|  |  |
| --- | --- |
| logo ngan.png | **MINISTRY OF EDUCATION AND TRAINING** |

|  |
| --- |
| **FPT UNIVERSITY** |
| Capstone Project Document |
| Office Rental Service |
|  |
| |  |  | | --- | --- | | **Group 6** | | | **Group Members** | Lê Xuân Tiến – Team Leader – SE60897  Nguyễn Vũ Hoàng Quốc – Team Member – SE61112  Trương Tiến Thành – Team Member – SE61052  Trần Lê Tuấn – Team Member – 60350 | | **Supervisor** | Mr. Nguyễn Trọng Tài | | **Ext Supervisor** | N/A | | **Capstone Project Code** | ORS | |
| - Hồ Chí Minh City, May 2015 - |

*This page is intentionally left blank*

Table of Contents

[A. Introduction 8](#_Toc419298696)

[1. Project Information 8](#_Toc419298697)

[2. Introduction 8](#_Toc419298698)

[3. Current Situation 8](#_Toc419298699)

[4. Problem Definition 8](#_Toc419298700)

[5. Proposed Solution 8](#_Toc419298701)

[5.1 Feature functions 8](#_Toc419298702)

[5.2 Advantage and disadvantage 8](#_Toc419298703)

[6. Functional Requirements 8](#_Toc419298704)

[7. Role and Responsibility 9](#_Toc419298705)

[B. Project Management Plan (PMP) 10](#_Toc419298706)

[1. Problem Definition 10](#_Toc419298709)

[1.1 Name of this Capstone Project 10](#_Toc419298713)

[1.2 Problem Abstract 10](#_Toc419298714)

[1.3 Project Overview 10](#_Toc419298715)

[2. Project organization 11](#_Toc419298716)

[2.1 Software Process Model 11](#_Toc419298719)

[2.2 Roles and Responsibilities 12](#_Toc419298720)

[2.3 Tools and Techniques 13](#_Toc419298721)

[3. Project management plan 13](#_Toc419298722)

[2.3.1 Tasks 13](#_Toc419298724)

[2.3.2 Task sheet 16](#_Toc419298725)

[2.3.3 All Meeting Minutes 16](#_Toc419298726)

[4. Coding Convention 16](#_Toc419298727)

[C. Software Requirements Specifications (SRS) 17](#_Toc419298728)

[1. User Requirement Specification 17](#_Toc419298729)

[1.1 Guest requirement 17](#_Toc419298730)

[2. System Requirement Specification 17](#_Toc419298731)

[2.1 External Interface Requirements 17](#_Toc419298732)

[2.2 System Overview Use Case: 18](#_Toc419298733)

[2.1 System Overview Use Case: 19](#_Toc419298734)

[3. Software System Attribute 21](#_Toc419298735)

[3.1 Reliability 21](#_Toc419298736)

[3.2 Availability 21](#_Toc419298737)

[3.3 Security 21](#_Toc419298738)

[3.4 Maintainability 21](#_Toc419298739)

[3.5 Portability 22](#_Toc419298740)

[3.6 Performance 22](#_Toc419298741)

[4. Conceptual Diagram 22](#_Toc419298742)

[5. Other material (if any) 24](#_Toc419298743)

[D. Software Design Description (SDD) 25](#_Toc419298744)

[1. Design Overview 25](#_Toc419298745)

[2. System Architectural Design 26](#_Toc419298746)

[2.1 Choice of System Architecture 26](#_Toc419298747)

[3. Component Diagram 26](#_Toc419298748)

[4. Detailed Description Explanation 26](#_Toc419298749)

[4.1 Class Diagram 26](#_Toc419298750)

[4.2 Class Diagram Explanation 28](#_Toc419298751)

[4.3 Interaction Diagram 28](#_Toc419298752)

[5. User Interface Design 29](#_Toc419298753)

[5.1 Login 29](#_Toc419298754)

[6. Database Design 31](#_Toc419298755)

[6.1 Logical Diagram 31](#_Toc419298756)

[6.2 Data Dictionary 33](#_Toc419298757)

[7. Algorithms 34](#_Toc419298758)

[7.1 Suggest products which rented by customers who rented current product 34](#_Toc419298759)

[E. System Implementation & Test (SIT) 36](#_Toc419298760)

[1. Introduction 36](#_Toc419298761)

[1.1 System Overview 36](#_Toc419298762)

[1.2 Test Approach 36](#_Toc419298763)

[2. Database Relationship Diagram 36](#_Toc419298764)

[2.1 Physical Diagram 36](#_Toc419298765)

[2.2 Data Dictionary 36](#_Toc419298766)

[3. Performance Measures 36](#_Toc419298767)

[3.1 Clustering Performance 36](#_Toc419298768)

[4. Test Plan 36](#_Toc419298769)

[4.1 Features to be tested 36](#_Toc419298770)

[4.2 Features not to be tested 36](#_Toc419298771)

[5. System Testing Test Case 37](#_Toc419298772)

[5.1 Guest Test Case 37](#_Toc419298773)

[6. Other material 38](#_Toc419298774)

[F. Software User’s Manual 39](#_Toc419298775)

[1. Installation Guide 39](#_Toc419298776)

[1.1 Setting up environment at server side 39](#_Toc419298777)

[1.2 Deployment at server side 39](#_Toc419298778)

[1.3 Client side environment setting 39](#_Toc419298779)

[2. User’s Guide 39](#_Toc419298780)

[2.1 Admin –Staff Login 40](#_Toc419298781)

[G. Appendix 40](#_Toc419298782)

**List of Figures**

[**Figure 1: Waterfall Model** 14](#_Toc419298675)

[*Figure 2: Use Case Overview* 20](#_Toc419298676)

[*Figure 3: <Guest>Overview Use Case* 21](#_Toc419298677)

[*Figure 4: <Guest> Register* 21](#_Toc419298678)

[*Figure 68: Entity Relationship Diagram* 25](#_Toc419298679)

[*Figure 71: Component Diagram* 28](#_Toc419298680)

[*Figure 72: Class Diagram* 29](#_Toc419298681)

[*Figure 73: Add Exchangeable Toy Sequence Diagram* 31](#_Toc419298682)

[*Figure 106: Login* 31](#_Toc419298683)

[**Figure 157: Physical Database Diagram** 34](#_Toc419298684)

[**Figure 158: Extract Deployment Package** 41](#_Toc419298685)

[**Figure 12: Admin - Staff Login Page** 42](#_Toc419298686)

**List of Tables**

[Table 1: Roles and Responsibilities 9](#_Toc419298687)

[Table 2: Hardware Requirement for Server 11](#_Toc419298688)

[Table 3: Hardware Requirement for Web User 11](#_Toc419298689)

[*Table 4: <Staff>Cancel Exchange* 21](#_Toc419298690)

[*Table 54: Entity Data Dictionary* 24](#_Toc419298691)

[*Table 55: Entity Data Dictionary* 33](#_Toc419298692)

[*Table 56: Data Dictionary of Role* 34](#_Toc419298693)

[*Table 77: Basic Search Test\_Case* 37](#_Toc419298694)

[**Table 9: Login Step** 40](#_Toc419298695)

**Definitions, Acronyms, and Abbreviations**

|  |  |
| --- | --- |
| **Name** | **Definition** |
| ORS | Office Rental Service |
| SRS | Software Requirement Specification |
| Admin | Administrator |
| SMS | Short message service |
| Info | Information |

# Report No 1 - Introduction

## Project Information

* Project name: **Office Rental Service**
* Project Code: **ORS**
* Product Type: **Website Application**
* Start Date: **May 11th, 2015**
* End Date: **August 22nd, 2015**

## Introduction

Nowadays, a lot of companies are established every day. When new companies are established or some companies want to expand their companies, they need to find where they put their offices. To find a suitable office for them, they need you to add some equipment they need, repair some equipment… All of that thing will get you a lot of time to do it and it is very complex to make contract with those companies. With this system, you can easily to manage your offices, equipment and contracts and it takes you less time than now.

## Current Situation

In Vietnam, to find a suitable office, company would rather consider the following actions:

* If company has large resource, and require a specific requirement and equipment, they would build a new office. This will takes lots of time and money, only adapted for foreign companies.
* Go to a broker company, fill in the form and ask for help to find a suitable office. This usually takes some days to find and cost fee for broker companies.
* Search the internet for office. There are some website in Vietnam has information about the office for rent. But the information is usually not richly or sometimes outdated.

## Problem Definition

To build a new office requires a lot of time, effort, and spend a lot of costs. Besides, there are many new office building has been built recently, but had not been rented, and many company has unused office space. The matching between the demand and the supply is critical required. But using tradition approach for searching office, we have some limitations:

* There are some classifieds website that put the office information. However, the office information is lack, the search criteria is not very effective, and the information about the amenities is hardly found.
* Every company choose for themselves the best office suite with their company: location, area, office facilities, finding a best match will be take time, costs. This work usually made by a third party broker company, and it will take days to finish and we would pay a fee to that company.
* When company want to rent the office or have a rented office but conditions where infrastructure does not guarantee, the company business will be very difficult to contact the lessors to consider repairing office suite company conditions, causing company to consume resources.

## Proposed Solution

The website support customer to select appropriate and process office rental online. The website also support manager to track their business.

### Feature functions

* Our website offers many methods for customer to search office (by size, price, place, amenities…). When they find approtiate office, our staff will arrange to meet them at that office. After check around, we can make a contract right away. The contract will be tracked by our website for further support.
* If customers request repair something in their office, the website will notify our staff to make sure they will fix that problem as soon as posible.
* Our system also supports the addition of devices such as power sockets, fax machines… at the request of customers.

### Advantage and disadvantage

The advantages and disadvantages of the proposed solution:

* Advantages:
  + User friendly and specialized interface.
  + Providing search engines criteria: price, location, amenities...
  + Providing full information about the office. Support map for customers
  + Easy to make an appointment
  + When a deal is made, the contract is tracked by website, so the office state is update in real time for further search.
  + Can request for repair, or add new equipment based on the contract.
  + The repair request status will be update by notification system
  + The repair request is easily manage by automatic assign system
  + The contract making and repair status update is manage by mobile system so the staff will easy to update when go to the offices.
* Disadvantages:
  + The partner who own the office must contact us to post their information.
  + The mobile system require internet connection.
  + The function of notification system is limited.
  + The matching solution based on location and needs will take more time than search by keyword in traditional approach

## Functional Requirements

The functional requirements of the system are based on four main actor as below:

* Guess
  + Search for office base on places, category, amenities
  + View office detail
  + Register information to make an appointment
  + Login
* Customer:
  + Review office
  + Request rental appointment
  + Request repair
* Admin:
  + View list/Create/Update/Delete Manager.
  + View list/Create/Update/Delete Staff.
* Manager:
  + Approve/Cancel/Delete contract
  + Assign repair
  + View Statistic
* Staff:
  + Create contract
  + View list/Create/Update/Delete office
  + Approve/Delete rental
  + Update status request repair.
  + Approve/Delete review

## Role and Responsibility

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Full Name** | **Role** | **Position** | **Contact** |
| 1 | Nguyễn Trọng Tài | Project Manager | Instructor | [taint@fpt.edu.vn](mailto:taint@fpt.edu.vn) |
| 2 | Lê Xuân Tiến | Developer | Team Leader | [tienlxse60897@fpt.edu.vn](mailto:tienlxse60897@fpt.edu.vn) |
| 3 | Nguyễn Vũ Hoàng Quốc | Developer | Team Member | [quocnvhse61112@fpt.edu.vn](mailto:quocnvhse61112@fpt.edu.vn) |
| 4 | Trương Tiến Thành | Developer | Team Member | [thanhttse61052@fpt.edu.vn](mailto:thanhttse61052@fpt.edu.vn) |
| 5 | Trần Lê Tuấn | Developer | Team Member | [tuantl60350@fpt.edu.vn](mailto:tuantl60350@fpt.edu.vn) |

Table 1: Roles and Responsibilities

# Project Management Plan (PMP)



## Problem Definition



### Name of this Capstone Project

* **Official name**: Office Rental Service
* **Vietnamese name:** Dịch vụ cho thuê văn phòng
* **Abbreviation:** ORS

### Problem Abstract

Almost medium and small companies don’t build a new office. They usually look up on internet to rent office. But they can’t find a suitable office for them. Or the price to hire a third party broker company is too high. Besides, the office they rent doesn’t provide the necessary equipment and the time to repair is too long. So, our system will solve that problem. It not only helps you find a suitable office but also has other useful services.

### Project Overview

#### Current Situation

There are some current websites such as vanphongthue.com.vn, rongbay.com, timvanphong.vn, etc. All of them have some advantages and disadvantages.

* Advantages:
  + Friendly and specialized interface.
  + Providing search engines criteria.
  + Direct consultation with the manager.
* Disadvantages:
  + Only supply direct meeting, not arrange for a meeting.
  + Not regularly updated information.
  + Not support map.
  + Few pictures describing office.
  + Not provide equipment.

#### The Proposed System

The system will help customers find suitable office. If you don’t find a suitable office, you can make a request. When our system find some office that meet your requirement, it will send email for you. If you need repair office, our system will send staff to fix that problem as soon as possible. Moreover, our system also provide rental equipment for you.

In more detail, the system would contain following features:

##### Website

* Guess can search office and view detail. If they want to make appointment, they need to register.
* Customer can review office, request appointment, request office (if they can’t find ones), request rental equipment and request repair.
* Admin can manage staff and manager.
* Manager can create and delete contract. They also can assign staff to repair for customer and view statistic of system.
* Staff can view list, create, update and delete office. When staff finish repair for customer, they can report to manager. They also can approve or delete rental equipment.

##### Mobile Application

* Staff can check appointment task and update status request repair.

#### Boundaries of the System

* The system is intended for customers who want to rent office (such as businessman, medium or small company,…).
* The language of the system is Vietnamese
* The complete product includes:
  + The website, for admin, staff, customers and guest to interactive with the system.
  + Mobile application for staff to check task and update status request repair.
  + All the process involved document.

#### Development Environment

* **Hardware requirement for server computer**

|  |  |  |
| --- | --- | --- |
| Windows | Minimum Requirements | Recommended |
| Internet Connection | Cable, Wi-Fi (2Mbps) | Cable, Wi-Fi (8 Mbps) |
| Operating System | Window server 2008 | Window server 2012 |
| Computer Processor | Intel® Celeron® Processor 2957U (1.40 GHz) | Intel® Core™ i3-3210 Processor |
| Computer Memory | 1GB RAM | 4GB or more |

Table 2: Hardware Requirement for Server

* **Hardware Requirement for Web User**

|  |  |  |
| --- | --- | --- |
| **Web** | **Minimum Requirements** | **Recommended** |
| **Internet Connection** | 2Mbps | 4Mbps |
| **Web Browser** | Firefox, Chrome, IE 8 | Firefox, Chrome, IE 9 |

Table 3: Hardware Requirement for Web User

All computers must be connected to the Internet.

* **Hardware requirement for mobile app**

|  |  |  |
| --- | --- | --- |
| Windows | Minimum Requirements | Recommended |
| Internet Connection | 2Mbps | 4 Mbps |
| Operating System | Android 4.0 | Android 4.4 |
| Hardware |  |  |
| Memory | 10MB | 10MB |

Table 4: Requirement for mobile app

* **Software requirements**
  + Operating system: Windows 7, or above;
  + Framework: Hibernate with Java Persistence;
  + Modeling Tool: Visual Paradigm Community;
  + IDE: Intellji Idea;
  + DBMS: Microsoft SQL Server;
  + Source Control: GitHub, Git for windows, Tortoise Git.

## Project organization



### Software Process Model

We choose waterfall model.

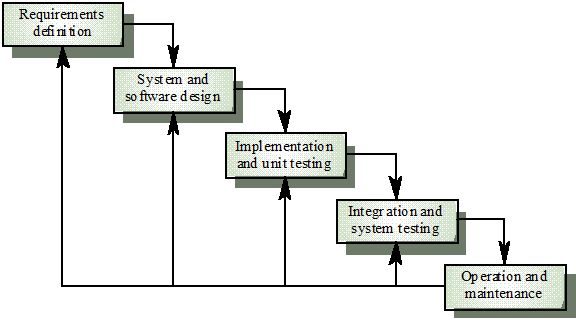


Figure 1: Software process model

This model is easy to manage and understand. For our project, we don’t have a lot of time so we use this model to help us release our project on time. This model uses for short project and it suitable for our project, which is small with 4 months and requirements are easy to clear.

### Roles and Responsibilities

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Full name** | **Role in Group** | **Responsibilities** |
| **1** | Nguyễn Trọng Tài | Supervisor/ Project manager | * Specify user requirement. * Control the development process. * Support technical and business analysis. * Review document and product application. |
| **2** | Lê Xuân Tiến | Team Leader, Developer, Tester | * Monitor process * Create project plan and distribute tasks * Clarify requirements. * Design database. * Prepare documents. * GUI Design. * Create code guide and form. * Coding. * Testing. * Deploy final product. |
| **3** | Nguyễn Vũ Hoàng Quốc | Developer, Tester | * Clarify requirements. * Prepare documents. * Review Database. * GUI Design. * Create test cases. * Coding. * Testing. |
| **4** | Trương Tiến Thành | Developer, Tester | * Clarify requirements. * Prepare documents. * Review Database. * GUI Design. * Create test cases. * Coding. * Testing. |
| **5** | Trần Lê Tuấn | Developer, Tester | * Clarify requirements. * Prepare documents. * Review Database. * GUI Design. * Create test cases. * Coding. * Testing. |

Table 5: Role and Responsibilities

### Tools and Techniques

* *Front-end*: CSS3, HTML5, JavaScript, JQuery, Ajax, AngularJS, Java Android, PhoneGap.
* *Back-end*: Java 1.7, Hibernate, Java Persistence.
* *Web Server*: Tomcat 7.
* *Developing Tool*: JetBrains IntelliJ IDEA 14, Eclipse.
* *Database Management System:* Microsoft SQL Server 2008.
* *Source Control:* Git-1.9.5-preview, TortoiseGit-1.8.14.0.
* *Modeling Tool*: Visual Paradigm 12.0 Commnunity.
* *Document Tool*: Microsoft Office 2013.

## Project management plan



### Software development life cycle

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Phase** | **Description** | **Deliverables** | **Resource**  **needed** | **Dependencies**  **and Constrains** | **Risks** |
| **Requirement**  **Analysis** | - Collect requirements.  - Analyze requirements  -Identify and clarify requirements for the system in general. | -Introduction of proposed system. (report 1)  -Project Task Plan. (report 2)  -Software requirement specification. (report 3) | 40 man-days | N/A | - Missing requirement  - Unclear scope of project  - Lack of member share  - Time resource not well managed |
| **Design** | - Design database  - Architecture design for the system  - Detail design using top-down break down  - Choose Architecture style | - Software Design Document  - Base code structure  - Architecture notes  (report 4) | 74 man-days | Depend on “Requirement Analysis” | - Lack of experience.  - Not fulfil requirement. |
| **Implement** | - Implement all system functions | - Source code  - Final deploy packet | 120 man-days | - Software requirement specification.  - Software Design Document | - Lack of member experience.  - Different of member skill |
| **Testing** | - Create test plan  - Create test cases  - Perform unit test cases and system integration test | - Test plan  - Test cases  - Test report  (report 5) | 80 man-days | - Coding is finished  - Based on SRS | - Lack of experience leads to lack of test cases  - Time schedule |
| **Operation and maintenance** | - Create user’s manual, includes server deployment and end-user manual | - Report 6: System User’s Manual | 20 man-days | - Testing is finished  - Based on SRS | - Lack of experience can lead to hard-to-understand document for user  - Time deadline |

Table 6: Software development lifecycle

### Phase Detail

Below are all the major tasks that need to be performed sequentially during the development of the system.

#### Phase 1: Requirement Analysis

|  |  |  |
| --- | --- | --- |
| **Task** | **Description** | **Author** |
| **1. Collect and analyze requirements** | Find the traditional approach for the service, and current IS system for office rental services. Analyze their strengths and weaknesses | TienLX, QuocNVH, ThanhTT, TuanTL |
| **2. Identify and clarify main functions.** | Define main flows which system will build to adapt, and main functions for system. | TienLX, QuocNVH, ThanhTT, TuanTL |
| **3. Create project management plan** | Define the overview of the project  Create project plan | TienLX, QuocNVH |

Table 7: Phase 1: Requirement Analysis

#### Phase 2: Design

|  |  |  |
| --- | --- | --- |
| **Task** | **Description** | **Author** |
| **1. Defined overall system design** | Design basis system architecture design | TienLX |
| **2. Create entity-relationship diagram** | From the defined use cases, create the ERD for the system | TienLX, QuocNVH |
| **3. Create logical database diagram** | Define the logical database diagram based on ERD | ThanhTT, TuanTL |
| **4. Create class diagrams and definition** | Create the classes with attributes and functions, along with its definition | TienLX, QuocNVH, ThanhTT, TuanTL |
| **5. Create interaction diagrams** | Include sequences diagrams and activity diagrams | TienLX, QuocNVH, ThanhTT, TuanTL |
| **6. Create interface design** | Design the user interface for the system applications | TienLX, QuocNVH, ThanhTT, TuanTL |
| **7. Define main algorithms** | Analyze and define the required algorithms for the system | TienLX, QuocNVH, ThanhTT, TuanTL |

Table 8: Phase 2: Design

#### Phase 3: Implement

|  |  |  |
| --- | --- | --- |
| **Task** | **Description** | **Author** |
| **1. Design physical database** | Define the physical database design on SQL Server, and create database dictionary | TienLX, QuocNVH, ThanhTT, TuanTL |
| **2. Define base structure** | Define the base structure for the program | TienLX |
| **3. Coding admin functions** | Include manage account | TuanTL |
| **4. Coding manager functions** | Include manage repair list and manage contract list | ThanhTT, QuocNVH |
| **5. Coding staff functions** | Include manage office | TienLX |
| **6. Coding customer functions** |  |  |
| **6.1. Coding request office flow** | Include search office, request office, request appointment | BE: QuocNVH  FE: TienLX |
| **6.2. Coding contract signing flow** | Include the assign flow for manager and view assigned flow for staff | ThanhTT |
| **6.3. Coding contract extend/cancel flow** | Include request extend contract and cancel contract | QuocNVH |
| **6.4. Coding request repair flow** | Include request repair office for customers | TuanTL |
| **7. Coding additional mobile app** | Include view assigned job for staff, view assigned and update the repair job for staff | BE: QuocNVH, TuanTL  FE: TienLX, ThanhTT |

Table 9: Phase 3: Implement

#### Phase 4: Testing

|  |  |  |
| --- | --- | --- |
| **Task** | **Description** | **Author** |
| **1. Create test plan** | Create the test plan document for the system | TienLX |
| **2. Create unit testing** | Create the test cases for each functions describe in the implementation |  |
| **2.1. For admin functions** | Create test cases for functions in manage account | TienLX |
| **2.2. For manager functions** | Create test cases for functions in manage repair and contract | TuanTL |
| **2.3. For staff functions** | Create test cases for functions in manage office | ThanhTT |
| **2.4. For customer functions** | Create test cases for functions for customer | QuocNVH |
| **2.5. For mobile app** | Create test cases for functions for mobile app | TuanTL |
| **3. Testing and fix bugs** | Base on the test cases, test the functions and fix bugs | TienLX, QuocNVH, ThanhTT, TuanTL |
| **4. System integration test** | Test the overall of the system, measure the performance of the system | TienLX, QuocNVH, ThanhTT, TuanTL |

Table 10: Phase 4: Testing

#### Phase 5: Operation and maintenance

|  |  |  |
| --- | --- | --- |
| **Task** | **Description** | **Author** |
| **1. Create installation guide** | Define the system requirement for the server and client. document installation guild | TienLX, QuocNVH |
| **2. Create User guide** | Define the user manual for operating the system | TienLX, QuocNVH, ThanhTT, TuanTL |
| **3. Maintenance and fix bugs** | Tracking the system and fix the remains bugs while installation | ThanhTT, TuanTL |

Table 11: Phase 5: Operation and maintenance

### All Meeting Minutes

Refer to Meeting Minutes folder.

## Coding Convention

**Indentation:**

* Four spaces should be used as the unit of indentation
* Tabs must be set exactly every 8 spaces
* Avoid lines longer than 80 characters
* Break after a comma.

**Declarations:**

* One declaration per line
* Put declarations only at the beginning of blocks
* No space between a method name and the parenthesis "(" starting its parameter list
* Open brace "{" appears at the end of the same line as the declaration statement
* Closing brace "}" starts a line by itself indented to match its corresponding opening statement, except when it is a null statement the "}" should appear immediately after the "{"

**White Space:**

* Between sections of a source file
* Between class and interface definitions
* Between methods

**Naming Conventions:**

* Class names should be noun
* Methods should be verbs

Reference: <http://www.oracle.com/technetwork/java/codeconvtoc-136057.html>

# Software Requirements Specifications (SRS)

## User Requirement Specification

The system has five actors including: guest, customer, staff, admin and system.

|  |  |
| --- | --- |
| **Actor** | **Description** |
| Guest | Person join to website but not login into system. |
| Customer | Person who want to rent offices and logged into system |
| Admin | Person who manage account and ban/ unban account |
| Manager | Person who manage request, contact customer, schedule staff…. |
| Staff | Person who in charge of create office, receive and process the appointment/ repair/ rental/ return office request |
| System | System will manage the schedule to process contract time, send notification, suggest office for customer |

Table 1: Overall user requirement specification

### Guest requirement

* **Register:** Guest need to register to become member of system.
* **Search office:** Customer can search suitable office by criteria such as price, location, name….
* **Login**: Guest uses email and password to login into the system to search or request office.

### Customer requirement

* **Search office:** Customer can search suitable office by criteria such as price, location, name….
* **Send request:** Customer can send request about book appointment, repair something in office, rental, and request cancel before expire…
* **Rate and comment office:** Customer can rate and comment satisfaction about office they renting.

### Admin requirement

* **Logout:** When finish all activities at website they can log out of system.
* **Manager member:** Admin can add, edit, remove or ban/ unban account.

### Manager requirement

* **Contact with customer:** Manager can notify customer when task is done.
* **Manager contract:** Manager can create, edit contract.
* **Monitor request:** Manager can confirm about request repair, rental, appointment.
* **Schedule staff:** manager can schedule staff to contact with customer when request has been approved by manager.

### Staff requirement

* **Logout:** When finish all activities at website they can log out of system.
* **Contact with customer:** staff have to check task list about the appointment with customer and change status of the appointment when finish his task.
* **Monitor office:** staff can add, edit or remove comment about office.
* **Repair amenities:** staff have to check task list about the request repair amenities and change status of that request after repair complete.

### System requirement

* **Send notification mail:** system will send notification mail to customer when appointment has been approved and scheduled or request repair has been accepted.
* **Send notification SMS (short message service):** system will send notification mail to customer when their contract will be expired in one month.
* **Suggest office:** system will suggest some offices which is nearly suitable with what customer searching.
* **Schedule request:** system will auto schedule all request about appointment and repair when those request has been approved by manager.

## System Requirement Specification

### External Interface Requirements

#### User Interfaces

* User interface must be friendly, simple.
* All functions should be showed clearly and don’t make user confuse.

#### Hardware Interfaces

* The system will use the standard hardware and data communications resources of a standard computer.

#### Software Interfaces

* The website run in Firefox and Chrome browsers.

#### Communications Protocol

* The website uses:
* HTTP/HTTPS protocol for communication between the web browser and the web server.
* TCP/IP network protocol for communication with HTTP protocol.
* WAP protocol for sending message for customer.

### System Overview Use Case:

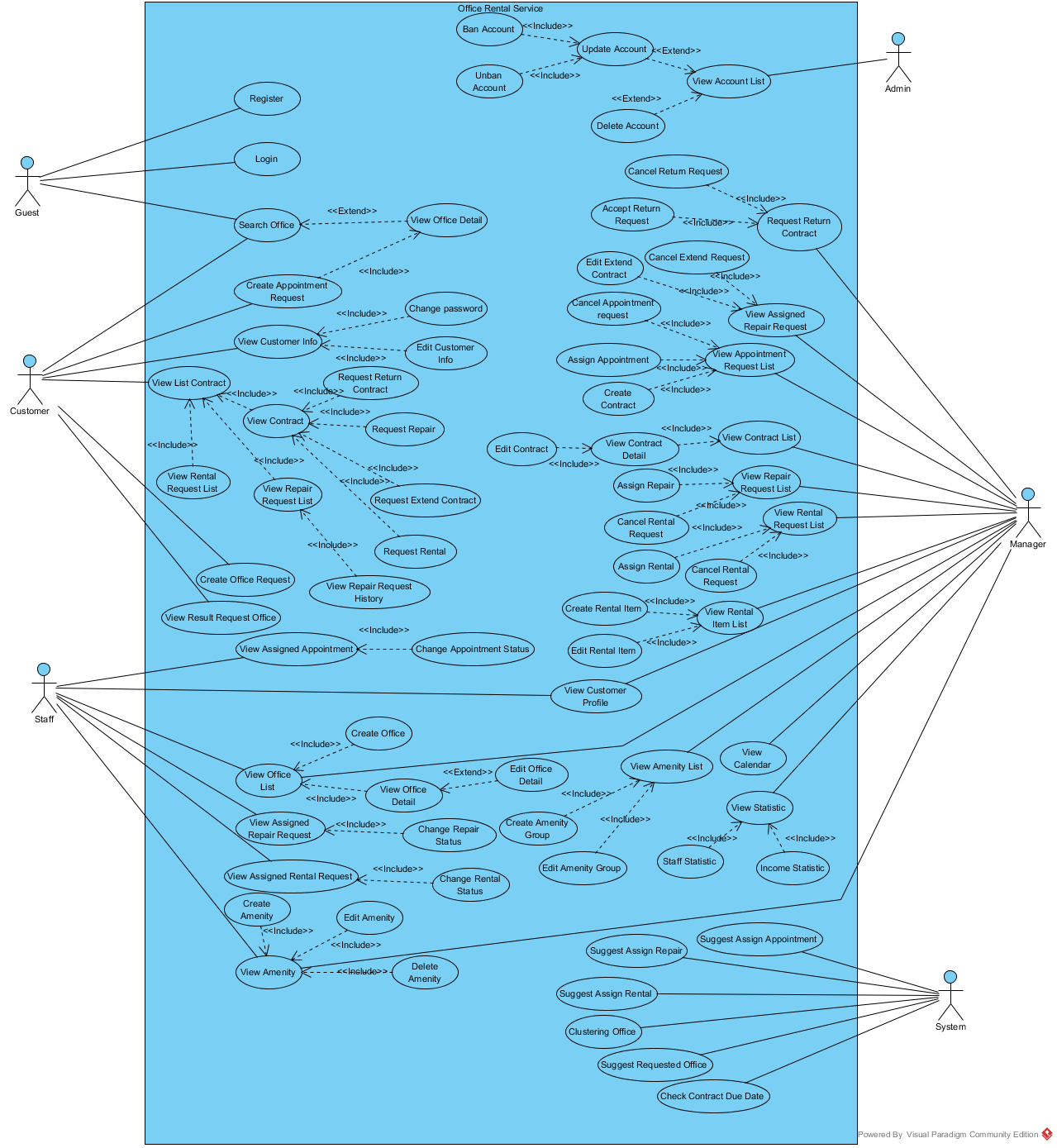


Figure 1: Use Case Overview

### List of Use Case:

## Software System Attribute

### Usability

* Staff should need less than one training week to interact with system.

### Reliability

* Information should be updated every day.

### Availability

* System uptime at about 99%

### Security

* All input data should be validated before saving to database.
* All privacy information, such as password, should be encrypted to ensure security.
* User should be authenticated and authorized when accessing to the system.

### Maintainability

* The system should be divided into modules and code. It would be easy to maintain and upgrade.

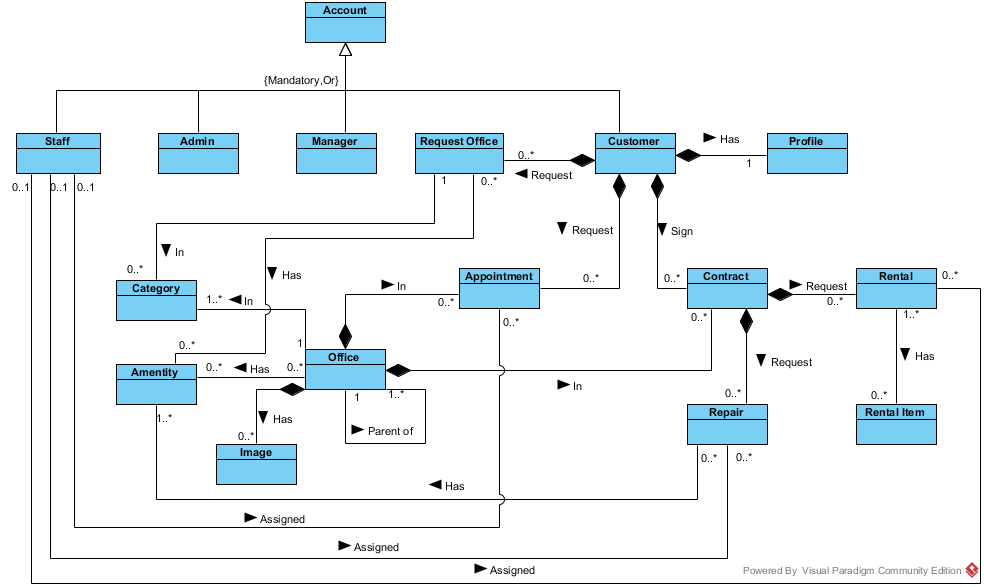
### Portability

* N/A

### Performance

* All requests should be response in no more than 1 minute.

## Conceptual Diagram



**Data dictionary:**

|  |  |
| --- | --- |
| **Entity Data dictionary: describe content of all entities** | |
| **Entity Name** | **Description** |
| Account | Describe account of user of system. |
| Staff | Describe account type staff |
| Admin | Describe account type admin |
| Manager | Describe account type manager |
| Customer | Describe account type customer |
| Profile | Describe the contact information for customer |
| Office | Describe information of the office |
| Category | Describe the category of the office |
| Amenity | Describe the amenities of the office |
| Image | List of images of the office |
| Appointment | Describe the information of the appointment of customer for the office |
| Contract | Describe the information of contract between customer and office |
| Request Office | Describe the stored office request of customer |
| Repair | Describe the repair request for each contract |
| Rental | Describe the rental request for contract |
| Rental Item | Describe the rental items which available for rental request |

# Report 4: Software Design Description

## Design Overview

This document describes the technical and user interface design of MSSC 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.

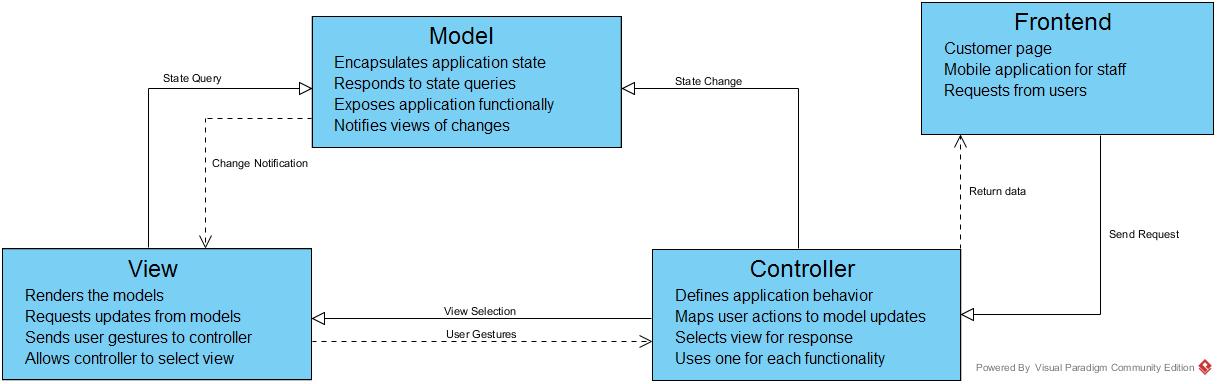
Document overview:

* 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: overview some main user interface of system.
* Section 6: describe fully attributed ERD.
* Section 7: describe in details all algorithms used in the system.

## System Architectural Design

### Choice of System Architecture

The system is developed using MVC architecture, combine with front end – back end RESTful Web service interface architecture.



## Component Diagram

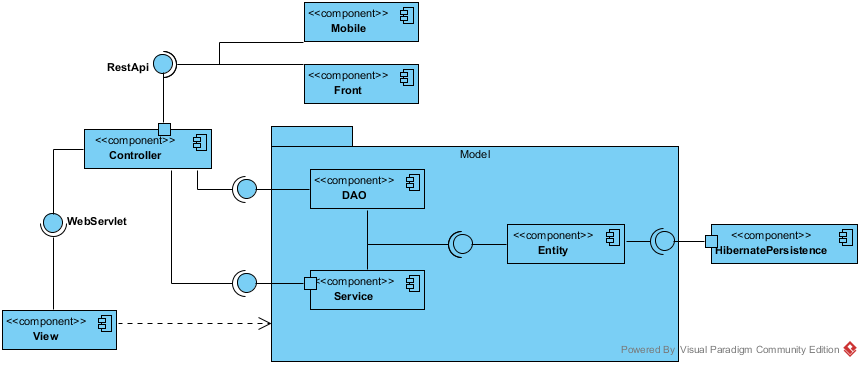


Figure 1: Component Diagram

## Detailed Description Explanation

### Class Diagram

#### Entity Package Class Diagram

Figure 2: Class Diagram

**Class Diagram Explanation**

##### Class 1

Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
|  |  |  |  |

Method

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
|  |  |  |  |

##### Class 2

Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
|  |  |  |  |

Method

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
|  |  |  |  |

#### DAO Package Class Diagram

Figure 3: Class Diagram

**Class Diagram Explanation**

##### Class 1

Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
|  |  |  |  |

Method

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
|  |  |  |  |

##### Class 2

Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
|  |  |  |  |

Method

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
|  |  |  |  |

#### Service Package Class Diagram

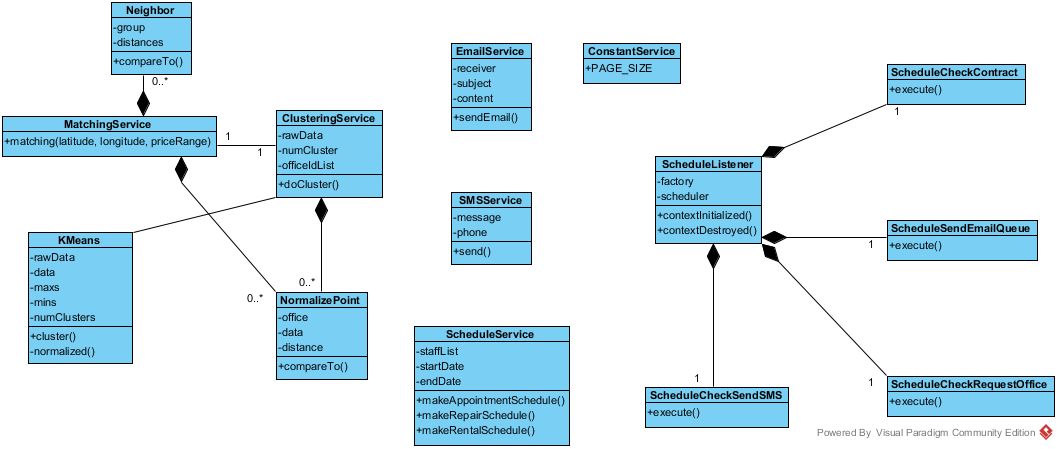


Figure 4: Service Package Class Diagram

**Class Diagram Explanation**

##### EmailService

Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
| receiver | String | private | The recipient of the email |
| subject | String | private | The subject of the email |
| content | String | private | The content of the email |

Method

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
| sendEmail | boolean | public | Send the email with the subject and content to the receiver |

##### SMSService

Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
| message | String | private | The SMS message will be sent |
| phone | String | private | The phone number of the recipient |

Method

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
| send() | String | public | Send the SMS message to the recipient. Return “SUCCESS” if sent success, otherwise return the error message form the service provider |

##### ConstantService

Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
| PAGE\_SIZE | int | public | The constant for the page size of the list in admin pages |

##### ScheduleService

Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
| staffList | List<Staff> | private | The list of all staff for the service |
| startDate | DateTime | private | The start date of the week for the schedule |
| endDate | DateTime | private | The end date of the week for the schedule |

Method

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
| makeAppointmentSchedule() | Map | public | Return the schedule hash map with appointment id and staff username for the unassigned appointment |
| makeRepairSchedule() | Map | public | Return the schedule hash map with repair id and pre-arranged repair for the unassigned repair |
| makeRentalSchedule() | Map | public | Return the schedule hash map with rental id and pre-arranged rental for the unassigned rental |

##### KMeans

Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
| rawData | double[][] | private | The raw data for k-means algorithm |
| data | double[][] | private | The normalized data for k-means algorithm |
| maxs | double[] | private | The array for max value of each dimension of k-means |
| mins | double[] | private | The array for min value of each dimension of k-means |
| numClusters | int | private | The number of cluster (k) in k-means algorithm |

Method

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
| cluster | double[] | public | Do the k-means algorithm. Return the array of group id for each of input data |
| normalized | double[][] | private | Normalize the raw data |

##### ClusteringService

Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
| rawData | double[][] | private | The input raw data of all office |
| numCluster | int | private | The calculated k-means number |
| officeIdList | int[] | private | List of all office id |

Method

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
| doCluster | int[] | public | Do the k-means clustering for all office. The result is saved to database also return the array of office group by all office |

##### NormalizePoint

Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
| office | Office | private | Use to store the office for clustering service |
| data | double[] | private | The normalize data for current office |
| distance | double | private | The distance between the data point and query point in MatchingService |

Method

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
| compareTo() | int | public | Return the compare result of 2 NormalizePoint |

##### MatchingService

Method

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
| matching() | List<Office> | public | Do the matching group for the input office point and the all office data. Return the list of matched office group sort by relevant |

##### Neighbor

Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
| group | int | private | The index of the office group |
| distances | List<Double> | private | The list of distance of query point and office in group for the KNN matching service |

Method

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
| compareTo() | int | public | Compare distance between query point and two list of data |

##### ScheduleListener

Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
| factory | SchedulerFactory | private | The factory for initiate the scheduler |
| scheduler | Scheduler | private | The class contain the instance of the scheduler factory |

Method

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
| contextInitialized() | void | public | Override context listener when the app initialized |
| contextDestroyed() | void | public | O Override context listener when the app destroyed |

##### ScheduleCheckContract

Method

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
| execute | void | public | Execute the schedule for check the contract due date. Cancel the contract if the contract at due date |

##### ScheduleCheckRequestOffice

Method

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
| execute | void | public | Execute the schedule for check the Request office. Stored the found office and queue to send an email to customer |

##### ScheduleCheckSendSMS

Method

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
| execute | void | public | Execute the schedule for check the appointment date. Send a SMS to user if the appointment date is near |

##### ScheduleCheckEmailQueue

Method

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
| execute | void | public | Execute the schedule for check the email in queue to send |

#### Controller Package Class Diagram

Figure 5: Class Diagram

**Class Diagram Explanation**

##### Class 1

Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
|  |  |  |  |

Method

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
|  |  |  |  |

##### Class 2

Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
|  |  |  |  |

Method

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
|  |  |  |  |

#### View Package Class Diagram

Figure 6: Class Diagram

**Class Diagram Explanation**

##### Class 1

Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
|  |  |  |  |

Method

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
|  |  |  |  |

##### Class 2

Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
|  |  |  |  |

Method

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
|  |  |  |  |

#### Front Package Class Diagram

Figure 7: Class Diagram

**Class Diagram Explanation**

##### Class 1

Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
|  |  |  |  |

Method

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
|  |  |  |  |

##### Class 2

Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
|  |  |  |  |

Method

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
|  |  |  |  |

#### Mobile Package Class Diagram

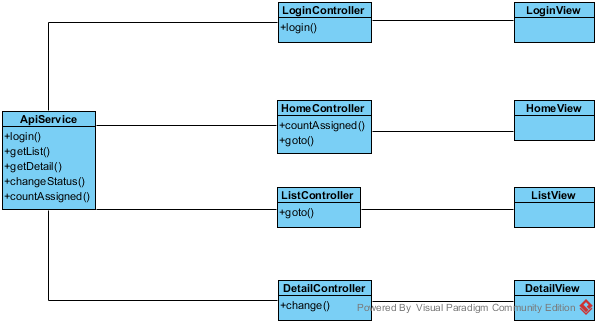


Figure 8: Mobile Package Class Diagram

**Class Diagram Explanation**

##### ApiService

Method

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
| login() | function | public | Call the login api and return the result. |
| getList() | function | public | Call the getList api to return the list of job for current staff (appointment/rental/repair) |
| getDetail() | function | public | Call the getDetail api to return the detail of current job |
| countAssigned() | function | public | Call the countAssigned api to return the number of assigned for current job |
| changeStatus() | function | public | Call the changeStatus api to change the status of current job |

##### LoginController

Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
| username | string | private | The inputted username |
| password | string | private | The inputted password |

Method

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
| login() | function | private | Call the login function of ApiService |

##### LoginView

This is the presentation view of LoginController

##### HomeController

Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
| title | string | private | The title of the job (appointment/rental/repair) |
| badge | Array | private | The array of number of assigned job |

Method

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
| countAssigned() | function | private | Call the countAssigned function of ApiService to get the job number |
| goto() | function | private | Go to the list page of the job |

##### HomeView

This is the presentation view of HomeController

##### ListController

Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
| title | string | private | The title of the job (appointment/rental/repair) |
| data | Array | private | The array of the detail of job |

Method

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
| goto() | function | private | Go to the detail page of the job |

##### ListView

This is the presentation view of ListController

##### DetailController

Attribute

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
| title | string | private | The title of the job (appointment/rental/repair) |
| data | object | private | The detail of job |

Method

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Visibility** | **Description** |
| change() | function | private | Call the api to change the status of the job |

##### DetailView

This is the presentation view of DetailController

### Interaction Diagram

#### <Manager> View Appointment Request List

**Summary:** This diagram used to describe the process of manager view all appointment requests.

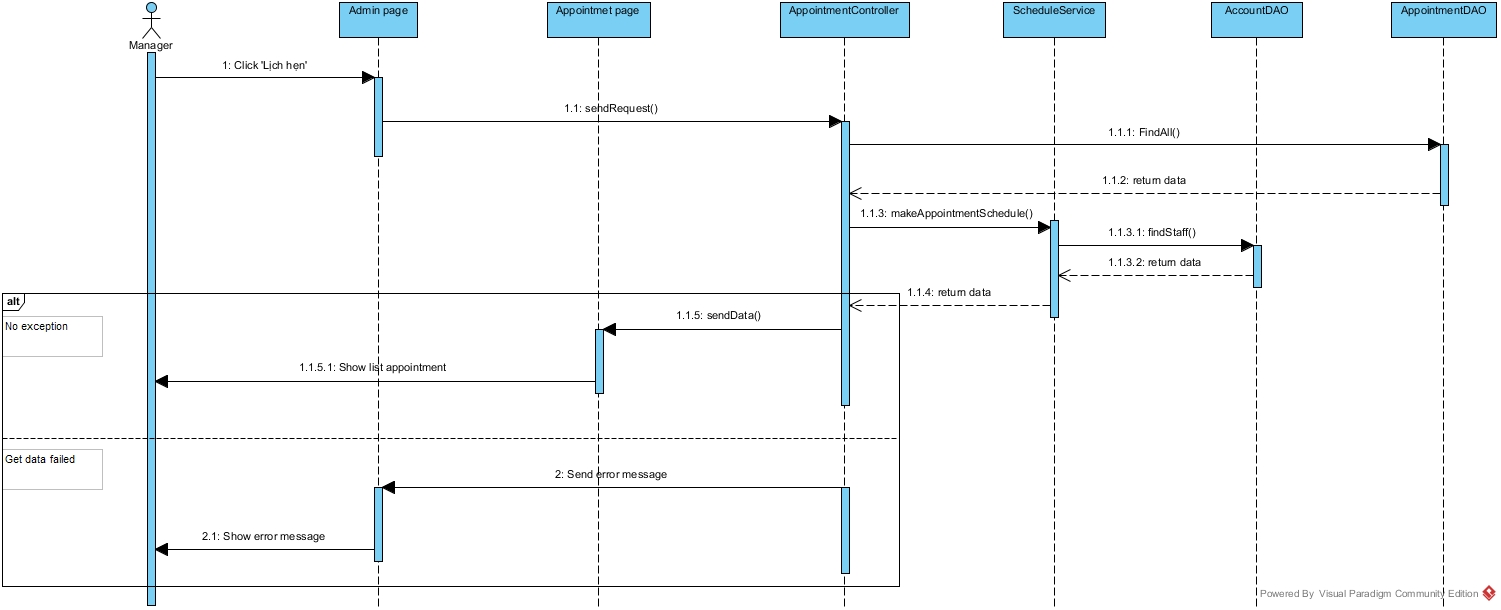


Figure 13: View Appointment Request List

#### <Manager> Assign Appointment

**Summary:** This diagram used to describe the process of manager assign appointment requests to staff.

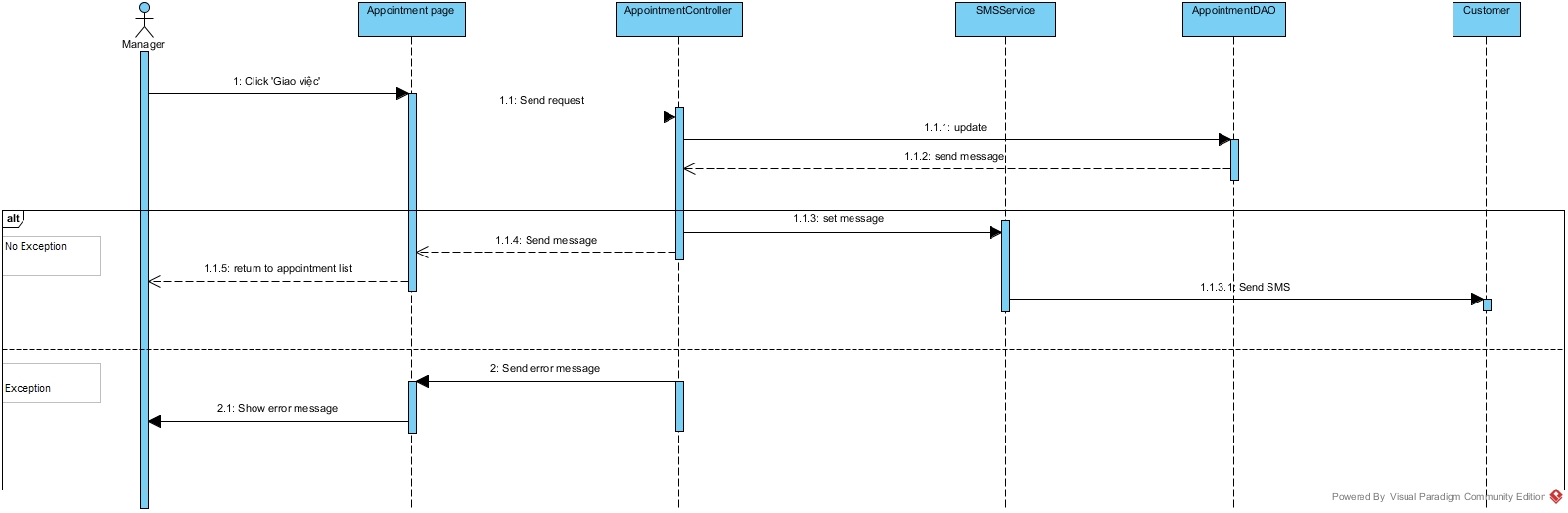


Figure 13: Assign Appointment

#### <Manager> Cancel Appointment

**Summary:** This diagram used to describe the process of manager cancel appointment

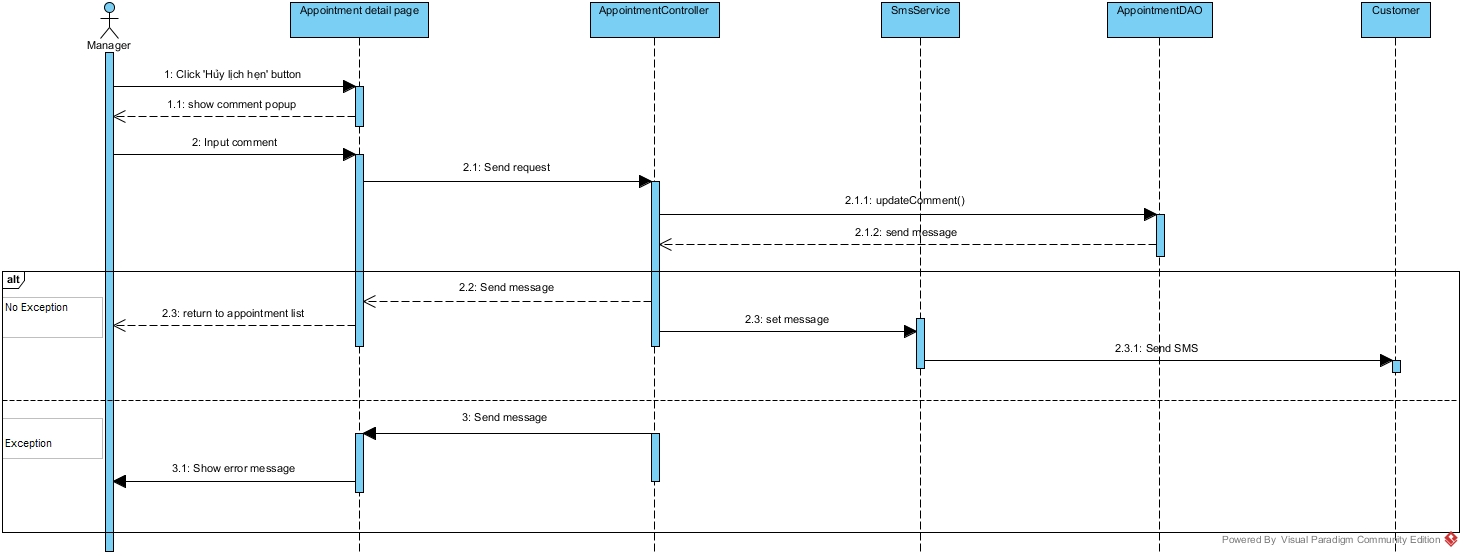


Figure 13: Cancel Appointment

#### <Manager> Create Contract

**Summary:** This diagram used to describe the process of manager create contract

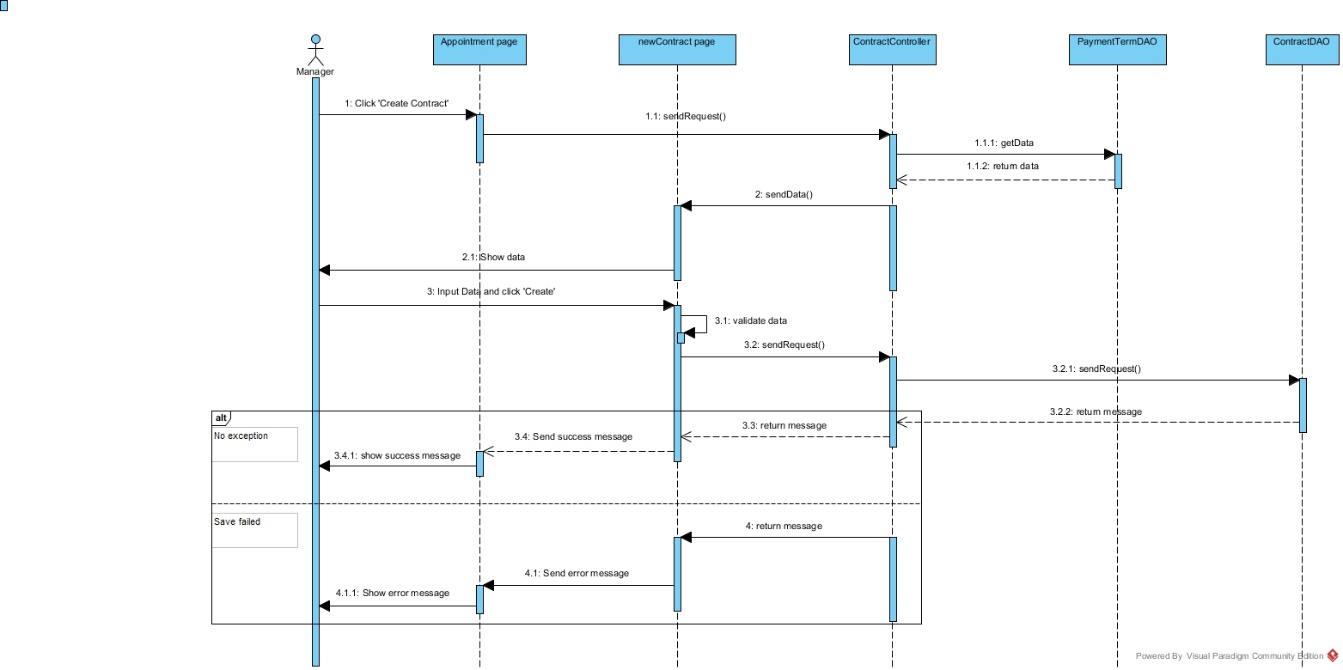


Figure 13: Create Contract

#### <Manager> View Contract List

**Summary:** This diagram used to describe the process of manager view contract list

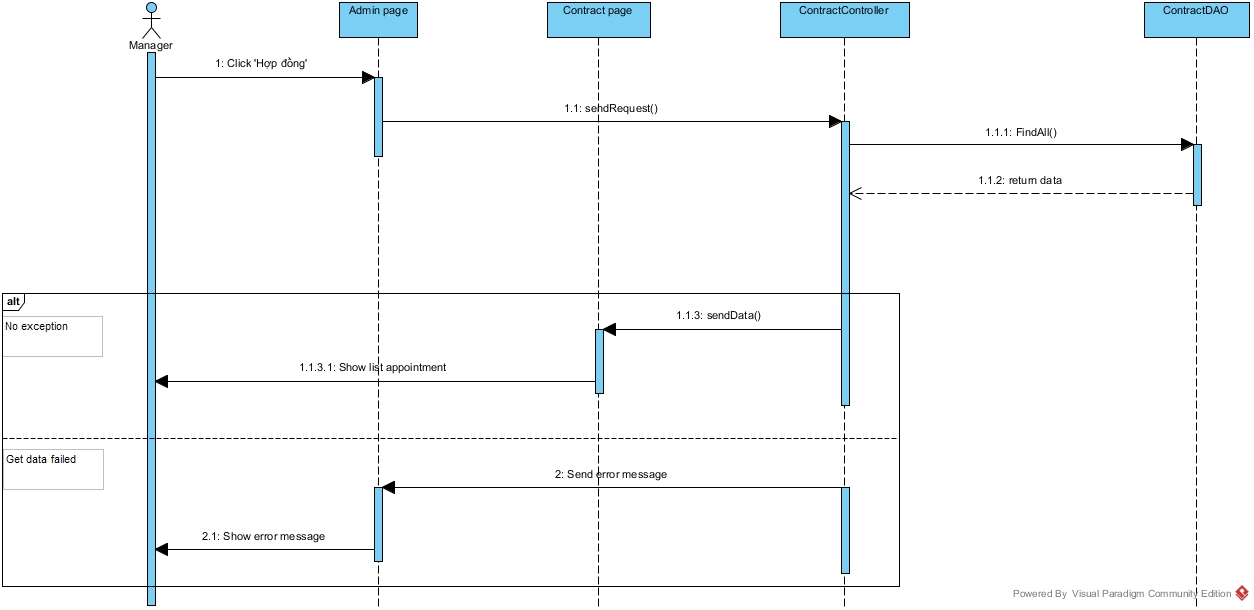


Figure 13: View Contract List

#### <Manager> View Contract Detail

**Summary:** This diagram used to describe the process of manager view contract detail

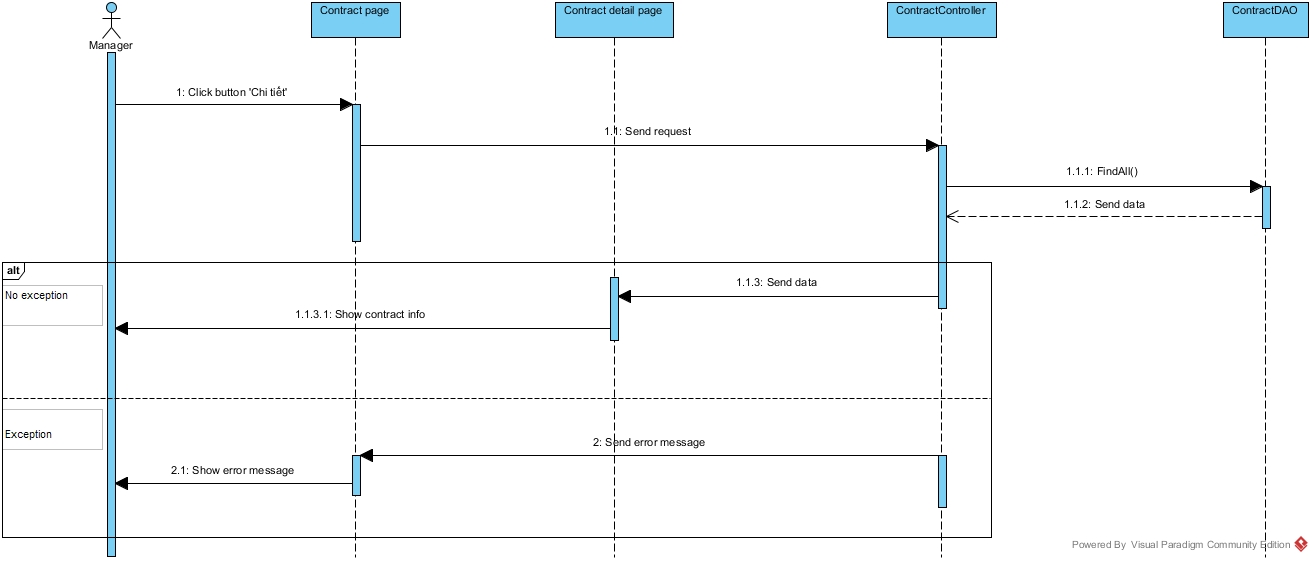


Figure 13: View Contract Detail

#### <Manager> Edit Contract

**Summary:** This diagram used to describe the process of manager edit contract.

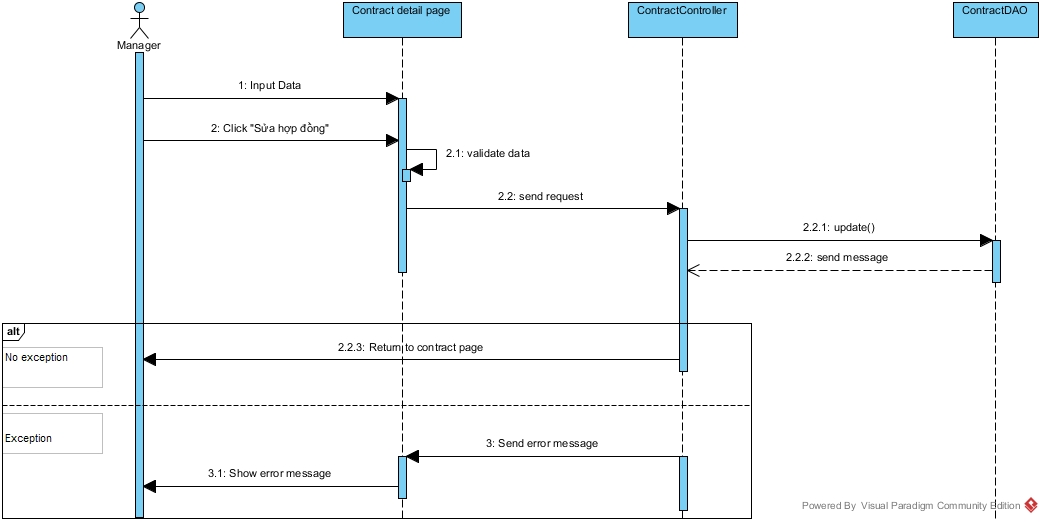


Figure 13: Edit Contract

#### <Manager> View Repair Request

**Summary:** This diagram used to describe the process of manager view repair requests.

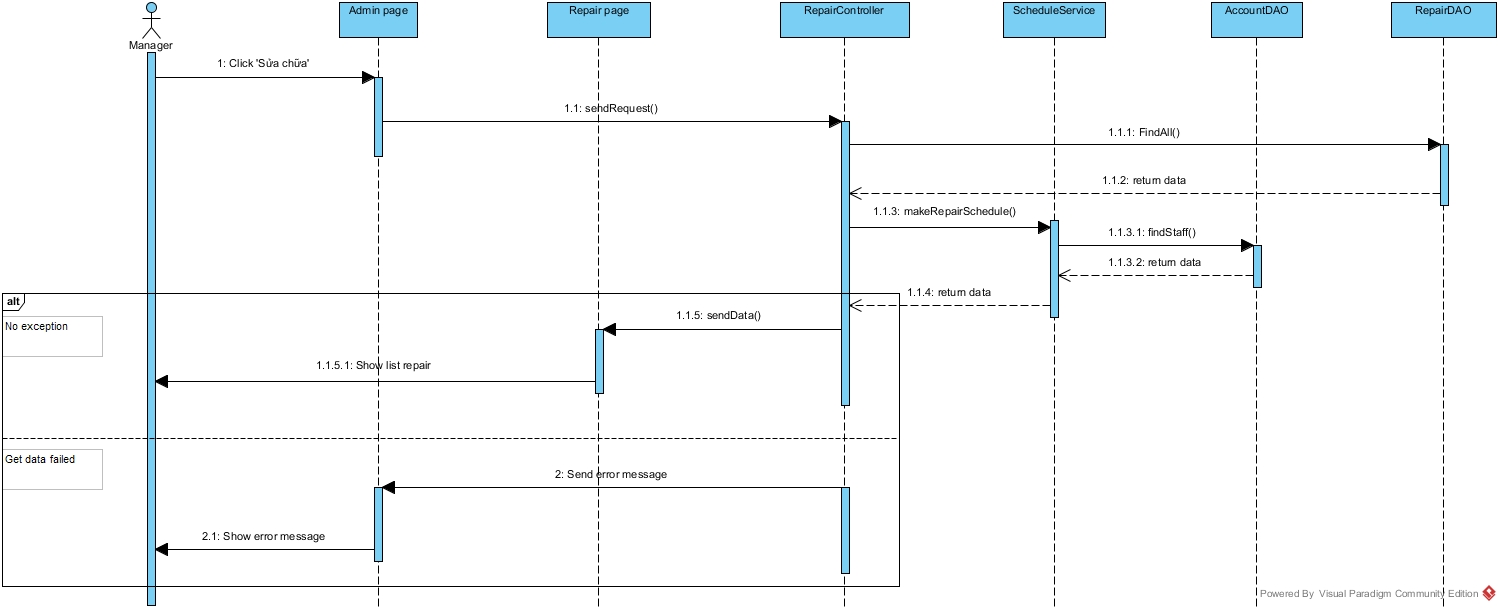


Figure 13: View Repair Request

#### <Manager> Assign Repair Request

**Summary:** This diagram used to describe the process of manager assign repair requests to staff.

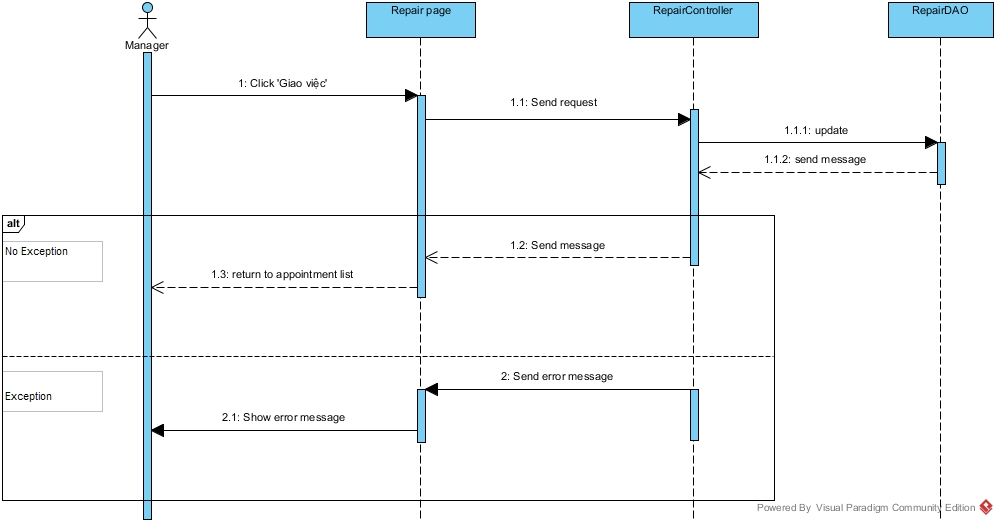


Figure 13: Assign Repair Request

#### <Manager> Cancel Repair Request

**Summary:** This diagram used to describe the process of manager cancel repair requests.

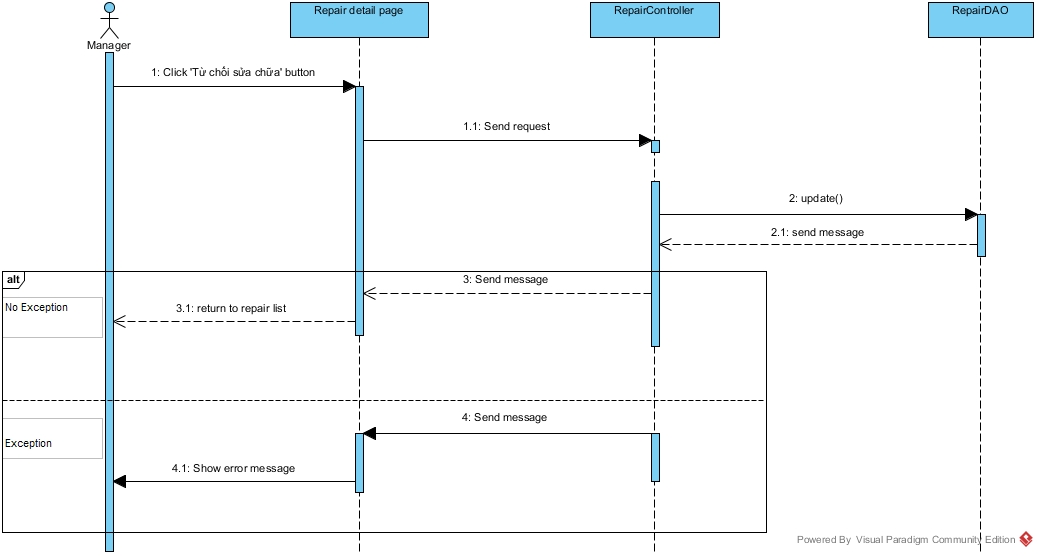


Figure 13: Cancel Repair Request

#### <Manager> View Rental Request

**Summary:** This diagram used to describe the process of manager view rental requests.

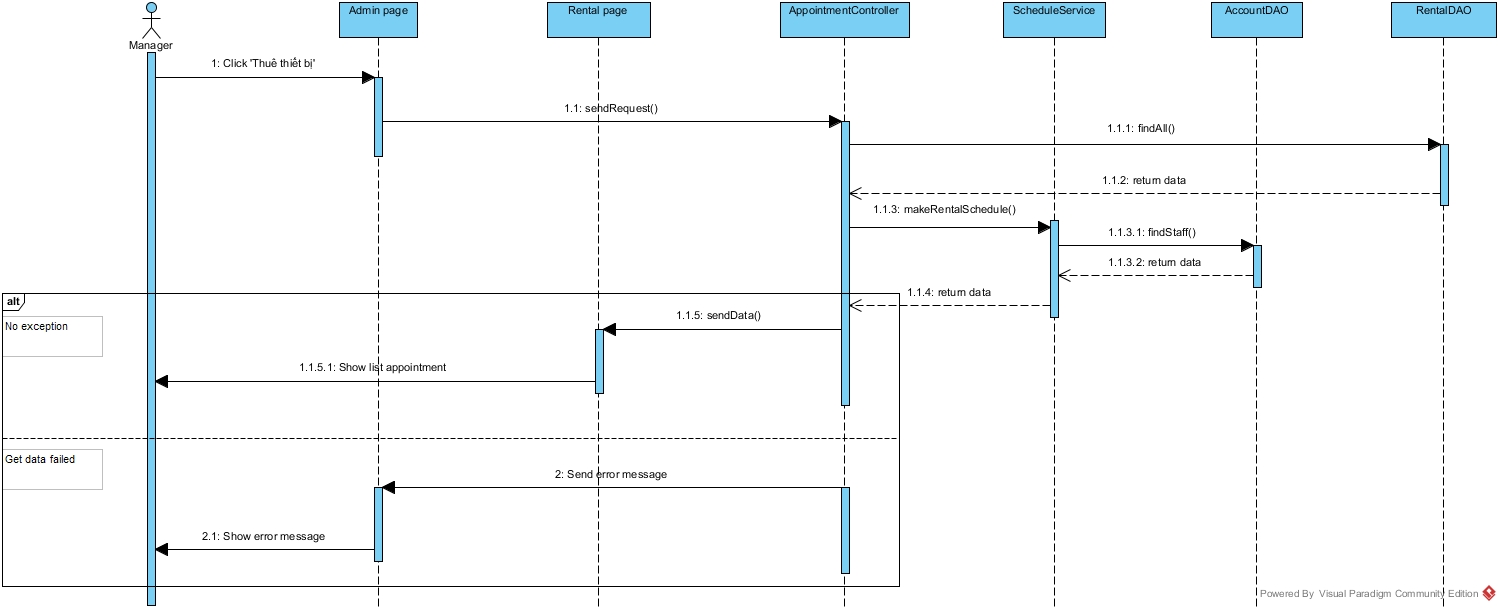


Figure 13: View Rental Request

#### <Manager> Assign Rental Request

**Summary:** This diagram used to describe the process of manager assign rental requests to staff.

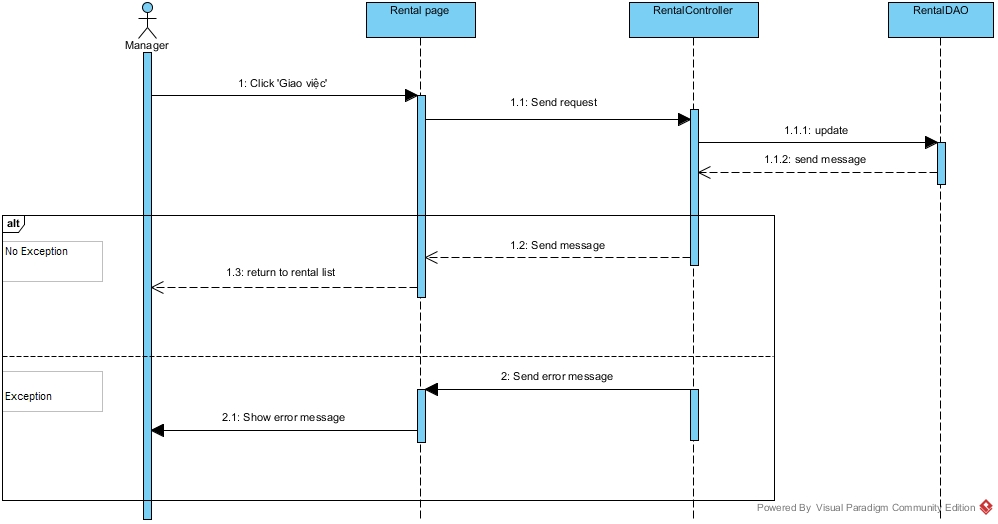


Figure 13: Assign Rental request

#### <Manager> Cancel Rental Request

**Summary:** This diagram used to describe the process of manager cancel rental request.

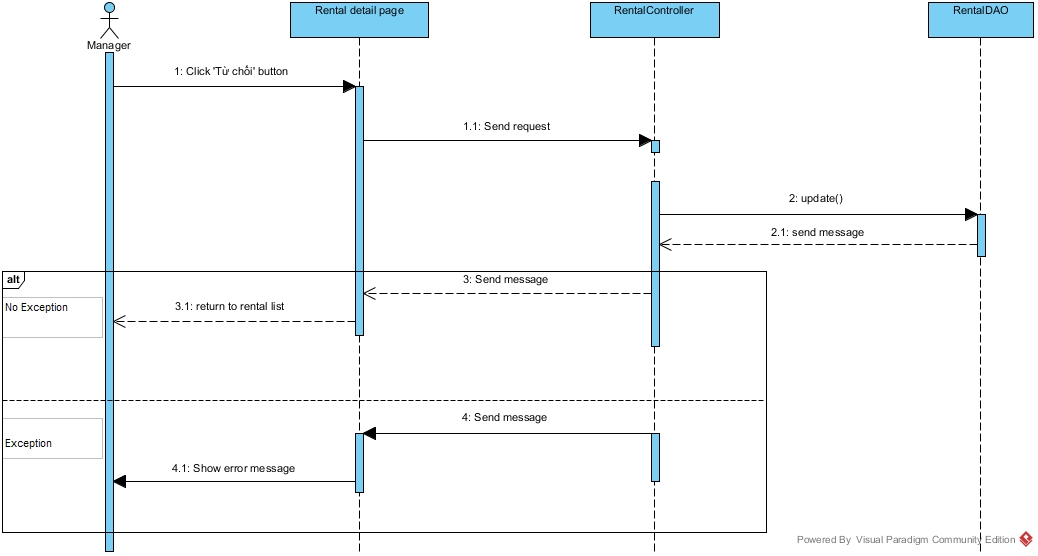


Figure 13: Cancel Rental Request

#### <Customer> View Result Request Office

**Summary:** This diagram used to describe the process of customer view result request office.

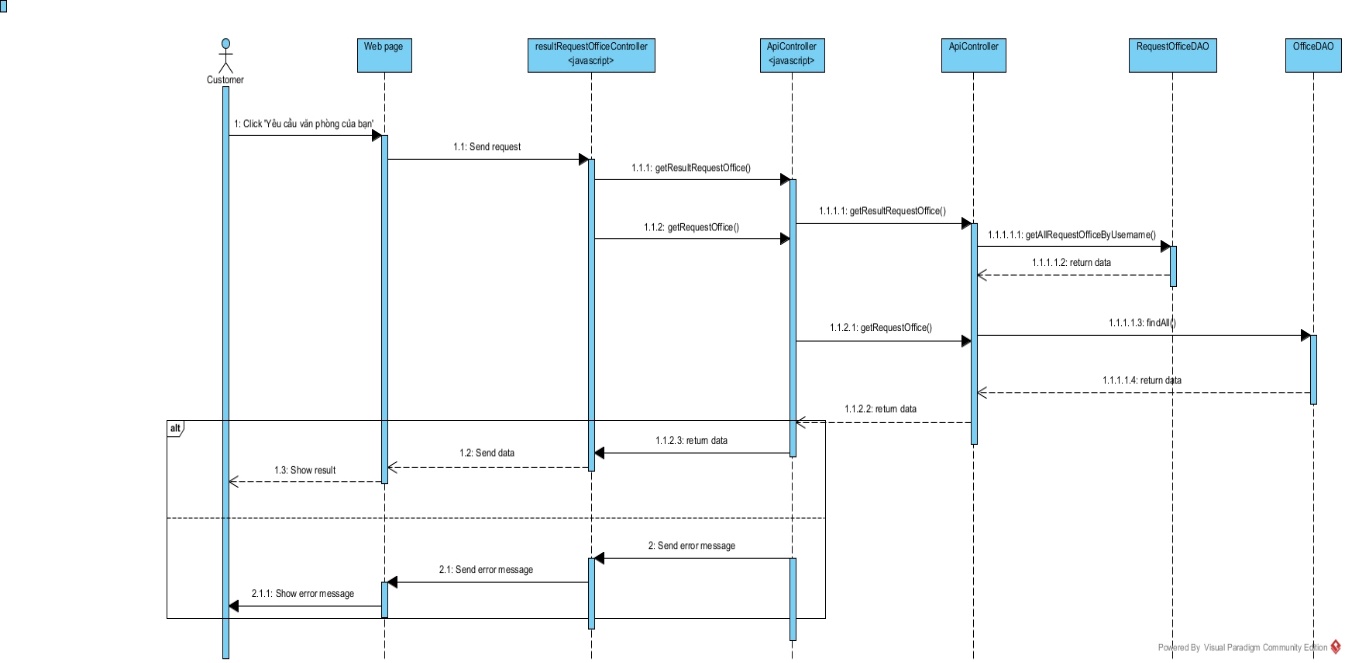


Figure 13: View Result Request Office

#### <System> Check Contract Due Date

**Summary:** This diagram used to describe the process of system check contract due date.

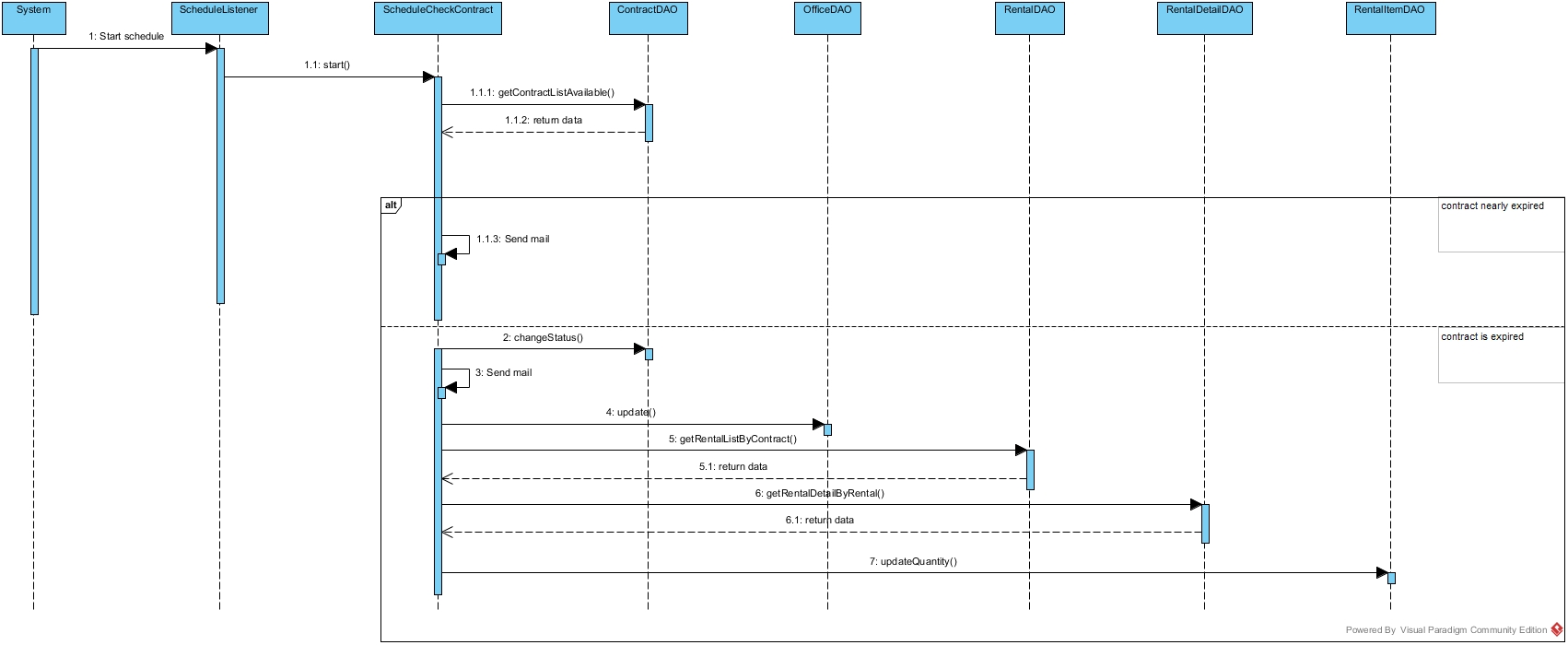


Figure 13: Check Contract Due Date

#### <System> Suggest Request Office

**Summary:** This diagram used to describe the process of system suggest request office to customer.

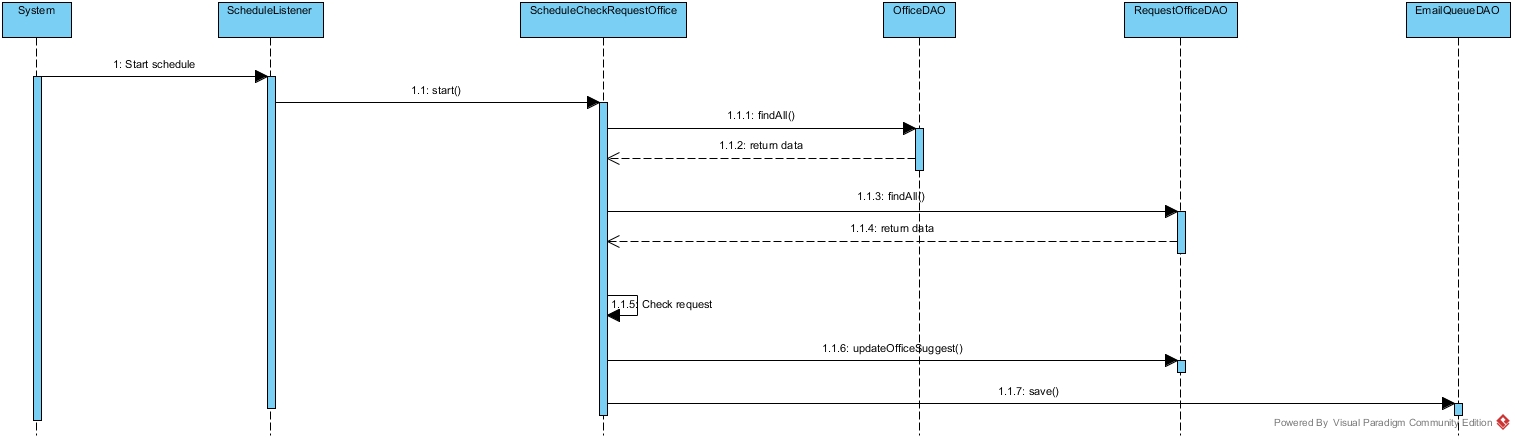
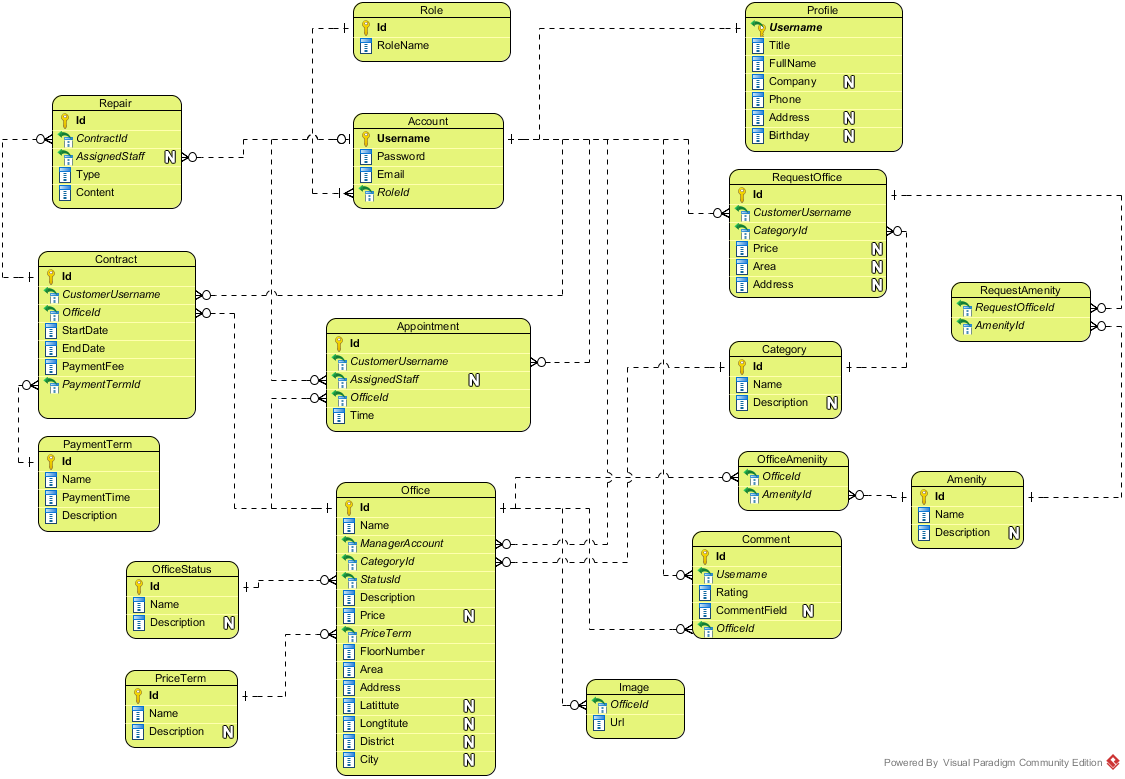


Figure 13: Suggest Request Office

## Database Design

### Logical Diagram



**Figure 17: Logical Database Diagram**

### Data Dictionary

|  |  |
| --- | --- |
| **Entity Data dictionary: describe content of all entities** | |
| **Entity Name** | **Description** |
| Account | Describe all accounts of users in the system. |
| Account Status | Describe status of account. One account has only one status. |
| Amenity | Describe all amenities in system. |
| Amenity Group | Describe groups of amenity. One amenity is only in one group. |
| Appointment | Describe all appointment requests of customer. |
| Appointment Status | Describe status of appointment request. One appointment has only one status. |
| Category | Describe category of office. One office is only in one category. |
| Contract | Describe all contracts in the system. |
| Contract Status | Describe status of contract. One contract has only one status. |
| Email Queue | Describe all emails have to send to customer in the system. |
| Office | Describe all offices in the system. |
| Office Amenity | Describe all amenities in one office. One office can have more than one amenity. |
| Office Group | Describe group of the office for searching office. One office is only in one group |
| Office Status | Describe status of the office. One office has only one status. |
| Payment Term | Describe time for the payment |
| Price Term | Describe the way that money is calculated base on. |
| Profile | Describe information of the account |
| Rental | Describe all rental requests from customer |
| Rental Detail | Describe all rental items for one rental request |
| Rental Item | Describe all rental items in system |
| Rental Status | Describe status of rental request. One rental request has only one status. |
| Repair | Describe all repair requests from customer |
| Repair Detail | Describe all amenities need to repair for one repair request |
| Repair Status | Describe status of repair request. One repair request has only one status. |
| Request Amenity | Describe all amenities in one request office |
| Request Office | Describe all requests office from customer |
| Role | Describe role of account. One account has only one role. |

#### Table Account

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Entity name** | **Attributes** | **Description** | **Type** | **Null** |
| Account | Username{PK} | Unique identifier of an account, username for login to system | nvarchar(50) | No |
| Password | Password for login to system | varchar(50) | No |
| Email | Email of this account | varchar(50) | No |
| RoleId{FK} | Role of this account. Foreign key references to table Role | int | No |
| StatusId{FK} | Status of this account. Foreign key references to table Account Status | int | No |
| Unique Key: Email | | | | |

#### Table Account Status

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Entity name** | **Attributes** | **Description** | **Type** | **Null** |
| Account Status | Id{PK} | Unique identifier of an account status, automatically increase | int | No |
| Name | Name of this status | nvarchar(50) | No |
| Description | Description for this status | nvarchar(50) | Yes |

#### Table Amenity

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Entity name** | **Attributes** | **Description** | **Type** | **Null** |
| Amenity | Id{PK} | Unique identifier of an amenity, automatically increase | int | No |
| Name | Name of this amenity | nvarchar(50) | No |
| Description | Description for this amenity | nvarchar(250) | Yes |
| Weight | Weight of amenity in one group | int | Yes |
| AmenityGroupId{FK} | Group of this amenity. Foreign key references to table Amenity Group | int | Yes |
| Priority | Priority of amenity in one group | int | Yes |
| Unique Key: Weight | | | | |

#### Table Amenity Group

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Entity name** | **Attributes** | **Description** | **Type** | **Null** |
| Amenity Group | Id{PK} | Unique identifier of an amenity group, automatically increase | int | No |
| Name | Name of amenity group | nvarchar(50) | No |
| Description | Description for amenity group | nvarchar(250) | Yes |

#### Table Appointment

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Entity name** | **Attributes** | **Description** | **Type** | **Null** |
| Appointment | Id{PK} | Unique identifier of an appointment, automatically increase | int | No |
| CustomerUsername{FK} | Username of customer. Foreign key references to table Account | nvarchar(50) | No |
| AssignedStaff{FK} | Staff who is responsibility for the appointment. Foreign key references to table Account | nvarchar(250) | Yes |
| OfficeId{FK} | Office where customer want to make appointment. Foreign key references to table Office. | int | No |
| Time | Time for appointment | datetime | No |
| StatusId{FK} | Status of the appointment. Foreign key references to table appointment status. | int | No |
| CreateTime | Time when customer create appointment | datetime | No |
| UpdateTime | Time when manager update the appointment | datetime | Yes |
| Comment | Comment for the appointment | nvarchar(250) | Yes |

#### Table Appointment Status

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Entity name** | **Attributes** | **Description** | **Type** | **Null** |
| Appointment Status | Id{PK} | Unique identifier of an appointment status, automatically increase | int | No |
| Name | Name of appointment status | nvarchar(50) | No |
| Description | Description for appointment status | nvarchar(50) | Yes |

#### Table Category

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Entity name** | **Attributes** | **Description** | **Type** | **Null** |
| Category | Id{PK} | Unique identifier of a category for office, automatically increase | int | No |
| Name | Name of category | nvarchar(50) | No |
| Description | Description for category | nvarchar(250) | No |
| Unique Key: Name | | | | |

#### Table Contract

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Entity name** | **Attributes** | **Description** | **Type** | **Null** |
| Contract | Id{PK} | Unique identifier of a contract, automatically increase | int | No |
| CustomerUsername{FK} | Username of customer. Foreign key references to table Account | nvarchar(50) | No |
| OfficeId{FK} | Office where customer want to make contract. Foreign key references to table Office. | int | No |
| StartDate | Date when contract starts. | date | No |
| EndDate | Date when contract ends. | date | No |
| PaymentFee | Fee for the contract | int | No |
| PaymentTerm{FK} | Payment term for the contract. Foreign key references to table Payment Term. | int | No |
| StatusId{FK} | Status of the contract. Foreign key references to table contract status. | int |  |
| Comment | Comment for the appointment | nvarchar(250) | Yes |

#### Table Contract Status

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Entity name** | **Attributes** | **Description** | **Type** | **Null** |
| Contract Status | Id{PK} | Unique identifier of a contract status, automatically increase | int | No |
| Name | Name of contract status | nvarchar(50) | No |
| Description | Description for contract status | nvarchar(50) | Yes |

#### Table Email Queue

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Entity name** | **Attributes** | **Description** | **Type** | **Null** |
| Email Queue | Id{PK} | Unique identifier of an email queue, automatically increase | int | No |
| Username{FK} | Username of the customer. Foreign key references to table Account. | nvarchar(50) | No |
| OfficeIds | List office that system suggests to customer | nvarchar(50) | Yes |
| CreateTime | Time when customer create request | datetime | Yes |

#### Table Office

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Entity name** | **Attributes** | **Description** | **Type** | **Null** |
| Office | Id{PK} | Unique identifier of an office, automatically increase | int | No |
| Name | Name of the office. | nvarchar(50) | No |
| CategoryId{FK} | Category of the office. Foreign key references to table Category. | int | No |
| StatusId | Status of the office. Foreign key references to table Office Status. | int | No |
| Description | Description for the office | ntext | No |
| Price | Price of the office | bigint | Yes |
| PriceTerm{FK} | Price term of the office. Foreign key references to table Price Term. | int | No |
| FloorNumber | Number of floor in the office | int | No |
| Area | Area of the office | float | No |
| ImageUrls | Link of the images of office | text | Yes |
| OtherDetail | Some detail of the office | ntext | Yes |
| Address | Address of the office | nvarchar(150) | No |
| Latitude | Latitude of the office | float | Yes |
| Longitude | Longitude of the office | float | Yes |
| District | District of the office | nvarchar(50) | Yes |
| City | City of the office | nvarchar(50) | Yes |
| ParentOfficeId{FK} | Parent Office ID of the office. Foreign key references to table Office. | int | Yes |
| CreateDate | Date when office create in database | datetime | No |
| ViewCount | Number views when customer see detail of office | int | Yes |

#### Table Office Amenity

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Entity name** | **Attributes** | **Description** | **Type** | **Null** |
| Office Amenity | Id{PK} | Unique identifier of a contract status, automatically increase | int | No |
| OfficeId{FK} | Id of office. Foreign key references to table Office. | int | No |
| AmenityId{FK} | Amenity of this office Foreign key references to table Amenity. | int | No |

#### Table Office Group

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Entity name** | **Attributes** | **Description** | **Type** | **Null** |
| Office Group | OfficeId{PK, FK} | Id of office. Foreign key references to table Office. | int | No |
| OfficeGroup | Group after system run algorithm of this office | int | No |

#### Table Office Status

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Entity name** | **Attributes** | **Description** | **Type** | **Null** |
| Office Status | Id{PK} | Unique identifier of an office status, automatically increase | int | No |
| Name | Name of office status | nvarchar(50) | No |
| Description | Description for office status | nvarchar(250) | Yes |

#### Table Payment Term

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Entity name** | **Attributes** | **Description** | **Type** | **Null** |
| Payment Term | Id{PK} | Unique identifier of a payment term, automatically increase | int | No |
| Name | Name of term for payment | nvarchar(50) | No |
| PaymentTime | Number months of the payment | int | No |
| Description | Description for payment term | nvarchar(100) | Yes |
| Unique Key: PaymentTime | | | | |

#### Table Price Term

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Entity name** | **Attributes** | **Description** | **Type** | **Null** |
| Price Term | Id{PK} | Unique identifier of a price term, automatically increase | int | No |
| Name | Name of term for price | nvarchar(50) | No |
| Description | Description for price term | nvarchar(250) | Yes |
| Unique Key: Name | | | | |

#### Table Profile

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Entity name** | **Attributes** | **Description** | **Type** | **Null** |
| Profile | Username{PK, FK} | Unique identifier of an account. Foreign key references to table Account. | nvarchar(50) | No |
| Title | Gender of the profile | char(4) | No |
| FullName | Full name of this profile | nvarchar(50) | No |
| Company | Company of this profile | nvarchar(100) | Yes |
| Phone | Phone of this profile | varchar(15) | No |
| Address | Address of this profile | nvarchar(150) | Yes |
| Birthday | Birthday of this profile | datetime | Yes |

#### Table Rental

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Entity name** | **Attributes** | **Description** | **Type** | **Null** |
| Rental | Id{PK} | Unique identifier of a rental request, automatically increase | int | No |
| ContractId{FK} | Contract which customer send rental request. Foreign key references to table Contract. | int | No |
| AssignStaff{FK} | Username of staff who responsibility for this rental request. Foreign key references to table Account. | nvarchar(50) | Yes |
| StatusId{FK} | Status of this rental request. Foreign key references to table Rental Status. | int | No |
| Description | Description of rental | ntext | Yes |
| CreateTime | Time when customer create this rental request | datetime | No |
| UpdateTime | Time when manger or staff change status | datetime | Yes |
| AssignedTime | Time when manager assign to staff | datetime | Yes |

#### Table Rental Detail

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Entity name** | **Attributes** | **Description** | **Type** | **Null** |
| Rental Detail | Id{PK} | Unique identifier of a rental detail, automatically increase | int | No |
| RentalItemId{FK} | Item rental for the rental request. Foreign key references to table Rental Item. | int | No |
| RentalId{FK} | Id of Request rental. Foreign key references to table Rental. | int | No |
| UnitPrice | Price of one item | float | Yes |
| Quantity | Quantity of rental item for the rental request | int | Yes |

#### Table Rental Item

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Entity name** | **Attributes** | **Description** | **Type** | **Null** |
| Rental Item | Id{PK} | Unique identifier of a rental item, automatically increase | int | No |
| Name | Name of the item | nvarchar(150) | No |
| Description | Description for the item | ntext | Yes |
| Price | Price of the item | float | Yes |
| Quantity | Quantity of the item | int | Yes |
| ImageUrl | Link image of the item | nvarchar(250) | Yes |

#### Table Rental Status

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Entity name** | **Attributes** | **Description** | **Type** | **Null** |
| Rental Status | Id{PK} | Unique identifier of a rental status, automatically increase | int | No |
| Name | Name of rental status | nvarchar(50) | No |
| Description | Description for rental status | nvarchar(50) | Yes |

#### Table Repair

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Entity name** | **Attributes** | **Description** | **Type** | **Null** |
| Repair | Id{PK} | Unique identifier of a repair request, automatically increase | int | No |
| ContractId{FK} | Contract which customer send repair request. Foreign key references to table Contract. | int | No |
| AssignStaff{FK} | Username of staff who responsibility for this repair request. Foreign key references to table Account. | nvarchar(50) | Yes |
| RepairStatusId{FK} | Status of this repair request. Foreign key references to table Repair Status. | int | No |
| Description | Description of repair | ntext | Yes |
| CreateTime | Time when customer create this repair request | datetime | No |
| UpdateTime | Time when manger or staff change status | datetime | Yes |
| AssignedTime | Time when manager assign to staff | datetime | Yes |

#### Table Repair Detail

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Entity name** | **Attributes** | **Description** | **Type** | **Null** |
| Repair Detail | Id{PK} | Unique identifier of a repair detail, automatically increase | int | No |
| RepairId {FK} | Id of repair request. Foreign key references to table Repair. | int | No |
| AmenityId{FK} | Amenity of this repair request. Foreign key references to table Amenity. | int | No |

#### Table Repair Status

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Entity name** | **Attributes** | **Description** | **Type** | **Null** |
| Repair Status | Id{PK} | Unique identifier of a repair status, automatically increase | int | No |
| Name | Name of repair status | nvarchar(50) | No |
| Description | Description for repair status | nvarchar(50) | Yes |

#### Table Request Amenity

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Entity name** | **Attributes** | **Description** | **Type** | **Null** |
| Request Amenity | Id{PK} | Unique identifier of a request amenity, automatically increase | int | No |
| RequestOfficeId{FK} | Id of request office. Foreign key references to table Request Office. | int | No |
| AmenityId{FK} | Amenity of this request. Foreign key references to table Amenity. | int | No |

#### Table Request Office

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Entity name** | **Attributes** | **Description** | **Type** | **Null** |
| Request Office | Id{PK} | Unique identifier of a request office, automatically increase | int | No |
| CustomerUsername{FK} | Username of customer. Foreign key references to table Account | nvarchar(50) | No |
| CategoryId{FK} | Category of the office. Foreign key references to table Category. | int | Yes |
| Price | Price of the office. | int | Yes |
| Area | Area of the office | float | Yes |
| District | District of the office | nvarchar(50) | Yes |
| CreateDate | Time when customer create this request. | datetime | No |
| ScheduleDate | Date when system check this request | datetime | Yes |
| OfficeSuggested | List offices that are suitable with request | nvarchar(250) | Yes |

#### Table Role

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Entity name** | **Attributes** | **Description** | **Type** | **Null** |
| Role | Id{PK} | Unique identifier of a role, automatically increase | int | No |
| RoleName | Name of role | nvarchar(50) | No |
| Unique Key: RoleName | | | | |

## Algorithms

### Clustering data

#### Definition

Given the offices which have different location and price range. Place those offices into different groups that have similarity

#### Define Problem

* Office in near location may have similarity in price range. But the location may vary and hard to define all location.
* Group the office by location and price range will make the user easier to find the similarity offices

#### Solution

To solve the problem, we use the well-known K-Means algorithm.

The step to perform the algorithm:

1. Defined the k number (number of cluster)
   * The k number is calculate by the square root of total of data (n). This make sure that there is group, and each group has about data
2. Normalize the data into the n-dimensions Euclidean space, all data based on [0, 1] distance. Each of office data is called “point”
   * In this problem, we use 3 statics: latitude, longitude and price range of the office for 3-dimensions
   * The price range is divided into 5 range: Call as 0, 100.000 VND – 200.000 VND as 1, etc., larger than 500.000 VND as 5
   * In each dimension, calculate the min and max. Then the normalize dimension data is calculate by:

normalize: dimension normalized data

data: the raw data in dimension

1. Select random point into k group, each group has at least 1 point
2. Calculate the mean points of each group
3. Calculate the distance of each data point to each group’s mean point, and change the group of data point to the group that has the nearest mean
   * The distance between A(a1, a2, … an) and X(x1, x2, … xn) is calculate by Euclidean distance:

Where n is the number of dimension

1. Repeat step 4 until the group in step 5 is unchanged

#### Complexity

* In total, the complexity of this algorithm is NP-hard. But with current algorithm maximum complexity limit to n\*10 (n is total of data)

#### Flow chart

Calculate K

Normalize data

Group data in K groups

Calculate the mean of each group

Group based on minimum distance to mean

Group changed?

End

Start

Yes

No

### Matching data

#### Definition

Given the search office criteria, take out the result group (as calculate above) that have the most similarity with the criteria.

#### Define Problem

* The search office may be different from all of the office in system, and user want to make their decision with vary of result.

#### Solution

With the clustering data using K-Means, the best solution is KNN (K-Nearest-Neighbor) algorithm.

The steps to perform the algorithm:

1. Define k number. In this solution, we choose k at least by haft of the average office in each group of K-Means algorithm. Particular 2\* .
2. Normalize the query point (input data) to the same dimension of sample data (by above function)
3. Calculate the distance between query point and each of data in sample (all office) by Euclidean distance (as above).
4. Sort the calculated distance, select the minimum k distances for comparison.
5. The group of query point will be group that have the most frequent existence in above k distances. If there is more than 1 group that have largest existence, the selected group will be the group that have minimum total distance (in the k selected distances only)

#### Complexity

* In total, the complexity of this algorithm is O(n)

# System Implementation & Test (SIT)

## Introduction

### System Overview

This section provides in detail all necessary information about test plans, test cases, test result, test environments, pass/fail criteria and risks estimations as well as a checklist to cover all possible cases of CTS system.

### Test Approach

Goal: Check all the features in CTS system and record remain bug to fix.

Type: White box testing, Black-box Testing.

Size: System Component.

Technique: Check list.

## Database Relationship Diagram

### Physical Diagram

### Data Dictionary

## Performance Measures

### Clustering Performance

## Test Plan

### Features to be tested

We will carry out test based on core workflow of system. All main functions will be tested carefully and clearly following phases.

* Guest: add to cart, view cart, update cart, view product detail, basic search, and advance search, register

### Features not to be tested

* Login, Logout.

## System Testing Test Case

### Guest Test Case

#### Search Event

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ID** | **Test Case Description** | **Pre-Condition** | **Test Case Procedure** | **Expected output** | **Result** | **Test Date** |
| BS01 | Basic Search with valid inputted key word | N/A | 1 – Input “bup be” into search fields.  2 – Press enter key | - After step 2, System redirected to search page and display 3 results:   1. Thú Bông\_Disney Alice in Wonderland. 2. Cún đốm (Búp bê & Thú bông category). 3. Gấu bông teddy (Búp bê & Thú bông category) | Passed | 30/11/2014 |
| BS02 | Basic Search with wrong key word | N/A | 1 – Input “abc” into search fields.  2 – Press enter key | - After step 2, System redirected to search page and display message “Hiện tại không có sản phẩm mà bạn muốn tìm” | Passed | 30/11/2014 |

Table 16: Basic Search Test\_Case

## Other material

N/A

# Software User’s Manual

## Installation Guide

### Setting up environment at server side

The following software must be installed into the server machine:

#### Hardware requirements

Server computer for deploying with the minimum configuration:

* CPU Intel® Core 2 Duo
* 2GB RAM.
* 30GB of hard disk.
* Internet

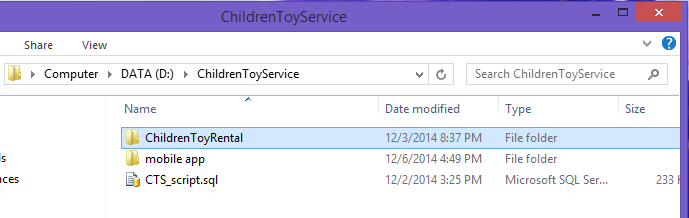
#### Software requirements

* Web Server: Internet Information System – IIS 7.
* Operating system: Windows Server 2008, or above
* SQL Server 2012 Express, or above: used to create and manage database of web application.

### Deployment at server side

#### Prepare deployment package

* Extract the deployment package to a folder on the server.
* For example: D:\ChildrenToyService



**Figure 11: Extract Deployment Package**

#### Database Deployment

### Client side environment setting

Client side should have one of these following browser:

* Google Chrome
* Firefox
* Internet Explorer

## User’s Guide

This user guide describes how to use the application based on the process of children toy rental service website

### Admin –Staff Login

**Figure 12: Admin - Staff Login Page**

|  |  |
| --- | --- |
| **Step** | **Description** |
| 1 | Enter “Email” và “Mật khẩu”  (E.g: Email: admin@gmail.com. Mật khẩu: 12345678) |
| 2 | Click “Đăng nhập” |

**Table 17: Login Step**

# Appendix