**MINISTRY OF EDUCATION AND TRAINING**

**FPT UNIVERSITY**

Capstone Project Document Overview

**Office Rental Service**

|  |  |
| --- | --- |
| **Group 03** | |
| **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 |

-Ho Chi Minh City, May 2015 -

*This page is intentionally left blank*

**

**CAPSTONE PROJECT REGISTER**

Class: Duration time: from 11/5/2015…. To ../../2015…..

(\*) Profession: <Software Engineer> Specialty: <ES> <IS> ⌧

(\*) Kinds of person make registers: Lecturer ⌧ Students

1. Register information for supervisor (if have)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Full name** | **Phone** | **E-Mail** | **Title** |
| Supervisor 1 | Nguyễn Trọng Tài |  | taint@fpt.edu.vn | Mr. |

2. Register information for students (if have)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Full name** | **Student code** | **Phone** | **E-mail** | **Role in Group** |
| Student 1 |  |  |  |  |  |
| Student 2 |  |  |  |  |  |
| Student 3 |  |  |  |  |  |
| Student 4 |  |  |  |  |  |

3. Register content of Capstone Project

(\*) 3.1. Capstone Project name:

English: Office Rental Service.

Vietnamese: Dịch vụ cho thuê văn phòng.

Abbreviation: ORS

Building an online system allows to rent the office. The system also support to manage different types of office (apartment, house) and its amenities..., manage contract, manage request for repair and maintenance.

(\*) 3.2. Main proposal content (including result and product)

1. Theory and practice (document):

* Apply the software development process and the UML (OOA, OOD, OOP should be applied)
* N-tier/MVC and Object-relational mapping should be applied
* Web-based application

1. Program:

* Main features should be included:
  + Member management
  + Office Management (by category, by amenities...)
  + Rental management
  + Contract management
  + Request for repair and maintenance management
  + Searching, statistic

1. Other products:

N/A

4. Other comment (propose all relative thing if have)

N/A

|  |  |
| --- | --- |
| **Supervisor (If have)**  *(Sign and full name)* | HCM city, date 11/4/2015  **On behalf of Registers**  *(Sign and full name)* |

# Table of Contents

[Table of Contents 5](#_Toc430353668)

[List of Tables 7](#_Toc430353669)

[List of Figures 7](#_Toc430353670)

[Definitions, Acronyms, and Abbreviations 7](#_Toc430353671)

[A. Software Project Management Plan 9](#_Toc430353672)

[1. Problem Definition 9](#_Toc430353673)

[1.1 Name of this Capstone Project 9](#_Toc430353674)

[1.2 Problem Abstract 9](#_Toc430353675)

[1.3 Project Overview 9](#_Toc430353676)

[2. Project Organization 12](#_Toc430353677)

[2.1 Software Process Model 12](#_Toc430353678)

[2.2 Tools and Techniques 12](#_Toc430353679)

[B. Software Requirement Specification 13](#_Toc430353680)

[1. User Requirement Specification 13](#_Toc430353681)

[1.1 Guest requirement 13](#_Toc430353682)

[1.2 Customer requirement 13](#_Toc430353683)

[1.3 Owner requirement 13](#_Toc430353684)

[1.4 Admin requirement 13](#_Toc430353685)

[1.5 Manager requirement 13](#_Toc430353686)

[1.6 Staff requirement 14](#_Toc430353687)

[1.7 System requirement 14](#_Toc430353688)

[2. System Overview Use Case 14](#_Toc430353689)

[3. Conceptual Diagram 16](#_Toc430353690)

[C. Software Design Description 17](#_Toc430353691)

[1. Design Overview 17](#_Toc430353692)

[2. System Architectural Design 18](#_Toc430353693)

[3. Component Diagram 19](#_Toc430353694)

[4. Detailed Description Explanation 20](#_Toc430353695)

[4.1 Class Diagram 20](#_Toc430353696)

[4.2 Sequence Diagram 22](#_Toc430353697)

[5 Database Design 24](#_Toc430353698)

[5.1 Entity-Relationship Diagram 24](#_Toc430353699)

[5.2 Data Dictionary 25](#_Toc430353700)

[6 Algorithms 25](#_Toc430353701)

[6.1 Clustering data 25](#_Toc430353702)

[6.2 Matching data 27](#_Toc430353703)

[D. System Implementation & Test 29](#_Toc430353704)

[1. Physical Diagram 29](#_Toc430353705)

[2 Data Dictionary 30](#_Toc430353706)

[2.1 Table Account 30](#_Toc430353707)

[2.2 Table Account Status 31](#_Toc430353708)

[2.3 Table Amenity 31](#_Toc430353709)

[2.4 Table Amenity Group 31](#_Toc430353710)

[2.5 Table Appointment 31](#_Toc430353711)

[2.6 Table Appointment Status 32](#_Toc430353712)

[2.7 Table Category 32](#_Toc430353713)

[2.8 Table Contract 32](#_Toc430353714)

[2.9 Table Contract Status 32](#_Toc430353715)

[2.10 Table Email Queue 33](#_Toc430353716)

[2.11 Table Office 33](#_Toc430353717)

[2.12 Table Office Amenity 34](#_Toc430353718)

[2.13 Table Office Group 34](#_Toc430353719)

[2.14 Table Office Status 34](#_Toc430353720)

[2.15 Table Payment Term 34](#_Toc430353721)

[2.16 Table Price Term 35](#_Toc430353722)

[2.17 Table Profile 35](#_Toc430353723)

[2.18 Table Rental 35](#_Toc430353724)

[2.19 Table Rental Detail 35](#_Toc430353725)

[2.20 Table Rental Item 36](#_Toc430353726)

[2.21 Table Rental Status 36](#_Toc430353727)

[2.22 Table Repair 36](#_Toc430353728)

[2.23 Table Repair Detail 37](#_Toc430353729)

[2.24 Table Repair Status 37](#_Toc430353730)

[2.25 Table Request Amenity 37](#_Toc430353731)

[2.26 Table Request Office 37](#_Toc430353732)

[2.27 Table Role 38](#_Toc430353733)

[E. Demonstrations Workflow 39](#_Toc430353734)

[1. <Manager> Create contract 39](#_Toc430353735)

[2. <Customer> Request repair 40](#_Toc430353736)

[3. <Customer> Request appointment 41](#_Toc430353737)

[F. Task sheet 42](#_Toc430353738)

# List of Tables

[Table 1: Definitions 8](#_Toc430353739)

[Table 2: Hardware Requirement for Server 11](#_Toc430353740)

[Table 3: Hardware Requirement for Web User 11](#_Toc430353741)

[Table 4: Requirement for mobile app 11](#_Toc430353742)

[Table 5: Conceptual diagram data dictionary 17](#_Toc430353743)

[Table 6: Component dictionary 19](#_Toc430353744)

[Table 7: Data dictionary 25](#_Toc430353745)

# List of Figures

[Figure 1: Proposed System 10](#_Toc430353746)

[Figure 2: Software Process Model 12](#_Toc430353747)

[Figure 3: System Overview Use Case 15](#_Toc430353748)

[Figure 4: Conceptual diagram 16](#_Toc430353749)

[Figure 5: ORS System Architectural 18](#_Toc430353750)

[Figure 6: Component Diagram 19](#_Toc430353751)

[**Figure 7: Class diagram** 20](#_Toc430353752)

[Figure 8: Search Office Sequence Diagram 22](#_Toc430353753)

[Figure 9: Create New Office Sequence Diagram 22](#_Toc430353754)

[**Figure 10: Assign Appointment Sequence Diagram** 23](#_Toc430353755)

[Figure 11: ERD Diagram 24](#_Toc430353756)

[Figure 12: Physical Database Diagram 29](#_Toc430353757)

[Figure 13: <Manager> Create contract 39](#_Toc430353758)

[Figure 14: <Customer> Request repair 40](#_Toc430353759)

[Figure 15: <Customer> Request appointment 41](#_Toc430353760)

# Definitions, Acronyms, and Abbreviations

|  |  |
| --- | --- |
| **Name** | **Definition** |
| ORS | Office Rental Service |
| IDE | Integrated Development Environment |
| DBMS | Database Management System |
| WAP | Wireless Application Protocol |
| SRS | Software Requirement Specification |
| Admin | Administrator |
| SMS | Short message service |
| Info | Information |
| FE | Front-end |
| BE | Back-end |

Table 1: Definitions

# Software Project Management Plan

## Problem Definition

### Name of this Capstone Project

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

### Problem Abstract

Nowadays, the way companies find and rent an office, is complex and hard to find a suitable office. It means they need to do many steps before they can rent a suitable office. About the office owner, they have to contact with customer, manage contract, manage repair request,… on the paper. It means there is human mistake.

To solve those problems already mentioned above, we need to make the way customer find an office and the way customer manage their request effectively. About the office owner, their contracts, requests for their office,… are easy to manage.

### 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.
  + Direct consultation with the manager.
* Disadvantages:
  + Only supply direct meeting, not arrange for a meeting.
  + Not regularly updated information.
  + Not support map.
  + Not provide additional equipment and maintenance.

#### The Proposed System

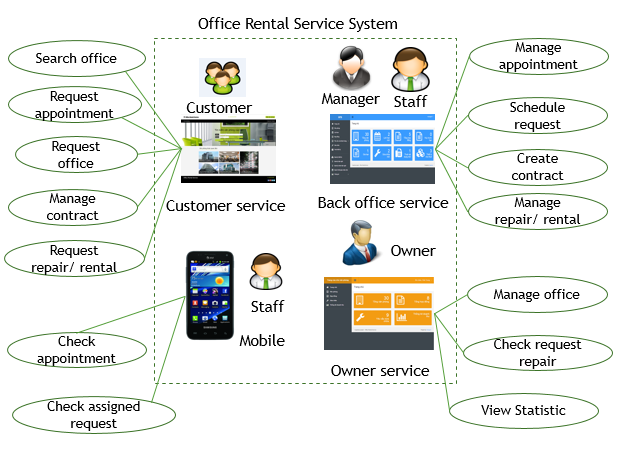


Figure 1: Proposed System

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.
* Owner can manage their office, and accept/request repair request.
* 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, manager, staff, customers, guest and owner 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, Ubuntu Server 12 LTS | Window Server 2012, Ubuntu Server 14.04.2 LTS |
| Computer Processor | Intel Core i5 | Intel Core i7 |
| Computer Memory | 4GB RAM | 6GB 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 | Wifi (2Mbps) | Wifi (4Mbps) |
| Operating System | Android 4.0 | Android 4.2 |
| Hardware | Touchscreen | Touchscreen |
| Memory | 512 MB or more | 1 GB or more |

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.

## 2. Project Organization

### 2.1 Software Process Model

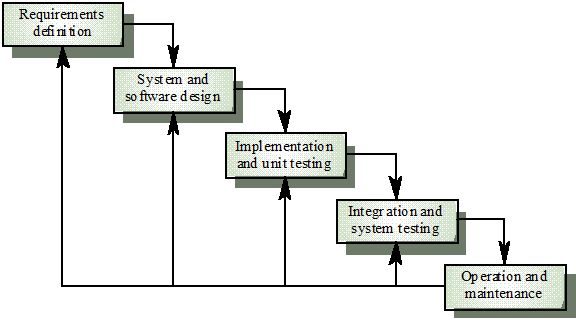


Figure 2: Software Process Model

[1]Reference: Modelling and System Design. [Online] http://www.pling.org.uk/cs/msd.html

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.

### 2.2 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, Android Studio.
* *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.

# Software Requirement Specification

## User Requirement Specification

The system has seven actors include guest, customer, owner, manager, 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 |
| Owner | Person who own the offices, and create a office on website for rent |
| 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/ extend/ return office request |
| System | System will manage the schedule to process contract time, send notification, suggest office for customer |

### Guest requirement

* **Register:** guest need to register to become member of system.
* **Search office:** guest can search suitable office by criteria such as price and location.
* **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 and location.
* **Send request:** customer can send request about book appointment, repair something in office, rental, request cancel before expire or request extend the contract.
* **Request office:** customer can send request office if they don’t find a suitable office.

### Owner requirement

* **Manage office:** owner can create and delete their own office.
* **Manage repair:** owner can approve/ cancel customer repair request, or check for its status

### Admin requirement

* **Logout:** when finish all activities at website they can log out of system.
* **Manage account:** admin can add, edit, remove or ban/ unban account.

### Manager requirement

* **Manage office:** manager can edit, approve or delete office.
* **Contact with customer:** manager can notify customer when task is done.
* **Manage contract:** manager can create, edit contract.
* **Manage request:** manager can confirm about request repair, rental and appointment.
* **Assign staff:** manager can assign 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.
* **Check assign request:** 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 Overview Use Case

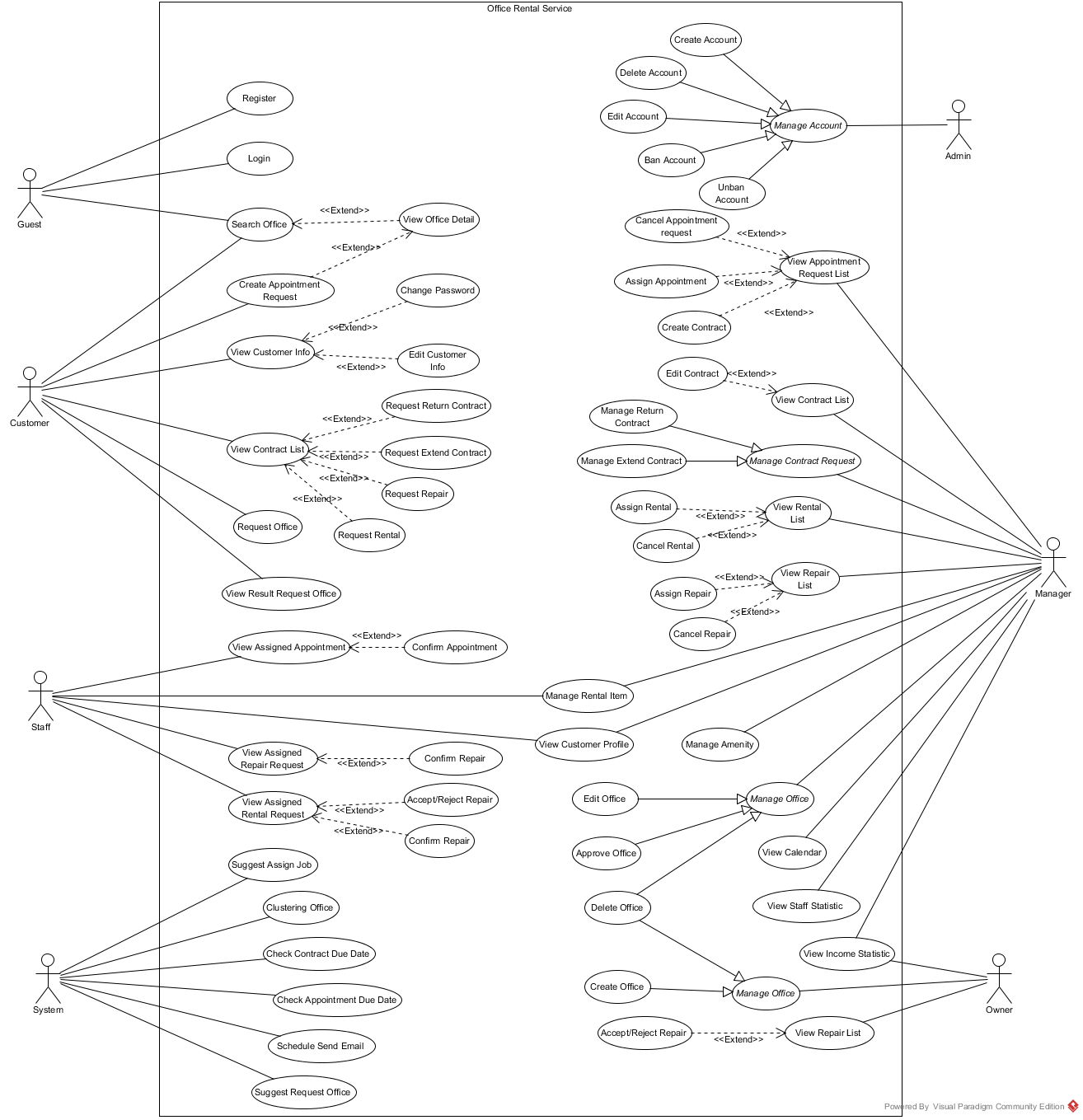


Figure 3: System Overview Use Case

## Conceptual Diagram

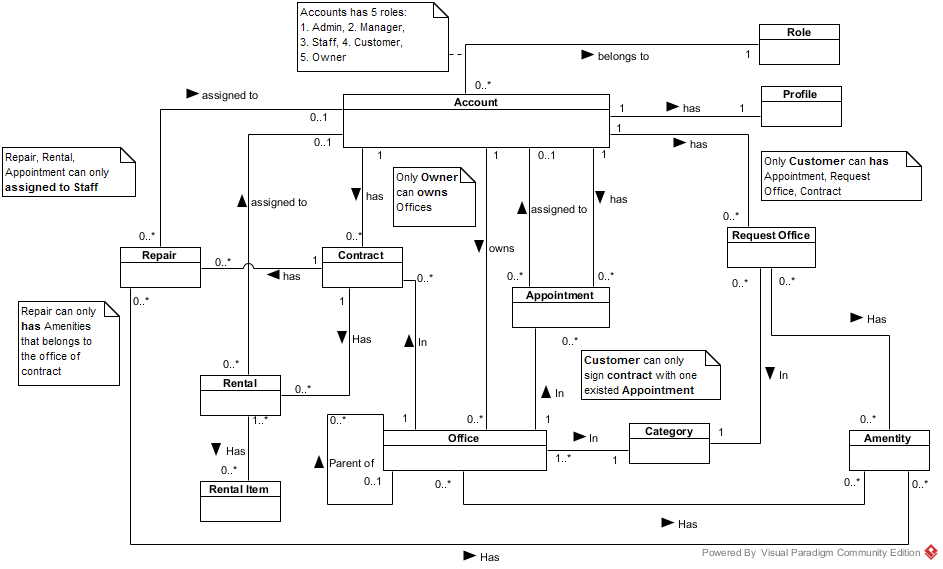


Figure 4: Conceptual diagram

**Data Dictionary**

|  |  |
| --- | --- |
| **Entity Data dictionary: describe content of all entities** | |
| **Entity Name** | **Description** |
| Account | Describe account of user of system. |
| Role | Describe account types of role |
| 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 |
| 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 |

Table 5: Conceptual diagram data dictionary

# 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

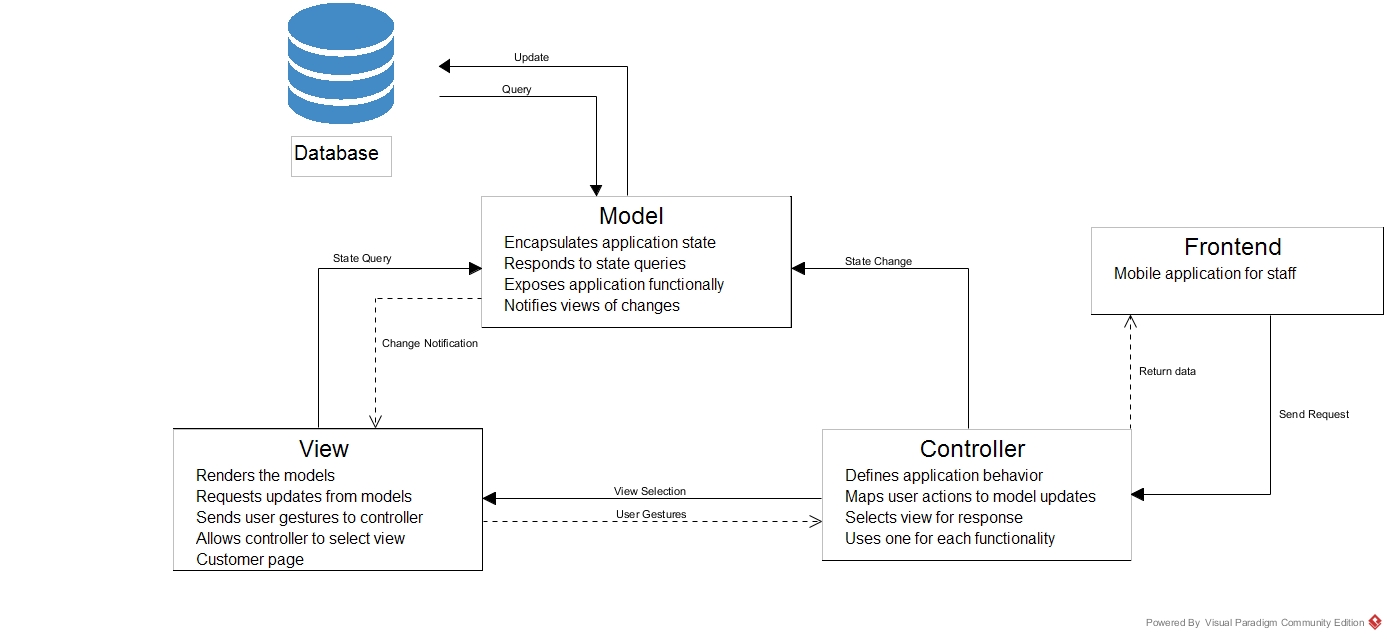


Figure 5: ORS System Architectural

[<http://msdn.microsoft.com/en-us/library/ff649643.aspx> ]

**Web application architecture description**

* **Model:** The model manages the behavior and data of the application domain, responds to requests for information about its state (usually from the view), and responds to instructions to change state (usually from the controller).
* **View:** The view manages the display of information.
* **Controller**: The controller interprets the mouse and keyboard inputs from the user, informing the model and/or the view to change as appropriate.
* **Frontend**: Mobile application for staffs send and receive request through controller.

## Component Diagram

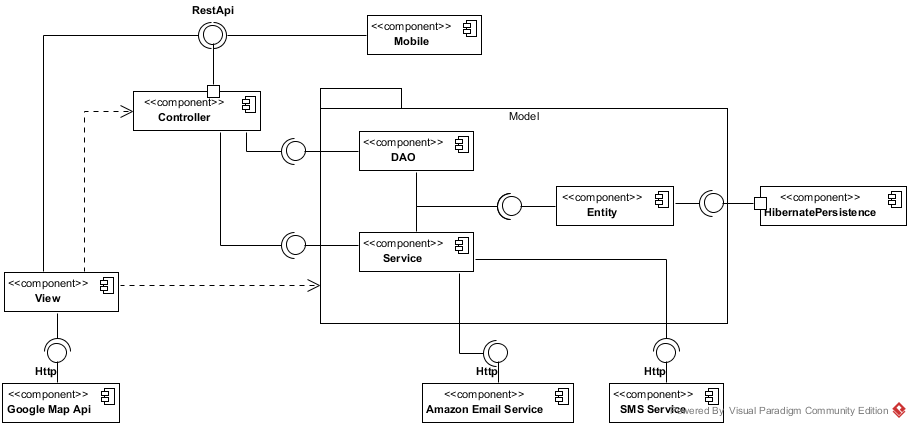


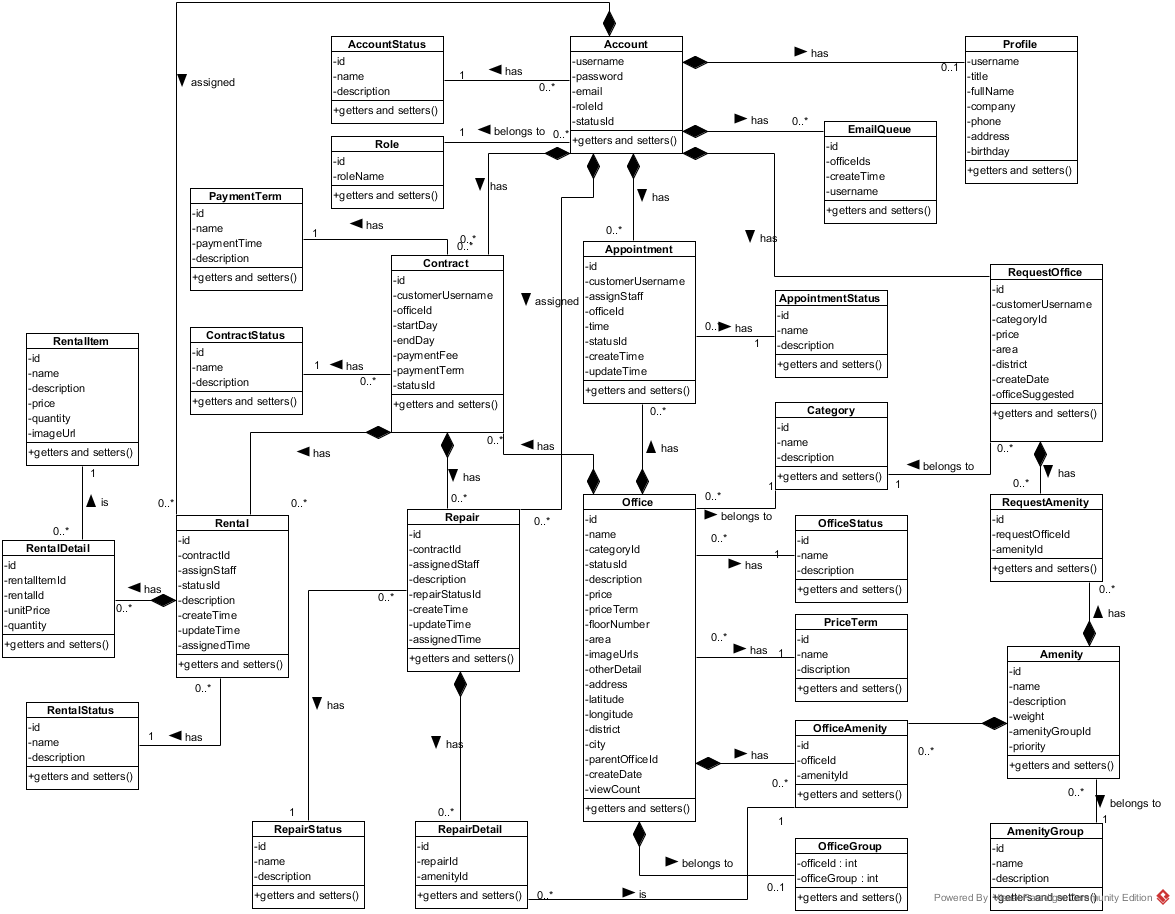
Figure 6: Component Diagram

|  |  |
| --- | --- |
| **Component dictionary: describe component** | |
| **Component Name** | **Description** |
| HibernatePersistence | Component provide by Hibernate to establish database connection |
| Entity | The database mapping classes component |
| DAO | Data Access Classes component, provide function to manipulate data |
| Service | Provide additional data manipulation and external data manipulation |
| Controller | Receive and handle the request from View, then call DAO/ Service to manipulate |
| View | Render the view for user |
| Mobile | The mobile application component |
| Google Map Api | The map component provide by Google |
| Amazon Email Service | The email service component provide by Amazon |
| SMS Service | The SMS service component provide by ESMS (<http://esms.vn/>) |

Table 6: Component dictionary

## Detailed Description Explanation

### Class Diagram



**Figure 7: Class diagram**

|  |  |
| --- | --- |
| **Class dictionary: describe Class** | |
| **Class Name** | **Description** |
| Account | Describe all accounts of users in the system. |
| AccountStatus | Describe status of account. One account has only one status. |
| Amenity | Describe all amenities in system. |
| AmenityGroup | Describe groups of amenity. One amenity is only in one group. |
| Appointment | Describe all appointment requests of customer. |
| AppointmentStatus | 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. |
| ContractStatus | Describe status of contract. One contract has only one status. |
| EmailQueue | Describe all emails have to send to customer in the system. |
| Office | Describe all offices in the system. |
| OfficeAmenity | Describe all amenities in one office. One office can have more than one amenity. |
| OfficeGroup | Describe group of the office for searching office. One office is only in one group |
| OfficeStatus | Describe status of the office. One office has only one status. |
| PaymentTerm | 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 |
| RentalDetail | Describe all rental items for one rental request |
| RentalItem | Describe all rental items in system |
| RentalStatus | Describe status of rental request. One rental request has only one status. |
| Repair | Describe all repair requests from customer |
| RepairDetail | Describe all amenities need to repair for one repair request |
| RepairStatus | Describe status of repair request. One repair request has only one status. |
| RequestAmenity | Describe all amenities in one request office |
| RequestOffice | Describe all requests office from customer |
| Role | Describe role of account. One account has only one role. |

### Sequence Diagram

#### 4.2.1 < Guest, Customer> Search office

**Summary:** This diagram used to describe the process of guest search for offices.

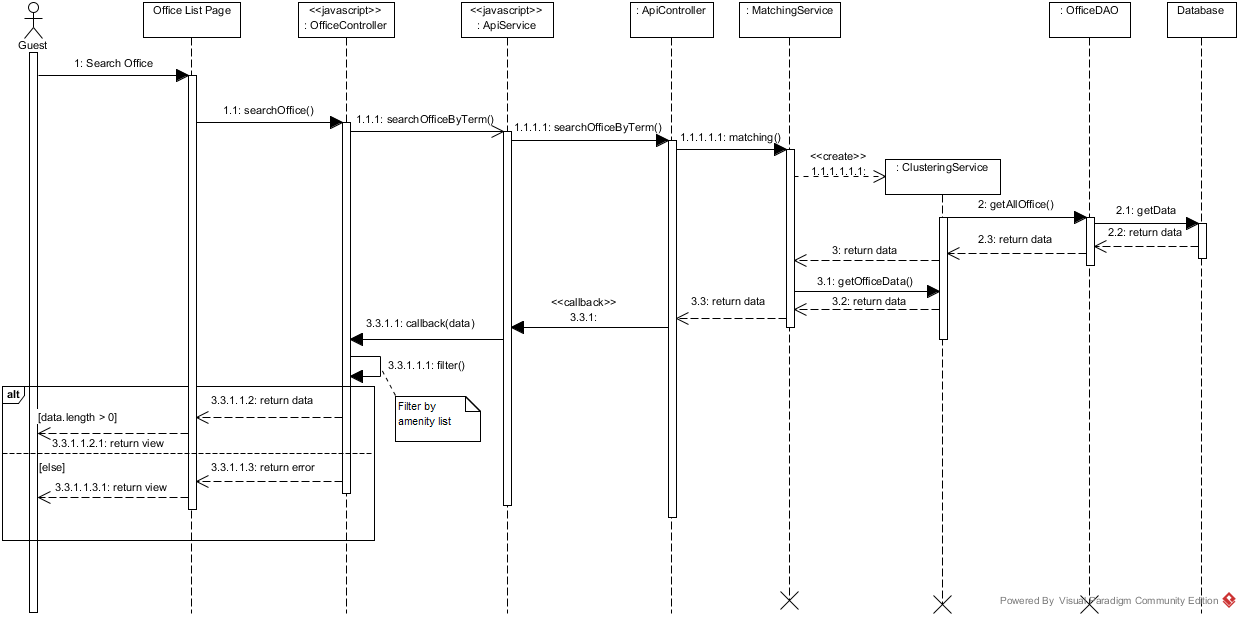


Figure 8: Search Office Sequence Diagram

#### <Owner> Create New Office

**Summary:** This diagram used to describe the process of owner create new office.

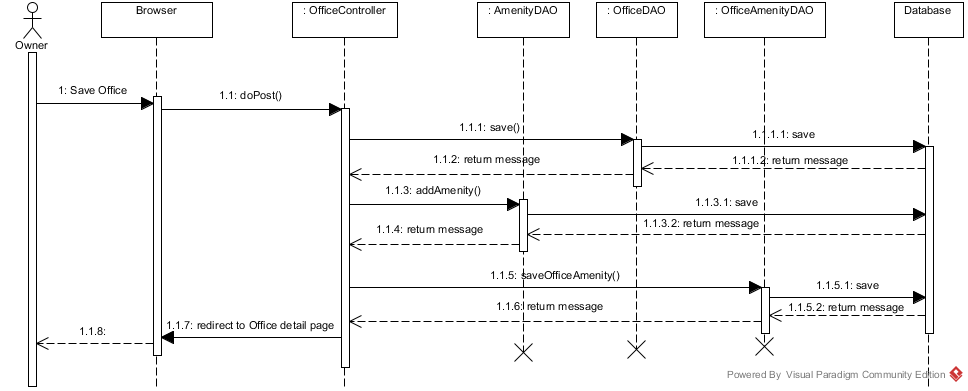
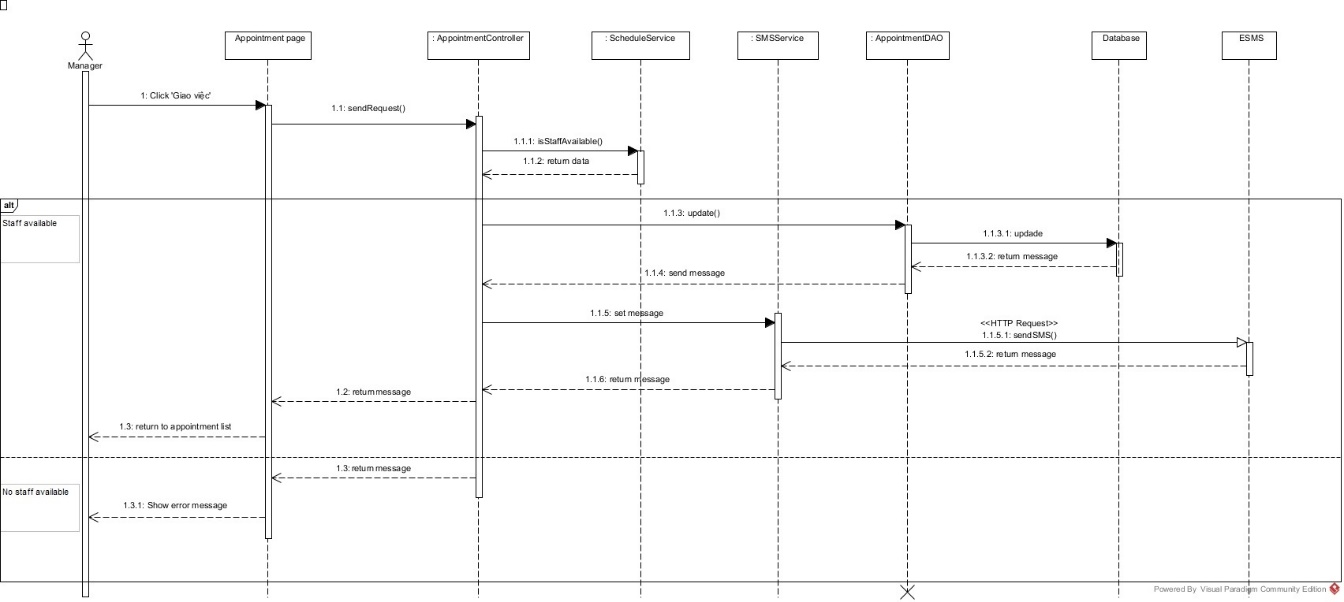


Figure 9: Create New Office Sequence Diagram

#### <Manager> Assign Appointment

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



**Figure 10: Assign Appointment Sequence Diagram**

## Database Design

### Entity-Relationship Diagram

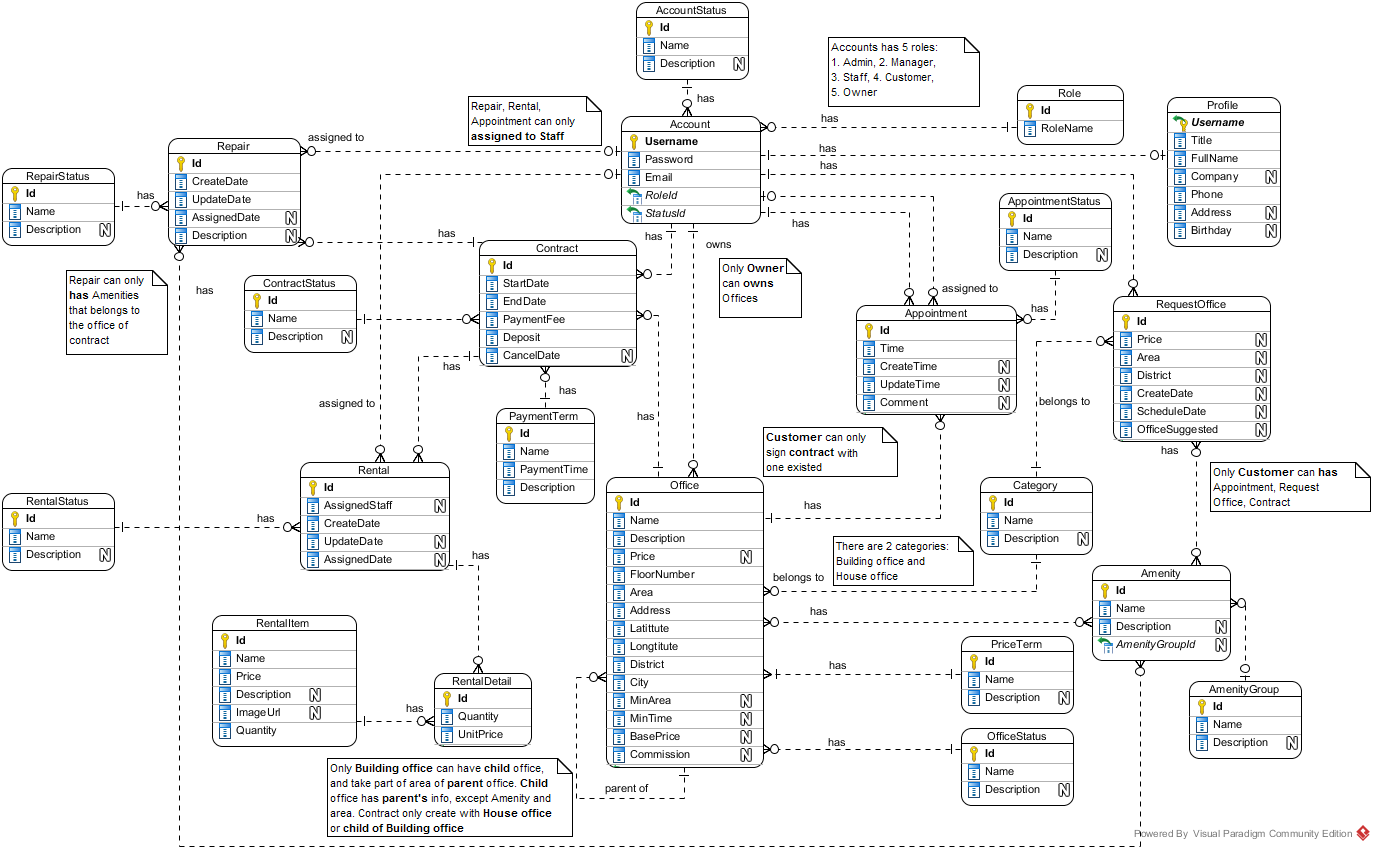


Figure 11: ERD Diagram

### 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. |
| 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 7: Data dictionary

## Algorithms

### Clustering data

#### Definition

K-means is a prototype based clustering technique defining the prototype in terms of a centroid which is considered to be the mean of a group of points and is applicable to objects in a continuous n-dimensional space. (Refer website http://www.hypertextbookshop.com)

#### 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

K-Nearest Neighbor algorithm (KNN) is a method for classifying objects based on the closest training examples in the feature space.  KNN is a type of instance-based learning, or lazy learning where the function is only approximated locally and all computation is deferred until classification. (Refer website http://www.datasciencecentral.com)

#### 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

## Physical Diagram

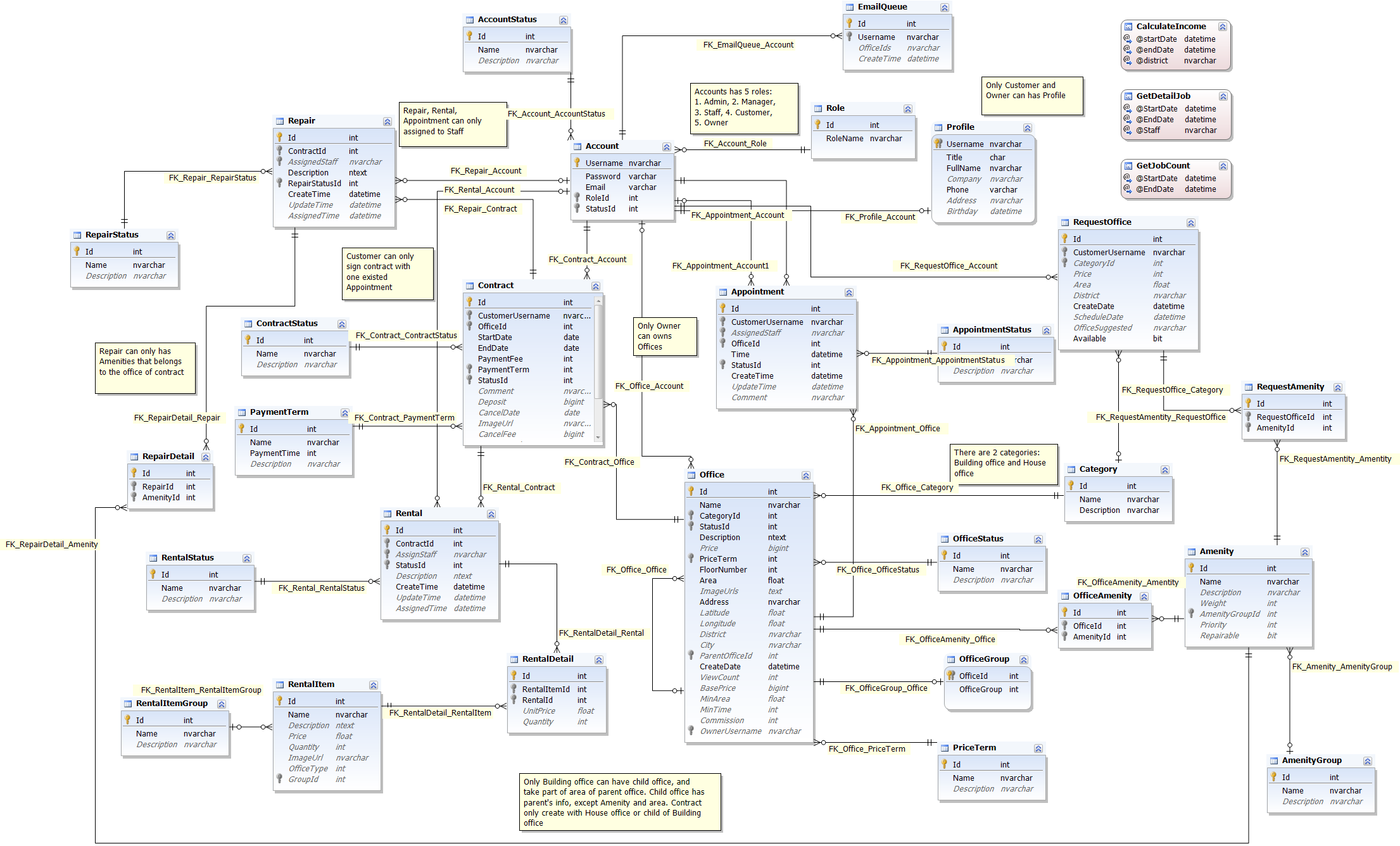


Figure 12: Physical Database Diagram

## 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. | int | No |
| StatusId{FK} | Status of this account. | int | No |
| Unique Key: Email | | | | |
| Foreign key:   * RoleId: Foreign key references to table Role * StatusId: Foreign key references to table Account Status | | | | |

### 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. | int | Yes |
| Priority | Priority of amenity in one group | int | Yes |
| Unique Key: Name | | | | |
| Foreign key:   * AmenityGroupId: Foreign key references to table Amenity Group | | | | |

### 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. | nvarchar(50) | No |
| AssignedStaff{FK} | Staff who is responsibility for the appointment. | nvarchar(250) | Yes |
| OfficeId{FK} | Office where customer want to make appointment. | int | No |
| Time | Time for appointment | datetime | No |
| StatusId{FK} | Status of the appointment. | 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 |
| Foreign key:   * CustomerUsername: Foreign key references to table Account. * AssignedStaff: Foreign key references to table Account. * OfficeId: Foreign key references to table Office. * StatusId: Foreign key references to table appointment status. | | | | |

### 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 |
| Unique Key: Name | | | | |

### 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. | nvarchar(50) | No |
| OfficeId{FK} | Office where customer want to make contract. | 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. | int | No |
| StatusId{FK} | Status of the contract. | int | No |
| Comment | Comment for the appointment | nvarchar(250) | Yes |
| Deposit | Money for deposit of contract | bigint | Yes |
| CancelDate | Time when contract has been accepted to cancel | date | Yes |
| ImageUrl | Image for contract | nvarchar(150) | Yes |
| CancelFee | Fee return when contract return | bigint | Yes |
| Foreign key:   * CustomerUsername: Foreign key references to table Account. * OfficeId: Foreign key references to table Office. * PaymentTerm: Foreign key references to table Payment Term. * StatusId: Foreign key references to table contract status. | | | | |

### 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 |
| Unique key: Name | | | | |

### 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 |
| Foreign key:   * Username: Foreign key references to table Account. | | | | |

### 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. | 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. | 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. | int | Yes |
| CreateDate | Date when office create in database | datetime | No |
| ViewCount | Number views when customer see detail of office | int | Yes |
| BasePrice | Base price for the office | bigint | Yes |
| MinArea | Minimum area when customer rent | float | Yes |
| MinTime | Minimum time when customer rent | int | Yes |
| OwnerName | Office’s owner name | nvarchar(150) | Yes |
| OwnerPhone | Office’s owner phone | nvarchar(15) | Yes |
| OwnerAddress | Office’s owner address | nvarchar(150) | Yes |
| Foreign key:   * CategoryId: Foreign key references to table Category. * PriceTerm: Foreign key references to table Price Term. * ParentOfficeId: Foreign key references to table Office. | | | | |

### 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 the office. | int | No |
| AmenityId{FK} | Amenity of this office | int | No |
| Foreign key:   * OfficeId: Foreign key references to table Office. * AmenityId: Foreign key references to table Amenity. | | | | |

### Table Office Group

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

### 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: Name | | | | |

### 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. | 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 |
| Foreign key:   * Username: Foreign key references to table Account. | | | | |

### 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. | int | No |
| AssignStaff{FK} | Username of staff who responsibility for this rental request. | nvarchar(50) | Yes |
| StatusId{FK} | Status of this rental request. | 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 |
| Foreign key:   * ContractId: Foreign key references to table Contract. * AssignStaff: Foreign key references to table Account. * StatusId: Foreign key references to table Rental Status. | | | | |

### 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. | int | No |
| RentalId{FK} | Id of Request rental. | int | No |
| UnitPrice | Price of one item | float | Yes |
| Quantity | Quantity of rental item for the rental request | int | Yes |
| Foreign key:   * RentalItemId: Foreign key references to table Rental Item. * RentalId: Foreign key references to table Rental. | | | | |

### 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 |
| Unique key: Name | | | | |

### 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 |
| Unique key: Name | | | | |

### 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. | int | No |
| AssignStaff{FK} | Username of staff who responsibility for this repair request. | nvarchar(50) | Yes |
| RepairStatusId{FK} | Status of this repair request. | 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 |
| Foreign key:   * ContractId: Foreign key references to table Contract. * AssignStaff: Foreign key references to table Account. * RepairStatusId: Foreign key references to table Repair Status. | | | | |

### 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. | int | No |
| AmenityId{FK} | Amenity of this repair request. | int | No |
| Foreign key:   * RepairId: Foreign key references to table Repair. * AmenityId: Foreign key references to table Amenity. | | | | |

### 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 |
| Unique key: Name | | | | |

### 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. | int | No |
| AmenityId{FK} | Amenity of this request. | int | No |
| Foreign key:   * RequestOfficeId: Foreign key references to table Request Office. * AmenityId: Foreign key references to table Amenity. | | | | |

### 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. | nvarchar(50) | No |
| CategoryId{FK} | Category of the office. | 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 |
| Foreign key:   * CustomerUsername: Foreign key references to table Account * CategoryId: Foreign key references to table Category. | | | | |

### 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 | | | | |

# Demonstrations Workflow

## <Manager> Create contract

**Summary:** This diagram shows how contract has been created.

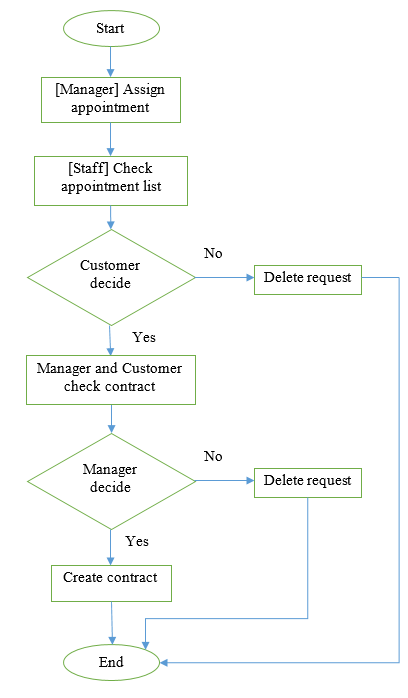


Figure 13: <Manager> Create contract

## <Customer> Request repair

**Summary:** This diagram shows how request repair has been handled

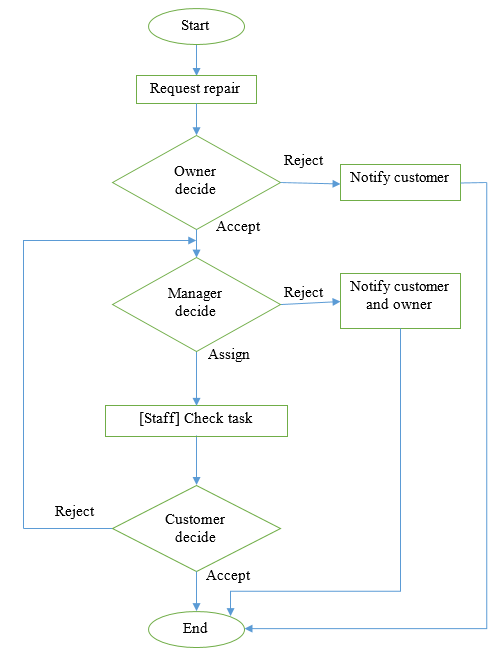


Figure 14: <Customer> Request repair

## <Customer> Request appointment

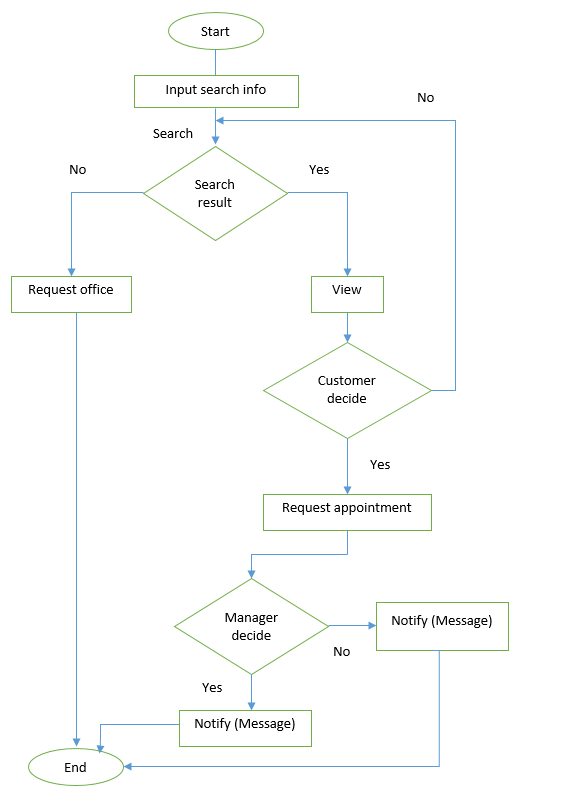
**Summary:** This diagram shows how customer find and get appointment

Figure 15: <Customer> Request appointment

# Task sheet

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **No.** | **Product Deliverables** | **Task** | **TienLX** | **QuocNVH** | **ThanhTT** | **TuanTL** | **Unit** | **Size** |
| 1 | Report1 - Introduction | Project Information | **O** |  |  |  |  |  |
| Introduction |  | **O** |  |  |  |  |
| Current Situation | **O** |  |  |  |  |  |
| Problem Definition |  |  |  | **O** |  |  |
| **Proposed Solution** |  |  |  |  |  |  |
| Feature functions |  |  | **O** |  |  |  |
| Advantages and disadvantages | **O** | **O** | **O** | **O** |  |  |
| Functional Requirements | **O** |  |  |  |  |  |
| Role and Responsibility | **O** |  |  |  |  |  |
| 2 | Report2- Software Project Management Plan | Problem Definition |  | **O** |  | **O** |  |  |
| **Project organization** |  |  |  |  |  |  |
| Software Process Model |  | **O** |  |  |  |  |
| Roles and Responsibilities | **O** |  |  |  |  |  |
| Tools and Techniques | **O** |  |  |  |  |  |
| **Project management plan** |  |  |  |  |  |  |
| Software development life cycle | **O** |  |  | **O** |  |  |
| Phase Detail | **O** |  | **O** |  |  |  |
| All Meeting Minutes |  | **O** | **O** |  |  |  |
| Coding Convention |  | **O** |  |  |  |  |
| 3 | Report 3- Software Requirement Specification | User Requirement Specification | **O** |  | **O** | **O** |  |  |
| **System Requirement Specification** |  |  |  |  |  |  |
| External Interface Requirements |  |  | **O** |  |  |  |
| **Functional Requirement** |  |  |  |  |  |  |
| <Guest>Overview Use Case |  |  | **O** |  |  |  |
| <Customer>Overview Use Case |  | **O** | **O** |  |  |  |
| <Staff> Overview Use Case | **O** |  |  |  |  |  |
| <Admin> Overview Use Case |  |  |  | **O** |  |  |
| <Manager> Overview Use Case | **O** | **O** |  | **O** |  |  |
| <System>Overview Use Case | **O** | **O** |  |  |  |  |
| Non-functional Requirement |  |  | **O** |  |  |  |
| Conceptual Diagram | **O** |  |  |  |  |  |
| 4 | Report 4- Software Design Description | Design Overview | **O** |  | **O** |  |  |  |
| System Architectural Design |  | **O** |  |  |  |  |
| Component Diagram | **O** |  |  |  |  |  |
| **Detailed Description Explanation** |  |  |  |  |  |  |
| **Class Diagram** |  |  |  |  |  |  |
| Overview | **O** |  | **O** |  |  |  |
| Entity package | **O** |  | **O** |  |  |  |
| DAO package |  |  |  | **O** |  |  |
| Service Package | **O** |  |  |  |  |  |
| Controller Package |  |  |  | **O** |  |  |
| Mobile Package | **O** |  |  |  |  |  |
| **Interaction Diagram** |  |  |  |  |  |  |
| <Guest> Sequence Diagram |  |  | **O** |  |  |  |
| <Customer> Sequence Diagram |  | **O** | **O** |  |  |  |
| <Staff> Sequence Diagram | **O** |  |  |  |  |  |
| <Admin> Sequence Diagram |  |  |  | **O** |  |  |
| <Manager> Sequence Diagram | **O** | **O** |  | **O** |  |  |
| <System> Sequence Diagram | **O** | **O** |  |  |  |  |
| **User Interface Design** |  |  |  |  |  |  |
| Guest Interface Design | **O** | **O** | **O** |  |  |  |
| Customer Interface Design | **O** | **O** | **O** |  |  |  |
| Manager Interface Design | **O** | **O** | **O** | **O** |  |  |
| Mobile Interface Design | **O** |  |  |  |  |  |
| **Database Design** |  |  |  |  |  |  |
| Logical Diagram |  |  |  | **O** |  |  |
| Data Dictionary |  | **O** |  |  |  |  |
| **Algorithms** |  |  |  |  |  |  |
| Clustering data | **O** |  |  |  |  |  |
| Matching data | **O** |  |  |  |  |  |
| 5 | Report 5 - Software Implementation and Test Document | **Setup Environment** | **O** |  |  |  |  |  |
| **Coding admin functions** |  |  |  |  |  |  |
| Login / Log out |  |  |  | **O** | 1 | 1 |
| Manage account |  |  |  | **O** | 3 | 1 |
| **Coding manager / staff / owner functions** |  |  |  |  |  |  |
| Manage contract |  | **O** |  |  | 2 | 4 |
| Manage appointment |  | **O** |  |  | 2 | 2 |
| Manage repair/rental |  |  | **O** |  | 4 | 2 |
| Manage office | **O** |  |  |  | 2 | 4 |
| Manage rental item |  |  | **O** |  | 2 | 1 |
| Manage amenity |  |  |  | **O** | 2 | 2 |
| Manage amenity group |  |  |  | **O** | 2 | 1 |
| Statistics |  |  |  | **O** | 2 | 3 |
| Calendar | **O** |  |  |  | 2 | 2 |
| Validation | **O** | **O** | **O** |  | 5 | 4 |
| Pagination | **O** |  |  |  | 3 | 1 |
| Change repair flow |  |  | **O** |  | 4 | 3 |
| Change create office flow | **O** |  |  |  | 1 | 2 |
| Print contract |  | **O** |  |  | 3 | 3 |
| **Customer functions** |  |  |  |  |  |  |
| Login / Register |  |  | **O** |  | 1 | 1 |
| Search office | **O** |  |  |  | 2 | 5 |
| View home page |  | **O** |  |  | 1 | 1 |
| View office detail |  |  | **O** |  | 1 | 2 |
| Request appointment | **O** |  |  |  | 1 | 1 |
| Manage contract |  |  | **O** |  | 2 | 3 |
| Manage repair/rental |  |  | **O** |  | 2 | 2 |
| Request repair |  |  | **O** |  | 1 | 2 |
| Request rental |  | **O** |  |  | 1 | 2 |
| Manage request office |  | **O** |  |  | 1 | 2 |
| **System functions** |  |  |  |  |  |  |
| Clustering office | **O** |  |  |  | 2 | 5 |
| Schedule |  | **O** |  |  | 3 | 5 |
| Suggest office |  | **O** |  |  | 2 | 4 |
| Suggest assign job | **O** |  |  |  | 2 | 4 |
| Delete outdated appointment |  | **O** |  |  | 2 | 3 |
| Handling conflict on job assign | **O** |  |  |  | 2 | 4 |
| **Mobile functions** |  |  |  |  |  |  |
| Coding front app | **O** |  |  |  | 2 | 1 |
| Coding mobile adapter | **O** |  |  |  | 1 | 2 |
| Coding backend | **O** |  |  |  | 2 | 2 |
| Introduction |  |  | **O** |  |  |  |
| Database Relationship Diagram | **O** |  |  |  |  |  |
| **Performance Measures** |  |  |  |  |  |  |
| Clustering Performance | **O** |  |  |  |  |  |
| Matching Performance | **O** |  |  |  |  |  |
| **Test Plan** |  |  |  |  |  |  |
| Features to be tested |  | **O** |  |  |  |  |
| Features not to be tested |  |  | **O** |  |  |  |
| **System Testing Test Case** |  |  |  |  |  |  |
| Overview |  |  | **O** |  |  |  |
| Guest Test Case | **O** |  |  |  |  |  |
| Customer Test Case |  | **O** | **O** |  |  |  |
| Manager Test Case |  | **O** | **O** | **O** |  |  |
| System | **O** | **O** |  |  |  |  |
| 6 | Report 6 - Software User's Manual | **Installation Guide** |  |  |  |  |  |  |
| Setting up environment at server side | **O** |  |  | **O** |  |  |
| Deployment at server side | **O** |  |  | **O** |  |  |
| Client side environment setting | **O** |  |  | **O** |  |  |
| **User Guide** |  |  |  |  |  |  |
| Guide for customer |  | **O** | **O** | **O** |  |  |
| Guide for manager on web application |  | **O** |  |  |  |  |
| Guide for staff on mobile application | **O** |  |  |  |  |  |
| Appendix |  |  |  | **O** |  |  |