 **MINISTRY OF EDUCATION AND TRAINING**

**FPT UNIVERSITY**

Capstone Project Document

**Barcode Order Eateries**

|  |  |
| --- | --- |
| **Nhóm số 5** | |
| **Group members** | Phạm Đăng Nam – Team Leader – SE61539  Nguyễn Khánh Linh – Team Member – SE61692  Nguyễn Đức Hoàng – Team Member – SE61505  Lưu Đức Phong – Team Member – SE61605 |
| **Supervisor** | Mr. Kiều Trọng Khánh |
| **Ext. Supervisor** | N/A |
| **Capstone Project code** | BOE |

-Ho Chi Minh City, *05/2017*-

*This page is intentionally left blank*

*With all our love and sincerity, we send our thanks from the bottom of heart to our teachers.*

# Table of Contents

[Table of Contents 4](#_Toc482791850)

[List of Figures 6](#_Toc482791851)

[Definitions, Acronyms, and Abbreviations 7](#_Toc482791852)

[A. Introduction 8](#_Toc482791853)

[1. Project Information 8](#_Toc482791854)

[2. Introduction 8](#_Toc482791855)

[3. Current Situation 8](#_Toc482791856)

[4. Problem Definition 9](#_Toc482791857)

[5. Proposed Solution 9](#_Toc482791858)

[5.1 Feature functions 10](#_Toc482791859)

[5.2 Benefit and drawback 11](#_Toc482791860)

[6. Functional Requirements 11](#_Toc482791861)

[7. Role and Responsibility 12](#_Toc482791862)

[B. Software Project Management Plan 12](#_Toc482791863)

[1. Problem Definition 12](#_Toc482791864)

[1.1 Barcode order eateries 12](#_Toc482791865)

[1.2 Problem Abstract 12](#_Toc482791866)

[1.3 Project Overview 12](#_Toc482791867)

[2. Project organization 15](#_Toc482791868)

[2.1 Software Process Model 15](#_Toc482791869)

[2.2 Roles and responsibilities 16](#_Toc482791870)

[2.3 Tools and Techniques 17](#_Toc482791871)

[3. Project Management Plan 17](#_Toc482791872)

[3.1 Product Backlog 17](#_Toc482791873)

[3.2 Sprint Backlog 20](#_Toc482791874)

[3.3 All Meeting Minutes 28](#_Toc482791875)

[4. Coding Convention 28](#_Toc482791876)

[C. Report No. 3 Software Requirement Specification 29](#_Toc482791877)

[1. User Requirement Specification 29](#_Toc482791878)

[1.1 Guest Requirement 29](#_Toc482791879)

**List of Tables**

[Table 1: Roles and Responsibilities 12](#_Toc483859263)

[Table 2: Hardware Requirement for Server 15](#_Toc483859264)

[Table 3: Hardware Requirement for Mobile 15](#_Toc483859265)

[Table 4: Roles and Responsibilities Details 17](#_Toc483859266)

[Table 5: Product Backlog 20](#_Toc483859267)

[Table 6: Sprint 1 Development 22](#_Toc483859268)

[Table 7: Sprint 2 Development 23](#_Toc483859269)

[Table 8: Sprint 3 Development 25](#_Toc483859270)

[Table 9: Sprint 4 Development 26](#_Toc483859271)

[Table 10:Sprint 5 Development 27](#_Toc483859272)

[Table 11: Sprint 6 Development 27](#_Toc483859273)

[Table 12: USE CASE -UC\_BOE01 - <Admin> Add account **Error! Bookmark not defined.**](#_Toc483859274)

[Table 13: USE CASE -UC\_BOE02 - <Admin> Update account 35](#_Toc483859275)

[Table 14: USE CASE -UC\_BOE03 - <Admin> Delete account 37](#_Toc483859276)

[Table 15:USE CASE -UC\_BOE04 - <Manager> Make statistic 39](#_Toc483859277)

[Table 16: USE CASE -UC\_BOE05 - <Manager> Add dishes 47](#_Toc483859278)

[Table 17: USE CASE -UC\_BOE06 - <Manager> Update dish 50](#_Toc483859279)

[Table 18: USE CASE -UC\_BOE07 - <Manager> Delete dishes 51](#_Toc483859280)

[Table 19: USE CASE -UC\_BOE11 - <Guest> Register 54](#_Toc483859281)

[Table 20: USE CASE - UC\_BOE10 - <Guest> Login 56](#_Toc483859282)

[Table 21: USE CASE -UC\_BOE11 - <Guest> Make order 58](#_Toc483859283)

[Table 22: USE CASE -UC\_BOE07 - <Chef > Approve cart **Error! Bookmark not defined.**](#_Toc483859284)

[Table 23: USE CASE -UC\_BOE08 - <Chef > Receive notification **Error! Bookmark not defined.**](#_Toc483859285)

[Table 24: USE CASE -UC\_BOE09 - <Waiter> Receive notification 69](#_Toc483859286)

# List of Figures

[Figure 1: Scrum model 16](#_Toc483473305)

[Figure 2: System Overview Use Case 31](#_Toc483473306)

[Figure 3: <Admin> Overview Use Case 31](#_Toc483473307)

[Figure 4: <Admin> Add account 31](#_Toc483473308)

[Figure 5: <Admin> Update account 34](#_Toc483473309)

[Figure 6: <Admin> Delete account 36](#_Toc483473310)

[Figure 7: <Manager> Overview Use Case 38](#_Toc483473311)

[Figure 8: <Manager> Add table **Error! Bookmark not defined.**](#_Toc483473312)

[Figure 9: <Manger> Delete table **Error! Bookmark not defined.**](#_Toc483473313)

[Figure 10: <Chef> Overview Use Case 59](#_Toc483473314)

[Figure 11: <Chef> Approve order **Error! Bookmark not defined.**](#_Toc483473315)

[Figure 12:<Chef> Receive notification **Error! Bookmark not defined.**](#_Toc483473316)

# Definitions, Acronyms, and Abbreviations

|  |  |
| --- | --- |
| **Name** | **Definition** |
| BOE | Barcode Order Eateries |

# A. Introduction

## 1. Project Information

* Project name: **Barcode order eateries**
* Project Code: **BOE**
* Product Type: **Web application and Mobile application**
* Start Date: **08/05/2017**
* End Date: **13/08/2017**

## 2. Introduction

Nowaday, the using labor of employers is not very effective in many restaurant’s process. When a lot of diners use service of restaurant, the number of waiters and chefts are not enought to do all job. Many diners must wait long time to be serve. Diners might not satisfied with this situation at restaurants. To solve this problem, many restaurants increase many employees. But, when a few diners is in restaurant, Many employees is free time.

We decide to provide a solution that we will develop BOE system in order to solve the problem. BOE system will help diners make order so easy, effecctive and fast. Diners will make order without depending on waiter. Food will be cooked faster and more exactly. The use of human resource will more effective.

## 3. Current Situation

Restaurant’s service is usually interested and compared by diners. If the quality of foods are same, they decide going to restaurant has service is better than. There are two usual models:

* Traditional restaurant:
* Waiter shows diners menu of restaurant.
* Diner makes order and waiter write down it.
* Order is taken to chef.
* Chef complete that order.
* Waiter take food and drink to table of diner.
* Diner make a signal about payment.
* Cashier make bill and waiter take it to diner.
* Diner pay.
* Alipay/Wechat in China using QR code idea:

Alipay/Wechat only focus on payment method. Business of Alipay on restaurants is same with traditional restaurant. But in model Alipay, diner using smartphone to scan QR code, then application define information about payment of that restaurant then pay money online.

## 4. Problem Definition

* In traditional model, we identified:
  + Strength:
    - Diners feel more comfortable and statisfy when waiter directly serve.
  + Weakness:
    - Each table need a waiter taking the order.
    - When number of table which diners are seating is more than number of waiter of restaurant diner must wait to order food. But when restaurant has a few diners, many waiters do not have task.
    - Chef interacts with diner about foods and drinks through waiter.
    - Waiter might write down order incorrect.
    - In traditional model, restaurant need one or more cashiers to do payment task.
* In Alipay model, we identified:
  + Strength:
    - Payment process is done easy and fast.
  + Weakness:
    - Only focus on payment process.

## 5. Proposed Solution

BOE system is devloped to help restaurants work effictively:

* Number of employees of restaurant is reduced.
* Diners can interact directly with chefs of restaurant about their order.
* Diners do some processes of restaurant such as making order, payment but using time less than.
* Making order process do not include waiter.
* Orders are always correct.
* Restaurants can not need cashier for payment process.

BOE system utilities to restaurants and their customer:

* Manager can offer strategies by study statisticals which are made by system.
* Manager save time in leading employee process.
* Diners have food and drink faster.
* Payment is done by online. This process is completed after a few seconds the order is accepted by chef.
* In both model traditional restaurant and restaurant use Alipay, diners have to wait waiter to make order. In traditional restaurant, when diners do payment process, they must wait cashier make a bill then waiter take it to their. If there are the returned money, diners wait waiter come back.

BOE system includes one web application and three mobile applications with following functions:

### Feature functions

* Manager:
  + Get statistics: manager can get statisctic about business of restaurant.
  + Manage table: manager able to manage all tables in restaurant.
  + Manage discount: manager can create many discount promotion campaigns of restaurant.
  + Manage membership: Manager can set many goodwill for diners of restaurant.
* Diners :
  + Make order: diners can order dishes which is in menu of restaurant.
* Chef :
  + Reject dish: chef rejects a dish in order of diner if this dish can not is cooked.
* Waiter:
  + Receive notification: Waiters are notified that they must delivery a dish to diners after this dish was cooked.
* Admin:
  + Creat account: admin able to create accounts.
  + Grant permissions: admin can grant permission.
* Timer schedule:
  + Do payment process.
  + Sent notification to user.
  + Do repayment process.
  + Check dish in cart of diner is available.
  + Suggest dishes follow favorite of diner.

### Benefit and drawback

* Benefit
  + Dinner can interact with chef directly
  + Using labor is effective.
  + Diners can make order and pay bill quickly.
* Drawback
  + Diners must have smartphone had installed application of restaurant to use service.
  + Diners must have account of mobile banking for payment.
  + In some case, dish in order of diners can not is cooked, it will be rejected.

## 6. Functional Requirements

* Diners component:
  + Make Order: Order dishes that diners want.
  + Receive notification: diners will be notified whenever dish has been rejected or dish in their order is not available.
* Chef component:
  + Reject dish: chef can reject dish if it is can not be cooked.
  + Change status of dish to done: Dish is cooked, chef must change it’s status to “done”.
  + Check food ingredients: Chef check food ingredients of restaurant and change their status, available or not available.
* Managers component:
  + Tracking statistic: manager able to tracking statistic of restaurant.
  + Manage dishes: manager can manage dishes.
* Waiter component:
  + Receive notification: Waiter able to receive notify when dishes are cooked and need to delivery to diners at the table.
* System component:
  + Send notification: System will send notification to diners when dishs have been rejected, notify to chefs many dish must be cooked, notify waiter whenever dishes needed to delivery.
  + Suggest menu: base on diner’s profile system will suggest.
  + Create statistic number.
  + Do payment: send information to third party.
  + Checking dish available: check dish in cart of diners is able to cooked.

## 7. Role and Responsibility

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *No* | *Full Name* | *Role* | *Position* | *Contact* |
| *1* | *Kiều Trọng Khánh* | *Product owner* | *Supervisor* | *khanhkt@fpt.edu.vn* |
| *2* | *Phạm Đăng Nam* | *Scrum master* | *Leader* | *nampdse61539@fpt.edu.vn* |
| *3* | *Nguyễn Khánh Linh* | *Developer* | *Member* | *linhnkse61692@fpt.edu.vn* |
| *4* | *Nguyễn Đức Hoàng* | *Developer* | *Member* | *hoangndse61505@fpt.edu.vn* |
| *5* | *Lưu Đức Phong* | *Developer* | *Member* | *phongldse61605@fpt.edu.vn* |

Table 1: Roles and Responsibilities

# B. Software Project Management Plan

## 1. Problem Definition

### 1.1 Barcode order eateries

* Official name: Barcode order eateries
* Vietnamese name: Đặt món ăn thông qua barcode thông minh.
* Abbreviation: BOE

### 1.2 Problem Abstract

* Our team don’t have enough experience in UX, UI.
* This is the first time we develop real project.
* We need more time to learn mobile programming.

### 1.3 Project Overview

#### 1.3.1 Current diagnostic diseases

Below are the problems encountered in this project:

* **Lack of UX/UI experience and mobile application experience**.
* ***Lack of experience in developing mobile programing****: Mobile programing is new fields for our team. We must spend time studing and training.*
* ***Lack of knowledge about QR code****: our team have not ever developed system use QR code.*
* ***Design system****: We have problem with design system such as database, network, security....*

#### 1.3.2 The Proposed System

* Diners will use smartphone to order foods and drinks instead of using traditional menu. In traditional restaurant, they use paper menu to show foods and dinks of restaurant. After a long time they are used, menu are dirty, old or frayed. Restaurant need to repalce it with new one. When restaurant update menu, they have to use new menu to replace old menu. So a small QR code is a smart choice to improve disadvantages of traditional menu. Price of QR code is cheaper than paper menu. Resteaurant will create QR code by theirself. Restaurant will update menu but they do not replace QR code. Diners use smartphone scan QR code and then menu displayed on their smartphone screen. Diners orders food and drink without using paper menu.
* Approaching for team’s issue:
* Each member training for another one about mobile programing.
* We discuss and identify the weaknesses and strengths of many current system’s UX/UI on both web application and mobile application: foody.vn, diadiemanuong.com...
* Learn characteristics of QR code.

##### 1.3.2.1 Web Application

Web application is where menu and dishes are managed. In addition, manager makes statistics about restaurant.

* **For managers:** 
  + Making statistics.
  + Making new menu.
  + Manage dishes of restaurant.
  + Manage QR code.

##### 1.3.2.2 Mobile Application

* **For diners:** 
  + Order foods.
  + Do payment process.
* **For chefs:** 
  + Reject dish which is can not be cooked.
  + Change status dish to done.
* **For waiter:** 
  + Receive notification about dish was cooked.

##### 1.3.2.3 API Application

#### 1.3.3 Boundaries of the System

The system should do:

* Allow diners scan QR code on table.
* Allow diners order dishes.
* Show menu on smartphone of diners after scaning QR code succesfully.
* Diners make order.
* Allow chefs reject unavailable dish.
* Chef update status of food ingredients.
* Notify diners if dish is rejected by chefs.
* Diners do payment after order is available.
* Allow managers make statistics.
* Allow manager manage dishes.
* Allow manage manage menu.
* Notify waiters about their task.
* Manage chefs, tables, waiters.

The system should not do:

* Do not manage diners.
* Do not manage payment.

#### 1.3.4 Future Plans

BOE system currently support only android device. We continuous improve and extend scop of system:

* Mobile application run on iOS, BerriOS
* Diners willdescribe the dishes will be cooked follow the way they want*.*
* BOE will extend, diners can use one mobile application in many restaurants apply BOE system.

#### 1.3.5 Development Environment

##### 1.3.5.1 Hardware requirements

***For server***

|  |  |  |
| --- | --- | --- |
| ***Windows*** | ***Minimum Requirements*** | ***Recommended*** |
| ***Internet Connection*** | *Cable, Wi-Fi (4 Mbps)* | *Cable, Wi-Fi (8 Mbps)* |
| ***Operating System*** | *Window Server 2008* | *Window Server 2008* |
| ***Computer Processor*** | *4 core, 2MB cache, 2.5GHz* | *4 core, 3MB cache, 2.6GHz* |
| ***Computer Memory*** | *4GB RAM* | *8GB or more* |

Table 2: Hardware Requirement for Server

***For mobile***

|  |  |  |
| --- | --- | --- |
| ***Windows*** | ***Minimum Requirements*** | ***Recommended*** |
| ***Internet Connection*** | *Wi-Fi* | *Wi-Fi* |
| ***Operating System*** | Android 5 | Android 5 |
| ***Mobile Processor*** | 4 core, 2GHz | 8 core, 2GHz |
| ***Mobile Memory*** | *1GB RAM* | *2GB or more* |

Table 3: Hardware Requirement for Mobile

##### 1.3.5.2 Software requirements

|  |  |  |
| --- | --- | --- |
| Software | Name / Version | Description |
| Modeling tool | Star UML 5.0 | Used to implement website and web service |
| IDE | Rubymine | Programming tools |
| DBMS | SQLite | Used to create & manage the database for system |
| Source control | Github | Used for source control |
| Web browser | Chrome 42 or above | Testing browser |

## 2. Project organization

### 2.1 Software Process Model

This project is developed under Srum model. We choose this model because of our situation and these following reasons:

* Prototypes are get feedback frequently.
* Srum adaptive approach is very suitable for lack of clarity in requirement.
* The project is tested very frequently, minimizing the risk of any failures in future.

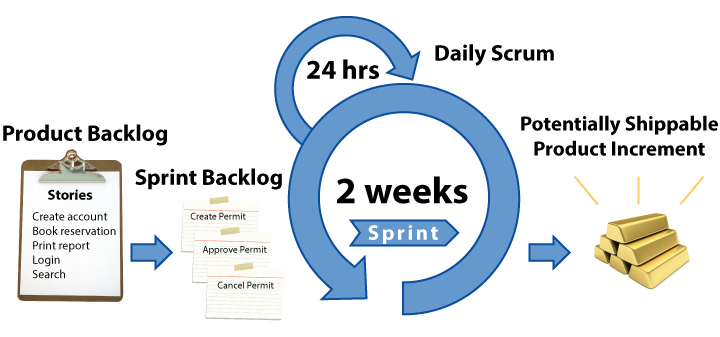


Figure 1: Scrum model

(<http://www.agilenutshell.com/scrum>)

### 2.2 Roles and responsibilities

|  |  |  |  |
| --- | --- | --- | --- |
| ***No*** | ***Full name*** | ***Role in Group*** | ***Responsibilities*** |
| ***1*** | *Kiều Trọng Khánh* | *Product owner* | * *Specify user requirement* * *Control the development process* * *Give out technique and business analysis support* |
| ***2*** | *Phạm Đăng Nam* | *Scrum master, BA, DEV, Tester* | * *Managing process* * *Designing database* * *Clarifying requirements* * *Prepare documents* * *GUI Design* * *Create test plan* * *Coding* * *Testing* |
| ***3*** | *Lưu Đức Phong* | *Scrum member, Tester* | * *Coding* * *Testing* * *Designing database* |
| ***4*** | *Nguyễn Khánh Linh* | *Scrum member, Tester* | * *Coding* * *Testing* * *Designing database* |
| ***5*** | *Nguyễn Đức Hoàng* | *Scrum member, Tester* | * *Coding* * *Testing* * *Designing database* |

Table 4: Roles and Responsibilities Details

### 2.3 Tools and Techniques

|  |  |  |
| --- | --- | --- |
|  | **Tools** | **Techniques** |
| Front-end | WebStorm | * HTML5 * CSS3 * ES6 * NodeJS * ReactJS |
| Back-end | RubyMine | * Rails * RESTFul * JSON * Ruby * PostgresSQL |
| Mobile Application | Android studio | * Java 8 * Android SDK |
| Database management system | SQLite | N/A |
|  | | |

## 3. Project Management Plan

### 3.1 Product Backlog

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Story ID** | **Features** | **Task ID** | **Task description** | **Sprint** |
| 1 | Create broduct backlog | 1 | Create product backlog | 1 |
| 2 | Create introduction document | 2.1 | Write introduction document | 1 |
| 2.2 | Review introduction document | 1 |
| 3 | Create the tasksheet | 3.1 | Create the tasksheet | 1 |
| 3.2 | Review the tasksheet | 1 |
| 4 | Studying QRcode | 4.1 | How to generate QRcode | 1 |
| 4.2 | How to scan QRcode | 1 |
| 5 | Create project management plan | 5.1 | Problem definition | 1 |
| 5.2 | Project organization | 1 |
| 5.3 | Project management plan | 1 |
| 5.4 | Coding convention | 1 |
| 5.5 | Review document | 1 |
| 6 | Build system structure | 6.1 | Backend structure | 1 |
| 6.2 | Web structure | 1 |
| 6.3 | Mobile structure | 1 |
| 7 | Create mobile prototype | 7 | Create Mobile prototype | 1 |
| 8 | Create software requirement specification | 8.1 | User Requirement Specification | 2 |
| 8.2 | System Requirement Specification – External interface requirement | 2 |
| 8.3 | System Requirement Specification – System overview Use Case | 2 |
| 8.4 | System Requirement Specification – List of Use Case | 2 |
| 8.5 | Software System Attribute | 2 |
| 8.6 | Conceptual Diagram | 2 |
| 9 | Create Software Description | 9.1 | Design Overview | 3 |
| 9.2 | System Architectural Design | 3 |
| 9.3 | Component Diagram | 3 |
| 9.4 | Detailed Description of Components | 3 |
| 9.5 | Sequence Diagram | 3 |
| 9.6 | User Interface Diagram | 3 |
| 9.7 | Database Design | 3 |
| 9.8 | Entity Diagram | 3 |
| 9.9 | Class Diagram | 3 |
| 10 | Implementation | 10.1 | Guest Sign\_Up | 2 |
| 10.2 | Guest Sign\_In | 2 |
| 10.3 | Authenticated User\_Sign out | 2 |
| 10.4 | Diner\_Edit profile | 3 |
| 10.5 | Timescheduler\_Send menu | 3 |
| 10.6 | Timescheduler\_Suggest dish | 3 |
| 10.7 | User\_Make Order | 3 |
| 10.8 | Timescheduler\_Notify | 4 |
| 10.9 | Chef\_Receive Notify | 4 |
| 10.10 | Diner\_Receive Notify | 4 |
| 10.11 | Waiter\_Receive Notify | 4 |
| 10.12 | Chef\_Reject Dish | 4 |
| 10.13 | Diner\_Pay | 4 |
| 10.14 | Admin\_Create account | 4 |
| 10.15 | Admin\_Update account | 4 |
| 10.16 | Admin\_Delete account | 4 |
| 10.17 | Manager\_Add table | 4 |
| 10.18 | Manager\_Update table | 4 |
| 10.19 | Manager\_Delete table | 4 |
| 10.20 | Chef\_Update status of food ingredients | 4 |
| 10.21 | Timescheduler\_Repayment | 4 |
| 10.22 | Manager\_Make statistic | 4 |
| 11 | Create Software Test Documentation | 11.1 | Test Plan | 5 |
| 11.2 | Test Cases | 5 |
| 12 | Quality Assurance | 12.1 | Quality Assurance for Backend | 5 |
| 12.2 | Quality Assurance for Web | 5 |
| 12.3 | Quality Assurance for Mobile | 5 |
| 13 | Create Software User’s Manual | 13.1 | Installation Guide | 6 |
| 13.2 | User’s Guide | 6 |

Table 5: Product Backlog

### 3.2 Sprint Backlog

#### 3.2.1 - Sprint 1 (09/05/2017 – 22/05/2017) Project Initiation

#### 3.2.1.1 Goal

- Sprint 1 must complete following tasks:

+ Create Product Backlog

+ Create Introduction document

* Write Introduction document
* Review Introduction document

+ Create the tasksheet

* Write the tasksheet.
* Review the tasksheet.

+ Studying QRCode

* How to generate QRCode
* How to scan QRCode

+ Write Project Management Plan

* Problem definition
* Project organization
* Project management plan
* Coding convention
* Review document

+ Build System Structure

* Backend Structure
* Web Structure
* Mobile Structure

+ Create Mobile prototype

+ Milestone 1, Meeting and Review

#### 3.2.1.2 Development

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Task** | **Implement** | **Reviewer** |
| 1 | Create Product Backlog |  |  |
| 2 | Write introduction document |  |  |
| 3 | Write the task sheet |  |  |
| 4 | How to generate QRCode |  |  |
| 5 | How to scan QRCode |  |  |
| 6 | Problem definition |  |  |
| 7 | Project organization |  |  |
| 8 | Project management plan |  |  |
| 9 | Coding convention |  |  |
| 10 | Backend Structure |  |  |
| 11 | Web Structure |  |  |
| 12 | Mobile Structure |  |  |
| 13 | Create Mobile prototype |  |  |

Table 6: Sprint 1 Development

#### 3.2.2 – Sprint 2 (23/05/2017 – 05/06/2017)

##### 3.2.2.1 Goal

- Sprint 2 must complete following tasks

* Create Software Requirement Specification
  + User Requirement Specification – Gest Requirement
  + User Requirement Specification – Member Requirement
  + System Requirement Specification – External Interface Requirement
  + System Requirement Specification – System overview Use Case
  + System Requirement Specification – List of Use Case
  + Software System Attribute
  + Conceptual Diagram
* Implementation
* Unanthorized User\_Register
* Unauthorized User\_Log In
* Diner\_Sign out
* Diner\_Edit Profile
* Milestone 2, Meeting and Review

##### 3.2.2.2 Development

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Task** | **Implement** | **Reviewer** |
| 1 | User Requirement Specification |  |  |
| 2 | System Requirement Specification – External Interface Requirement |  |  |
| 3 | System Requirement Specification – System overview |  |  |
| 4 | System Requirement Specification – List of Use Case |  |  |
| 5 | Software System Attribute |  |  |
| 6 | Conceptual Diagram |  |  |
| 7 | Unanthorized User\_Register |  |  |
| 8 | Unauthorized User\_Log In |  |  |
| 9 | Diner\_Sign out |  |  |
| 10 | Diner\_Edit Profile |  |  |

Table 7: Sprint 2 Development

#### 3.2.3 – Sprint 3 (06/06/2017 – 12/06/2017; 20/06/2017 – 26/06/2017)

##### 3.2.3.1 Goal

- Sprint 3 must complete following tasks

* Create Software Dexcription
  + - Design Overview
    - System Architectural Design
    - Component Diagram
    - Detailed Description of Components
    - Sequence Diagram
    - User Interface Diagram
    - Database Design
    - Entity Diagram
    - Class Diagram
* Implementation
  + - Time scheduler\_Send menu
    - Timescheduler\_Suggest dish
    - Diner\_Make Order
    - Chef\_Reject Dish
    - Chef\_Update status of food ingredients.
    - Chef\_Receive Notification
    - Time scheduler\_Send Notification
    - Diner\_Receive Notification
    - Waiter\_Receive Notification
* Milestone 3, Meeting and Review

3.2.3.2 Development

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Task** | **Implement** | **Reviewer** |
| 1 | Design Overview |  |  |
| 2 | System Architectural Design |  |  |
| 3 | Component Diagram |  |  |
| 4 | Detailed Description of Components |  |  |
| 5 | Sequence Diagram |  |  |
| 6 | User Interface Diagram |  |  |
| 7 | Database Design |  |  |
| 8 | Entity Diagram |  |  |
| 9 | Class Diagram |  |  |
| 10 | Timescheduler\_Send menu |  |  |
| 11 | Timescheduler\_Suggest dish |  |  |
| 12 | Diner\_Make Order |  |  |
| 13 | Chef\_Reject Dish |  |  |
| 14 | Chef\_Update status of food ingredients |  |  |
| 15 | Timescheduler\_Send Notification |  |  |
| 16 | Chef\_Receive Notification |  |  |
| 17 | Diner\_Receive Notification |  |  |
| 18 | Waiter\_Receive Notification |  |  |

Table 8: Sprint 3 Development

#### 3.2.4 – Sprint 4 (27/06/2017 – 10/07/2017)

##### 3.2.4.1 Goal

- Sprint 4 must complete following tasks

+ Implementation

* + - Admin\_Create account
    - Admin\_Update account
    - Admin\_Delete account
    - Manager\_Add QR code
    - Manager\_Print QR code
    - Diner\_Payment
    - Manager\_Make statistic
    - Manager\_Add Dish
    - Manager\_Delete Dish
    - Manager\_Update Dish
    - Manager\_Add Discount
    - Manager\_Update Discount
    - Manager\_Delete Discount
    - Manager\_Add membership
    - Manager\_Delete membership
    - Manager\_Update membership
    - Time scheduler\_Repayment
    - Manager\_Recieve Notification

+ Milestone 4, Meeting and Review

3.2.4.2 Development

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Task** | **Implement** | **Reviewer** |
| 1 | Admin\_Create account |  |  |
| 2 | Admin\_Update account |  |  |
| 3 | Admin\_Delete account |  |  |
| 4 | Manager\_Add QR code |  |  |
| 7 | Diner\_Payment |  |  |
| 8 | Manager\_Make statistic |  |  |
| 9 | Manager\_Add Dish |  |  |
| 10 | Manager\_Delete Dish |  |  |
| 11 | Manager\_Update Dish |  |  |
| 12 | Manager\_Add Discount |  |  |
| 13 | Manager\_Update Discount |  |  |
| 14 | Manager\_Update Discount |  |  |
| 15 | Manager\_Add membership |  |  |
| 16 | Manager\_Delete membership |  |  |
| 17 | Manager\_Update membership |  |  |
| 18 | Time scheduler\_Repayment |  |  |
| 19 | Manager\_Recieve Notification |  |  |

Table 9: Sprint 4 Development

#### 3.2.5 – Sprint 5 (12/07/2017 – 24/07/2017)

##### 3.2.5.1 Goal

- Sprint 5 must complete following tasks

+ Create Software Test Documentation

* Test plan
* Test cases

+ Quality Assurance

* + - Quality Assurance for Backend
    - Quality Assurance for Web
    - Quality Assurance for Mobile

+ Milestone 5, Meeting and Review

#### 3.2.5.2 Development

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Task** | **Implement** | **Reviewer** |
| 1 | Test plan |  |  |
| 2 | Test cases |  |  |
| 3 | Quality Assurance for Backend |  |  |
| 4 | Quality Assurance for Web |  |  |
| 5 | Quality Assurance for Mobile |  |  |

Table 10:Sprint 5 Development

#### 3.2.6 – Sprint 6 (25/07/2017 – 07/08/2017)

##### 3.2.6.1 Goal

- Sprint 6 must complete following tasks:

+ Create Software User’s Manual

* + - Installation Guide
    - User’s Guide

+ Milestone 6, Meeting and Review

#### 3.2.6.2 Development

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Task** | **Implement** | **Reviewer** |
| 1 | Installation Guide |  |  |
| 2 | User’s Guide |  |  |

Table 11: Sprint 6 Development

### 3.3 All Meeting Minutes

All meeting documents could be found [here](https://drive.google.com/drive/u/2/folders/0ByY0NXCJg23lR1dBazJXdWpJd2c)

## 4. Coding Convention

* **Ruby:** use to develop backend and API
  + Naming convention:
    - Name is identified in English.
    - Use snake-case for symbols, method and variables.
    - Use camel-case for classes and modules.
    - Use snake-case for naming file and directories.
  + Indentation:
    - Use to spaces per indentation level.
    - Do not  use “**;**” to separate statements and expressions.
    - Avoid single-line methods.
  + Class:
    - Prefer modules to classes with only class methods.
    - Split multiple mixins into separate statements.
    - Use **attr** family of function to define trivial accessors and mutators.
  + Comment:
    - Write comments in English.
    - Use one space between the leading # character of comment and text comment.
* **Java(Android):** Using for mobile development
  + Naming convention:
    - Use camel-case style for variables function name (lowercase for first letter).
    - Use camel-case style for interface and class name (Uppercase for first letter).
    - Constants should be all uppercase with words.
  + Indentation:
    - Four spaces should be used as the unit of indentation.
    - Use eight spaces to continue indentation.
  + Class:
    - Use **this** to refer current class instance.
* **JavaScript:** Using for frontend website
  + Naming convention:
    - Use camel-case style for variables and functions name (Lowercase for first letter).
    - Use camel-case for classes (Uppercase for first letter).
  + Indentation:
    - Use two spaces per indentation level.
    - Avoid single-line method.
  + Class:
    - Use constructor function for initialize a class.
    - Use export and import to reuse modules.
    - Use this to refer current class instance.
* References:
* [4] Java: <http://www.oracle.com/technetwork/java/codeconvtoc-136057.html>
* [5] Ruby: <https://github.com/bbatsov/ruby-style-guide>

# C. Software Requirement Specification

## 1. User Requirement Specification

### 1.1 Guest Requirement

User is a person who does not have access to the system. User can use some function in the system. These are some function user can use:

* + - * + Login
        + Register
        + Make order

### 1.2 Diner Requirement

Make order

Edit profile

Receive notification

### 1.3 Chef Requirement

* Receive notification
* Reject dish.
* Update status of food ingredients.

### 1.4 Waiter Requirement

* Receive notification

### 1.5 Manager Requirement

* Make statistic
* Manage table:
  + Add table

### Admin Requirement

* Manage account:
  + Create account
  + Update account
  + View account

### TimeScheduler Requirement

* Send menu:
  + Suggest dish
* Notify
* Repayment

### Online Banking Requirement

### Authenticated Requirement

* Log out

## 2. System Requirement Specification

### 2.1 External Interface Requirement

#### 2.1.1 User Interface

* The user interface uses Vietnamese as main language for all role on both Mobile and Web application.
* The user interface displays best on resolution at 1024x720 or screen size at 5.5 inch.

##### 2.1.2 Hardware Interface

N/A

##### 2.1.3 Software Interface

N/A

##### 2.1.4 Communication Protocol

* Use HTTP protocol 1.1 for communication between the web browser and the web server.
* Use HTTP protocol 1.1 for communication between the mobile application and the web service.

### 2.2 System Overview Use Case

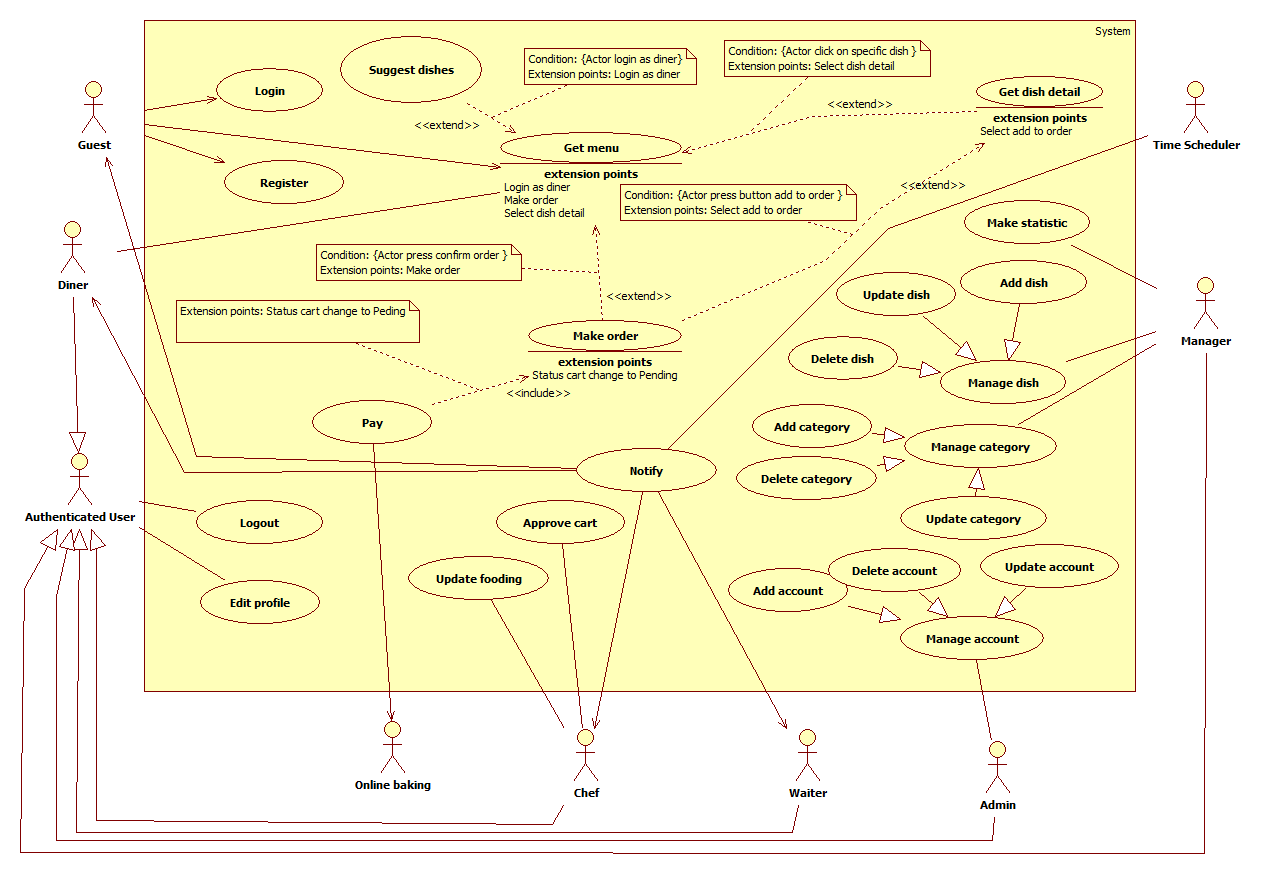


Figure 2: System Overview Use Case

### 2.3 List of Use Case

#### <Admin> Overview Use Case

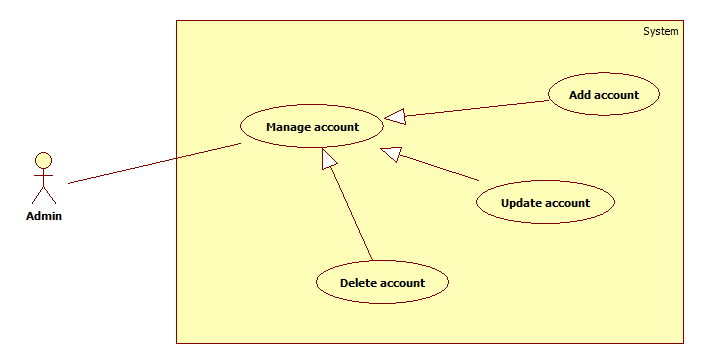


Figure 3: <Admin> Overview Use Case

##### <Admin> Add account (UC\_BOE01)

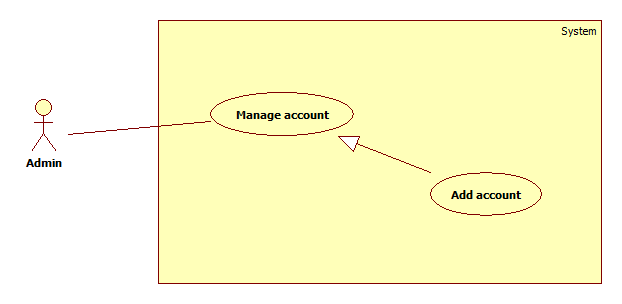
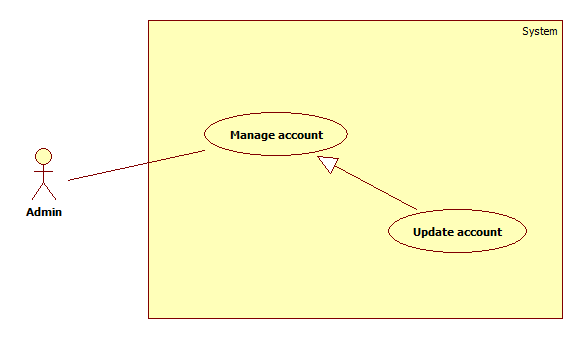


Figure 4: <Admin> Add account

|  |  |  |  |
| --- | --- | --- | --- |
| **USE CASE – UC\_BOE01** | | | |
| **Use Case No.** | UC\_BOE01 | **Use Case Version** | 2.0 |
| **Use Case Name** | Add account | | |
| **Author** | NamPD | | |
| **Date** | 22/05/2017 | **Priority** | High |
| **Actor:**   * Admin   **Summary:**   * This use case allows actor add an account to the system.   **Goal:**   * A new account has been added to system with username, password and role are same as provided by admin. New account have status is active.   **Triggers:**   * Actor sends creating account command.   **Preconditions:**   * Actor must login as admin role. * Username not existed on system before.   **Post Conditions:**   * **Success:** New account is added to the system with its role. * **Fail:** Show error message.   **Main Success Scenario:**   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 1 | Actor goes to adding account view. | System requires information for a account:   * **Username**: free text input, required, length 6–25 characters. * **Email**: free text input, required, length 6–50 characters, email regex. * **~~Role~~**~~: drop box, required.~~ | | 2 | Actor inputs all information of new user. |  | | 3 |  | System validate inputted information on page.  [Exception 1,2,4,5~~,7~~] | | 3 | Actor sends command to add new account.  [Alternative 1] |  | | 4 |  | System validates information on server.  [Exception 3] | | 5 |  | New account is created and show message account has been created successfully. |   **Alternative Scenario:**   |  |  |  | | --- | --- | --- | | Step. | Actor Action | System Response | | 1 | Actor clear all information. | Keep adding account view. |   **Exceptions:**   |  |  |  | | --- | --- | --- | | No. | Cause | System Response | | 1 | Missing “Username” field. | System shows error message: “Please input username”. | | 2 | Invalid username format. | System shows error message: “Username is not valid. Please choose another one. Username is contain 6 – 25 characters, not include !@#$%^&\*()”. | | 3 | Username existed. | System shows error message: “Username existed is system. Please choose another one.”. | | 4 | Missing “Email” feild. | System shows error message: “Please input email”. | | 5 | Email is not valid | System shows error message: “Email is not valid, the valid email in form abc@mail.com”. | | 6 | Email has been used | System shows error message:”Email has been used by another account.”. | | ~~7~~ | ~~Missing “Role” feild.~~ | ~~System shows error message: “Vui lòng chọn role của tài khoản”.~~ |   **Relationships:**  **N/A**  **Business Rules:**   * Roles of account: Manager~~, chef or waiter.~~ * Only one role each account. * Username is unique. * Email is not duplicated with other account’s email has been in system. * Status of new account is “Active”. * ~~A mail is created and sent to email address of user who own new account.~~ | | | |

Table 12: USE CASE -UC\_BOE01 - <Admin> Add account

##### <Admin> Update account (UC\_BOE02)

Figure 5: <Admin> Update account

|  |  |  |  |
| --- | --- | --- | --- |
| **USE CASE – UC\_BOE02** | | | |
| **Use Case No.** | UC\_BOE02 | **Use Case Version** | 2.0 |
| **Use Case Name** | Update account | | |
| **Author** | NamPD | | |
| **Date** | 22/05/2017 | **Priority** | Normal |
| **Actor:**   * Admin   **Summary:**   * User is changed with new role in system.   **Goal:**   * Current role of account be replaced by a new role.   **Triggers:**   * Actor send updating command.   **Preconditions:**   * Actor must login as admin role. * Account must be existed on system.   **Post Conditions:**   * **Success:** Account updated with new role. * **Fail:** System show error message “Fail to update account.”   **Main Success Scenario:**   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 1 | Actor is at Update account view. | System requires information from admin:   * **Role**: drop box, required. | | 2 | Actor choose role for that account. |  | | 3 | Actor sends command to save. |  | | 4 |  | System show confirm dialog. | | 5 | Actor click “Đồng ý”.  [Alternative 1] |  | | 6 |  | Role is changed and saved. | | 7 |  | Keep updating account view and display new role of that account. |   **Alternative Scenario:**   |  |  |  | | --- | --- | --- | | Step. | Actor Action | System Response | | 1 | Actor click “Hủy bỏ”. | Hide dialog and come back Updating account view. |   **Exceptions:**  N/A  **Relationships:**  N/A  **Business Rules:**   * Account will be updated must have status “Active”. * A account have only one role. * Roles of account: Manager, ~~chef or waiter.~~ | | | |
| Table 13: USE CASE -UC\_BOE02 - <Admin> Update account | | | |

##### <Admin> Delete account (UC\_BOE03)

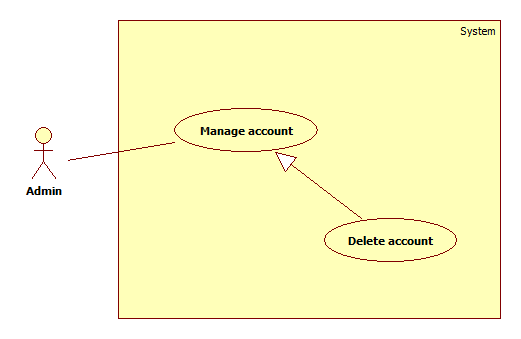


Figure 6: <Admin> Delete account

|  |  |  |  |
| --- | --- | --- | --- |
| **USE CASE – UC\_BOE03** | | | |
| **Use Case No.** | UC\_BOE03 | **Use Case Version** | 2.0 |
| **Use Case Name** | Delete account | | |
| **Author** | NamPD | | |
| **Date** | 22/05/2017 | **Priority** | Normal |
| **Actor:**   * Admin   **Summary:**   * This use case allows actor delete an existed account.   **Goal:**   * The account’s status will change to “Deactive”, can not use to login to system.   **Triggers:**   * Actor sent deleting account command.   **Preconditions:**   * Actor must login as admin role. * Account must existed on system.   **Post Conditions:**   * **Success:** Deleted account’s status set to false, can not login after. * **Fail:** System shows error message “Fail to delete account”.   **Main Success Scenario:**   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 1 |  | System display confirm dialog. | | 2 | Actor click “Đồng ý”.  [Alternative 1] |  | | 3 |  | System validates information.  [Exception 1] | | 4 |  | System set status of account to “Deactive”. |   **Alternative Scenario:**   |  |  |  | | --- | --- | --- | | Step. | Actor Action | System Response | | 1 | Actor send cancel command | Hide dialog and come back list account view. |   **Exceptions:**   |  |  |  | | --- | --- | --- | | No. | Cause | System Response | | 1 | Admin’s account is undeletable | System shows error message: “Can not delete admin’s account”. |   **Relationships:**   * N/A   **Business Rules:**   * Admin account with id is 1 can not be deleted. * Account has been choosen to be delete must have status “Active”**.** | | | |

Table 14: USE CASE -UC\_BOE03 - <Admin> Delete account

#### <Manager> Overview Use Case

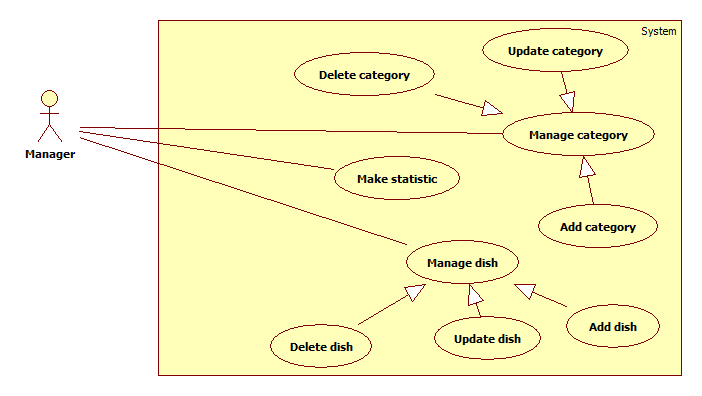


Figure 7: <Manager> Overview Use Case

##### <Manager> Make statistic (UC\_BOE0)

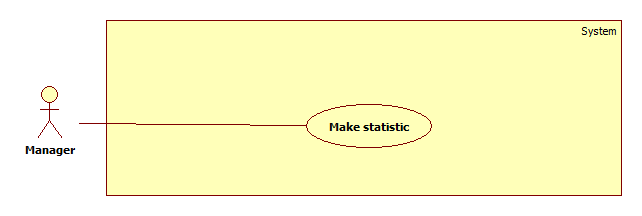


Figure 8: <Manager> Make statistic

|  |  |  |  |
| --- | --- | --- | --- |
| **USE CASE – UC\_BOE04** | | | |
| **Use Case No.** | UC\_BOE04 | **Use Case Version** | 2.0 |
| **Use Case Name** | Make statistic | | |
| **Author** | NamPD | | |
| **Date** | 22/05/2017 | **Priority** | High |
| **Actor:**   * Manager   **Summary:**   * This use case allows actor to create a statistic report.   **Goal:**   * Base on data on database, system collect and create report as user request. Report will be displayed to actor.   **Triggers:**   * Manager send make statistic command.   **Preconditions:**   * Actor must login as manager role. * System must run at least 01 day before the day manager create report. * Data is existed on server.   **Post Conditions:**   * **Success:** New report will display to manager’s view. * **Fail:** Display message “Some thing went wrong. Can not create report.”.   **Main Success Scenario:**   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 1 |  | Collect information and summarize to a report and display to manager.  [Alternative 1] | | 2 |  | Display report to manager. |   **Alternative Scenario:**   |  |  |  | | --- | --- | --- | | Step. | Actor Action | System Response | | 1 | Information required not found | Display message: “Required information can not found.“ |   **Exceptions:**  N/A  **Relationships:**  N/A  **Business Rules:**   * System will automatically create report with required information. * Report will be display in chart. * One criteria each time make report. | | | |

Table 15:USE CASE -UC\_BOE04 - <Manager> Make statistic

##### <Manager> Add category (UC\_BOE05)

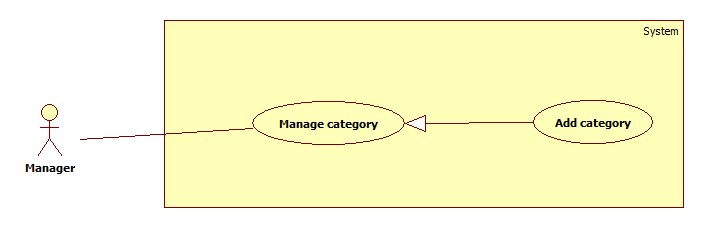
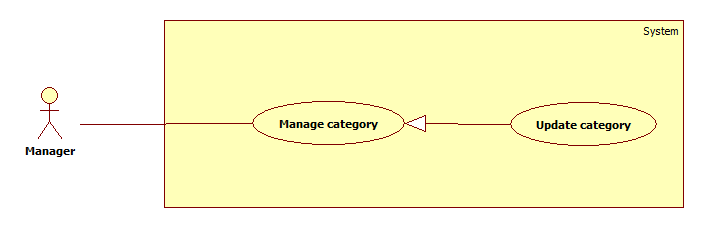


Figure 9: <Manager> Add category

|  |  |  |  |
| --- | --- | --- | --- |
| **USE CASE – UC\_BOE05** | | | |
| **Use Case No.** | UC\_BOE05 | **Use Case Version** | 2.0 |
| **Use Case Name** | Add category | | |
| **Author** | NamPD | | |
| **Date** | 22/05/2017 | **Priority** | High |
| **Actor:**   * Manager   **Summary:**   * This use case allows manager add a dish’s category to system.   **Goal:**   * Create a new category with status is active and applied in restaurant.   **Triggers:**   * Actor send add category command.   **Preconditions:**   * Actor must login as manager role. * There is no category has same name existed on system.   **Post Conditions:**   * **Success:** New category with active status display in top of the list of dish on manage dish view. * **Fail:** Display message “Some thing went wrong. Can not create report.”.   **Main Success Scenario:**   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 2 | Actor is at add category view | System requires information from manager:  **Category name**: free text field, required, length 2–25 characters.  [Exception 1] | | 3 | Manager inputs name of category. |  | | 4 | Manager send command save. |  | | 5 |  | Server validate information.  [Exception 1,2] | | 6 |  | Server save new category. | | 7 |  | System display message to notify that creating new category is successful. |   **Alternative Scenario:**  N/A  **Exceptions:**   |  |  |  | | --- | --- | --- | | No. | Cause | System Response | | 1 | Manager inputs Invalid category’s name format. | System shows error message: “Invalid format of category name. Only character A-Z and 0 – 9 allowed.”. | | 2 | Name of new category is exited. | System shows error message: “Category is exited! Please choose another one”. |   **Relationships:**  N/A  **Business Rules:**   * Status of new category is “Active”. * Name of category is only one. * The category new created with no dish will not show on menu of diner mobile application. * After create new category, manager can add dish into it immediately. * Category with at least one dish will be show on diner mobile application. | | | |

Table 16: USE CASE -UC\_BOE05 - <Manager> Add category

##### <Manager> Update dish (UC\_BOE06)

 Figure 10: <Manager> Update category

|  |  |  |  |
| --- | --- | --- | --- |
| **USE CASE – UC\_BOE06** | | | |
| **Use Case No.** | UC\_BOE06 | **Use Case Version** | 2.0 |
| **Use Case Name** | Update category | | |
| **Author** | NamPD | | |
| **Date** | 22/05/2017 | **Priority** | Normal |
| **Actor:**   * Manager   **Summary:**   * This use case allows actor update name of an existed category.   **Goal:**   * The current name of category has been replaced by new name.   **Triggers:**   * Manager send update category command.   **Preconditions:**   * Actor must login as manager role. * The selected category must have status is active.   **Post Conditions:**   * **Success:** New informations will be apply for choosen category. * **Fail:** System will display message “Fail to update category.”   **Main Success Scenario:**   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 1 | Manager at update category view | System requires information from manager:  **Category name**: free text field, required, length 2–25 characters. | | 2 | Manager inputs information. |  | | 3 |  |  | | 4 | Manager send save command |  | | 5 |  | System validates information on server.  [Exception 1,2] | | 6 |  | System saves new name for that category.  [Exception 3] | | 7 |  | System forwards to category view page. |   **Alternative Scenario:**  N/A  **Exceptions:**   |  |  |  | | --- | --- | --- | | No. | Cause | System Response | | 1 | Invalid “Category name” format. | System shows error message: “Invalid format of category name. Only character A-Z and 0 – 9 allowed.”. | | 2 | Currupted “Category name”. | System shows error message: “Category is exited! Please choose another one”. |   **Relationships:**  N/A  **Business Rules:**   * Category’s name is only one. * When updating new name is successfull, category is shown with new name. * Updated category will have status is active as it’s status before update. * The new name will be show in diner mobile application immediately when diner get menu or refresh menu. | | | |

Table 17: USE CASE -UC\_BOE06 - <Manager> Update category

##### <Manager> Delete category (UC\_BOE07)

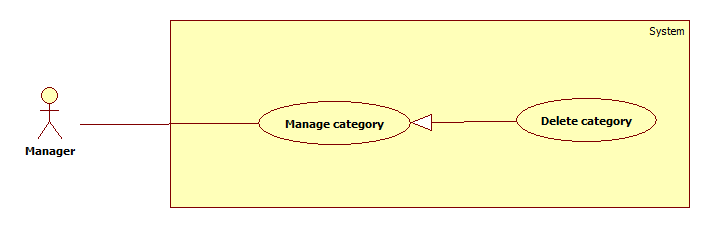


Figure 11: <Manager> Delete category

|  |  |  |  |
| --- | --- | --- | --- |
| **USE CASE – UC\_BOE07** | | | |
| **Use Case No.** | UC\_BOE07 | **Use Case Version** | 2.0 |
| **Use Case Name** | Delete category | | |
| **Author** | NamPD | | |
| **Date** | 22/05/2017 | **Priority** | Normal |
| **Actor:**   * Manager   **Summary:**   * Actor is allow to remove a category from menu.   **Goal:**   * Selected category will be set status to deactive and not showing on menu of diner mobile application, also all dishes belong to selected category.   **Triggers:**   * Actor send delete category command.   **Preconditions:**   * Actor must login as manager role. * Category’s status is “Active”. * Category must existed on system.   **Post Conditions:**   * **Success:** Deleted category status is deactive, guest or diner at make order phase will not see that category. * **Fail:** System will display message “Fail to delete category”.   **Main Success Scenario:**   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 3 |  | System show confirm dialog. | | 4 | Actor click “Đồng ý”.  [Alternative 1] |  | | 5 |  | System set that category’s status to false. |   **Alternative Scenario:**   |  |  |  | | --- | --- | --- | | Step. | Actor Action | System Response | | 1 | Actor send cancel command |  | |  |  | Dialog is hidden. Keep current view. |   **Exceptions:**  N/A  **Relationships:**  N/A  **Business Rules:**   * Category was deleted have status is “Unactive”. * All dishes belong to that category are changes status to “Unactive”. * Diner can not see the deactive category and all dishes belong to deactive category also. * The deactive category can be set to be a active one by manager later | | | |

Table 18: USE CASE -UC\_BOE07 - <Manager> Delete category

##### <Manager> Add dishes (UC\_BOE05)

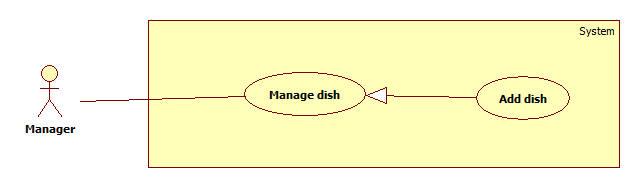
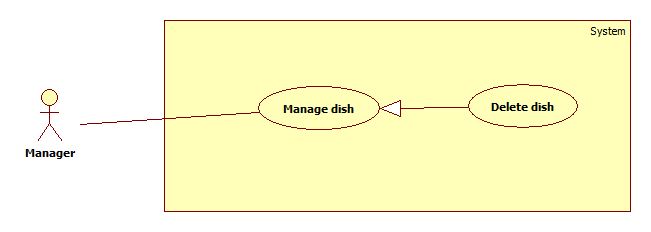


Figure 12: <Manager> Add dishes

|  |  |  |  |
| --- | --- | --- | --- |
| **USE CASE – UC\_BOE05** | | | |
| **Use Case No.** | UC\_BOE05 | **Use Case Version** | 2.0 |
| **Use Case Name** | Add dishes | | |
| **Author** | NamPD | | |
| **Date** | 22/05/2017 | **Priority** | High |
| **Actor:**   * Manager   **Summary:**   * This use case allows manager to add a new dish to system.   **Goal:**   * A new dish will be added with status is active and appear on the menu of diner mobile application in its category.   **Triggers:**   * Actor send add dish command   **Preconditions:**   * Actor must login as manager role.   **Post Conditions:**   * **Success:** New dishes with active status display in top of the list of dishes on manage dishes view. * **Fail:** Show message “Can not add new dish.”.   **Main Success Scenario:**   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | |  | Actor goes to add dish view | System requires information from manager:   * **Dish name**: free text field, required, length 2–45 characters. * **Category**: drop box, required. * **Decription**: free text area, text editor, length 10-300 characters. * **Price**: free text field, required, only numberic. * **Main material**: dropbox, required. * **Image**: file input, required. | |  | Manager inputs information for new dish. |  | |  |  | System validate information.  [Exception 1, 2, 3, 4, 5, 6] | |  | Actor send save command.  [Alternative 1] |  | |  |  | System validate information.  [Exception 1, 2, 3, 4, 5, 6] | |  |  | New dish is saved to system. | |  |  | Forward to dish detail view. Display information of new dish. |   **Alternative Scenario:**   |  |  |  | | --- | --- | --- | | Step. | Actor Action | System Response | | 1 | Actor send reset command |  | | 2 |  | Dialog is hidden. Reset field. |   **Exceptions:**   |  |  |  | | --- | --- | --- | | No. | Cause | System Response | | 1 | Invalid “Dish name” format. | System shows error message: “Only character A-Z and 0 – 9 allowed. Can not empty”. | | 2 | Invalid “Image” format. | System shows error message: “Only image file such as .png .jpg allowed”. | | 3 | Require category | System shows error message: “Dish’s category can not be empty”. | | 4 | Invalid price format. | System shows error message: “Price must number and not empty”. | | 5 | Manager has not been chosen material for new dish. | System shows error message: “Dish’s material can not be empty”. | | 6 | Dish’s name is existed. | System shows error message: “Dish’s name is existed. Please choose another one”. |   **Relationships:**  N/A  **Business Rules:**   * Dish’s name is unique. * New dish’s status is active. * The image will be resize and upload to server. * The material use in case chef announce out of material. All dish related to that materal will be hide from menu and refund if needed. * System validate information twice, in client side for checking blank informations on required fields, in server side for checking duplicate dish name. | | | |

Table 19: USE CASE -UC\_BOE05 - <Manager> Add dishes

##### <Manager> Update dish (UC\_BOE06)

Figure 13: <Manager> Update dish

|  |  |  |  |
| --- | --- | --- | --- |
| **USE CASE – UC\_BOE06** | | | |
| **Use Case No.** | UC\_BOE06 | **Use Case Version** | 2.0 |
| **Use Case Name** | Update dish | | |
| **Author** | NamPD | | |
| **Date** | 22/05/2017 | **Priority** | Normal |
| **Actor:**   * Manager   **Summary:**   * Managet can change information of existed dish.   **Goal:**   * The selected dish will have new information according to inputted information of manager.   **Triggers:**   * Actor send update command.   **Preconditions:**   * Actor must login as manager role. * Dish must be existed on system.   **Post Conditions:**   * **Success:** New informations will be apply for choosen dish. * **Fail:** System will display message “Fail to update dish.”   **Main Success Scenario:**   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 1 | Actor is at update dish view | System requires information from manager:   * **Dish name**: free text field, required, length 2–45 characters. * **Category**: drop box, required. * **Decription**: free text area, text editor, length 10-300 characters. * **Price**: free text field, required, only numberic. * **Main material**: dropbox, required. * **Image**: file input, required. | |  | Manager inputs new information. |  | |  |  | System validates information.  [Exception 1, 2, 3, 4, 5, 6] | |  | Actor send save command.  [Alternative 1] |  | |  |  | System validates information.  [Exception 1, 2, 3, 4, 5, 6] | |  |  | System save new formation. | |  |  | System forwads to dish’s detail view page. |   **Alternative Scenario:**   |  |  |  | | --- | --- | --- | | Step. | Actor Action | System Response | | 1 | Actor send cancel command |  | |  |  | Hide dialog and come back list dishes view. |   **Exceptions:**   |  |  |  | | --- | --- | --- | | No. | Cause | System Response | | 1 | Invalid “Dish name” format. | System shows error message: “Only character A-Z and 0 – 9 allowed. Can not empty”. | | 2 | Invalid “Image” format. | System shows error message: “Only image file such as .png .jpg allowed”. | | 3 | Require category | System shows error message: “Dish’s category can not be empty”. | | 4 | Invalid price format. | System shows error message: “Price must number and not empty”. | | 5 | Manager has not been chosen material for new dish. | System shows error message: “Dish’s material can not be empty”. | | 6 | Dish’s name is existed. | System shows error message: “Dish’s name is existed. Please choose another one”. |   **Relationships:**  N/A  **Business Rules:**   * System validate information twice, in client side for checking blank informations on required fields, in server side for checking duplicate dish name. * If actor update information including price, system will add new price to history table with date is the day informations have been change. | | | |
| Table 20: USE CASE -UC\_BOE06 - <Manager> Update dish | | | |

##### <Manager> Delete dishes (UC\_BOE07)

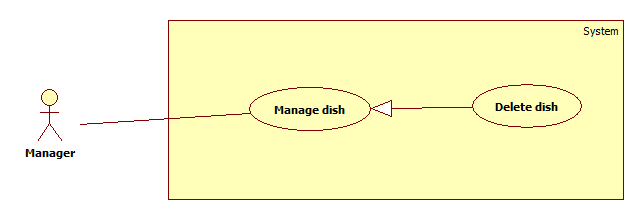


Figure 14: <Manager> Delete dishes

|  |  |  |  |
| --- | --- | --- | --- |
| **USE CASE – UC\_BOE07** | | | |
| **Use Case No.** | UC\_BOE07 | **Use Case Version** | 2.0 |
| **Use Case Name** | Delete dishes | | |
| **Author** | NamPD | | |
| **Date** | 22/05/2017 | **Priority** | Normal |
| **Actor:**   * Manager   **Summary:**   * This use case allows actor delete existed dishes.   **Goal:**   * Selected dish to be deleted will not showing to diner on the menu of diner mobile application, status set to deactive   **Triggers:**   * Actor press Delete dish button.   **Preconditions:**   * Actor must login as manager role. * These dishes must existed on system.   **Post Conditions:**   * **Success:** Deleted dishes’ status set deactive, guest or diner at make order phase will not see these dishes. * **Fail:** System will display message “Fail to delete”.   **Main Success Scenario:**   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 1 | Actor choose dishes need to delete. |  | | 2 | Actor sends command to delete. | Show confirm dialog. | |  | Actor click “Đồng ý”.  [Alternative 1] |  | | 3 |  | System set these dishes’ status to false. |   **Alternative Scenario:**   |  |  |  | | --- | --- | --- | | Step. | Actor Action | System Response | | 1 | Actor send cancel command. | Hide dialog and come back list account view. |   **Exceptions:**  N/A  **Relationships:**  N/A  **Business Rules:**   * After actor sends command to delete, system will set the status of these dishes to false. Diner or guest can not see these dishes until those status set to true. | | | |

Table 21: USE CASE -UC\_BOE07 - <Manager> Delete dishes

#### <Guest> Overview Use Case

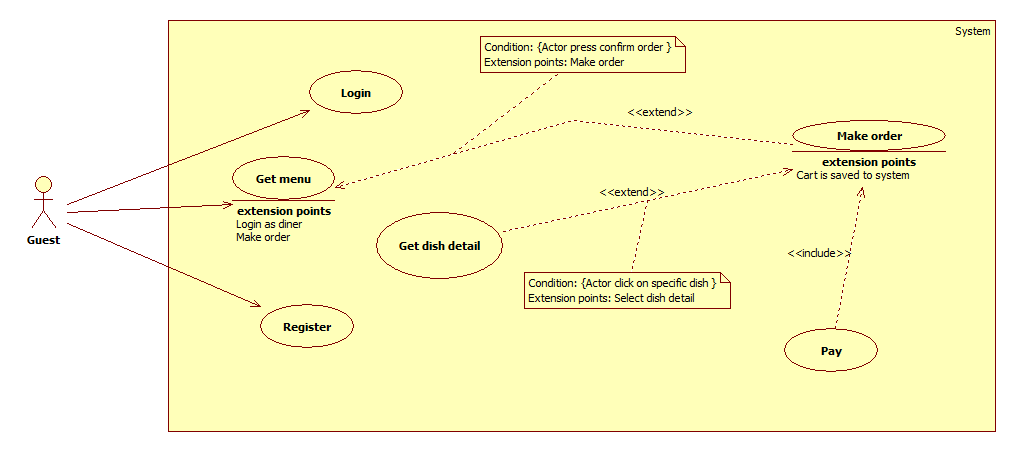


Figure 15: <Guest> Overview Use Case

##### <Guest> Register (UC\_BOE11)

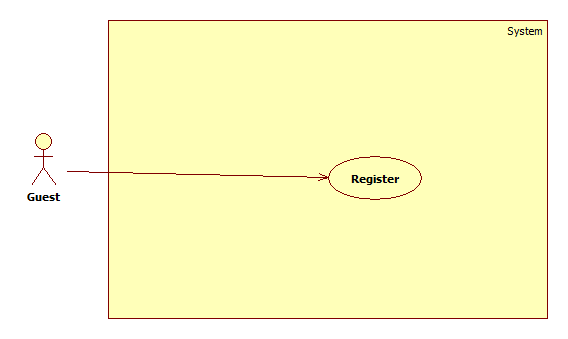


Figure 16:<Guest> Register

|  |  |  |  |
| --- | --- | --- | --- |
| **USE CASE – UC\_BOE11** | | | |
| **Use Case No.** | UC\_BOE11 | **Use Case Version** | 2.0 |
| **Use Case Name** | Register | | |
| **Author** | HoangND | | |
| **Date** | 22/05/2017 | **Priority** | Normal |
| **Actor:**   * Guest.   **Summary:**   * This use case helps guest registering a new account.   **Goal:**   * A new account have status is true, role is diner will be created.   **Triggers:**   * Actor send register command.   **Preconditions:**   * Username as actor inputted must not existed in system.   **Post Conditions:**   * **Success:** Newaccount is created successfully. * **Fail:** System will display message “Something went wrong. Can not create account now.”.   **Main Success Scenario:**   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 1 | Actor goes to register view. | System requires identity information from actor:   * **Username**: free text input, required, length 6–25 characters. * **Password**: free text input, required, length 6–30 characters. * **Confirm password**: free text input, required, length 6–30 characters. | | 2 | Actor inputs information |  | |  |  | System validates inputted information [Exception 1, 3, 4] | | 3 | Actor send register command to the system |  | |  |  | System validates inputted information [Exception 2] | | 4 |  | System signed actor in with user role and information inputed. |   **Alternative Scenario:**  N/A  **Exceptions:**   |  |  |  | | --- | --- | --- | | No. | Cause | System Response | | 1 | Invalid username format. | System shows error message: “Username cannot be blank and must be 6 – 25 characters”. | | 2 | Username already existed. | System shows error message: “Account is already exist”. | | 3 | Invald password format. | System shows error message: “Username cannot be blank and must be 6 – 30 characters”. | | 4 | Confirm password not matching | System show error message: “Confirm password do not match”. |   **Relationships:**  N/A  **Business Rules:**   * Password will be encrypted before saving to system. * After registration, actor will be redirected to home screen on mobile application, can make order immediately. * Account just registed will have status is active and role is diner. | | | |

Table 22: USE CASE -UC\_BOE11 - <Guest> Register

##### <Guest> Login (UC\_BOE10)

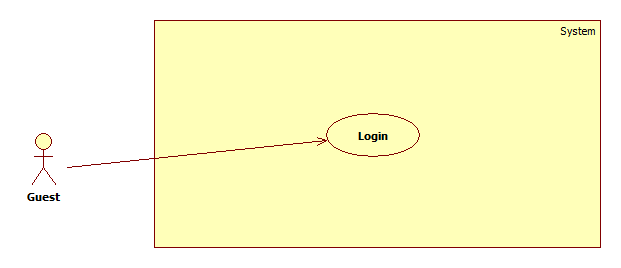


Figure 17: <Guest> Login

|  |  |  |  |
| --- | --- | --- | --- |
| **Use Case No.** | UC\_BOE10 | **Use Case Version** | 2.0 |
| **Use Case Name** | Login | | |
| **Author** | HoangND | | |
| **Date** | 22/05/2017 | **Priority** | Normal |
| **Actor:**   * Guest.   **Summary:**   * This use case helps actor login to their existed account.   **Goal:**   * This function use username and password actor send to compare with data in database. If validated, system allow actor use function as their role.   **Triggers:**   * Actor send login command.   **Preconditions:**   * Account must be existed in the system.   **Post Conditions:**   * **Success:** Actor login successfully. * **Fail:** System will display message “Something go wrong. Can not log you in.”.   **Main Success Scenario:**   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 1 | Actor goes to login view. | System requires identity information from actor:   * **Username**: free text input, required, length 6–25 characters. * **Password**: free text input, required, length 6–30 characters, only accept email format. | | 2 | Actor inputs information |  | |  |  | System validates inputted information [Exception 1,2] | | 3 | Actor send login command to the system |  | | 4 |  | System validates inputted information [Exception 3] |   **Alternative Scenario:**  N/A  **Exceptions:**   |  |  |  | | --- | --- | --- | | No. | Cause | System Response | | 1 | Invalid username format. | System shows error message: “Username cannot be blank”. | | 2 | Invalid password format. | System shows error message: “Password cannot be blank”. | | 3 | Wrong indentity information. | Wrong indentity information, system shows error message: “Invalid username /password.” |   **Relationships:**  N/A  **Business Rules:**   * If wrong username/password, system display message and stay in the login view. * If validated usrename/password, system will log actor in and redirect actor to main view of their role. | | | |

Table 23: USE CASE - UC\_BOE10 - <Guest> Login

##### <Guest,Diner> Make order (UC\_BOE12)

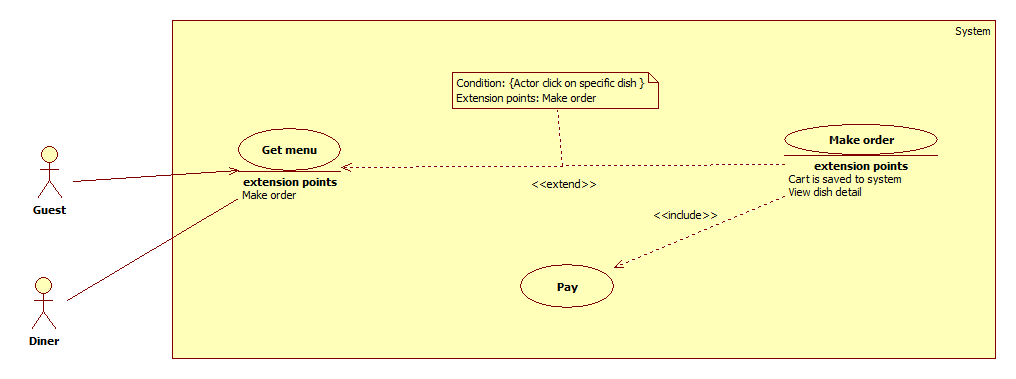


Figure 18: <Guest,Diner> Make order

|  |  |  |  |
| --- | --- | --- | --- |
| **USE CASE – UC\_BOE12** | | | |
| **Use Case No.** | UC\_BOE12 | **Use Case Version** | 2.0 |
| **Use Case Name** | Make order | | |
| **Author** | HoangND | | |
| **Date** | 22/05/2017 | **Priority** | Very High |
| **Actor:**   * Guest or Diner.   **Summary:**   * This use case allows actor to make order and do payment.   **Goal:**   * This function help actor create a cart contains all order items they ordered. * Cart have status is Waiting payment.   **Triggers:**   * Actor send make order command.   **Preconditions:**   * On cart must have at least one dish with quantity bigger than 0.   **Post Conditions:**   * **Success:** Chef receive notification about new order. Actor receive notification about payment. * **Fail:** N/A   **Main Success Scenario:**   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 1 | Actor send confirm cart command.  [Alternative 1] |  | | 2 |  | System check for material of these dishes.  [Exception 1] | | 3 |  | System display invoice contains information of order actor just made. | | 4 | Actor send continue command |  | | 5 |  | System direct actor to sandbox of mobile banking third party. | | 6 | Actor do payment.  [Alternative 2] |  | | 7 |  | System check for payment information. Set status for this cart is “Pending” and send notification to chef. |   **Alternative Scenario:**   |  |  |  | | --- | --- | --- | | Step. | Actor Action | System Response | | 1 | Actor cancel cart | System hide confirm dialog. Show cart detail view. | | 2 | Actor do not do payment. | System redirect user to cart deail view. | | 3 | Actor choose continue meal | System refund money actor paid for out of material dishes. | | 4 | Actor choose cancel meal | System refund money for unfinish dishes on order. |   **Exceptions:**   |  |  |  | | --- | --- | --- | | No. | Cause | System Response | | 1 | Out of material | Ask actor for decision continue or cancel meal.  [Alternative 3,4] |   **Relationships:**   * Extend to Get menu: Actor is able to make order when actor recived menu from system. * Include Pay: Actor has to do payment after make order.   **Business Rules:**   * After actor make order, information of just make order will contain on a cart with status is Waiting payment. * The ordinal of dishes that actor order will be preserve by system and serve sequentially according to that ordinal if payment success. * System checks for payment informations:   + If system do not receive any informaton of payment, cart will be cancel, not save in system.   + If system receive payment success message, system change status of cart to Pending. * After payment success, system will recognize cart as order with status is Pending and send notify to chef and save to system. * The total of order will be saved with 1000 unit of VND in system. * If actor create another cart within 30 minutes later, in mobile application will combination current cart with last cart, in system will recognize as 02 orders. * System will check for material in this usecase twice, one when create cart, one after payment. * If out of material for any dishes contains on order, system will ask for actor decision:   + If actor choose continue meal, system refund money actor paid for out of material dishes.   + If actor choose stop meal, system refund money for unfinish dishes on order. | | | |

Table 24: USE CASE -UC\_BOE11 - <Guest,Dinner> Make order

#### <Chef> Overview Use Case

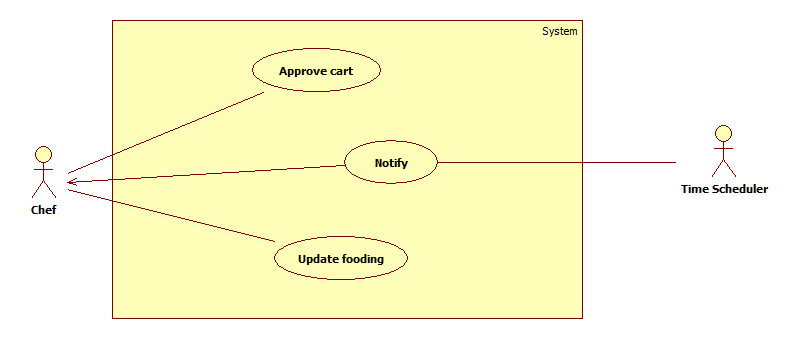


Figure 19: <Chef> Overview Use Case

##### <Chef> Receive notification (UC\_BOE08)

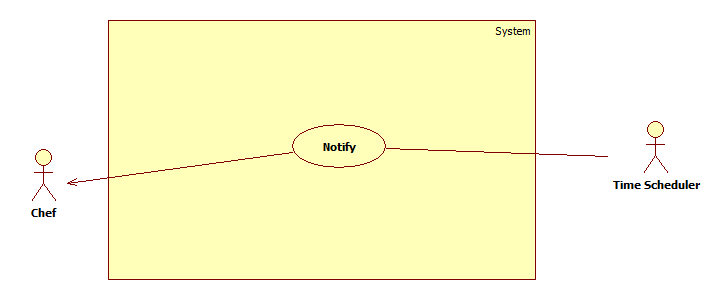


Figure 20:<Chef> Receive notification

|  |  |  |  |
| --- | --- | --- | --- |
| **USE CASE – UC\_BOE08** | | | |
| **Use Case No.** | UC\_BOE08 | **Use Case Version** | 2.0 |
| **Use Case Name** | Receive notification | | |
| **Author** | PhongLD | | |
| **Date** | 22/05/2017 | **Priority** | High |
| **Actor:**   * Chef.   **Summary:**   * Server sends information about dishes what is ordered by diners.   **Goal:**   * Dishes grouped by name and display to chef application. * Chef knows dish’s name and dish’s quantity to be cooked.   **Triggers:**   * System recorded a order with status Pending.   **Preconditions:**   * Actor must login as chef role. * Diner, Guest finished payment for ordering their food.   **Post Conditions:**   * **Success:** A new notification appear on chef’s view. * **Fail:** N/A   **Main Success Scenario:**   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 1 |  | Server automatically sends information about dishes have been ordered.  [Exception 1] | | 2 | Chef click to choice Dish Tab. |  | | 3 |  | Show all dishes have been ordered. |   **Alternative Scenario:**  N/A  **Exceptions:**   |  |  |  | | --- | --- | --- | | Step. | Actor Action | System Response | | 1 | App can not connect to server. | Show error message: “Can not connect to server”. |   **Relationships:**  N/A  **Business Rules:**   * System must group dishes by name and display to chef. * Chef should only care about what and how many dishes need to cook. | | | |

Table 25: USE CASE -UC\_BOE08 - <Chef > Receive notification

##### <Chef> Update fooding (UC\_BOE08)

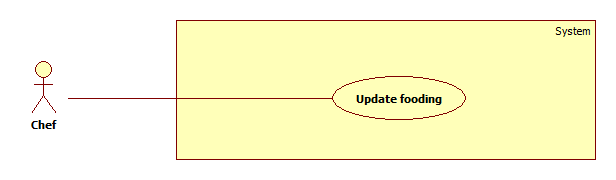


Figure 21: <Chef> Update fooding

|  |  |  |  |
| --- | --- | --- | --- |
| **USE CASE – UC\_BOE08** | | | |
| **Use Case No.** | UC\_BOE08 | **Use Case Version** | 2.0 |
| **Use Case Name** | Update fooding | | |
| **Author** | NamPD | | |
| **Date** | 22/05/2017 | **Priority** | High |
| **Actor:**   * Chef   **Summary:**   * This use case allows actor to hide a dish from menu.   **Goal:**   * This function change the status of choosen dish to false. Diner or guest can not see that dish. The day after, status of that dish back to true.   **Triggers:**   * Chef detect that can not cook that dish.   **Preconditions:**   * Actor must login as chef role. * That dish must existed on system.   **Post Conditions:**   * **Success:** Status the choosen dish will be set to false. The next day, it’s status set back to true. * **Fail:** Status the choosen dish not set to false. The next day, it’s status not set back to true.   **Main Success Scenario:**   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 1 | Chef at list dishes view. |  | | 2 | Chef choose the dish need to hide from menu and press “Ẩn” |  | |  |  | Show confirm dialog. | |  | Actor click “Đồng ý”.  [Alternative 1] |  | | 3 |  | System change status of dish to false. |   **Alternative Scenario:**   |  |  |  | | --- | --- | --- | | Step. | Actor Action | System Response | | 1 | Actor send cancel command | Hide dialog and come back list dishes view. |   **Exceptions:**  N/A  **Relationships:**  N/A  **Business Rules:**   * Chef chose dish need to hide and press “Ẩn”. * System display confirm dialog.   + If chef press “Hủy bỏ” system hide dialog   + If chef press “Đồng ý” system change status of dish to false. * The next day, status of dish back to true as normal. | | | |

Table 26: USE CASE -UC\_BOE08 - <Chef > Update fooding

#### <Diner> Overview Use Case

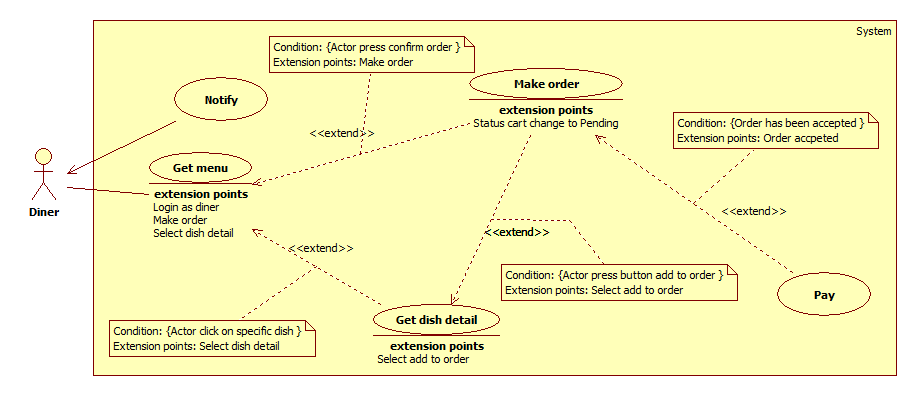


Figure 22: <Diner> Overview Use Case

##### <Diner> Make order (UC\_BOE09)

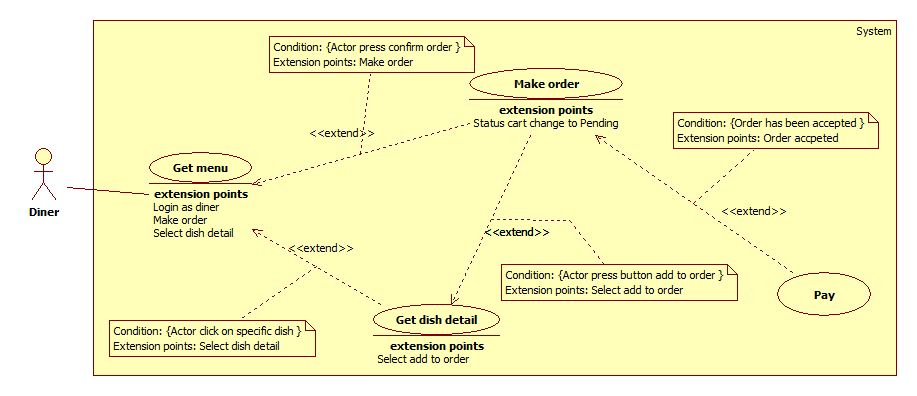


Figure 23: <Diner> Make order

|  |  |  |  |
| --- | --- | --- | --- |
| **USE CASE – UC\_BOE09** | | | |
| **Use Case No.** | UC\_BOE09 | **Use Case Version** | 2.0 |
| **Use Case Name** | Make order | | |
| **Author** | LinhNK | | |
| **Date** | 22/05/2017 | **Priority** | Very High |
| **Actor:**   * Diner.   **Summary:**   * This use case allows diner choices dishes what they want to eat in restaurant.   **Goal:**   * New order is created. It contains dishes are in restaurant.   **Triggers:**   * Menu was shown on the diner’s view.   **Preconditions:**   * Restaurant must have at least one dish in menu. * Cart of diner must have least one dish with quantity bigger than 0.   **Post Conditions:**   * **Success:** Waiter’s view receive a new notification. * **Fail:** Waiter’s view not receive a new notification.   **Main Success Scenario:**   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 1 | Diner choice dishes they want eating.  [Alternative 1] |  | | 2 |  | Dish what diner choice is added to cart. | | 3 | Diner choice tab cart. |  | | 4 |  | Display information of cart.  [Alternative 3, 4] | | 5 | Diners clicks “Thanh Toán” button.  [Exception 1] |  | | 6 |  | Display information about new order. | | 7 | Diners clicks “Thanh Toán” button.  [Exception 1] |  | | 8 |  | Forward to payment’s view. | | 9 | Diners inputed information to done payment process.  [Exception 2] |  | | 10 |  | Show dialog to notify making order is succesful. |   **Alternative Scenario:**   |  |  |  | | --- | --- | --- | | Step. | Actor Action | System Response | | 1 | Diner click to view detail of dish. |  | |  | Dish detail is shown. | | Diner clicks “Chọn Món” button. | That dish is add to cart. | | 2 | Diner clicks “Tiếp Tục” button. |  | |  | Forward to payment view. | | Diner clicks “Hủy” button. |  | |  | Cart is cleared. | |  | Forward to menu view. | | 3 | Diner clicks “Decrease” buttons. |  | |  | Dish’s quantity is decreased 1 value.  [Alternative 5] | | 4 | Diner clicks “Increase” buttons |  | |  | Dish’s quantity is ccreased 1 value. | | 5 | Dish’s quantity is 0. |  | |  | Remove dish out of cart |   **Exceptions:**   |  |  |  | | --- | --- | --- | | No. | Cause | System Response | | 1 | Dish what diner choice is not available.  [Alternative 2, 3] | System displays message to notify about dish is not availabe. | | 2 | Payment process is not successful. | System displays error message. |   **Relationships:**   * Payment process folow payment usecase.   **Business Rules:**   * Dish is ordered must have status “Available”. * Range of dish’s quantity: from 1 to 4. * Dish is checked available again before diner do payment process. | | | |

Table 27:USE CASE -UC\_BOE09 - <Diner> Make order

##### <Diner> Receive notification (UC\_BOE10)

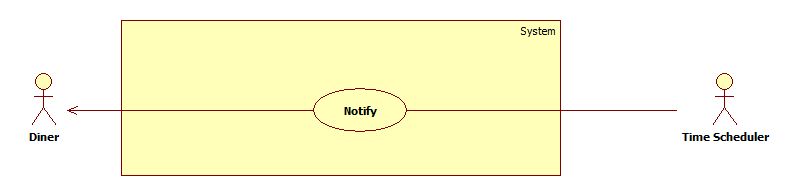


Figure 24:<Diner> Receive notification

|  |  |  |  |
| --- | --- | --- | --- |
| **USE CASE – UC\_BOE10** | | | |
| **Use Case No.** | UC\_BOE10 | **Use Case Version** | 2.0 |
| **Use Case Name** | Receive notification | | |
| **Author** | LinhNK | | |
| **Date** | 22/05/2017 | **Priority** | High |
| **Actor:**   * Diner   **Summary:**   * This use case allows diner to receive notification about orders if it was accept/reject by chef.   **Goal:**   * Notification will be send to diner including status accept/reject of order they have made.   **Triggers:**   * A cart change status to Cooking/Reject.   **Preconditions:**   * Actor must login as diner role. * Chef finished accept/reject.   **Post Conditions:**   * **Success:** A new notification appear on diner’s phone. * **Fail:** Nonotification appear on diner’s phone. N/A   **Main Success Scenario:**   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 1 | Chef’s phone connected to wifi. |  | | 2 |  | Notification display in diner’s phone.  [Alternative 1] |   **Alternative Scenario:**   |  |  |  | | --- | --- | --- | | Step. | Actor Action | System Response | | 1 | Notification show order has been rejected. | Show app allow diner to make order again. |   **Exceptions:**  N/A  **Relationships:**  N/A  **Business Rules:**   * Diner open application on their device and it is connecting internet. * A request is sent to server. If there are many notification is for them, server is sent to diner. * Device of diner make signal after it receive a new notification. * Diner can click on that notification to read all information. | | | |

Table 28: USE CASE -UC\_BOE08 - <Chef > Receive notification

#### <Waiter> Overview Use Case

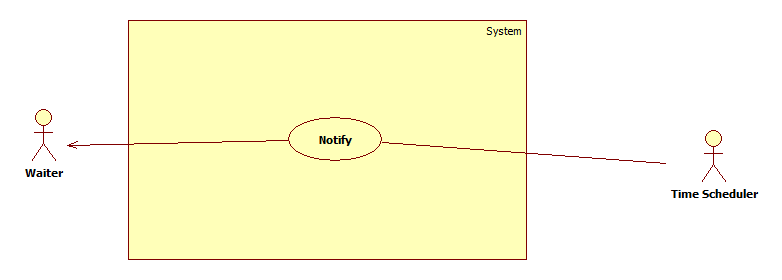


Figure 25: <Waiter> Overview Use Case

##### <Waiter> Receive notification (UC\_BOE09)

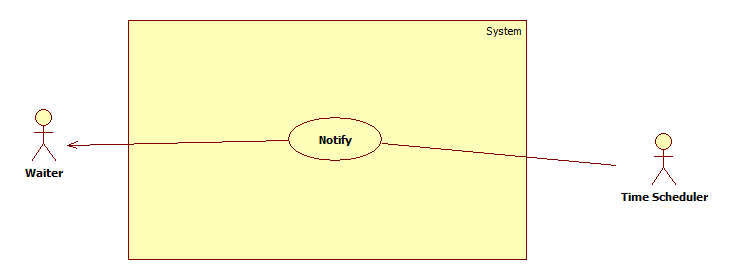


Figure 26: <Waiter> Receive notification

|  |  |  |  |
| --- | --- | --- | --- |
| **USE CASE – UC\_BOE09** | | | |
| **Use Case No.** | UC\_BOE09 | **Use Case Version** | 2.0 |
| **Use Case Name** | Receive notification | | |
| **Author** | PhongLD | | |
| **Date** | 22/05/2017 | **Priority** | High |
| **Actor:**   * Time sheduler.   **Summary:**   * This use case allows Waiter to receive notify if dishes need bring to diner.   **Goal:**   * Notification will be send to waiter including information which dish need bring out to diner or guest.   **Triggers:**   * Order change status to Finished.   **Preconditions:**   * Actor must login as waiter role. * There were a dish need to bring out.   **Post Conditions:**   * **Success:** Waiter’s view receive a new notification. * **Fail:** Waiter’s view not receive a new notification.   **Main Success Scenario:**   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 1 | Chef at waiter’s view. |  | | 2 |  | Notification is send to waiter’s view | | 3 | Waiter swipe notification to the right. | System hide notification. |   **Alternative Scenario:**  N/A  **Exceptions:**  N/A  **Relationships:**  N/A  **Business Rules:**   * After chef mark cart to status Finished. A notficication about that dish send to waiter with information to identify that dish. * Waiter swipe notification to right to mark as bringing out and done the order. | | | |
| Table 29: USE CASE -UC\_BOE09 - <Waiter> Receive notification | | | |

## 3. Software System Attribute

## 3.1 Usability

* Admin and manager interface of web application are written in Vietnamese including all dialogs and messages.
* Diner, waiter and chef interface of mobile application are written in Vietnamese including all dialogs and messages.
* The system needs no more than one day of training for staffs.

## 3.2 Reliability

* System should not store any customers’ banking informations.
* Web application is only accessed by admintrators and managers.

## 3.3 Availability

* Data will be backed up to make sure that if it have problem while running then all necessary data must be protected and restore as soon as possible.

## 3.4 Security

* Each role of user has a specific permission to interact with the system.
* System always checks for authorization and authentication before doing anything.
* Only Admin can grant permission to accounts.

## 3.5 Maintainability

* N/A

## 3.6 Portability

N/A

## 3.7 Performance

* The notification is sent less than 2 seconds after it is made by user.

## 4. Conceptual Diagram

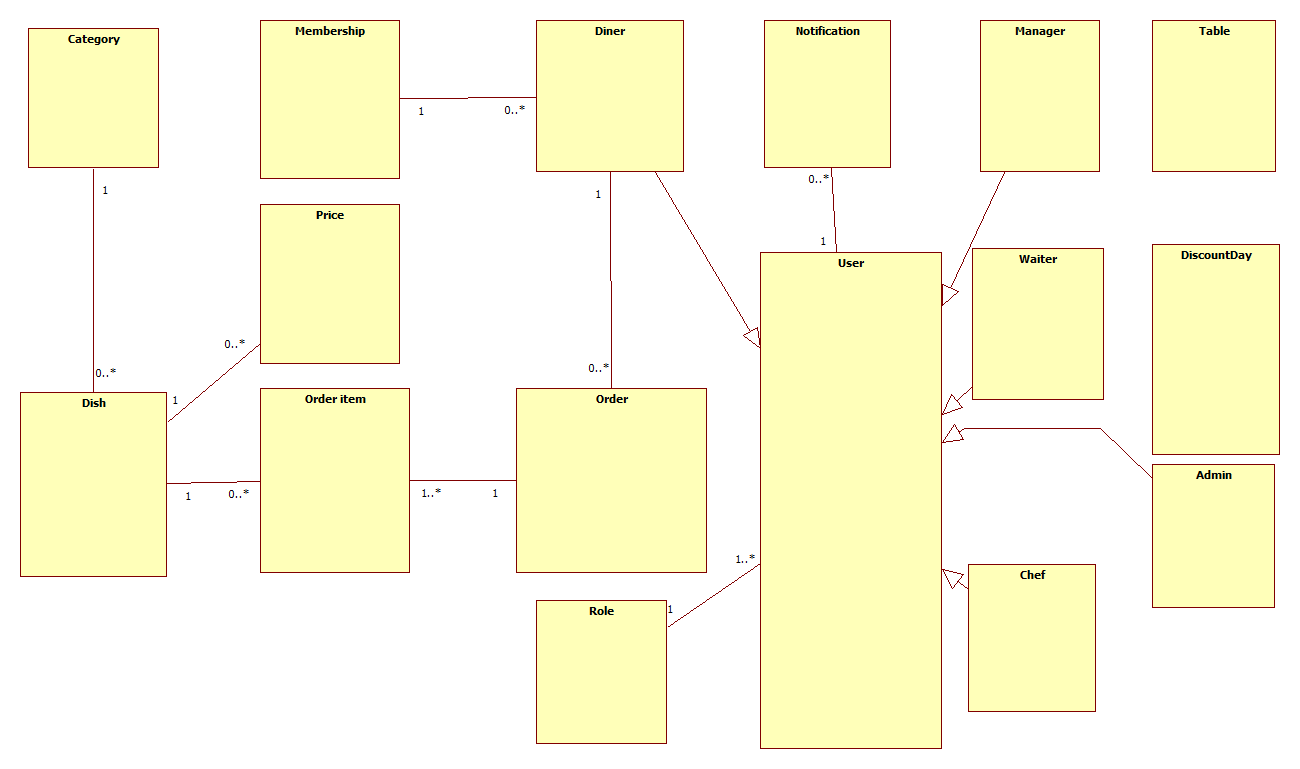


Figure 27: Conceptual diagram

**Data Dictionary**

|  |  |
| --- | --- |
| *Entity Data dictionary: describe all content of all entities* | |
| *Entity Name* | ***Description*** |
| *Category* | *Contain list of category.* |
| *Dish* | *Contain list of dish.* |
| *Price* | *Contain price of dish and history changed price of dish.* |
| *Order item* | *Contain information about items on orders have been made.* |
| *Order* | *Contain list of order made by diners or guests.* |
| *Membership* | *Contain information about list of membership and its’ welfare.* |
| *User* | *Contain information of all users in system.* |
| *Role* | *Contain information about role of all users in system.* |
| *Admin* | *Persons who administration system.* |
| *Manager* | *Person who manage system.* |
| *Chef* | *Person who approve order and make dish.* |
| *Diner* | *Person who allowed to order dish in system.* |
| *Waiter* | *Person who receive notification about bring out food to diners.* |
| *Notification* | *Contain all notification in the system.* |
| *Table* | *Contain information about table in system.* |
| *Discount day* | *Contain information about discount for specific dish on specific day.* |

Table 30: Conceptual Diagram Data Dictionary

# D. Report No. 4 Software Design Description

## Design Overview

* This document describes the technical and user interface design of **BOE system**. It includes the architectural design, the detailed design of common functions and business functions and the design of database model.
* The architectural design describes the overall architecture of the system and the architecture of each main component and subsystem.
* The detailed design describes static and dynamic structure for each component and functions. It includes class diagrams, class explanations and sequence diagrams for each use cases.
* The database design describes the relationships between entities and details of each entity.
* Document overview:
  + Section 2: gives an overall description of the system architecture design.
  + Section 3: gives component diagrams that describe the connection and integration of the system.
  + Section 4: gives the detail design description which includes class diagram, class explanation, and sequence diagram to details the application functions.
  + Section 5: describe screens design.
  + Section 6: describe a fully attributed ERD.
  + Section 7: describe algorithms**.**

## System Architectural Design

### 2.1 Web application architecture description

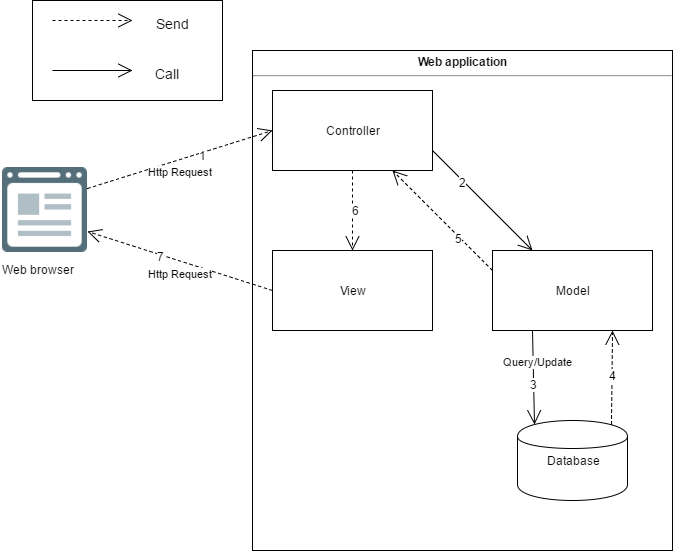
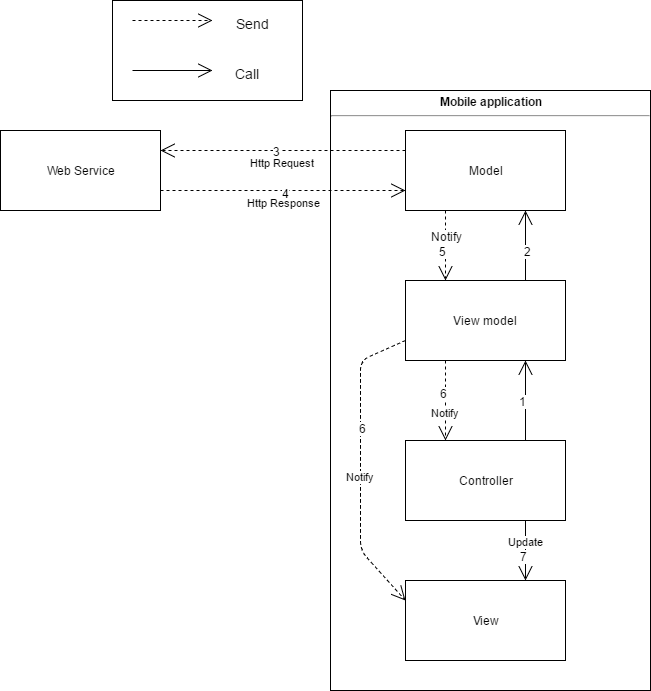
**

Figure 28: Web Application Architecture

The web application uses MVC as an architecture. We choose this architecture for a couple of reasons:

* Web Application includes the Web services (public API for mobile application), with the MVC architecture, we can use the business code in Web service without repeat the code (reusable).
* The architecture divides the application into 3 layer. This increases reusability and testability. Enable test driven development.
* Adapt change to a layer cause less effect the other layers. Support scum model with change regularly.

### 2.2 Mobile application architecture description



The web application uses MVVM as an architecture. We choose this architecture for a couple of reasons:

* Loosing the coupling between view and controller.
* The View will be notified to update and display data.
* The View Model will contain view logic information to display in view. This help increase reusability of View Model.
* Develop Mobile Application using MVVM have increase maintainability, testability, extensibility.

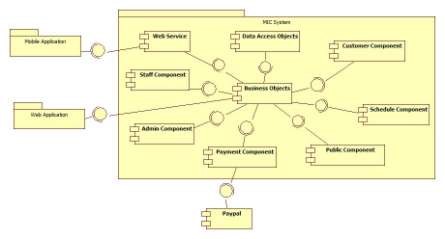
## Component Diagram

<Thể hiện việc chia hệ thống thành các component. Nội dung này dựa trên kiến trúc đã đề ra ở phần trên để chia cho phù hợp và đúng mô hình> **Ghi chú:** Xem lại bộ quy ước kí hiệu của UML 2.0 trước khi vẽ các mối quan hệ cũng như hiểu rõ thiết kế để vẽ chính xác. Nếu tool không phù hợp thì nhóm nên dùng Paint để vẽ

<Mô tả từng thành phần trong hình vẽ theo bảng biểu bên dưới.>

|  |  |
| --- | --- |
| **Component dictionary: describe component** | |
| **Component Name** | **Description** |
|  |  |

*Ví dụ*



**Figure 10 Component Diagram**

|  |  |
| --- | --- |
| *Component Dictionary: Describes components* | |
| *Web Application* | *Web application package: View, Controller* |
| *Mobile Application* | *Mobile application package* |
| *PayPal* | *Handle payment process with PayPal API* |
| *Payment Component* | *Component to handle payment process* |
| *Web Service* | *Provide API for mobile applications to interact with the system.* |
| *Staff Component* | *Component to handle staff activities in the system* |
| *Customer Component* | *Component to handle customer activities in the system* |
| *Public Component* | *Component to handle guest activities in the system* |
| *Admin Component* | *Component to handle admin activities in the system* |
| *Schedule Component* | *Component to handle scheduler in the system* |
| *Business Objects* | *Common objects to handle domain business operations for each components* |
| *Data Access Objects* | *Component to handle interaction between the system and database* |

***Table 10 Component Dictionary***

## 4. Detailed Description

### 4.1 Class Diagram

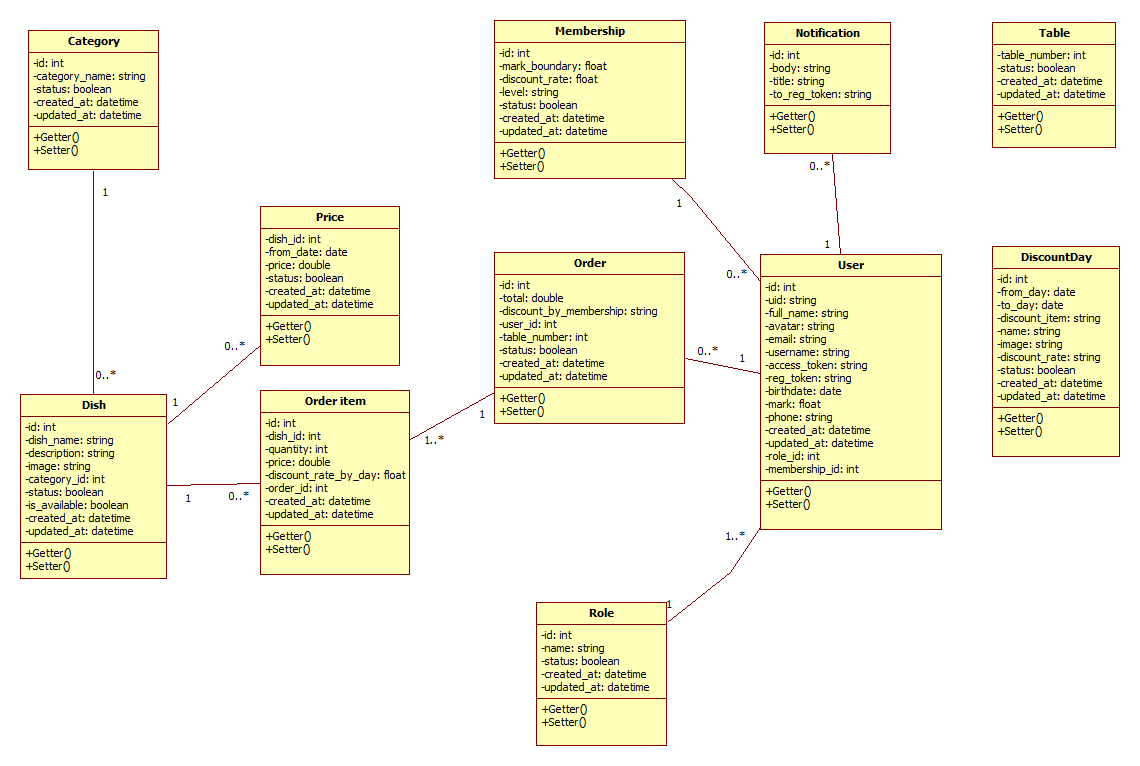


Figure 29: Class Diagram

|  |  |  |
| --- | --- | --- |
| *Class dictionary: describe Class* | | |
| *Class Name* | ***Mapping column with Conceptual diagram*** | ***Description*** |
| *Category* | *Category* | *Contain list of category* |
| *Dish* | *Dish* | *Contain list of dish.* |
| *Price* | *Price* | *Contain price of dish and history changed price of dish.* |
| *Order item* | *Order item* | *Contain information about items on orders have been made.* |
| *Order* | *Order* | *Contain list of order made by diners or guests.* |
| *Membership* | *Membership* | *Contain information about list of membership and its’ welfare.* |
| *User* | *User* | *Contain information of all users in system.* |
| *Role* | *Role* | *Contain information about role of all users in system.* |
| *Admin* | *Admin* | *Persons who administration system.* |
| *Manager* | *Manager* | *Person who manage system.* |
| *Chef* | *Chef* | *Person who approve order and make dish.* |
| *Diner* | *Diner* | *Person who allowed to order dish in system.* |
| *Waiter* | *Waiter* | *Person who receive notification about bring out food to diners.* |
| *Notification* | *Notification* | *Contain all notification in the system.* |
| *Table* | *Table* | *Contain information about table in system.* |
| *Discount day* | *Discount day* | *Contain information about discount for specific dish on specific day.* |

Table 31: Class dictionary

### 4.2 Class Diagram Explanation

#### 4.2.1 Category

*Attribute*

|  |  |  |  |
| --- | --- | --- | --- |
| ***Attribute*** | ***Type*** | ***Visibility*** | ***Description*** |
| *id* | *int* | *Private* | *Unique identifier of category* |
| *category\_name* | *string* | *Private* | *Name of category* |
| *status* | *boolean* | *Private* | *Status of category* |
| *created\_at* | *datetime* | *Private* | *Date time when create category* |
| *updated\_at* | *datetime* | *Private* | *Date time when update category* |

*Method*

|  |  |  |  |
| --- | --- | --- | --- |
| ***Method*** | ***Return type*** | ***Visibility*** | ***Description*** |
| *Getter* | *Attribute type* | *Public* | *Get attribute value* |
| *Setter* | *Void* | *Public* | *Set value of attribute* |

#### 4.2.2 Dish

*Attribute*

|  |  |  |  |
| --- | --- | --- | --- |
| ***Attribute*** | ***Type*** | ***Visibility*** | ***Description*** |
| *id* | *int* | *Private* | *Unique identifier of dish* |
| *dish\_name* | *string* | *Private* | *Name of dish* |
| *description* | *string* | *Private* | *Description of dish* |
| *image* | *string* | *Private* | *Illustration of dish* |
| *status* | *boolean* | *Private* | *Status of dish* |
| *is\_available* | *boolean* | *Private* | *Dish is available or unavailable for order* |
| *category\_id* | *int* | *Private* | *Dish belong to which category* |
| *created\_at* | *datetime* | *Private* | *Date time when create dish* |
| *updated\_at* | *datetime* | *Private* | *Date time when update dish* |

*Method*

|  |  |  |  |
| --- | --- | --- | --- |
| ***Method*** | ***Return type*** | ***Visibility*** | ***Description*** |
| *Getter* | *Attribute type* | *Public* | *Get attribute value* |
| *Setter* | *Void* | *Public* | *Set value of attribute* |

#### 4.2.3 Price

*Attribute*

|  |  |  |  |
| --- | --- | --- | --- |
| ***Attribute*** | ***Type*** | ***Visibility*** | ***Description*** |
| *id* | *int* | *Private* | *Unique identifier of a price* |
| *from\_date* | *date* | *Private* | *Date of changing price* |
| *status* | *string* | *Private* | *Status of changing price* |
| *created\_at* | *datetime* | *Private* | *Date time when create dish* |
| *updated\_at* | *datetime* | *Private* | *Date time when update dish* |

*Method*

|  |  |  |  |
| --- | --- | --- | --- |
| ***Method*** | ***Return type*** | ***Visibility*** | ***Description*** |
| *Getter* | *Attribute type* | *Public* | *Get attribute value* |
| *Setter* | *Void* | *Public* | *Set value of attribute* |

#### 4.2.4 Order item

*Attribute*

|  |  |  |  |
| --- | --- | --- | --- |
| ***Attribute*** | ***Type*** | ***Visibility*** | ***Description*** |
| *id* | *int* | *Private* | *Unique identifier of order item* |
| *dish\_id* | *int* | *Private* | *Which dish order item is* |
| *price* | *double* | *Private* | *Price of item* |
| *discount\_rate\_by\_day* | *float* | *Private* | *Discount for order item* |
| *quantity* | *int* | *Private* | *Quantity of order item* |
| *order\_id* | *int* | *Private* | *Order which order item belong to* |
| *created\_at* | *datetime* | *Private* | *Date time when create order item* |
| *updated\_at* | *datetime* | *Private* | *Date time when update order item* |

*Method*

|  |  |  |  |
| --- | --- | --- | --- |
| ***Method*** | ***Return type*** | ***Visibility*** | ***Description*** |
| *Getter* | *Attribute type* | *Public* | *Get attribute value* |
| *Setter* | *Void* | *Public* | *Set value of attribute* |

#### 4.2.5 Order

*Attribute*

|  |  |  |  |
| --- | --- | --- | --- |
| ***Attribute*** | ***Type*** | ***Visibility*** | ***Description*** |
| *id* | *int* | *Private* | *Unique identifier of order* |
| *total* | *double* | *Private* | *The total amount of money for order* |
| *price* | *double* | *Private* | *Price of item* |
| *discount\_date\_by\_membership* | *float* | *Private* | *Discount for order by membership* |
| *user\_id* | *int* | *Private* | *Order belong to which user* |
| *table\_number* | *int* | *Private* | *Order was made in which table* |
| *status* | *boolean* | *Private* | *Status of order* |
| *created\_at* | *datetime* | *Private* | *Date time when create oder* |
| *updated\_at* | *datetime* | *Private* | *Date time when update order* |

*Method*

|  |  |  |  |
| --- | --- | --- | --- |
| ***Method*** | ***Return type*** | ***Visibility*** | ***Description*** |
| *Getter* | *Attribute type* | *Public* | *Get attribute value* |
| *Setter* | *Void* | *Public* | *Set value of attribute* |

#### 4.2.6 Membership

*Attribute*

|  |  |  |  |
| --- | --- | --- | --- |
| ***Attribute*** | ***Type*** | ***Visibility*** | ***Description*** |
| *id* | *int* | *Private* | *Unique identifier of membership* |
| *mark\_boundary* | *double* | *Private* | *The boundary of membership when go over boundary, user reach this membership* |
| *discount\_rate* | *float* | *Private* | *Rate of discount for membership user reached.* |
| *level* | *string* | *Private* | *Name of membership* |
| *status* | *boolean* | *Private* | *Status of membership* |
| *created\_at* | *datetime* | *Private* | *Date time when create membership* |
| *updated\_at* | *datetime* | *Private* | *Date time when update membership* |

*Method*

|  |  |  |  |
| --- | --- | --- | --- |
| ***Method*** | ***Return type*** | ***Visibility*** | ***Description*** |
| *Getter* | *Attribute type* | *Public* | *Get attribute value* |
| *Setter* | *Void* | *Public* | *Set value of attribute* |

#### 4.2.7 User

*Attribute*

|  |  |  |  |
| --- | --- | --- | --- |
| ***Attribute*** | ***Type*** | ***Visibility*** | ***Description*** |
| *id* | *int* | *Private* | *Unique identifier of user* |
| *uid* | *string* | *Private* | *User id of user on system* |
| *full\_name* | *string* | *Private* | *Full name of user* |
| *avatar* | *string* | *Private* | *URL of user’s avatar* |
| *email* | *string* | *Private* | *Email of user* |
| *username* | *string* | *Private* | *Username of user in system* |
| *access\_token* | *string* | *Private* | *Access token of user in system to identify which user using function* |
| *reg\_token* | *string* | *Private* | *Reg token of user in system to send notification* |
| *birthday* | *date* | *Private* | *Birthday of user* |
| *mark* | *float* | *Private* | *Mark to compare with boundary on membership.* |
| *phone* | *string* | *Private* | *Phone number of user* |
| *role\_id* | *int* | *Private* | *Role which user is in* |
| *membership\_id* | *int* | *Private* | *Which membership is user in* |
| *created\_at* | *datetime* | *Private* | *Date time when create dish* |
| *updated\_at* | *datetime* | *Private* | *Date time when update dish* |

*Method*

|  |  |  |  |
| --- | --- | --- | --- |
| ***Method*** | ***Return type*** | ***Visibility*** | ***Description*** |
| *Getter* | *Attribute type* | *Public* | *Get attribute value* |
| *Setter* | *Void* | *Public* | *Set value of attribute* |

#### 4.2.8 Role

*Attribute*

|  |  |  |  |
| --- | --- | --- | --- |
| ***Attribute*** | ***Type*** | ***Visibility*** | ***Description*** |
| *id* | *int* | *Private* | *Unique identifier of role* |
| *name* | *string* | *Private* | *Name of role* |
| *status* | *string* | *Private* | *Status of role* |
| *created\_at* | *datetime* | *Private* | *Date time when create role* |
| *updated\_at* | *datetime* | *Private* | *Date time when update role* |

*Method*

|  |  |  |  |
| --- | --- | --- | --- |
| ***Method*** | ***Return type*** | ***Visibility*** | ***Description*** |
| *Getter* | *Attribute type* | *Public* | *Get attribute value* |
| *Setter* | *Void* | *Public* | *Set value of attribute* |

#### 4.2.9 Role

*Attribute*

|  |  |  |  |
| --- | --- | --- | --- |
| ***Attribute*** | ***Type*** | ***Visibility*** | ***Description*** |
| *id* | *int* | *Private* | *Unique identifier of role* |
| *name* | *string* | *Private* | *Name of role* |
| *status* | *string* | *Private* | *Status of role* |
| *created\_at* | *datetime* | *Private* | *Date time when create role* |
| *updated\_at* | *datetime* | *Private* | *Date time when update role* |

*Method*

|  |  |  |  |
| --- | --- | --- | --- |
| ***Method*** | ***Return type*** | ***Visibility*** | ***Description*** |
| *Getter* | *Attribute type* | *Public* | *Get attribute value* |
| *Setter* | *Void* | *Public* | *Set value of attribute* |

### 4.3 Interaction Diagram

#### 4.3.x Tên Interaction Diagram

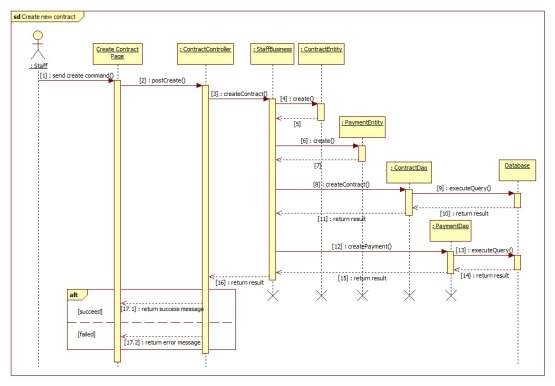
<Sử dụng **sequence diagram** **là chủ yếu để trình bày nội này**. Sequence diagram cần kết hợp giữa các class đã trình bày ở trên kết hợp với các kiến trúc đã được thuyết minh để có mô hình phù hợp. Đối với ứng **dụng điện thoại di động thì nên sử dụng activity diagram**>

**Summary:** <Nên có phần tóm tắt trước diagram để trình bày về mục đích của diagram trước khi thể hiện hình vẽ>.

*Ví dụ*

##### *4.3.1.1 Create new contract*

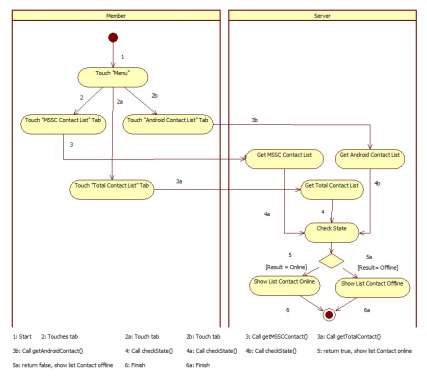
*Summary: this diagram show process of staff creates new contract*



***Figure 12 Sequence diagram - <Staff> Create new contract***

##### *4.3.1.2 <Member> View Friend List*

***Summary:*** *This diagram shows how member views all contacts that include MSSC contacts and android cell phone contacts.*



***Figure 13: <Member> View Friend List***

## 5. Interface

### 5.1 Component interface

<Mô tả các interface như của web service hay các signature của core flow được sử dụng trong hệ thống>

Nội dung được đặc tả theo dạng bảng như sau

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Signature | Description | Input | Output | Output Format | Exception |
| Tên hàm | Mô tả mục đích | Tham số truyền | Kết xuất khi hàm xử lý xong | Kiểu dữ liệu | Xử lý lỗi |

*Ví dụ*

##### *Web Service Interface*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Signature* | *Description* | *Input* | *Output* | *Output Format* | *Exception* |
| *public ResponseObject getCheckConnection(R r)* | *Check server status* | *Request object r* | *Json Boolean the status of server* | *Boolean* | *JsonProcessi ngException* |
| *...* |  |  |  |  |  |

### 5.2 User Interface Design

<Chụp và mô tả màn hình>.

**Lưu ý phải đánh số đặc tả các control trên giao diện cùng với các thành phần trong ràng buộc**

*Ví dụ*

### 5.3 Guest Interface Design

#### 5.3.1 Login



***Figure 14: Login***

Fields

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ***No*** | ***Field***  ***Name*** | ***Description*** | ***Read only*** | ***Mandatory*** | ***Control Type*** | ***Data***  ***Type*** | ***Length*** |
| *1* | *Username* | *Fill user name* | *No* | *Yes* | *Textbox* | *String* | *N/A* |
| *2* | *Password* | *Fill password* | *No* | *Yes* | *Password* | *String* | *N/A* |

Buttons/Hyperlinks

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***No*** | ***Function*** | ***Description*** | ***Validation*** | ***Outcome*** |
| *3* | *Signin* | *Log-in into the system* | *N/A* | *Transfer to home page* |

## 6. Database Design

### 6.1 Entity relationship diagram (ERD)

### 6.2 Data Dictionary

<Mô tả về các thực thể>

|  |  |
| --- | --- |
|  | **Entity Data dictionary: describe content of all entities** |
| **Entity Name** | **Description** |
|  |  |

<Mô tả các thành phần bên trong thực thể>

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Entity name** | **Attributes** | **Description** | **Domain** | **Null** |
| Tên | Thuộc tính 1 {PK} | Mô tả | Kiểu dữ liệu | Y/N |
| ... | ... | ... | ... |

**Table 12: Detail Data Dictionary**

\* Business integrity constraint:

<Mô tả các ràng buộc về toàn vẹn dữ liệu để đảm bảo nghiệp vụ>

## 7. Algorithms

<Các thành phần thuật toán - các giải pháp để giải quyết phần core flow mà nhóm đã áp dụng>

**Chú ý**

* Không nhất thiết phải là thuật toán nổi tiếng mà có thể là cách tổ chức dữ liệu cũng như giải thuật do nhóm đang thực hiện ở bên trong hệ thống: ghi rõ bản chất, phân tích về độ phức tạp, nếu tham khảo phải ghi rõ nguồn
* Cách giải quyết hay cách áp dụng các qui trình nghiệp vụ hay cách chuyển đổi bài toán khi làm bằng tay - chưa áp dụng máy tính và chương trình để cho thấy việc áp dụng giải bài toán hay giải quyết vấn đề rồi chuyển đổi cách đó sang thành chương trình máy tính *Ví dụ*

### 7.1 Document Breakdown

##### 7.1.1 Definition

*Document breakdown is the way to break the document into many small parts. Each part has it own title and contents of it. And the final data has tree structure.*

##### 7.1.2 Define Problem

*All content of document is quite difficute for manage so we must re-construc structure of document for using.*

##### 7.1.3 Solution

*To solve this problem, we should follow these steps:*

* *Convert (save) document DOCX file as html type by using Microsoft Word save as Web Filtered.*
* *Import both html file and directory that incluses all pictures of document.*
* *Using xpath to get data of html file as we need, include h1, h2, h3,…, image, text content,..*
* *Save them with structure as below: -TitleA: contentA*

*---TitleA1: contentA1*

*------TitleA1.1: contentA1.1*

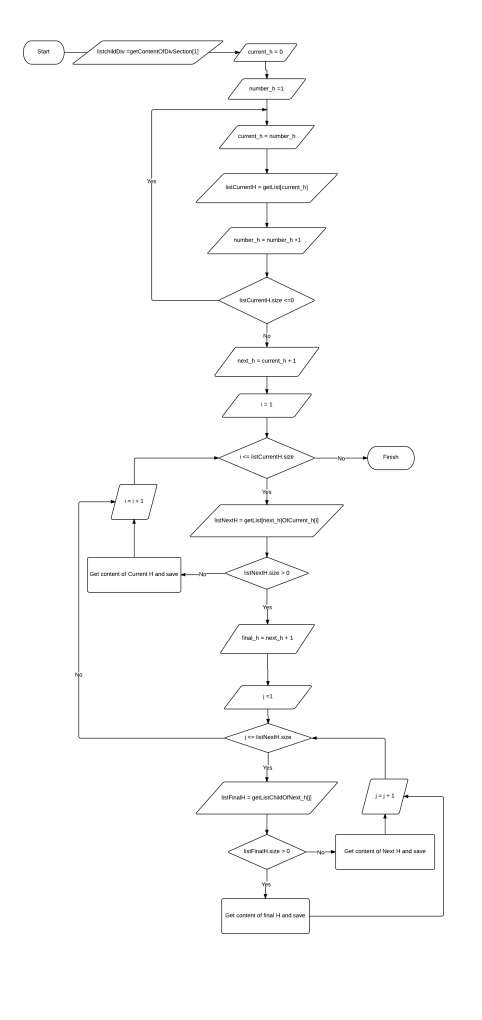
*------TitleA1.2: contentA1.2*

*---TitleA2: contentA2*

##### 7.1.4 Complexity

- *In total, the complexity of this algorithm is* 

##### 7.1.5 Flowchart



**Figure 15: Breakdown document flow chart**

#### *7.2 String Comparison*

***7.2.1 Define Problem***

*Given two strings. Calculate their matching percent.*

##### *7.2.2 Requirement*

* *Robustness to changes of word order: two strings which contain the same words, but in a different order, should be recognised as being similar.*
* *Language independence: the algorithm should work not only in English, but in many different languages.*

##### *7.2.3 Solution*

* *If a string contains many words, break it into a list of words.*
* *For each word, we find out how many adjacent character pairs are contained in it.*
* *Create a function pairs(s) which returns a list of adjacent character pairs of string s.*
* *Then, we use below formula to calculate matching percent.*



##### *7.2.4 Example*

*Calculate the matching percent of 2 strings: France and French.*

* *Upper case 2 strings:*

+ *France FRANCE.*



+ *French FRENCH.*

* *Break string into list of adjacent character pairs:*

+ *FRANCE*



+ *FRENCH*

* *Calculate its matching percent.*



# E. System Implementation & Test

## 1. Introduction

### 1.1 Overview

<Mô tả tống quát mục đích test chủ yếu với thời gian và scope và số lượng nhân lực thì nhóm áp dụng phương pháp gì cho việc test>

*Ví dụ*

*This section provides in detail all necessary information about implementation information and testing procedure of MSSC includes test plans, test cases, test result and risks estimations.*

**1.2 Test Approach**

<Phương pháp kiểm thử của nhóm : black box, white box ...>

**2. Database Relationship Diagram**

#### 2.1 Physical Diagram

<Vẽ database khi cài đặt vật lý trên các RDBMS: chú ý bố cục cũng nhu kích thước cho dễ đọc>

#### 2.2 Data Dictionary

<Mô tả thành phần theo bảng biểu bên dưới>

|  |  |
| --- | --- |
| **Data dictionary: describe content of all tables** | |
| **Table Name** | **Description** |
| Tên | Explanation |

<Mô tả thành phần chi tiết>

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Entity name** | **Attributes** | **Description** | **Domain** | **Null** |
| Tên | Thuộc tính 1 {PK} | Mô tả | Kiểu dữ liệu | Y/N |
| ... | ... | ... | ... |

**Table 13: Attribute Data Dictionary**

**3. Performance Measures**

<Cách nhóm ước lượng việc đo đạc hệ thống>

*Ví dụ*

#### *3.1 Clustering Performance*

 *Clustering is performed by running K Mean Algorithm which has complexity of : O(n \* k \* I \* d)*

o *n : number of points* o *k : number of cluster* o *I : number of iteration* o *d : number of attributes (3) Clustering take almost the time of process that we can ignore the time needed to load data from database, digitalize data.*

*The speed of clustering will vary and increase dramatically when n increase. The purpose of this project is not about optimizing K-Mean Algorithm so it is accepted to let the process run till it completes. Moreover, the clustering is designed to run by staff, wait time is acceptable.*

1. **Test Plan**

<Đưa ra kế hoạch test>

*Ví dụ*

*The purpose of this section is to verify and ensure that MSSC meets its design specification and other requirements from user. The following part will describe which features to be tested and which will not.*

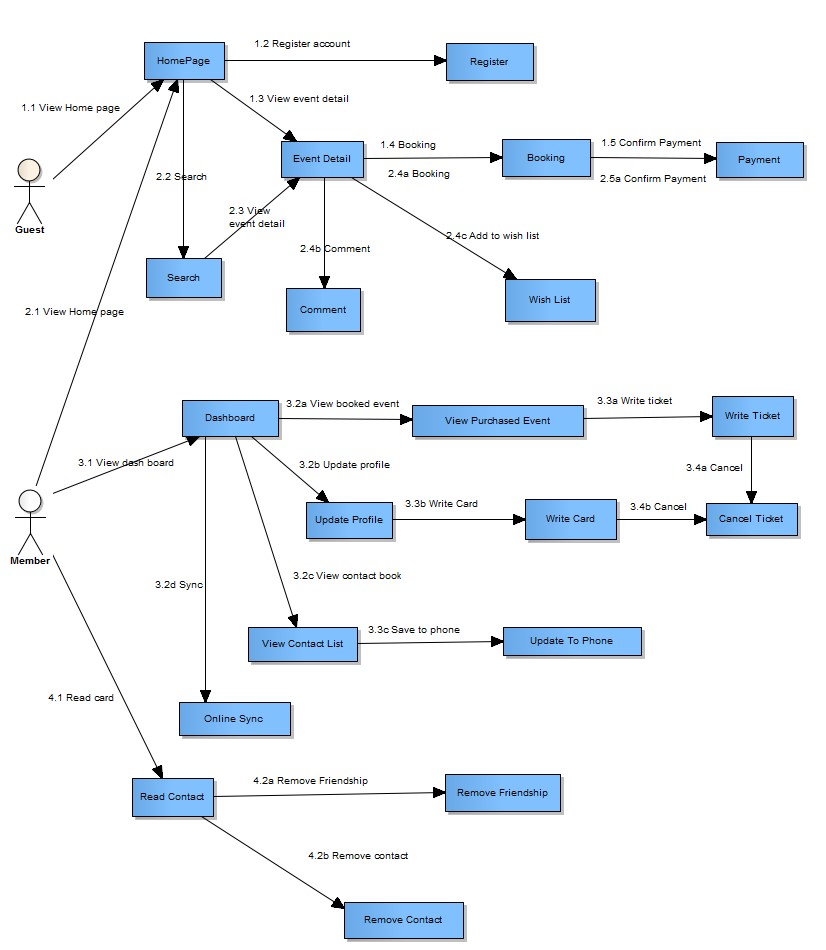
**4.1 Features to be tested** <Tính năng sẽ kiểm thử> **4.2 Features not to be tested**

<Tính năng sẽ không kiểm thử>

1. **System Testing Test Case**

**<Nên vẽ các workflow tính năng sẽ test để dể hình dung, chú ý dàn trang in ngang, chú ý đánh số, ngày tháng, kết quả, không sao chép>**

*Ví dụ*



#### *Figure 16: Guest, Member Core Flow*

MSSC - Introduction

***5.1 Guest Test Case***

***5.1.1 Search Event***

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ***ID*** | ***Test Case Description*** | ***Test Case Procedure*** | ***Expected output*** | ***Inter-test Case Dependence*** | ***Result*** | ***Test Date*** | ***Note*** |
|  |  |  |  |  |  |  |  |

MSSC - Introduction

**F. Software User’s Manual**

**1. Installation Guide**

#### 1.1 Setting up environment at server side

The following software must be installed into the server machine:

##### 1.1.1 Hardware requirements

<Yêu cầu phần cứng server, chú ý xem lại các report trước để nhất quán>

##### 1.1.2 Software requirements

<Yêu cầu phần mềm server, chú ý xem lại các report trước để nhất quán>

#### 1.2 Deployment at server side

<Mô tả quá trình triển khai lên server thực tế, gợi ý có thể gồm các bước sau, chú ý khi làm phải chụp hình cụ thể để hướng dẫn cũng như so sánh kết quả thành công>

**1.2.1 Prepare deployment package**

**1.2.2 Configure Server before deploy**

**1.2.3 Deploy web application on server**

|  |  |  |
| --- | --- | --- |
|  |  | 47 |

**1.3 Setting up the environment at client side 1.3.1 Setting up for computer**

<Ghi rõ phiên bản tối thiểu để sử dụng>

**2. User Guide**

<Viết hướng dẫn sử dụng cho người dùng>

**G. Appendix**

<Các thành phần tham khảo của tài liệu chú ý tham khảo thêm cách ghi tại [http://www.khoahocviet.info/meresci/vi/meresci03d4.html>](http://www.khoahocviet.info/meresci/vi/meresci03d4.html)