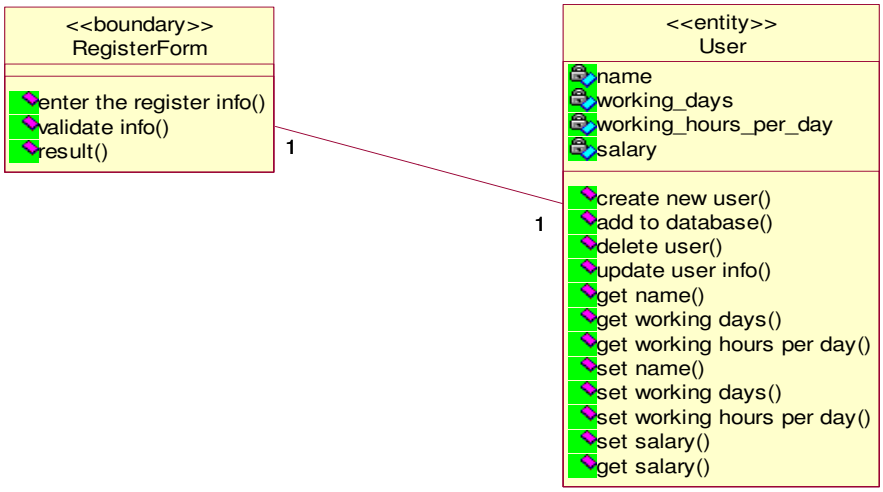

Figure 2.2.32: Register - VOPC

2.2.3 Analysis-Class-To-Analysis-Mechanism Map

Analysis Class	Analysis Mechanism(s)
LoginForm	Error detection /handling /reporting
ChangeProfileForm	Error detection /handling /reporting
UserCatalog	Persistency, Security
MenuViewer	None
DishCatalog	Persistency, Security
ReportViewer	None
ReportCatalog	Persistency, Security
MaintainMenuForm	Error detection /handling /reporting
Dish	Security
MaintainUserForm	Error detection /handling /reporting
User	Security
UserCatalog	Persistency, Security
ReportManageForm	None
Report	Persistency, Security
ActiveDish	Distribution, Process control and Synchronization
ActiveDishViewer	None
CreateOrderForm	None
Bill	Persistency
BillingForm	None
BillingSystem	Security
StatisticForm	None
StatisticEngine	Redundancy, Error detection /handling /reporting
ChatWindow	Process control and Synchronization
RegisterForm	Error detection /handling /reporting

3 Restaurant Manager Design

3.1 Identify Design Elements

3.1.1 Subsystem Context Diagram

3.1.1.1 Billing subsystem

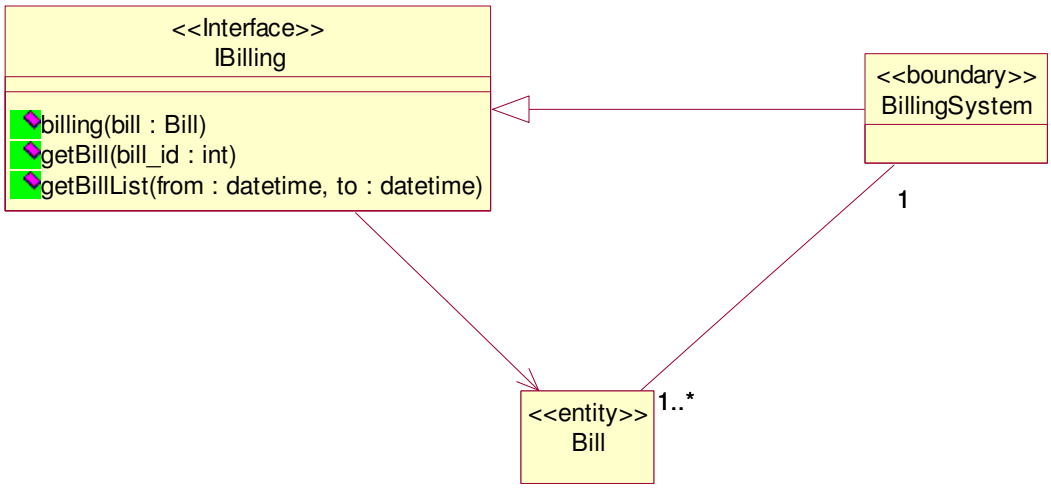

Figure 3.1.1: Billing subsystem

Subsystem Interface Descriptions

IBilling: encapsulates the communication with the billing system

Billing: store the given bill into database with adding some information.

getBill: return the bill corresponding to the given bill id.

getBillList: return the list of bills that in the given time interval (from "from" to "to" time).

3.1.1.2 Maintain Catalog Subsystem

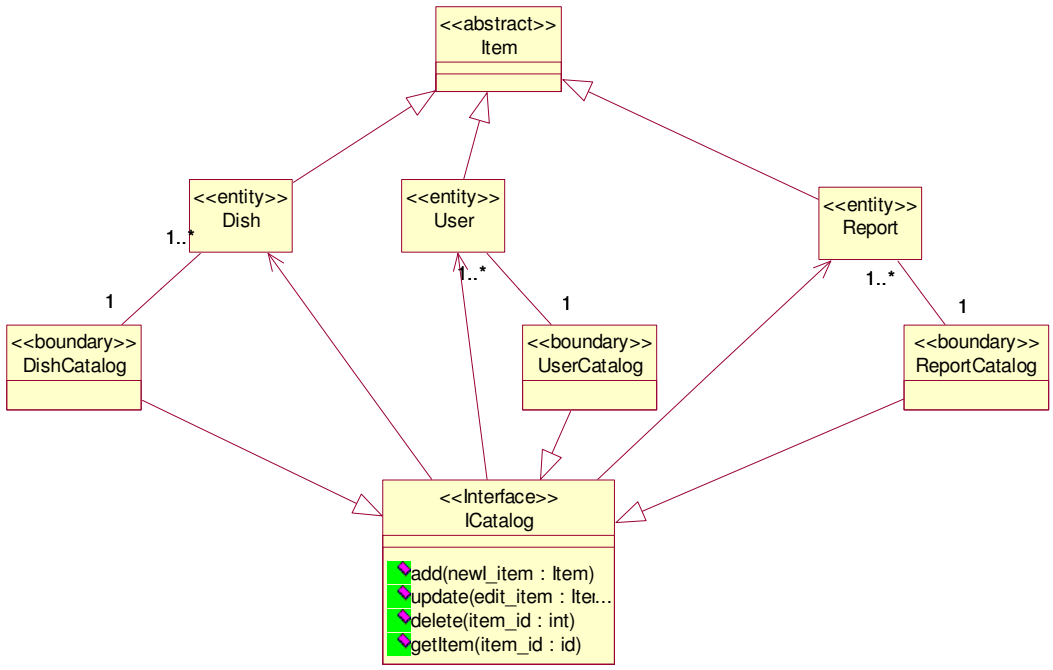

Figure 3.1.2: Maintain Catalog subsystem

Subsystem Interface Descriptions

IMaintainCatalog: encapsulates the action of add, update or delete a data item.

Data: may be a Dish, a Report or a User object.

add: remove data item from the catalog

update: change the data item in catalog to the given data item

delete: remove a data item from the catalog with given item_id.

3.1.2 4Analysis-Class-to-Design-Element Map

Analysis Class	Design Element
LoginForm	LoginForm
ChangeProfileForm	ChangeProfileForm
UserCatalog	UserCatalog subsystem

	ICatalog interface
MenuViewer	MenuViewer
DishCatalog	DishCatalog subsystem
ReportViewer	ReportViewer
ReportCatalog	ReportCatalog sybsystem
MaintainMenuForm	MenuForm
	AddNewDishForm
	EditDishForm
	DeleteDishForm
Dish	Dish
	Item abstract
MaintainUserForm	UserForm
	AddNewUserForm
	EditUserForm
	DeleteUserForm
User	User
ReportManageForm	ReportForm
	WriteReportForm
	EditReportForm
	DeleteReportForm
Report	Report
ActiveDish	ActiveDish
ActiveDishViewer	ActiveDishViewer
CreateOrderForm	CreateNewOrderForm
Bill	Bill
BillingForm	BillingForm
BillingSystem	BillingSystem subsystem
	IBilling interface
StatisticForm	StatisticForm
StatisticEngine	StatisticEngine
ChatWindow	ChatWindow
RegisterForm	RegisterForm

3.1.3 Design-Element-to-Owning-Package Map

Design Element	**“Owning” packet**

LoginForm	Controller::GUI controller
ChangeProfileForm	
MenuViewer	
ReportViewer	
MenuForm	
AddNewDishForm	
EditNewDishForm	
DeleteDishForm	
UserForm	
AddNewUserForm	
EditUserForm	
DeleteUserForm	
ReportForm	
RegisterForm	
CreateNewOrderForm	
ActiveDishViewer	
BillingForm	
StatisticForm	
ChatWindow	
RegisterForm	
WriteReportForm	
EditReportForm	
DeleteReportForm	
DishCatalog	Controller::Subsystem
UserCatalog	
BillingSystem	
ReportCatalog	
ICatalog	
IBilling	
Dish	Controller::Restaurant Elements
Item	
User	
Report	
ActiveDish	
Bill	
StatisticEngine	Controller::Activity

3.1.4 Architectural Components and Their Dependencies

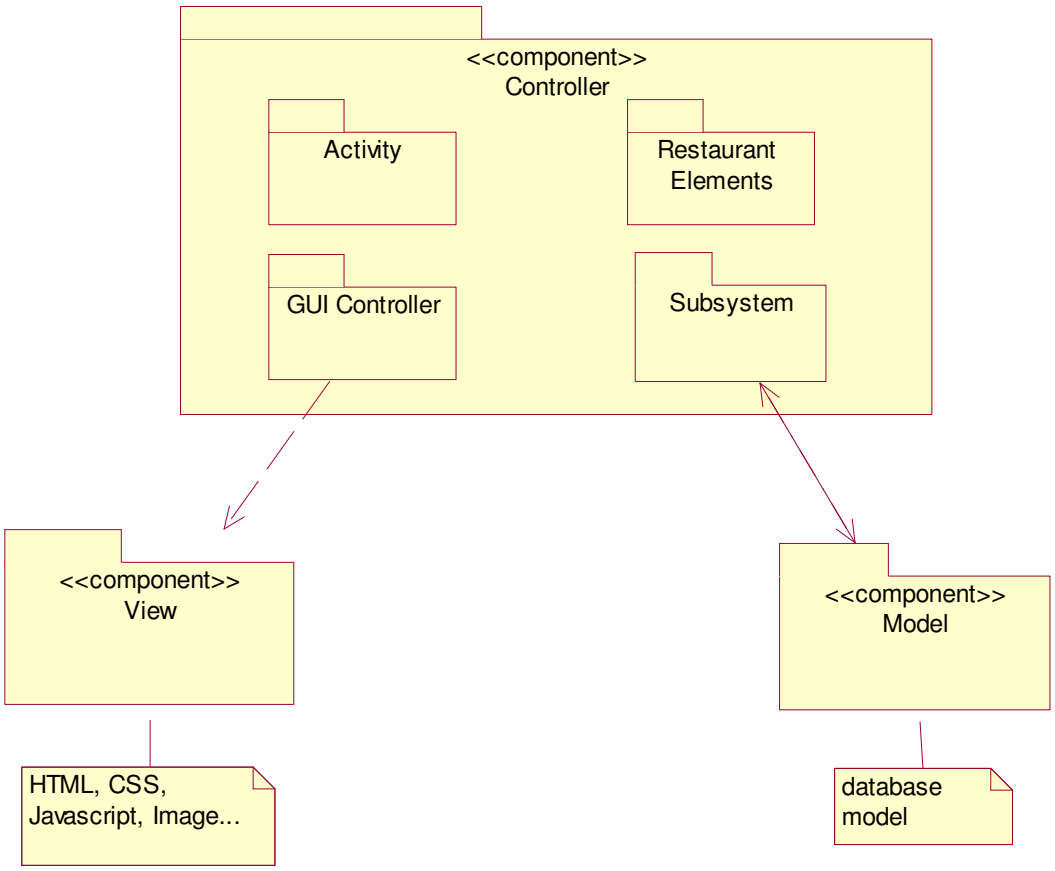

Figure 3.1.3: Architectural Components

3.1.5 Packages and Their Dependency

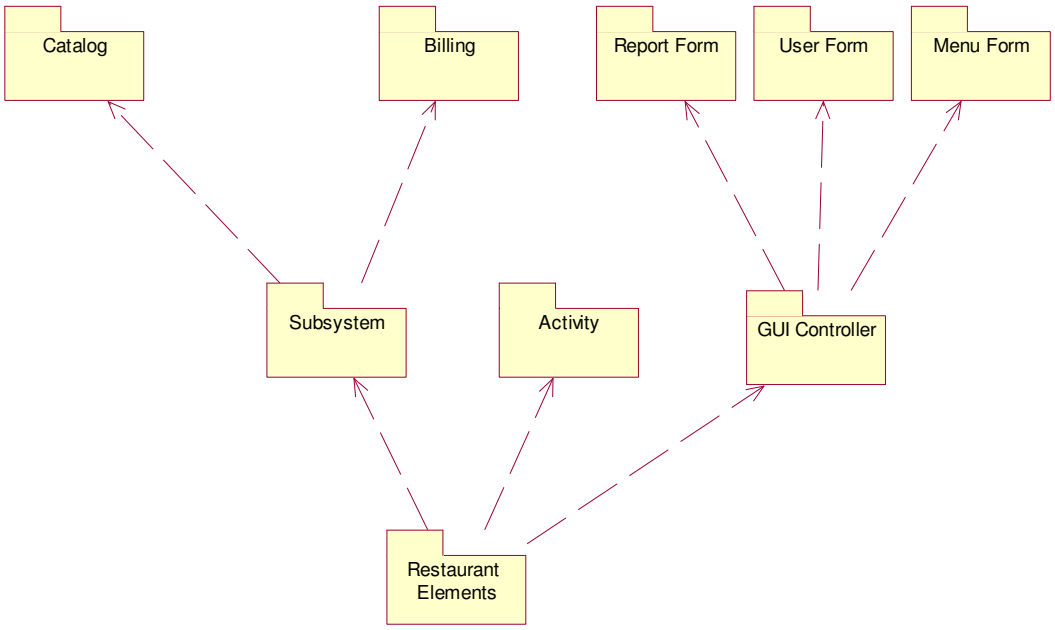

Figure 3.1.4: Packages Dependencies

Package Descriptions

Catalog: contains the interface ICatalog, UserCatalog, DishCatalog, ReportCatalog.

Billing: contains the interface IBiling, BillingSystem, BillingForm

Subsystem: import Catalog and Billing packages.

Activity: contains StatisticEngine.

Report Form: contains ReportForm, WriteReportForm, EditReportForm, DeleteReportForm, ReportViewer.

User Form: contains UserForm, AddNewUserForm, EditUserForm, DeleteUserForm.

Menu Form: contains Dish Form, AddNewDishForm, EditDishForm, DeleteDishForm, MenuViewer.

GUI Controller: contains RegisterForm, StatisticForm, BillingForm, …. And imports Report Form, User Form and ReportForm packages.

Restaurant Elements: Import all above package and contains some other elements.

3.2 Describe the Run-time Architecture

This part describes the Restaurant Manager System Concurrency

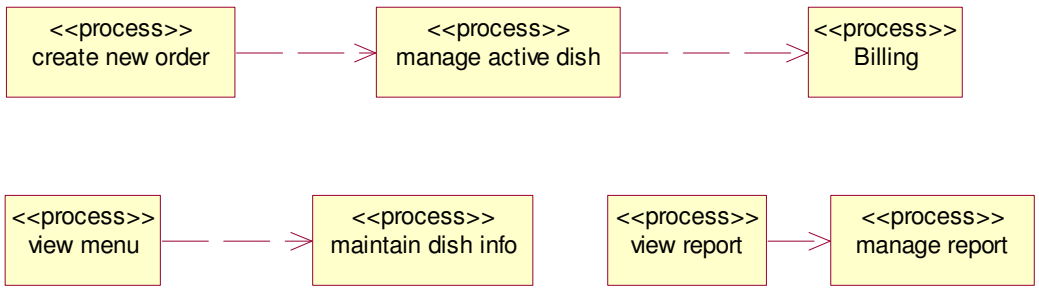

Figure 3.2.1: Process Model

3.3 Describe Distribution

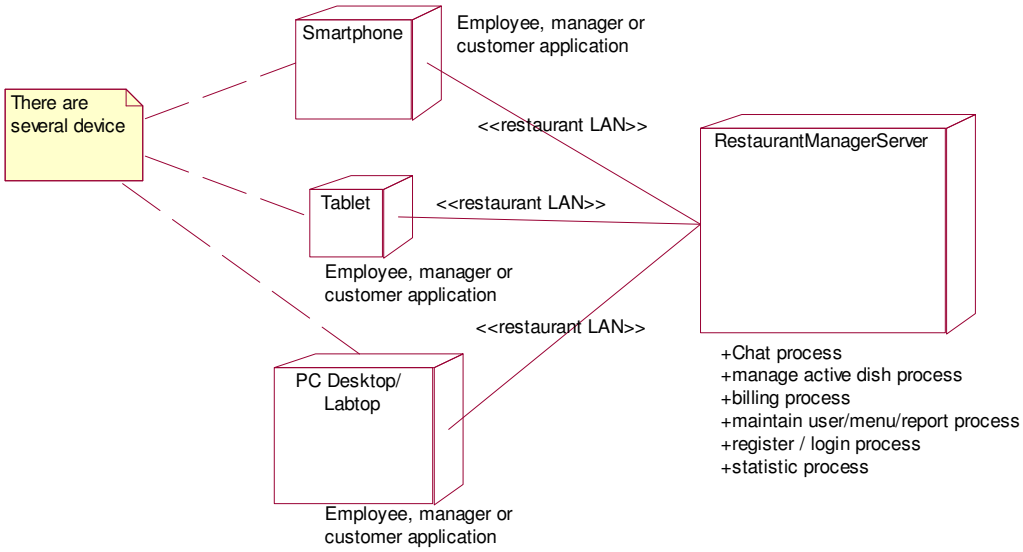

Figure 3.3.1: Deployment Model

3.4 Use Case Design

This part describes the use case in the consideration of every analysis mechanism that were defined before.

None

3.5 Subsystem Design

None.

3.6 Class Design

This part describes the all class structure and their description also the total class diagram and their dependency.

3.6.1 Describe each class or interface

3.6.1.1 LoginForm

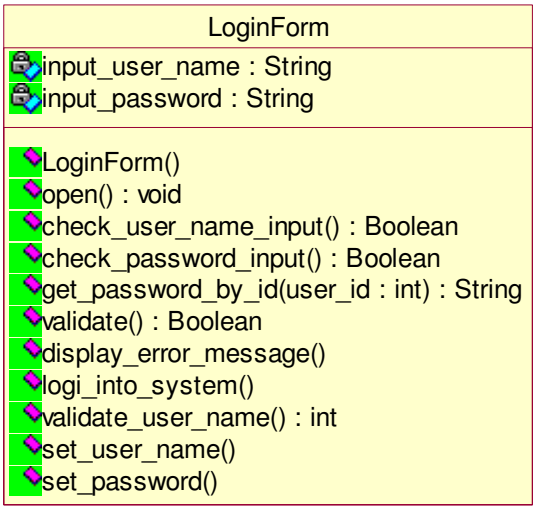

Figure 3.6.1: Class "LoginForm"

3.6.1.2 ChangeProfileForm

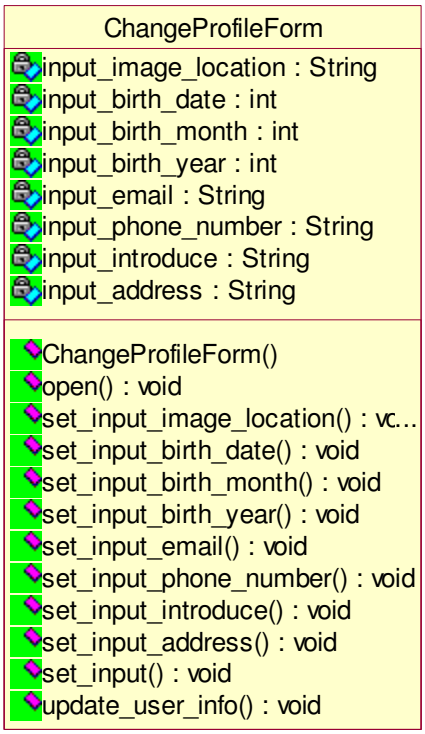

Figure 3.6.2: Class "ChangeProfileForm"

3.6.1.3 ICatalog (interface)

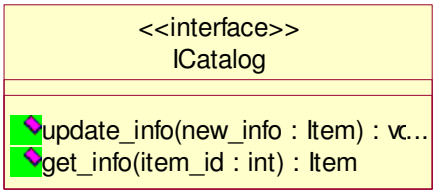

Figure 3.6.3: Interface "ICatalog"

3.6.1.4 UserCatalog

Implements the interface "ICatalog"

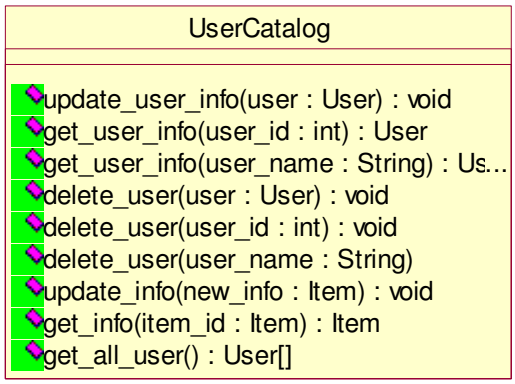

Figure 3.6.4: Class "UserCatalog"

3.6.1.5 MenuViewer

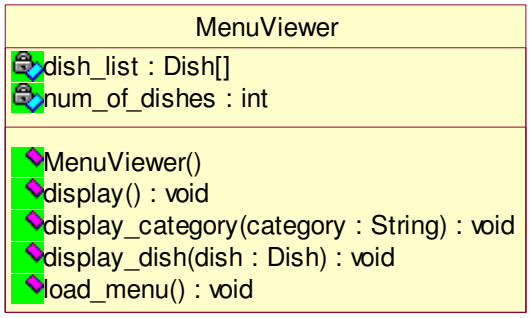

Figure: 3.6.5: Class "MenuViewer"

3.6.1.6 DishCatalog

Implements the interface "ICatalog"

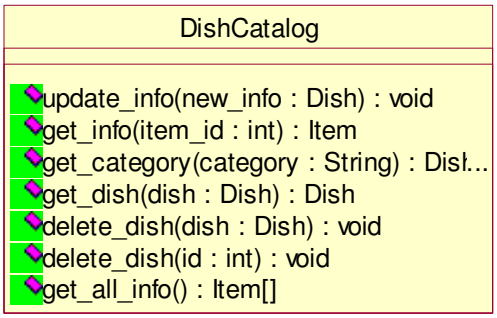

Figure 3.6.6: Class DishCatalog

3.6.1.7 ReportViwer

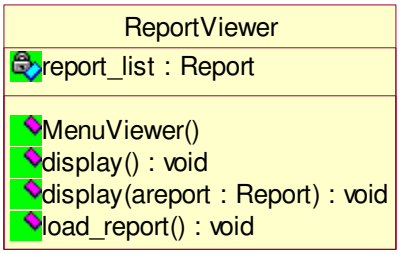

Figure 3.6.7: Class "ReportViewer"

3.6.1.8 ReportCatalog

Implements the interface "ICatalog"

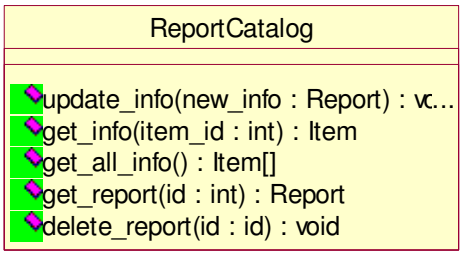

Figure 3.6.8: Class "ReportCatalog"

3.6.1.9 MenuForm

Inherits the class "MenuViewer" with some modifies in the display function which are adding add/edit/delete button for prompt maintaining.

3.6.1.10 AddNewDishForm

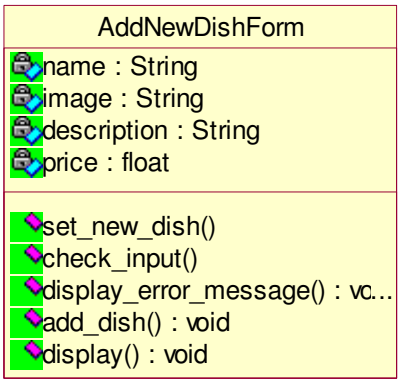

Figure 3.6.9: Class "AddNewDishForm"

3.6.1.11 EditDishForm

Inherits the class AddNewDishForm with override in the display function to modify the form input to editable.

3.6.1.12 DeleteDishForm

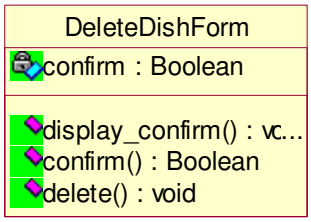

Figure 3.6.10: Class "DeleteDishForm"

3.6.1.13 Item

Is only a dummy abstract class.

3.6.1.14 Dish

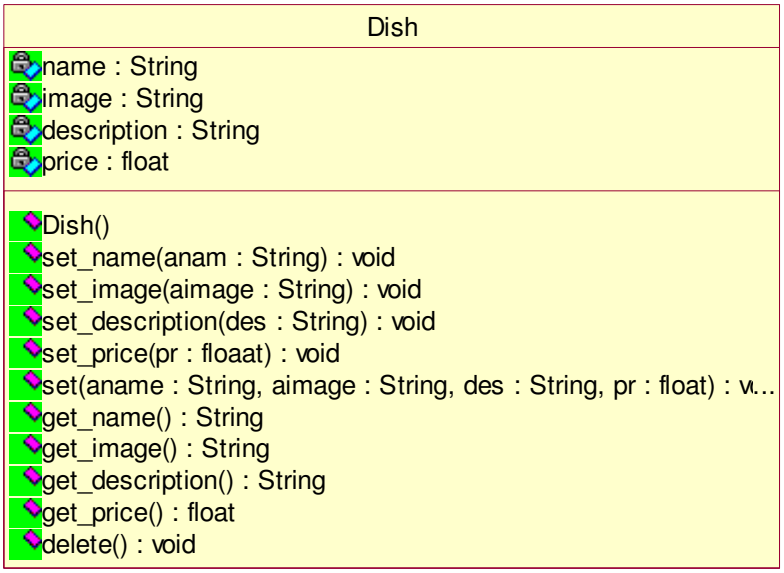

Figure 3.6. 11: Class "Dish"

3.6.1.15 UserForm

This class inherits class "ChangeProfileForm" and add some new follow attributes: name, working_days, hour_work_per_day, salary. The display function of MenuForm is modified with disable in some attributes that the manage can't modify.

3.6.1.16 AddNewUserForm

This class inherits the class “UserForm” with overriding in the display function by modifying the input form to editable.

3.6.1.17 DeleteUserForm

This class inherits the class “DeleteDishForm” with overriding in the display function.

3.6.1.18 User

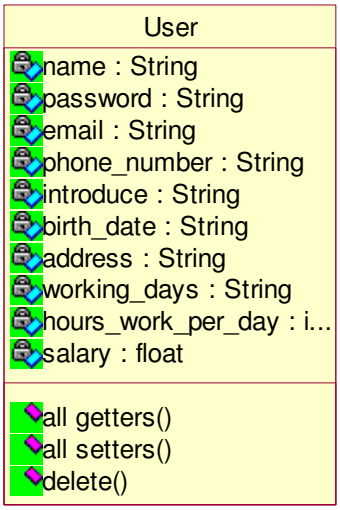

Figure 3.6.12: Class “User”

3.6.1.19 ReportForm

This class inherit the class “ReportViewer” and override in display function with adding the buttons to prompt the manager manages the reports.

3.6.1.20 WriteReportForm

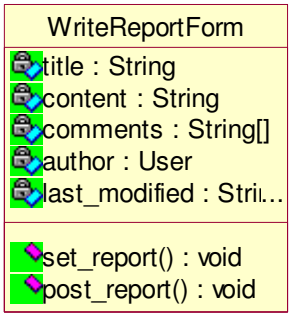

Figure 3.6.13: Class “WriteReportForm”

3.6.1.21 EditReportForm

This class inherits the class "WriteReportForm" with overriding in the post function by changing the input forms to editable.

3.6.1.22 DeleteReportForm

This class inherits the class "DeleteDishForm" with some modifies in the display_confirm function.

3.6.1.23 Report

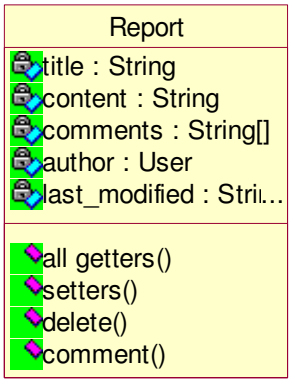

Figure 3.6.14: Class "Report"

3.6.1.24 ActiveDish

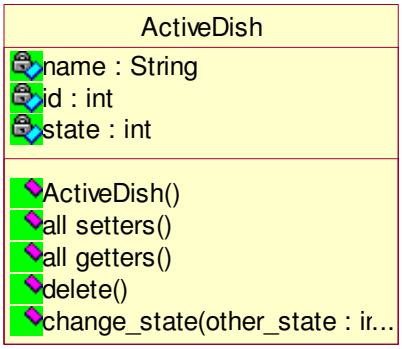

Figure 3.6.15: Class "ActiveDish"

3.6.1.25 ActiveDishViewer

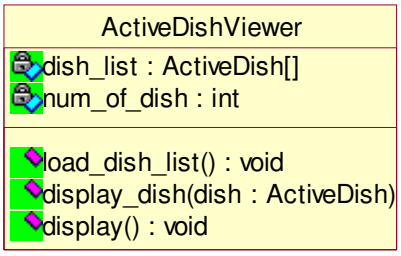

Figure 3.6.16: Class "ActiveDishViewer"

3.6.1.26 CreateNewOrderForm

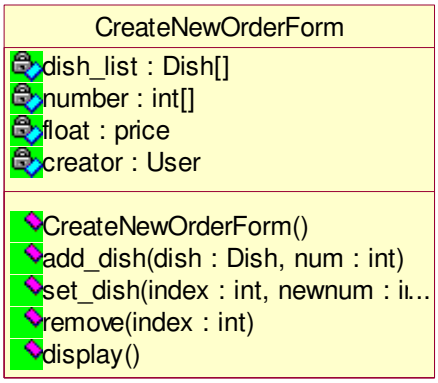

Figure 3.6.17: Class "CreateNewOrderForm"

3.6.1.27 Bill

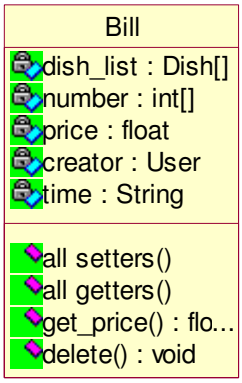

Figure 3.6.18: Class "Bill"

3.6.1.28 BillingForm

This class inherits the class "CreateOrderForm" with overrding in the display function so that the user is prompted for billing.

3.6.1.29 IBilling

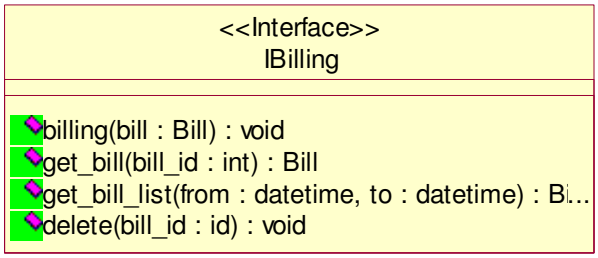

Figure 3.6.29: Interface "Billing"

3.6.1.30 BillingSystem

This class implements the interface "IBilling".

3.6.1.31 Statistic Form

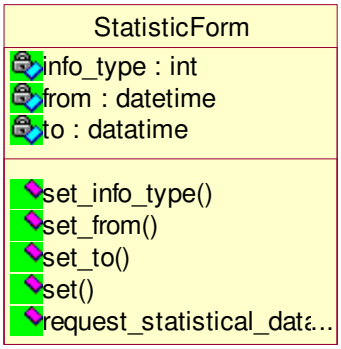

Figure 3.6.30: Class "StatisticForm"

3.6.1.32 StatisticEngine

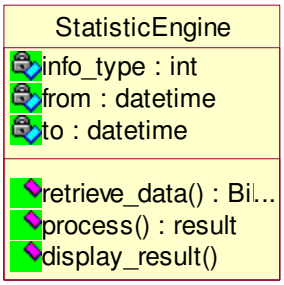

Figure 3.6.31: Class "StatisticEngine"

3.6.1.33 ChatWindow

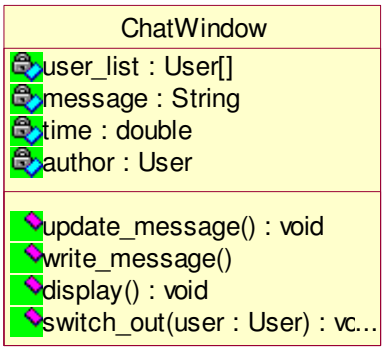

Figure 3.6.32: Class "ChatWindow"

3.6.1.34 RegisterForm

This class inherits the class "AddNewUserForm" but the input form is different. So all function will be overrided.

3.6.2 Class diagram in total

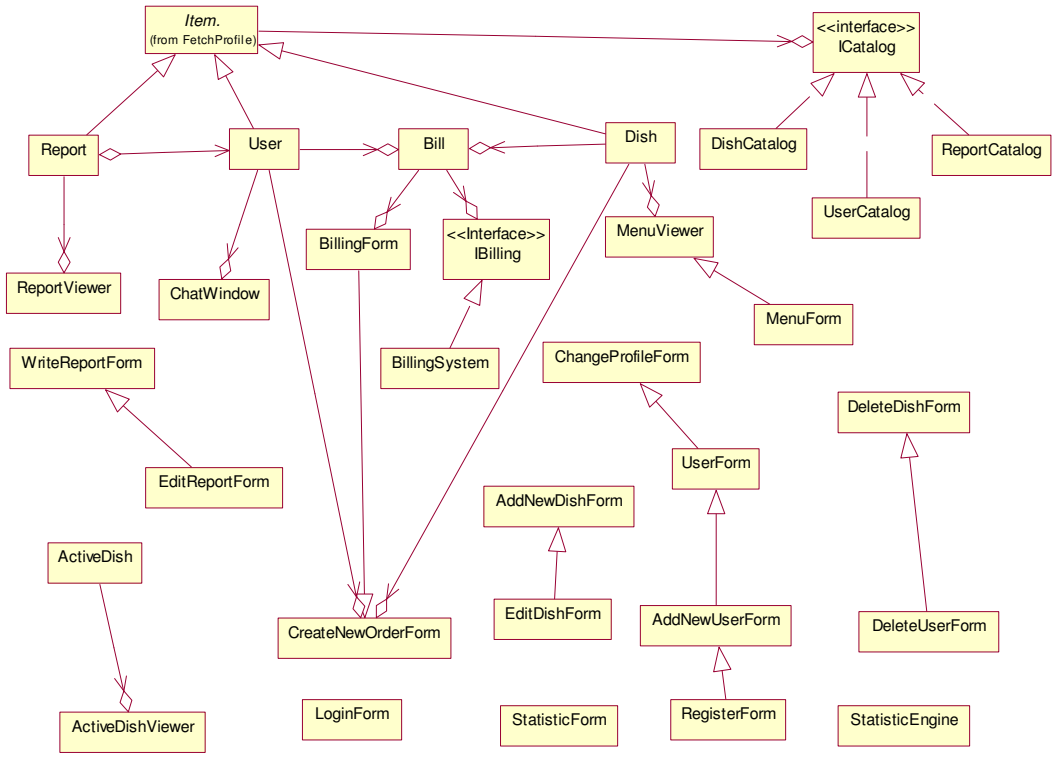

Figure 3.6.33: The total class diagram